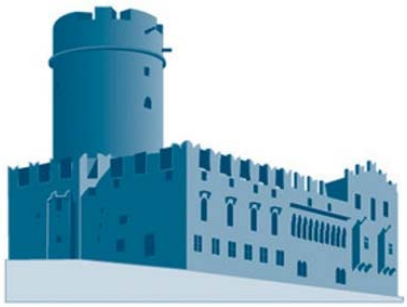


SDN4FNS 2013

2013 Workshop on Software Defined Networks for Future Networks and Services

November 11-13, 2013
Trento, ITALY

IEEE
Future Directions Committee

IEEE COMMUNICATIONS SOCIETY

IEEE computer society

WIRELESS WORLD RESEARCH FORUM

IEEE Catalog number: CFP13SDN-ART ISBN: 978-1-4799-2781-4

IEEE Conference Publication Form for 2013 IEEE SDN for Future Networks and Services (SDN4FNS)

IEEE Catalog Number: CFP13SDN-ART (Article)
CFP13SDN-USB (USB)
ISBN: 978-1-4799-2781-4 (Article)
978-1-4799-2780-7 (USB)

Printed copies of this publication are available from:

Curran Associates, Inc
57 Morehouse Lane
Red Hook, NY 12571 USA

Phone: (845) 758-0400
Fax: (845) 758-2633
E-mail: curran@proceedings.com

Produced by IEEE eXpress Conference Publishing

For information on producing a conference proceedings and receiving an estimate, contact conferencepublishing@ieee.org

http://www.ieee.org/conferencepublishing

Message from the General Chair of SDN4FNS 2013

The Workshop SDN4FNS 2013 was kindly hosted by EIT ICT Labs in Trento (11th – 13th November, 2013).

The Workshop embraced several main aspects of Software Defined Networks, from technology to business models, from Users' perception to regulatory frameworks, bringing together a variety of interests and skills from all over the world. Main scope was to contribute elaborating a roadmap and a charter for IEEE Societies in this field.

The 32 accepted papers, together with the 3 keynote speeches, were organized into 3 technical Tracks: "Telecom and Internet SDN Scenarios", "Hardware and Software Innovation to enable SDN " and "Regulatory, Biz, Techno-Economic sustainability".

The scientific success of the workshop was due to hard work of a lot of people. Our appreciations goes to the authors of the papers, the keynote speakers, the Track Chairs and the members of the Technical Program and Steering Committees.

Special thanks to EIT ICT Labs for hosting the Workshop.

Antonio Manzalini
SDN4FNS 2013 General Chair

SDN4FNS 2013 Committees

General Chair

Antonio Manzalini (Telecom Italia)

Organizing Committee

Roberto Saracco (EIT ICT Laboratories)
Bobby Wong (IEEE Future Directions Program Director)
Ezio Zerbini (Ericsson)
David Soldani (Huawei Technologies)
Heiner Stuttgen (NEC Laboratories Europe)
Stephen F. Bush (GE Global Research)
Laura Meijere (EIT ICT Laboratories)

Publications Chairs

Franco Callegati (Università di Bologna)
Walter Cerroni (Università di Bologna)

Track Chairs

Telecom and Internet SDN Scenarios: Cagatay Buyukkoc (AT&T)
Hardware and Software Innovations to enable SDN and NFV: Eliezer Dekel (IBM Research-Haifa)
Regulatory, Biz, Techno-Economic sustainability: Andreas Gladisch (Deutsche Telekom - T-Labs)

Technical Program Committee

Henrik Abramowicz (Ericsson)
Niranth Amogh (Huawei Technologies)
Sergio Beker (Huawei Technologies)
Raouf Boutaba (University of Waterloo)
Cagatay Buyukkoc (AT&T)
Franco Callegati (Università di Bologna)
Jiannong Cao (Hong Kong Polytechnic Univ)
Walter Cerroni (Università di Bologna)
Prosper Chemouil (Orange Labs)
Alexander Clemm (Cisco Systems, Inc.)
Noel Crespi (Institut Mines-Télécom, Télécom SudParis)
Eliezer Dekel (IBM Research - Haifa)

Hans-Martin Foisel (Deutsche Telekom)
Erol Gelenbe (Imperial College London)
Alexander Gelman (IEEE Communications Society)
Andreas Gladisch (Deutsche Telekom - T-Labs)
Ernest Kaempfer (Intel Corporation)
Diego Lopez (Telefonica I+D)
Guido Maier (Politecnico di Milano)
Antonio Manzalini (Telecom Italia)
Carmen Mas Machuca (Technische Universität Munchen)
Julius Mueller (Technische Universität Berlin)
Saverio Niccolini (NEC Laboratories Europe)
Dan Pitt (Open Networking Foundation)
Fulvio Risso (Politecnico di Torino)
Elio Salvadori (Create-Net)
Fabian Schneider (NEC Laboratories Europe)
Marcus Schöller (NEC Laboratories Europe)
Stefano Secci (University Pierre et Marie Curie - Paris 6)
Wenyu Shen (NTT Network Innovation Laboratories)
David Soldani (Huawei Technologies)
Mehmet Ulema (Manhattan College)
Albert Vico Oton (Fundació i2CAT)
M. Can Vuran (University of Nebraska at Lincoln)
Hagen Woesner (BISDN GmbH)
Linda Jiang Xie (University of North Carolina at Charlotte)
Ezio Zerbini (Ericsson)
Nan Zhang (Aalto University)

2013 Software Defined Networks for Future Networks and Services - Technical Program

Monday, November 11

09:00 - 10:00

Keynote 1: China Mobile and SDN

Chih-Lin I (China Mobile)

10:00 - 11:30

Session 1: Software Defined Wireless and Mobile Networks

An SDN-based Network Architecture for Extremely Dense Wireless Networks
H. Ali-Ahmad, C. Cicconetti, A. de la Oliva, V. Mancuso, M. R. Sama, P. Seite and S. Shanmugalingam

A Virtual SDN-enabled LTE EPC Architecture: a case study for S-/P-Gateways functions
A. Basta, W. Kellerer, M. Hoffmann, K. Hoffmann and E. D. Schmidt

SDN as an enabler for inter-technology load balancing and admission control
S. Namal, I. Ahmad, A. Gurtov and M. Ylianttila

Enabling Secure Mobility with OpenFlow
S. Namal, A. Gurtov and M. Ylianttila

12:00 - 13:30

Session 2: Traffic Engineering, Optimization and Performance Evaluation

SDN for Inter Cloud Networking
M. Mechtri, D. Zeghlache, I. Houidi and W. Louati

Virtual Links Mapping in Future SDN-enabled Networks
R. Trivisonno, I. Vaishnavi, R. Guerzoni, Z. Despotovic, A. Hecker, S. Beker and D. Soldani

Scalable On-Demand Network Management Module for Software Defined Telecommunication Networks
J. Mueller, A. Wierz and T. Magedanz

Live Migration of Virtualized Edge Networks: Analytical Modeling and Performance Evaluation
F. Callegati and W. Cerroni

14:30 - 16:20

Session 3: SDN Challenges

Evolution and Challenges of Software Defined Networking
Á. Valdivieso, L. Barona and J. García Villalba

SDN Security: A Survey
S. Scott-Hayward, G. O'Callaghan and S. Sezer

Tearing down the Protocol Wall with SDN

J. Crowcroft, Y. Bar Geva and H. Oliver

Session Border Controller virtualization towards "service-defined" networks based on NFV and SDN
P. Paglierani and G. Monteleone

16:45 - 18:15

Panel 1: Performance Implications of SDN

Chair: Jean Walrand (University of California, Berkeley, USA)

Panelists:

Akash Deshpande (Cisco, USA)
Anja Feldmann (TU Berlin, Germany)
Slawomir Kuklinski (Orange Lab, Poland)
David Hausheer (TU Darmstadt, Germany)

Tuesday, November 12

08:30 - 09:30

Keynote 2: The Software Defined Operator

Axel Clauberg (Deutsche Telecom)

09:30 - 11:20

Session 4: Frameworks for SDN Architectures

Software Service Defined Network: Centralized Network Information Service
J. Zhu, W. Xie, Li Li, M. Luo and Wu Chou

OpenNaaS based SDN framework for dynamic QoS control
I. Bueno-Rodríguez, J. Aznar, E. Escalona, J. Ferrer Riera and J. García-Espín

PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks
M. F. Bari, S. Chowdhury, R. Ahmed and R. Boutaba

Towards a distributed SDN control: Inter-platform signalling among flow processing platforms
F. Salvestrini, G. Carrozzo and N. Ciulli

OpenFlow rules interactions: definition and detection
R. Bifulco and F. Schneider

11:45 - 13:30

Session 5: Advanced SDN Services and Validation

Using SDN for cloud services provisioning: the XIFI use-case
E. Escalona, J. Aznar, L. M. Contreras, O. González de Dios, G. Cossu, E. Salvadori and F. Facca

Boosting Cloud Communications Through A Crosslayer Multipath Protocol Architecture
M. Coudron, S. Secci, G. Maier, G. Pujolle and A. Pattavina

Enabling Information Centric Networking in IP Networks Using SDN
M. Vahlenkamp, F. Schneider, D. Kutscher and J. Seedorf

End to End Solution Testing over SDN - Future challenges for service providers
S. Chandra and R. K

EmPOWER: A Testbed for Network Function Virtualization Research and Experimentation
R. Riggio, T. Rasheed and F. Granelli

14:30 - 16:20

Session 6: Network Virtualization

Edge-to-Edge Virtualization and Orchestration in Heterogeneous Transport Networks
D. Siracusa, E. Salvadori and T. Rasheed

A Node Plug-in Architecture for Evolving Network Virtualization
Y. Kanada

Network Virtualisation Trends: virtuallly anything is possible by connecting the unconnected
E. Patouni, A. Merentitis, P. Panagiotopoulos, A. Glentis and N. Alonistioti

On the implementation of NFV over an OpenFlow infrastructure: Routing Function Virtualization
J. Batalle, J. Ferrer Riera, E. Escalona and J. García-Espín

CONTENT Project: Considerations towards a Cloud-based Internetworking Paradigm
K. Katsalis, T. Korakis, G. Landi, G. Bernini, B. Rahimzadeh Rofoee, S. Peng, M. Anastasopoulos, A. Tzanakaki, D. Christofi, M. Georgiades, R. Larsen, J. Ferrer Riera, E. Escalona and J. García-Espín

16:45 - 18:15

Panel 2: Hardware and Software Architectures for SDN

Chair: Alex Galis (University College London, United Kingdom)

Panelists:

Akihiro Nakao (University of Tokyo, Japan)
Thomas M. Bohnert (Zurich University, Switzerland)
Bruno Chatras (Orange Lab, France)

Wednesday, November 13

08:30 - 09:30

Keynote 3: Linear Evolution Leading to a Revolution: Technology Disrupting the Biz

Roberto Saracco (EIT ICT Laboratories, Italy)

09:30 - 11:20

Session 7: Future SDN Scenarios

IEEE Software Defined Network Initiative
M. Ulema, N. Amogh, R. Boutaba, C. Buyukkoc, A. Clemm, M. Vuran, L. J. Xie, A. Manzalini and R. Saracco

Software Networks at the Edge: a shift of paradigm
A. Manzalini and R. Saracco

Research Directions in Network Service Chaining
W. John, K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert and C. Meirosu

Some Controversial Opinions on Software-Defined Data Plane Services
F. Risso, A. Manzalini and M. Nemirovsky

Softwarization of Future Networks and Services - Next Generation SDNs
A. Galis, S. Clayman, L. Mamatas, J. Rubio-Loyola, A. Manzalini, S. Kukliński, J. Serrat and T. Zahariadis

11:45 - 13:15

Panel 3: On the business models, policies aspects and regulatory frameworks: Are we ready for SDN?

Chair: David Soldani (Huawei Technologies Duesseldorf GmbH & European Research Centre, Germany)

Panelists:

Ruediger Martin (DG CONNECT, EC, Beligium)
Bram Naudts (iMind, Belgium)
Eliezer Dekel (IBM Research, Israel)
Nicholas Economides (NYU, USA)
Dang Wenshuan (Huawei, China)

TABLE OF CONTENTS

An SDN-based Network Architecture for Extremely Dense Wireless Networks

Hassan Ali-Ahmad[1], Claudio Cicconetti[2*], Antonio de la Oliva[3],
Vincenzo Mancuso[4], Malla Reddy Sama[1], Pierrick Seite[1], Sivasothy Shanmugalingam[1],
[1]*Orange, France*, [2]*Intecs, Italy*, [3]*Universidad Carlos III de Madrid, Spain* [4]*IMDEA Networks, Spain*
[*] corresponding author e-mail: claudio.cicconetti@intecs.it
CROWD project: http://www.ict-crowd.eu/, @FP7CROWD

***Abstract*—Telecommunications networks are undergoing major changes so as to meet the requirements of the next generation of users and services, which create a need for a general revised architectural approach rather than a series of local and incremental technology updates. This is especially manifest in mobile broadband wireless access, where a major traffic increase is expected, mostly because of video transmission and cloud-based applications. The installation of a high number of very small cells is foreseen as the only practical way to achieve the demands. However, this would create a struggle on the mobile network operators because of the limited backhaul capacity, the increased energy consumption, and the explosion of signalling. In the FP7 project CROWD, Software Defined Networking (SDN) has been identified as a solution to tame extreme density of wireless networks. Following this paradigm, a novel network architecture accounting for MAC control and Mobility Management has been proposed, being the subject of this paper.**

I. Introduction

Recently, mobile users demand on data traffic is increasing dramatically. Operators statistics show that the usage of mobile data traffic has doubled during the last year [1]. This is expected to continue in this decade especially with the deployment of 4G networks, resulting in an explosion in mobile Internet traffic. In order to cope with such rapid explosion, mobile network operators have already started to push for denser, heterogeneous deployments [2]. Indeed, current technology needs to steer towards efficiency, to avoid unsustainable energy consumption and network performance implosion due to interference and signaling overhead.

In fact, interference due to uncoordinated resource sharing techniques represents a key limiting factor in the design of dense wireless networks, where resources are limited due to either the costs for licensed bands or the proliferation of hot spots in license-exempt bands. This situation calls for the deployment of *network controllers* with either *local* or *regional* scope, with the aim of orchestrating the access to wireless and backhaul resources of the various wireless network elements.

Moreover, mobility management plays a key role in dense environments, where mobiles can easily undergo several handovers during the same connectivity session. Therefore, there is a need to define novel mobility management mechanisms that are both distributed and offered dynamically. They should be distributed in order to avoid any network bottleneck or single point of failure, and to provide better reliability. They should be activated/deactivated dynamically as needed, in order to globally reduce their signaling load and to increase the achieved performances.

In this paper we present an architecture that tackles the two key challenges highlighted above: interference and mobility management in wireless dense networks. Furthermore, we show that a SDN-based networking approach can be suitably adopted to design the next generation of dense wireless mobile networks.

Regarding interference, in the context of small dense cells, research have shown that simply measuring performance via bit error rate, SINR distributions, or spectral efficiency directly, is no longer very relevant. Instead, rate distribution (user-perceived, i.e., accounting for load) or area spectral efficiency becomes relevant [3]. Therefore, smart cell association policies for dense networks should be based on *user-perceived* rates rather than Base Station (BS) signal strength. Moreover, the seminal work of Knopp and Humblet [4] showed that throughput performance can leverage the presence of time-variable interference conditions by utilizing the wireless medium in an opportunistic way. As a matter of fact, dense wireless deployments show the characteristics typically needed to fully exploit opportunistic gain, especially when network heterogeneity comes into play (e.g., irregular deployments of heterogeneous infrastructures elements including, micro, pico and femtocells are common and have been studied for cellular networks deployment [5], [6]).

Regarding Mobility Management, current IP mobility management solutions pose the following problems [7], which are exacerbated in dense networks: *sub-optimal routing*, since typical solutions rely on a central entity to forward packets to the current location of the terminal, hence providing paths that are generally longer than the direct path between the terminal and its communication peer; *scalability problems*, because existing mobile networks have to be dimensioned to support all the traffic traversing the central anchors and *reliability*, since centralized solutions share the problem of being more prone to reliability problems, as the central entity is potentially a single point of failure.

The FP7 project Connectivity management for eneRgy Optimised Wireless Dense networks (CROWD) has the ambitious objectives of tackling the two challenges above in the

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

specific context of extremely dense and heterogeneous wireless networks [8]. It is worth noting that the latter is a very high potential topic, which has been also selected as one of the five macro-scenarios of the future 5G of telecommunications by the FP7 project METIS[1]. To achieve the required level of flexibility and reconfigurability, and at the same time yielding an energy-efficient network infrastructure in both the radio access part and the backhaul, SDN has been recognized as a basic enabler and driver of innovation within the project.

In this paper we report the initial results obtained within CROWD. Specifically, in Section II we describe the proposed functional architecture, along with some example new functions that can be natively supported by the network. Section III proposes an illustration of the novel mobility management procedures defined so far in CROWD. Conclusions are drawn in Section IV.

II. Network Model and Control Functions

We now describe the network model and network architecture proposed in CROWD. The goal of the proposed architecture is to leverage the heterogeneity of dense wireless deployments, both in terms of radio condition and non-homogeneous technologies. Specifically, the CROWD architecture offers the tools to orchestrate the network elements in a way that intra-system interference is mitigated, channel opportunistic transmission/reception techniques can be enabled, and energy efficiency can be boosted. Furthermore, as detailed in Section III, the architecture accounts for innovative mobility management mechanisms.

As illustrated in the bottom part of Fig. 1, an extremely dense and heterogeneous network consists of two domains of (physical) network elements: backhaul and Radio Access Network (RAN). The latter is expected to become increasingly heterogeneous in the next years in terms of technologies (e.g., 3G, LTE, WiFi), cell ranges (e.g., macro-/pico-/femto-cells), and density levels (e.g., from macro-cell BS coverage in underpopulated areas to several tens or hundreds of potentially reachable BSs in hot spots). Such heterogeneity also creates high variability over time due to statistical multiplexing, mobility of users, and variable-rate applications (chiefly video streaming). Therefore, to achieve optimal performance at all locations at any time, reconfiguration of the network elements is required at all time scales, from very fast (i.e., order of ms to account for, e.g., indoor fading effects) to relatively long (i.e., order of hours to follow mobility patterns, e.g., commuting of citizens between work places and residential areas). This equally affects both the backhaul and the RAN.

We propose to tackle this issue by following an SDN-based approach to network control: the logic for network optimisation is delegated from the network elements to a set of *controllers*, which are virtual entities deployed dynamically over the physical devices depending (among the others) on the actual network load and the capacity constraints. These controllers are technology-agnostic and vendor-independent,

[1]https://www.metis2020.com/.

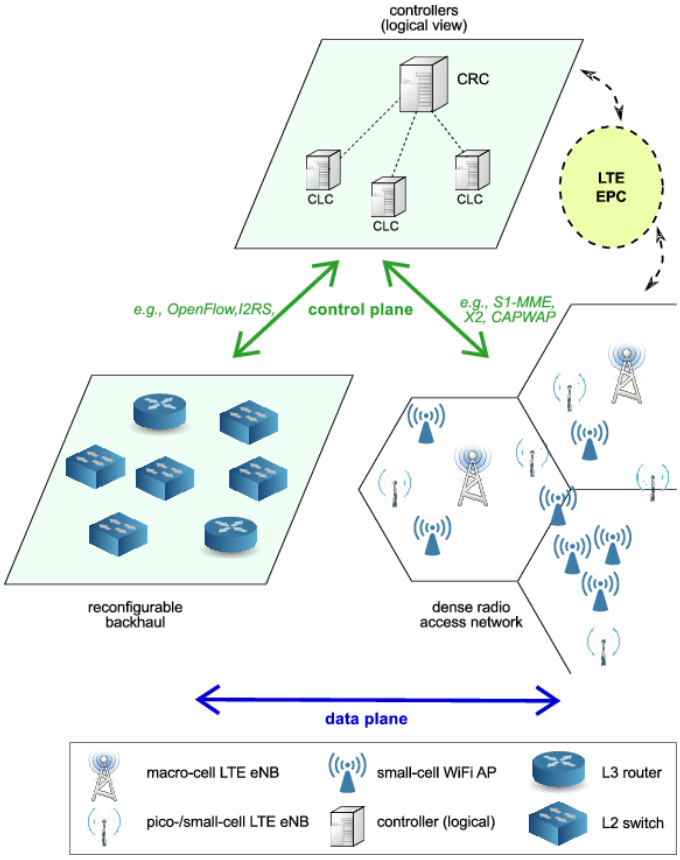


Fig. 1. Network control architecture.

which allows full exploitation of the diversity of deployment/equipment characteristics. Furthermore, they expose a so-called Northbound interface, which is an open Application Programming Interface (API) to the control applications. The Northbound interface does not need be concerned on either the details of the acquisition of the data from the network or the enforcement of decisions made. Instead, a Southbound interface is responsible for managing the interaction between controllers and network elements.

The control plane is shown in the top part of Fig. 1. As can be seen, two types of controllers are indicated: the CROWD Regional Controller (CRC), which is a logically centralised entity that executes long-term optimisations, and the CROWD Local Controller (CLC), which runs short-term optimisations. The CRC only requires aggregate data from the network, and it is in charge of the dynamic deployment and life cycle management of the CLCs, which require detailed and instantaneous data from the network. For this reason, CLCs only cover a limited number of BSs (called a *district* according to the CROWD project terminology).

The CLC can be hosted by a backhaul/RAN node itself, e.g., a macro-cell BS, so as to keep the fast optimisation intelligence to the edge of the network. On the other hand, the CRC is likely to run on dedicated hardware in some network operator data centre.

In Table I we report some key control applications that

TABLE I
EXAMPLE CONTROL APPLICATIONS.

Names	Objective	Description
LTE interference mitigation	Increased capacity	Dynamically reduce the downlink and uplink interference in a district with multiple tiers of LTE eNBs (macro-/pico-/femto-) via the adaptation of the ABSF patterns, based on the channel quality reports sent by the UEs (downlink) and on the physical measurements taken by the eNBs (uplink), as well as on the current traffic load and QoS constraints, if any.
WLAN optimisation	Increased capacity	Dynamically adapt some configuration parameters of the IEEE 802.11 access points, namely the transmission power and the medium access parameters (AIFS, CWmin, CWmax, TXOPmax) on a per-station basis, depending on the current load of access points, the current transmission rates of stations, and other PHY/MAC layer measurements available at the access points (e.g., retransmissions per station, SINR).
LTE access selection	Increased capacity or energy efficiency	Dynamically associate UEs to one of the LTE eNBs (macro-/pico-/femto-) in a district so as maximise either the capacity or the energy efficiency, under the constraints imposed by the backhaul network. Energy efficiency is achieved by aggregating UEs into as few eNBs as possible (and also possibly selecting those eNBs which are connected to the same backhaul network elements) so that unused devices can be switched off.
Power cycling	Energy efficiency	Periodic reconfiguration of the power cycling pattern of base stations, access points, and backhaul network elements to minimise energy consumption, based on the estimated utilisation and traffic load and/or based on a resource-on-demand availability paradigm.
Offloading	Increased capacity	Dynamic selection of best path, also including point of access selection, for a given traffic flow to maximise network capacity. Downlink and uplink traffic can use different points of access (e.g., uplink goes via the femto-cell eNB or 802.11 access point, while downlink is routed through the macro-cell eNB). Best path selection is made based on QoS and load balancing constraints.

have been identified in CROWD, which are enabled by the proposed network architecture. The control applications described in the table offer a sample of the wide spectrum of applications and technologies that can be targeted by means of SDN-based controllers. Specifically, the proposed control applications range from interference mitigation in LTE, using a multi-tier scheduling scheme among neighbor base stations, e.g., adopting the Almost Blank Sub-Frame (ABSF) paradigm, to innovative LTE access selection schemes, with association and re-association mechanisms that consider energy efficiency in addition to user Quality of Service (QoS). CROWD also targets the optimization of WLANs, which is achieved by fine tuning MAC/PHY parameters of access points and stations on a per-node basis. Since energy efficiency is fundamental for operators and for environmental issues, CROWD proposes, among others, a *Power cycling* control application, to dynamically reconfigure the network and the status of network nodes according to traffic demands. Eventually, CROWD proposes control applications for networks consisting of both LTE and 802.11 devices, e.g., the *Offloading* control application envisions the utilization of load balancing and relay techniques that span across multiple RATs and multiple technologies. The CROWD vision aims at providing a common set of functions as a Southbound interface which can be used by the control applications to build a new set of services such as the ones explained above. For example, in order to fulfil the Southbound requirements of the *WLAN Optimisation* function in Table I, we are studying possible extensions to the CAPWAP [9] protocol to carry the required commands and parameters to the 802.11 access point. Before discussing alternative approaches currently available and under study, we next show that an interesting business process can be built on top of the architecture proposed in CROWD and described above.

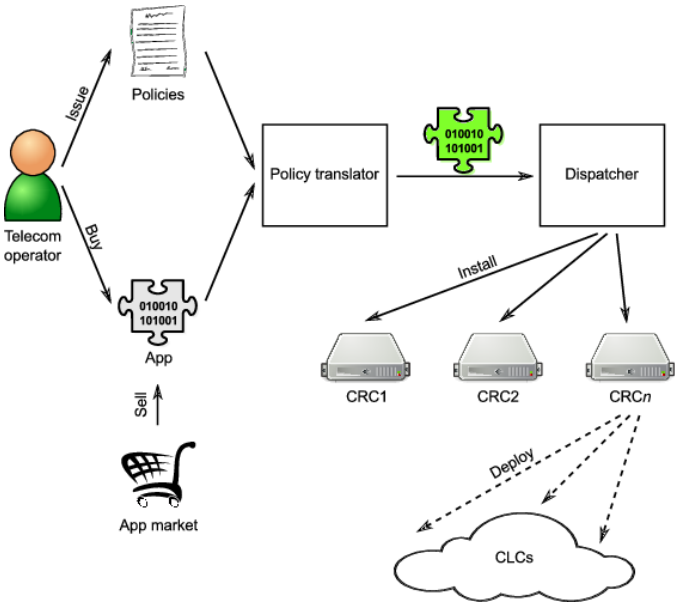


Fig. 2. High-level business process.

A. High-level business process

As illustrated in Fig. 2, the CROWD approach enables the creation of a market of control applications developed by third-party service providers. The telecom operator can

buy such control applications (flexible billing models could be also exploited, as inspired from cloud computing business models) and has to define a set of high-level policies which it wants to enforce on its network, or part thereof. For instance, the operator aims to maximise energy efficiency while the network is lightly loaded or to maximize capacity by favoring users with a high bit-rate, while guaranteeing others with a minimum level of QoS even at high loads. The control application and the policies are passed as input to a *policy translator* which configures the control application at a low level based on the high-level objectives expressed by the telecom operator. A *dispatcher* then installs the configured application on all the CRCs of the telecom operators' network. Each CRC dynamically partitions its network in a number of *districts*, deploying one CLC per district, depending on a number of factors, such as processing requirements of the control application (e.g., a complex application will require smaller districts otherwise the computation time would be too high for the optimisation to be effective), number of mobile devices and their QoS requirements and actual traffic (e.g., lightly loaded network may consist of larger districts, in terms of the number of base stations, compared to heavily loaded networks), backhaul characteristics and constraints (e.g., a backhaul bottleneck may suggest the creation of small districts, in terms of the number of base stations, to mitigate the performance penalty due to the controller signalling overhead). In general, more than one application can be in use within the same region/network.

B. Alternative approaches

We now review some alternative approaches, in a mature or emergent phase, which tackles similar architectural issues as CROWD, also clarifying the differences and synergies with the approach proposed in this project.

Operations Support Systems (OSSs) and their evolutions, e.g., UniverSelf[2] and SEMAFOUR[3]: OSSs deal with long-term optimisations of the network and are often integrated with a Business Support System (BSS) handling, e.g, billing data. Regardless of their actual implementation, which can be centralised or distributed, the goal is to take decisions based on high volumes of data which have a statistical, not instantaneous, meaning. Near-real-time services are also possible, but only for fault management via an integrated Network Management System (NMS) and based on pre-configured procedures. CROWD complements OSS since it provides short-term optimisations which react in real-time to the instantaneous changes to network usage. A closed-loop between OSS/BSS and CROWD is possible at a high level, by means of the policy translation function identified above.

Cloud Radio Access Network (C-RAN), e.g., iJoin[4]: C-RAN refers to the centralisation of digital signal processing into a dedicated server within the RAN, which then distributes the digital I/Q signal samples, e.g., via a Common Public Radio Interface (CPRI) bus or the Open Base Station Architecture Initiative (OBSAI) protocol. While this approach brings undoubtful performance advantages due to the enormous flexibility introduced, it requires a significant restructuring of the overall network, which is only justified for macro-cell base stations serving a high number of users. On the other hand, in CROWD we focus on wireless dense networks only, for which the most critical factor impacting penetration is the cost of backhaul improvement. Since backhaul demands in C-RAN are significantly higher than with traditional broadband mobile systems, we argue that C-RAN scenarios are different and non overlapping with the CROWD scenario, while there is not limitations in including C-RAN as part of the CROWD optimisations, at least for macro-cells.

Mobile cloud networking, e.g., MCN[5], TROPIC[6]: the efficient execution of cloud computing applications in a mobile network has unique requirements. Mobile cloud networking refers to the re-engineering of mobile architectures in order to support the efficient and elastic use of network resource, for example, by shifting the mobile network entities (e.g. BBUs (Base Band Units), RRHs (Remote Radio Heads), MME (Mobility Management Entity), etc.) into cloud platform, the network resources can be easily provision and optimize based on the demand. Since CROWD deals with the optimization of the network functions themselves, it does not represent an alternative to mobile cloud network solutions. Rather, the latter, when an in mature stage of development, could be integrated within CROWD as further optimization opportunities.

Self Optimising Network (SON): SON is a generic term referring to the "autonomic" execution of configuration/optimisation procedures at the base stations of a wireless networks, as an alternative to the traditional approach of manually tuning the values of parameters to achieve a given objective function. SON is seen as a basic enabler of large scale deployments of small cells, due to the prohibitive costs which would be incurred by planning, configuration, and optimisation if done manually for the many BSs in the network. SON is typically sold to telecom operators as a proprietary add-on, and it can only be used effectively in a given area if the hardware has been provided by the same manufacturer. While there are attempts to standardise SON, e.g., the X2 interface in LTE and the recently founded OSSII[7] alliance, such attempts are in their infancy. In CROWD we push the SON concept to the next step by proposing a open, i.e., vendor-neutral, architecture which provides a clean abstraction and network control functions via a set of APIs, which are then realised via the Southbound interface in an SDN approach.

III. Mobility functions and protocols

This section is devoted to the detailed analysis of one of the control applications defined in CROWD, the Mobility Management function, since it is one of the controller functions which

[2]http://www.univerself-project.eu/
[3]http://fp7-semafour.eu/
[4]http://www.ict-ijoin.eu/
[5]http://mobile-cloud-networking.eu/.
[6]http://www.ict-tropic.eu/.
[7]http://www.ossii.info/

highly affects the underlying architecture. The solution follows current trends towards distributed mobility management.

We first introduce Distributed Mobility Management (DMM), and then describe the SDN-based DMM implementation proposed in CROWD.

A. The DMM paradigm

Current networks architectures are deployed in a hierarchical manner, relying on a centralized gateway. Thus, the existing IP mobility management protocols are generally deployed following the same pattern. All the data traffic passes through a centralized mobility anchor, such as the Home Agent in Mobile IPv6 or the Localized Mobility Anchor in Proxy Mobile IPv6, and all the bindings are managed at this anchor as well. As the number of mobile nodes and the volume of the mobile data traffic increase, such centralized architectures may encounter several issues already discussed in Section I. In addition, existing IP mobility protocols are designed to be always active, managing all the services and all the traffic in the same way. They do not take into account that mobile nodes may not move during service session (which is 60% of the cases in operational networks [10]) or that a service may not require mobility functions at all. Such approaches may thus lead to non-optimal routing and large overhead due to tunneling mechanisms.

In contrast, novel flat IP architectures are now emerging, requiring the adaptation of current IP mobility management protocols to the evolution of the network. Indeed, there is a need to define novel mobility management mechanisms that are both distributed, to avoid bottlenecks, and offered dynamically, to globally reduce their signaling load and to increase the achieved performance.

Accordingly, the IETF chartered the DMM working group [11] in 2012. Efforts, from both industry and academia, are being performed on specifying DMM schemes, e.g., [12]–[15]. A common feature between different DMM schemes is the distribution of the mobility anchoring at access router level. Mobile nodes change dynamically their mobility anchor for new sessions, while keeping the previous anchors of ongoing sessions. When the sessions attached at a specific mobility anchor are terminated, mobile nodes deregister from that anchor. Assuming that most of the sessions are relatively short, most of the data traffic is routed optimally without tunneling [16].

One of the DMM requirements is to rely on the existing IP mobility protocols by extending and adapting them [7]. This is in order to benefit such standardized protocols before specifying new ones, and also to facilitate the migration of networks architectures. Accordingly, current DMM proposals in the literature focus on being MIPv6-based or PMIPv6-based as it is the general trend in IETF. However, we need to adapt the DMM concepts to our architecture that is SDN-based. Therefore, we propose an SDN-based DMM approach. It inherits some of the PMIPv6 concepts, being network-based and providing local mobility support for mobile nodes moving in a single operational domain, but in addition it splits the control and data planes providing the network with more flexibility in decision making. Hereafter, we discuss the approach and protocol operation in more details.

B. SDN-based DMM solution proposed in CROWD

The SDN solution proposed by CROWD for DMM relies on two main entities, the CLC and the CRC. Since the solution works at layer 3, it is required the definition of new APIs to control the IP layer configuration of the terminal's session anchor point (DMM-GW). Specifically, we need two new APIs: one to convey information on the mobile node to the CRC, such as the IPv6 prefix and DMM-GW assigned to it, and a second one to configure the IP layer of the DMM-GW, the prefixes reachable through the interfaces and to setup an IP-in-IP tunnel (which acts as a Southbound interface). Fig. 3(a), shows the initial attachment phase of the CROWD inter-district mobility solution. The attachment to the network begins by the terminal $MN1$ associating to a point of attachment belonging to the district. This event triggers the sending of an Logical Link Control (LLC) frame by the Access Point (AP), which is encapsulated in an OpenFlow message and forwarded to CLC_1. Through this message, CLC_1 is able to check on its local Binding Cache (BC) whether the mobile node $MN1$ is already attached to the district. If it is not the case, then it will contact the CRC, in order to check if the node is already registered in a previous district, and inter-district mobility is required. In the example depicted in Fig. 3(a), the terminal has not been attached previously to any CROWD network, so CLC_1 is free to assign it any of the available IPv6 prefixes on the district. Once the CLC has decided the prefix and DMM-GW to be assigned to the terminal, it proceeds to install the prefix in the DMM-GW ($DMM\ GW_1$). In this way, the CLC has an extra degree of flexibility, being able to assign arbitrarily the prefix to the selected gateway[8]. In addition, prefix, DMM-GW and terminal identification (e.g., MAC address) are notified to the CRC, that is able to keep track in this way of the previous attachments of the terminal.

Afterwards, the standard procedure followed by a terminal after successful attachment to a new network is performed. It includes the sending of a Router Solicitation (RS), in order to configure its IP address using IPv6 Stateless Auto-Configuration (SLAC). As in the case of the LLC message, the network encapsulates the RS message and sends it to the CLC. The CLC uses a Router Advertisement (RA) message to answer the RS, providing the prefix and default router ($DMM\ GW_1$) selected before. Hence, through the mediation of the CLC, highjacking the RA functionality of the network, we are able to control the IP level attachment of the terminal within the CROWD network. At this point in time, the CLC is able to compute the match rules and data path modifications required to forward the terminal's packets to the selected DMM-GW. These modifications are configured into the network through the OpenFlow protocol, requiring several message exchanges among CLC_1 and the different switches conforming

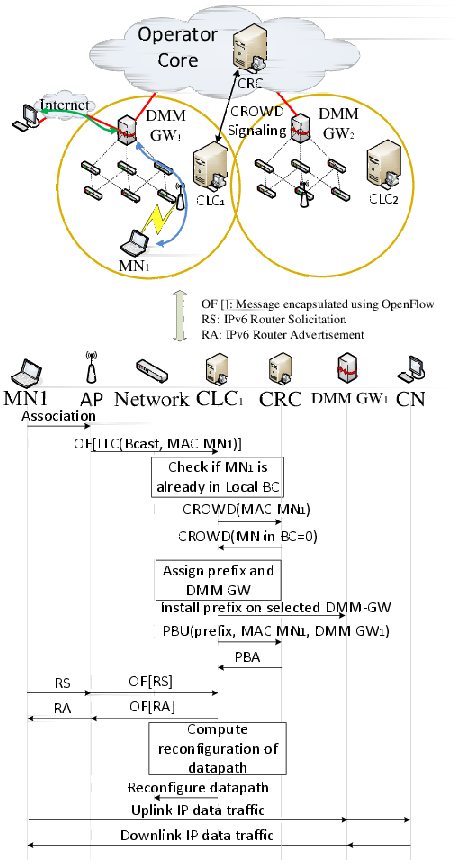
[8]Subject to the routing constraints imposed by the operator.

the path between the terminal and *DMM* GW_1. Once the data path is configured, packet originated at the terminal with layer 2 destination the DMM-GW, are transparently forwarded at layer 2. This behaviour is completely transparent to the layer 3 stack of the terminal, which sees the path between the terminal and the DMM-GW as a single hop.

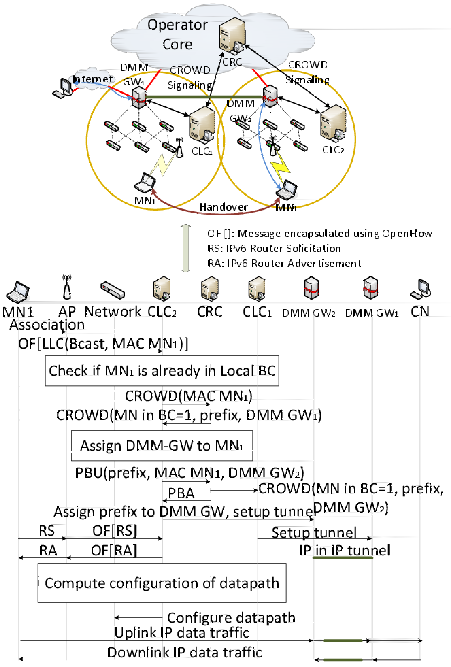
Fig. 3(b) presents the procedure of a handover between two different CROWD districts. The procedure assumes that the initial attachment process has been carried on as presented above. The procedure relies on the communication among the CLCs being orchestrated by the CRC. Basically, the CRC behaves as a data-base containing the list of previous DMM-GWs to be considered while performing handover. The configuration of the IP layer on the DMM-GWs and the tunnel setup among them is handled locally by the CLCs on each district. The procedure starts when the terminal attaches to an access point in a different district. As in previous cases, this event triggers the CLC (CLC_2) to check if the node is registered on its internal BC. As this is the first time the terminal attaches to the district, the CLC asks the CRC for previous registrations. In this case, the CRC has information regarding the terminal, informing the CLC of the prior connection of the terminal to *DMM* GW_1 and the prefix used. With this information, CLC_2 is able to decide the DMM-GW (*DMM* GW_2) to be used within this district and informs the CRC of this information. The CRC stores this information on its local BC for future reference. At this point in time several procedures are performed in parallel. First, the CRC informs CLC_1 of the new location of terminal $MN1$. With this information, CLC_1 configures *DMM* GW_1 with an IP-in-IP tunnel connection to *DMM* GW_2 and changes the routes at *DMM* GW_1 so that the prefix used by the terminal is routed through the tunnel. In parallel, CLC_2 configures the new prefix in *DMM* GW_2 and setups the IP-in-IP tunnel towards *DMM* GW_1. Once the tunnel is established, the configuration of the data path in the new network is performed as in previous cases. When all the procedure is complete, packets from the core network *CN* to terminal $MN1$ are forwarded first to *DMM* GW_1, which tunnels them to *DMM* GW_2. After *DMM* GW_2, the OpenFlow configured data path takes care of forwarding the packets to the appropriate location of $MN1$ within the district.

IV. Conclusion

In this paper we have introduced an approach based on SDN for the efficient and scalable realisation of control functions in extremely dense and heterogeneous wireless networks, as identified as part of the preliminary results in the FP7 project CROWD. First, we have illustrated the general network model, also including a critical review of the possible alternative solutions put forward in the market and within the research community, as well as a long-term business process supporting the paradigm envisioned. Second, we have focused on the issue of efficient and scalable mobility management, which is of paramount importance in the use case of interest mentioned above.



(a) Initial attachment.



(b) Inter-district Mobility: Handover.

Fig. 3. Inter-district Mobility procedures.

Acknowledgment

The research leading to these results has received funding from the European Unions Seventh Framework Programme

(FP7/2007-2013) under grant agreement no. 318115 (Connectivity management for eneRgy Optimised Wireless Dense networks – CROWD). Further details about the project can be found in the website http://www.ict-crowd.eu/. To receive news you can follow the twitter channel @FP7CROWD.

REFERENCES

[1] "Wireless intelligence (operators data): https://wirelessintelligence.com."

[2] P. Bhat, S. Nagata, L. Campoy, I. Berberana, T. Derham, G. Liu, X. Shen, P. Zong, and J. Yang, "LTE-Advanced: An operator perspective," *IEEE Communications Magazine*, pp. 104–114, 2012.

[3] J. Andrews, "Seven ways that HetNets are a cellular paradigm shift," *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 136–144, 2013.

[4] R. Knopp and P. Humblet, "Information capacity and power control in single-cell multiuser communications," in *Proceedings of IEEE ICC*, vol. 1, Jun. 1995, pp. 331–335 vol.1.

[5] G. J. Foschini, K. Karakayali, and R. A. Valenzuela, "Coordinating multiple antenna cellular networks to achieve enormous spectral efficiency," *IEE Proceedings*, vol. 153, pp. 548–555, 2006.

[6] X. Xu, G. He, S. Zhang, Y. Chen, and S. Xu, "On functionality separation for green mobile networks: concept study over LTE," *IEEE Commun. Mag.*, vol. 51, no. 5, pp. –, 2013.

[7] H. Chan, D. Liu, P. Seite, H. Yokota, and J. Korhonen, "Requirements for Distributed Mobility Management," DMM Working Group, IETF, Internet-Draft draft-ietf-dmm-requirements-07 (work in progress), 2013.

[8] A. Oliva, A. Morelli, V. Mancuso, M. Draexler, T. Hentschel, T. Melia, P. Seite, and C. Cicconetti, "Denser networks for the future internet, the CROWD approach," in *Mobile Networks and Management*. Springer Berlin Heidelberg, 2013, vol. 58, pp. 28–41. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37935-2_3

[9] M. Montemurro and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP)," RFC 5415 (Draft Standard), Internet Engineering Task Force, Dec. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5415.txt

[10] Cisco Systems Inc., "Cisco Visual Networking Index: Global mobile data traffic forecast update, 2009–2014," Cisco, Tech. Rep., 2010.

[11] IETF DMM WG, http://datatracker.ietf.org/wg/dmm/charter/.

[12] H. A. Chan, H. Yokota, J. Xie, P. Seite, and D. Liu, "Distributed and dynamic mobility management in mobile internet: Current approaches and issues," *Journal of Communications*, vol. 6, no. 1, 2011.

[13] P. Bertin, S. Bonjour, and J. Bonnin, "Distributed or centralized mobility?" in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*. IEEE, 2009, pp. 1–6.

[14] P. Seite and P. Bertin, "Distributed mobility anchoring," DMM Working Group, IETF, Internet-Draft draft-seite-dmm-dma-06 (work in progress), 2013.

[15] H. Ali-Ahmad, M. Ouzzif, P. Bertin, and X. Lagrange, "Distributed dynamic mobile IPv6: Design and evaluation," in *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, 2013, pp. 2166–2171.

[16] ——, "Comparative performance analysis on dynamic mobility anchoring and proxy mobile IPv6," in *15th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2012, pp. 653–657.

A Virtual SDN-enabled LTE EPC Architecture: a case study for S-/P-Gateways functions

Arsany Basta, Wolfgang Kellerer
Institute for Communication Networks
Technische Universität München
Germany
Email: arsany.basta, wolfgang.kellerer@tum.de

Marco Hoffmann, Klaus Hoffmann, Ernst-Dieter Schmidt
Nokia Siemens Networks
Munich
Germany
Email: marco.hoffmann, klaus.hoffmann, ernst-dieter.schmidt@nsn.com

***Abstract*—The recent initiative of Network Functions Virtualization (NFV) aims to deliver any data-plane processing or control-plane function in high volume data centers or network elements to decrease operational cost and increase deployment flexibility. In order to dynamically direct traffic flows between respective network elements, Software Defined Networking (SDN) can be seen as one enabler. In this paper, we focus on mobile core network nodes such as the MME, HSS, S- and P-Gateway as standardized for the LTE Evolved Packet Core (EPC). One straightforward solution for a virtualized EPC architecture would be to move all EPC network nodes completely into a data center and handle the data traffic via SDN-enabled switches. However, this solution would keep the conventional monolithic architecture unchanged. A possible split in the EPC functionality between a centralized data center and operator's transport network elements could be needed to provide the desired flexibility, performance and TCO reduction. Therefore, we have analyzed the EPC nodes and classified their functions according to their impact on data-plane and control-plane processing. We propose a mapping for these functions on four alternative deployment frameworks based on SDN and OpenFlow (OF). In addition, we investigate the current OF implementation's capability to realize basic core operations such as QoS, data classification, tunneling and charging. Our analysis shows that functions, which involve high data packet processing such as tunneling, have more potential to be kept on the data-plane network element, i.e. realized by an OpenFlow Switch. We argue for an enhanced OF network element NE+, which contains additional network functions next to the basic OpenFlow protocol.**

I. Introduction

In today's operator networks the deployment of new network services comes at a high cost for integration and operation as network functions typically come with separate hardware entities. Network Functions Virtualization (NFV) [1] is a recent operator's initiative aiming at addressing this problem. NFV is transforming the way how operators architect their networks towards deploying network services onto virtualized industry standard servers, which can be located for example in data centers. In this way, NFV involves delivering network functions as software that can run as virtualized instances and that can be deployed at locations in the network as required, without the need to install hardware equipment for each new service. With this migration from hardware to software running in a cloud environment, NFV is expected to lower not only equipment cost (capex) but also operational cost (opex). Services are expected to be deployed more flexibly and can be scaled up and down quickly. From an architecture perspective, NFV can be seen as complementary to technologies such as

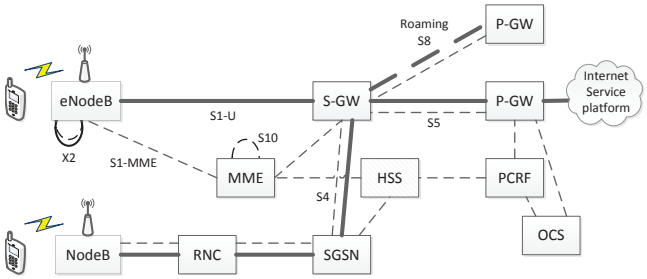


Fig. 1: LTE EPC functional architecture

Software Defined Networking (SDN) and cloud computing. Virtualized network functions might run in an operator's cloud environment and might be configured via OpenFlow. Whereas, these technology aspects are out of scope of the current NFV white paper [1], we address them by a thorough analysis of alternative architectures for functions deployment of an LTE EPC system in an operator cloud environment with SDN as a network enabler.

An emerging technology for supporting flexibility in networking is Software Defined Networking (SDN). The SDN concept provides an abstraction between data-plane forwarding hardware and control software. In this way, a logically centralized SDN controller allows to program the network on a per flow basis [2]. The OpenFlow protocol [3] is currently being standardized for the signaling between an SDN switch and an SDN controller. It describes the actions that can be performed on protocol header fields to install per flow rules in an SDN switch. While limitations in OpenFlow are being discussed in standardization and new versions are released regularly, these limitations and related performance aspects motivated this work to look into the deployment and split of virtualized network functions from a more general point of view with respect to realization of NFV for a mobile network.

In this paper, we focus on mobile core network nodes as standardized in the LTE Evolved Packet Core (EPC) [4]. In the current system deployment, core network nodes such as the Mobility Management Entity (MME), Home Subscriber System (HSS), Serving-Gateway (S-GW), Packet-Gateway (P-GW), Policy Charging and Rules Function (PCRF), Online Charging System (OCS) are typically deployed by separate dedicated hardware boxes. Thus, the application of NFV seems attractive for the EPC. Fig. 1 shows the functional architecture of the EPC. Mobile users' data-plane runs from eNodeB (LTE) or NodeB via SGSN (UMTS) through S-GW

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

and P-GW towards a service platform or the Internet. For mobility management, subscriber management, policy control and charging, control-plane nodes (MME, HSS, PCRF, OCS) complete the functional architecture. As it can be observed from this architecture, the S-GW and the P-GW play a central role in the EPC as they are responsible for both control-plane and data-plane handling including mobility management, QoS bearer handling, traffic policing and statistics. Hence, our analysis in this work for a realization of NFV is focusing on these two components.

The mobile network architecture, and in particular the LTE cellular system, has been in the focus of related work investigating the introduction of network virtualization or SDN. In [5] the authors discuss how the EPC architecture can benefit from virtualization mainly aiming at infrastructure sharing among operators. Realization solutions for a virtualized cellular network are presented in [6] with a focus on the access network. [7] addresses the flexibility to be gained in a cellular core network with the application of SDN technology. The authors propose a new architecture for the cellular core network based on inexpensive access switches, middleboxes and a centralized controller composing the data path for each user flow through a set of highly distributed middlebox functions. No core network function is addressed, in particular. The presentation in [8] addresses SDN and NFV for EPC. However, the EPC is migrated to an operator cloud as a whole, without the differentiation into separate network functions and their suitability for such migration. Overall, to the best of our knowledge, whereas the topic is gaining momentum, there is no state of the art analysis concerning EPC network functions with respect to NFV using SDN technology as the enabler.

The objective of this paper is to analyze the network functions of the LTE EPC architecture with respect to their potential deployment according to the Network Function Virtualization concept. We decompose and classify the functions according to their impact on control-plane and data-plane processing. We observe realization challenges with our theoretical and experimental study looking into the deployment of the identified functions based on SDN technology using the OpenFlow protocol. Those challenges mainly address the deployment of functions that involve a certain data processing pattern such as tunneling and monitoring being part of S-GW and P-GW EPC nodes in their mobility management, QoS and charging functions. Those functions demand for potentially new SDN component frameworks. For a deeper analysis, we propose four architectures, in which the identified network functions are mapped differently, split between an operator cloud, a middlebox and an SDN network element.

II. EPC Function Analysis based on basic LTE Scenarios

In this section, a thorough study of the S-GW and P-GW functionality is presented based on basic EPC scenarios. A scenario defines the steps to be carried out to provide a certain operation for users in a mobile network. From these scenarios, the main functions of the S- and P-GWs are derived.

A. EPC considered scenarios that involve S-/P-GWs

The *User Equipment (UE) attach* scenario serves the purposes of mutual user-network authentication, user location registration and establishment of a default bearer which has the basic policies and quality parameters [9]. The *UE detach* scenario occurs explicitly when a UE is switched off or implicitly when the network coverage is lost [10].

An EPC bearer is composed of two parts, namely the S1-U bearer and the S5/S8 bearer. The S1-U carries the traffic between the eNodeB and S-GW while the S5/S8 carries the traffic between the S-GW and P-GW. For a better resource utilization in case of user inactivity, the established *S1-U bearer* is *released* while the S5/S8 bearer is preserved to provide the ”always-on” feature of LTE [9]. Whenever a user becomes active once more, a *service request* would initiate re-establishing the S1-U bearer in case the user utilizes only the *default bearer* [10] [11]. A new *service request* is triggered to establish a *dedicated bearer* if additional quality parameters are needed. In this case, new S1-U and S5/S8 bearers are created that support the higher demand service [9] [12].

Addressing users' mobility, an *update to the tracking area (TA)*, which is a set of multiple coverage cells, takes place when an idle user moves to a new TA [9] [12]. The whole bearer is updated by both the S-GW and P-GW. While a *handover* typically takes place if an active user switches to 2G/3G network or moves to a cell covered by another eNodeB. The active session is transferred without interruption requiring bearer connection updates accordingly [9] [12]. Roaming scenarios are not presented here in this paper. However, they are included in our full analysis.

B. S-/P-GWs derived functions

In each aforementioned EPC scenario, we analyze and extract the main functions that are used in both the S- and P-GWs. In addition, we assign a different color for each function block to ease the architecture's visual illustration. We have classified these functions into seven categories. Table I shows the set of classified functions utilized in each scenario. Each function corresponds to a certain traffic pattern respective to its occurrence frequency which indicates a deployment requirement in our later analysis.

Control Signaling: receives and triggers signaling messages and calls the corresponding functions accordingly. Such signaling function is required by both S-GW and P-GW.

Resources Management Logic: manages the node's available data-plane resources as it assigns the bearer parameters such as the Tunnel End-point IDs (TEID). It also administrates the traffic shaping and users' policy enforcement as it allocates the bearer quality attributes based on users' QoS requirements as the Quality Class Identifier (QCI) or the Guaranteed Bit Rate (GBR) [13] for a dedicated bearer. This function is needed in both S-GW and P-GW, where the P-GW utilizes it additionally to assign IP addresses to users attaching to the network.

Data-plane Forwarding Rules: contains the data-plane forwarding rules obtained from the resources management logic for all established bearers. The rules include the users' enforced policies on the data-plane. In case of the S-GW, it is mapping between the S1-U bearer and S5/S8 bearer of each user, while the P-GW maps the S5/S8 bearer to the external IP packet data network.

TABLE I: S-/P-GWs Functions Classification in chosen EPC Scenarios

EPC Scenario	EPC Node	Signaling	Resources Management	U-plane Forwarding Rules	U-plane Forwarding	GTP	Filtering	Charging
UE Attach	S-GW	✔	✔	✔	✔	✔	-	-
	P-GW	✔	✔	✔	✔	✔	✔	✔
UE Detach	S-GW	✔	✔	✔	-	-	-	-
	P-GW	✔	✔	✔	-	-	-	-
S1-U Bearer Release	S-GW	✔	✔	✔	-	-	-	-
	P-GW	-	-	-	-	-	-	-
Service Request	S-GW	✔	✔	✔	✔	✔	-	-
(Default Bearer)	P-GW	-	-	-	-	-	-	-
Service Request	S-GW	✔	✔	✔	✔	✔	-	-
(Dedicated Bearer)	P-GW	✔	✔	✔	✔	✔	✔	✔
Tracking Area Update	S-GW	✔	✔	✔	-	-	-	-
	P-GW	✔	✔	✔	-	-	-	-
Intra Handover	S-GW	✔	✔	✔	✔	✔	-	-
with/without X2 Support	P-GW	-	-	-	-	-	-	-
Handover	S-GW	✔	✔	✔	✔	✔	-	-
from LTE to 2G/3G	P-GW	✔	✔	✔	✔	✔	✔	✔

Data-plane Forwarding: represents the switching fabric or hardware that handles the data traffic and carries out the data flows processing. It is present in S-GW and P-GW.

GTP Matching: Both the S-GW and P-GW use GPRS Tunneling Protocol (GTP), which is an IP tunneling protocol, to encapsulate the signaling and data traffic. GTP serves the purposes of supporting users' IP mobility, security provisioning and aggregating transport traffic. This function handles GTP headers' matching, encapsulation and decapsulation.

Data-plane Filtering and Classification: is responsible for identifying the packets based on users' profiles and policies for both uplink and downlink data traffic. This function is typically present at the P-GW.

Charging Control: mainly takes place in the P-GW only and can be classified into offline and online charging. An offline charging function collects Charging Data Records (CDR) in a post paid fashion while an online charging function allocates charging events on the data-plane that are triggered when a condition meets a user's profile [14]. Charging can have different models based on data volume, data usage time or event-based. The charging function is generally implemented by the Policy and Charging Enforcement Function (PCEF) located at the P-GW. Most notably, in case of roaming and offline charging, the visited S-GW takes over the accounting and charging function when connecting to the home P-GW as illustrated in Fig. 1.

III. USE CASES OF EPC FUNCTIONS REALIZED VIA SDN

In this section, we show how an SDN realization for the classified functions of the S-/P-GWs could be achieved. Since we aim at using OpenFlow, we evaluated the current OF switch API capabilities, i.e. OF controller southbound API, to provide each function category. In addition, we provide a framework for additional modules to be integrated to the basic OF controller framework, i.e. OF northbound API. For our investigation, we built an SDN testbed with 3 OF switches in a ring topology, one NEC PF5240 and two PICA8 Pronto 3290 switches. We adopted the Java based open source Floodlight OF controller [15]. All proposed frameworks are applicable starting from OF specification 1.0.

A. Control-plane related functions

Generally for control-plane only related functions, it is relatively straightforward to integrate and comply with the OF controller framework. Starting with the Control Signaling function, a module is needed to be added to the OF controller which performs the signaling management. We could point out that the Resources Management Logic could be mapped to the centralized switching module which is already a core function in any OF controller such as in Floodlight. The existing switching module would need to be slightly modified to include user profiles and policies and to utilize these in the resource management decisions.

B. Data-plane related functions

Intuitively, data-plane functions are highly dependent on the data traffic and accordingly would exploit more the OF switch and its traits. First by looking at the Data-plane Forwarding Rules and Data-plane Forwarding, both are main operations of a basic OF switch and they impose no additional effort. However, both S-GW and P-GW use GTP tunneling protocol to transport the signaling and data traffic. An OF switch running even the latest OF specification, version 1.3.1 [16], is limited to only layer 2/3 matching with the addition of TCP/UDP port fields. Hence, it is not possible currently to realize S-/P-GWs data forwarding with GTP header matching.

We use this finding to propose four different frameworks to realize functions such as GTP via SDN, illustrated in Fig. 2. The first framework argues for implementing the GTP function as a controller module as shown in Fig. 2a. This solution matches the current OF implementation, nevertheless it dictates forwarding every flow packet to the controller to be processed which could impose a huge data overhead.

The next approach would be to utilize a middlebox that accommodates the additional function as for example a middlebox hosting a GTP function as shown in Fig. 2b. The data flows need to go by the middlebox for additional processing. However, the main difference between the previous proposed framework is the flexibility of deploying middleboxes at the transport network close to the OF switches.

Another proposal would be to enhance the OF switch to include certain additional functions, as in this case a GTP

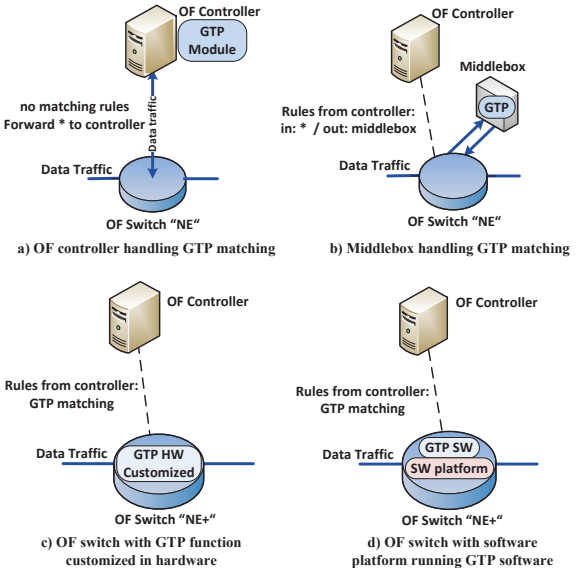


Fig. 2: SDN frameworks for the GTP Matching Function

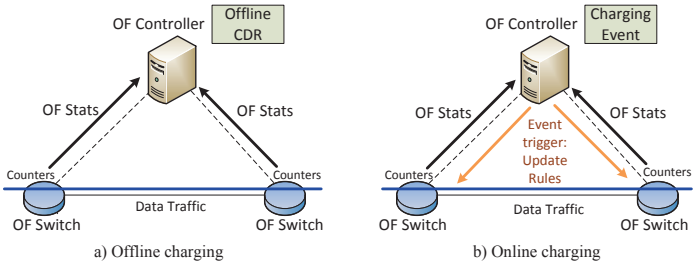


Fig. 3: SDN framework for the Charging Control Function

Matching function customized in hardware as shown in Fig. 2c. We call it an OF Network Element plus "NE+". This approach is needed for functions that cannot be provided currently by the OF specification or for which an implementation at the switch is beneficial for performance reasons (see Section IV). In this approach a hardware implementation in the NE+ is required, which implies that its flexibility is quite limited.

Fig. 2d illustrates a fourth alternative framework where we propose supplying each switching element with a programmable platform that enables functions to be deployed on it. Such platform should enable the software functions to interact with the data-plane and hence extend the OF switch capabilities. This is what we call an OF Network Element plus with a Programmable Software Platform "NE+ with SW-platform". The advantage of such element is the increased flexibility in extending the basic OF switch functionality.

Next to GTP, we have evaluated OF towards QoS support based on the Packet Filtering and Classification function. This would be quite relevant to realize a dedicated bearer for LTE. Packet filtering can be realized via matching rules which correspond to the service data flow filters specified in [17]. The filters are matched through the IP five tuple: (source IP address, destination IP address, source port, destination port, protocol ID). Our SDN testbed results concluded that QoS support is provided starting from OF 1.0. However, quality guarantees such as guaranteed bit rates and priority classes are switch dependent. The experiment included two flows from

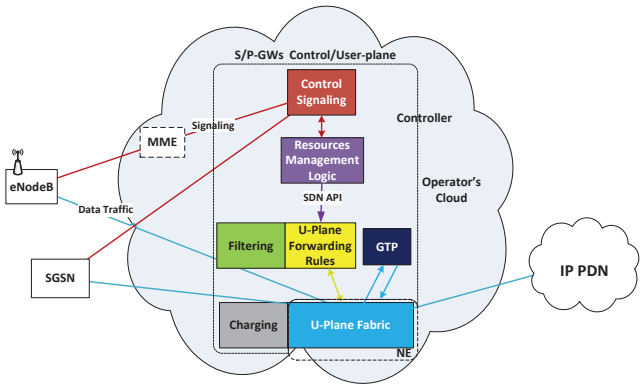


Fig. 4: Full Cloud Migration Architecture

two different applications, namely an HD video streaming and a file transfer, where alternative scheduling configurations were investigated at the switch such as priority queues or weighted fair queuing with guaranteed minimum bit rates.

Regarding the Charging function, we propose using the OF counters and statistics exchanged between the OF switch and controller to realize an SDN-enabled charging framework as shown in Fig. 3. In case of offline charging, an additional module is needed to collect the CDRs based on OF counters and stats as illustrated in Fig. 3a. It becomes more complex for online charging. As the OF switch keeps no state for the forwarded data flows, it is not possible to allocate charging events on a basic OF switch. Fig. 3b illustrates one possible deployment in the OF controller. An alternative solution would be to include the charging function either in a middlebox, hardware customized NE+ or NE+ with SW-platform.

IV. Deployment Architectures Based on SDN

As we can observe from the theoretical and experimental analysis of realizing EPC functions virtualization with SDN, we have to take a closer look at how data-plane related functions could be deployed. So far we have only considered implementation aspects related to OF protocol versions, which are subject to be revised. However, there are further aspects such as performance motivating to carefully consider where to place an EPC function. Therefore, in this section, we generalize the deployment discussion above by proposing four generic deployment architectures, showing alternatives for function placement either in a data center, i.e. operator cloud, or at transport network elements such as OF switches.

A. Full Cloud Migration Architecture

The first architecture shown in Fig. 4 deploys virtualized S-/P-GWs into an operator owned cloud. Since MME is handling only control-plane, it is virtualized and migrated to the cloud in all our proposed models. SDN technology would be used to handle signaling and data traffic incoming and outgoing from the cloud, as well as intra-cloud forwarding. This architecture corresponds to Fig. 2a, i.e. basic OF network element.

The high volume cloud infrastructure brings several advantages to traditional mobile networks. First, the cost savings are quite notable compared to proprietary hardware components. A second advantage would be the flexible dimensioning of

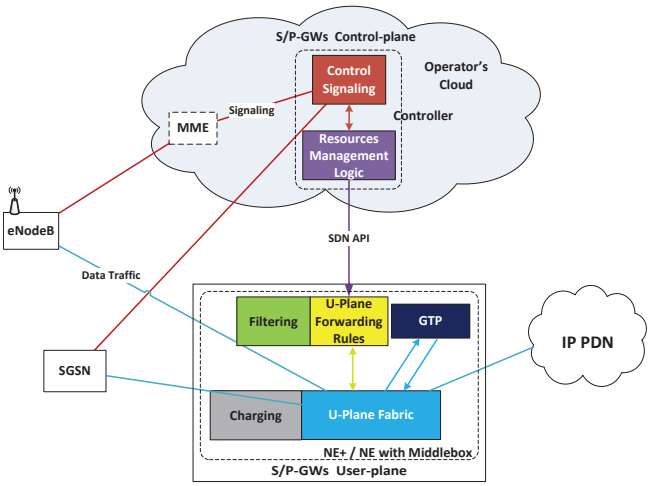


Fig. 5: Control-plane Cloud Migration Architecture

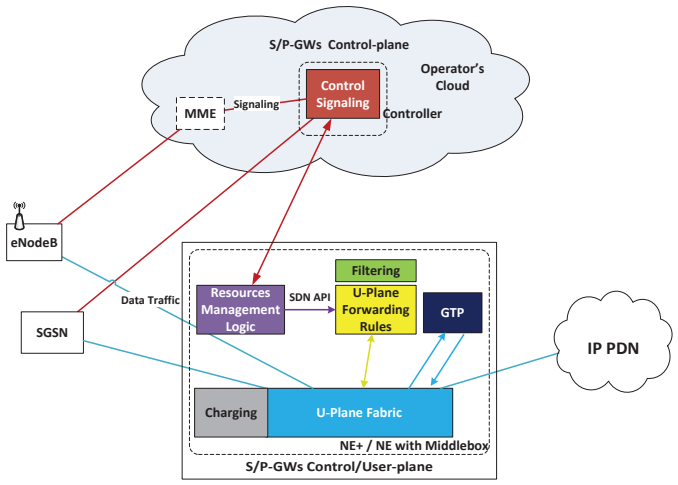


Fig. 6: Signaling Control Cloud Migration Architecture

EPC components, where a dynamic migration of the virtual components would allow for better resource utilization and would be a key factor in case of failures or disasters. Another point would be flexible upgrades by allocating more resources or adding a new functionality to the core. SDN would play a role to support such flexible deployment of resources or introducing new features.

However, such advantages are bounded by the cloud infrastructure domain and a larger scale flexibility covering the whole core network is not yet exploited in this architecture. Considering a use case of a service that requires a core node to be deployed in a rather nearer geographical location towards the users, this requirement could rise due to performance constraints such as the end-to end delay or due to on-demand or time-based services. The Cloud infrastructure performance would be another critical factor taking into account the high frequency of the control-plane operations taking place at the S-/P-GWs in addition to the large volume data traffic needed to be managed, specially at the P-GW.

B. Control-plane Cloud Migration Architecture

The second architecture employs a split in the S-/P-GWs functions to address the performance considerations above and to enable flexible deployment of the data-plane. As illustrated in Fig. 5, the control-plane management would be kept in the cloud, namely the signaling control and resources management logic functions. The data-plane would span over a more distributed infrastructure covering the whole core transport network. Therefore, the forwarding rules function is critical to be on data-plane elements together with other heavy data processing functions such as GTP, filtering and charging functions. This implies that to avoid forwarding all the data packets to the cloud, hardware customized NE+, NE+ with SW-platform or NE with middlebox is required. Keeping the SDN framework in mind, this proposed architecture still preserves the centralized control-plane with an SDN control API between the resources management logic function and the forwarding rules function.

This architecture allows for more flexibility and a higher degree of freedom by having the S-/P-GWs data-plane deployed on distributed elements. It would allow on-demand migration of the virtualized data-plane components based on the traffic volume or service requirements. Another key point, offloading the data traffic on distributed dedicated nodes would have a major influence on the performance. In addition, keeping the centralized control-plane would maintain the global network view, which would contribute to better data traffic shaping and load balancing in the core network.

Nevertheless, data-plane rules exchange could introduce an additional overhead that needs to be minimized. Additionally, it could also create a bottleneck at the centralized cloud control which would have a major influence on the whole core performance. It would also be restricted to relatively strict delay requirements to meet EPC standards.

C. Signaling Control Cloud Migration Architecture

A third architecture is illustrated in Fig. 6 where we propose migrating the signaling control function only to the cloud while allocating all other functions on the data-plane elements. From an SDN point-of-view, this architecture requires having an NE+ with multiple functions integrated whether customized in hardware or implemented in software. Alternatively, those functions could be offloaded to middleboxes. The main reason would be performance requirements, where this virtualized S-/P-GWs architecture would have the potential for decreasing the response time experienced in the data-plane. This proposed architecture opens the discussion to compare the overhead experienced by sending signaling triggers in this case compared to sending the data-plane rules in the previous control-plane cloud migration architecture.

Another advantage of moving the resources management logic to the distributed network elements is that it would enable the data-plane to be more independent from the cloud framework, which makes it more resilient in case of power cuts or cloud connection failures. Note that in this case, a default management logic needs to be installed and operated if such failures occur to keep the data-plane survivability.

However, in this current architecture, we could recognize that the cloud migration is quite minimal and we are not taking the full advantage of its powerful processing and storage resources. In addition, the logically centralized resources

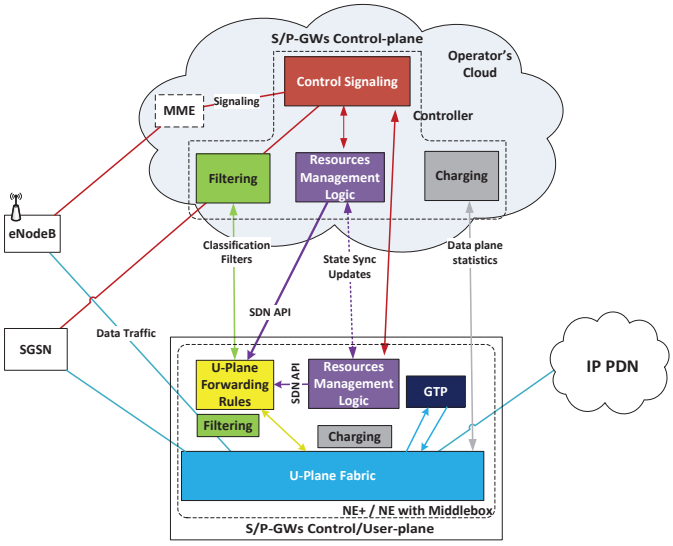


Fig. 7: Scenario based Cloud Migration Architecture

management and the global network view of the resources is not available anymore which makes application-oriented traffic engineering and network resource optimization less achievable.

D. Scenario based Migration Architecture

Based on the discussions of the previous architectures, we propose a forth virtualized S-/P-GWs architecture that aims to mix and match the function deployment based on the scenarios, as in Fig. 7. The functions are deployed in both the cloud infrastructure and data-plane nodes. EPC scenarios introduced in Table I are assigned to either deployed functions according to the scenario latency requirements or its arrival frequency [18], which influences the data volume exchanged between the cloud and data-plane nodes. A simple example would be activating the resources management function at the data-plane elements for delay-critical scenarios while scenarios which need high processing power would be handled by the resources management function at the cloud. Such approach certainly requires state synchronization and orchestration between the cloud and data-plane deployed functions as shown in the figure to avoid duplicate assignments or conflicting decisions.

As mentioned in Section III, the charging function could be placed in the cloud and use the OF statistics exchanged between the OF controller and switches to obtain offline charging records or implement online charging events. Additionally, the OF controller could map service flow filters to flow rules based on users' policies and profiles to implement the packet filtering function via SDN. It is noted that NE+ with SW-platform or NE with middlebox could show more potential in this proposed architecture to flexibly change the functions deployment. Nevertheless, if an optimal functions deployment classification is reached, fixed hardware customized NE+ would also fit.

The disadvantage in this proposed architecture is the introduced overhead of functions' state synchronization and the need to maintain an updated overall system state to preserve the S-/P-GWs performance. This point shall be further investigated along with the classification of scenarios to be assigned either to the cloud or the data-plane network elements.

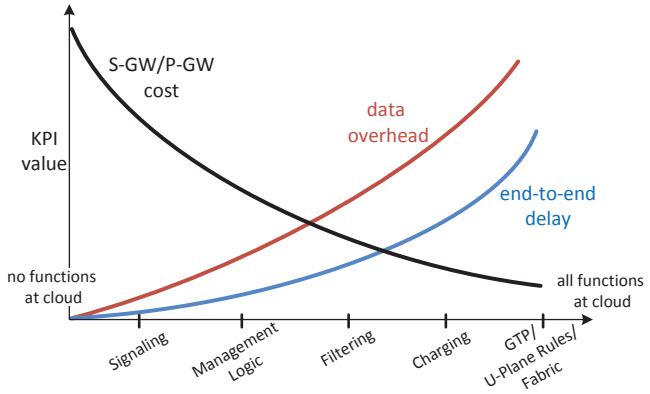


Fig. 8: Impact of incremental functions migration to the cloud

V. Discussion and Conclusion

In order to backup our qualitative study, we plan to derive quantitative parameters for the derived functions including the required delay upper bound, function call frequency, exchanged data volume or deployment cost. Fig. 8 illustrates the expected behavior of some chosen key criteria where migrating more functions to the cloud results in cost savings (black line). We assume a certain order of moving functions into the cloud according to their cost contribution based on our earlier investigations. Yet we argue that moving functions into the cloud may impose some additional data and delay overhead (red and blue lines). Note that the order of the functions might be different for each cost line. Given the right order, the cost function will be a decreasing function, however not necessarily steady, but with steps and possible flat parts. Our future prospective target is to find the optimal deployment solution via a quantitative evaluation of the function deployment split using an LTE simulation platform.

In this paper, we present a thorough study of the EPC S-GW and P-GW functions based on fundamental LTE scenarios. Our functions evaluation shows that for some data-plane related functions, in particular GTP tunneling, current OpenFlow shows limitations. This motivates us to propose four alternative concepts for an enhanced SDN network element NE+ allowing additional functions customized in hardware or software: (a) basic NE with a controller-based packet processing, (b) basic NE with a middlebox-based packet processing, (c) NE+ with a customized hardware function extension, (d) NE+ with a programmable software extension.

Furthermore, we detail our analysis and propose four deployment architecture alternatives between a cloud environment and SDN network elements. The first architecture classically deploys the whole S-/P-GWs in the cloud, while the second architecture proposes a split with control related functions deployed in the cloud and data related functions residing on the NEs. A further shift of functions to the data-plane NEs is presented in the third architecture leaving only the signaling function in the cloud, arguing for a potential performance boost. The fourth architecture presents a hybrid deployment where the functions are mapped to both the cloud and to NEs, being activated according to individual performance requirements. Each architecture alternative is critically evaluated to provide a guideline for system designers in addition to the design of the OF network elements.

REFERENCES

[1] *Network Function Virtualization, white paper*, SDN and OpenFlow World Congress, Darmstadt, Germany, October 22-24, 2012.

[2] *Software-Defined Networking: The New Norm for Networks, white paper*, ONF, April 2012.

[3] N. McKeown et al., *OpenFlow: enabling innovation in campus networks*, ACM SIGCOMM CCR, 38(2):6974, April 2008.

[4] *Enabling mobile broadband growth Evolved Packet Core, technical white paper*, Nokia Siemens Networks, 2009.

[5] A. Khan et al., *Network Sharing in Next Mobile Network: TCO Reduction, Management Flexibility and Operational Independence*, IEEE Communications Magazine, vol. 49, no. 10, pp. 134-142, Oct. 2011.

[6] Y. Zaki et al., *LTE mobile network virtualization*, Mobile Networks and Applications Journal, vol. 16, pp. 424-432, August 2011.

[7] X. Jin et al., *SoftCell: Taking Control of Cellular Core Networks*, Princeton Computer Science Technical Report TR-95-13, May 2013.

[8] J. Mueller et al., *SDN/Openflow Impacts on EPC Evolution*, 8th KuVS/42nd ITG 5.2.4 expert meeting, Koenigswinter, Germany, April 17, 2013. (slides at: http://www.kuvs-ngsdp.org/index.html)

[9] *General Packet Radio Service (GPRS) enhancements for Evolved Universal Terrestrial Radio Access Network (E-UTRAN) access*, 3GPP TS 23.401, release 10, 2012.

[10] R. Kreher and K. Gaenge, *LTE Signaling: Troublshooting and Optimzation*, Wiley, 2011.

[11] *Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS)*, 3GPP TS 24.301, 2009.

[12] P. Lescuyer and T.Lucidarme, *EVOLVED PACKET SYSTEM (EPS): THE LTE AND SAE EVOLUTION OF 3G UMTS*, Wiley, 2008.

[13] *LTE-capable transport: A quality user experience demands an end-to-end approach*, Nokia Siemens Networks, 2011.

[14] *Charging management; Charging architecture and principles*, 3GPP TS 32.240, release 11, 2013.

[15] Floodlight OpenFlow Conroller, *http://www.projectfloodlight.org/floodlight/*

[16] *OpenFlow Switch Specification Version 1.3.1*, ONF, September, 2012.

[17] *Policy and charging control architecture* , 3GPP TS 23.203, release 9, 2010.

[18] *Signaling is growing 50% faster than data traffic, technical white paper*, Nokia Siemens Networks, 2012.

SDN Based Inter-Technology Load Balancing Leveraged by Flow Admission Control

Suneth Namal*, Ijaz Ahmad†, Andrei Gurtov‡ and Mika Ylianttila§
*†§Department of Communications, University of Oulu, Finland
‡Department of Computer Science and Engineering, Aalto University, Finland
Email: [*namal,†iahmad]@ee.oulu.fi,‡gurtov@hiit.fi,§mika.ylianttila@oulu.fi

***Abstract*—In this paper, we have followed the idea of exploiting inter-technology load balancing leveraged by flow admission control with software defined networking (SDN). By means of the highly flexible interfaces, SDN is simply effective in the 5G network architecture where load balancing and dynamic flow admission control are considered as essence of networking. In one hand, load balancing presented in this paper reveals a drastic reduction of unsatisfied-user percentage almost by five times while, on the other hand, it enhances per-flow resource allocation more than 200%. In addition, this proposal substantially off-loads the core-network, avoids over-utilizing the cellular-network and reserves resources based on cut-off priority. Explicitly, it allows a certain degree of freedom to choose the network options based on user-preference and their priorities. Finally, we compare our load balancing algorithm leveraged by flow admission control with an analytical model to ensure the correctness and efficiency.**

I. Introduction

Deployment of virtualization and SDN technologies are foreseen mainly in the transport level, and user/data plane procedures will presumably remain intact. However, there are ongoing researches focusing on the possibilities of using SDN technologies for user-level management. Load balancing is such a classical networking function supported by routers, switches and load balancers [1]. Due to existence of various wireless technologies that are evolved in parallel with the emergence of heterogeneous 5G networks, the adaptive coupling based on load status and delay constraints has become a must in 5G network architecture which is inspired by the dynamically diversified traffic conditions [2], [3]. Load balancing based on IP addresses or other primitive methods are unlikely to overload the system heavily, though such basic functionalities are decades old and being proven inefficient for scaling and virtualizing network functions and applications.

SDN empowers isolation and redirection of traffic by means of the remote assistance of the controller. It could be realized by configuring the flow tables on switches and routers. Adding, modifying, and removing flow rules on networking infrastructure are the major function of remote controller. Intelligent flow configuration with the understanding of physical topology opens-up opportunities to freely develop network applications [2]. Thus, admitting new flows happens to repeat this process. Load balancing in SDN, leveraged by admission control is a technique of manipulating these flows. This could be realized at the centralized controller who can configure flow-paths, such that they optimally utilize network resources and enhance the user-experience. In large networks, it is understood that defining optimal path while admitting to a network is always efficient compared to that of regulating the existing flows. This is the driving force behind load balancing empowered by flow admission. It is empowered by three main components: flow tables, controller and secure channel. Existing load-balancing algorithms assume that the requests are entering to the network through a single point where the load balancers are placed though, there could be several such other choke points in a large network [4]. Because of this, the need for isolating such network functions from infrastructure is growing. Successful load-balancing leveraged by flow admission control optimizes resources, minimizes response time, maximizes throughput, and avoids overloading.

In this paper, we introduce a novel SDN based load balancing solution where flow capacity is defined by the number of requested physical resource blocks (PRBs). The results reveal a drastic reduction of the number of unsatisfied and angry users in the network and a substantial improvement of resources allocated per user. This paper is organized as follows. In Section II, we presents our SDN based load balancing architecture. Then, our simulation model is presented in Section III. Section IV describes simulation results and finally, in Section V, we conclude this paper.

II. SDN based load balancing architecture

SDN enablers, such as OpenFlow allow to retrieve packet count at switches and routers. In this architecture, the centralized controller or the internetworked set of distributed controllers retrieve the load status through the Load Balancing (LB) application part which is an application programming interface (API) on switches. Then, this information is directly communicated to the controller who will decide how to admit new flows. The ability to control flow tables of legacy networking elements by means of logically separated control plane with an efficient forwarding approach is the benefit of this programmability that allows to overcome load imbalance problem.

The SDN based common interfaces that enable inter-system communication is an essence that offers necessary platform for our solution. Managed handover is a must in applications like these. Typically, WLAN handover is initiated by the mobile stations by scanning the surrounding access points/base stations and by selecting the best one with the highest signaling

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

strength. However, the access point or base station with the highest signaling strength may not always have enough capacity or resources to occupy an additional station. Thus, delegating management rights to the network operating system to decide when and how to perform handover is more beneficial in terms of guaranteed connectivity, managed QoS, workload balancing, and flow and traffic management. Therefore, SDN is a promising solution that could be easily customized to trigger network originated handover combined with such programmable load balancing applications.

III. Simulation Model

In each time instant, the users move randomly across the map and measure the interference with path-loss model. We have assumed that the load-balancing application part at the controller determines switching between the systems and creates, modifies or deletes the flow-rules to optimize network resources and routing. The users could be either in inactive state, meaning the user is not connected to the network at all; active state, meaning the user is switched on, but there's no ongoing transmission; and connected state, meaning the user is on, and an active IP data transfer is ongoing.

A. Flow of Load Balancing Algorithm

The users with the highest PRB demand are considered to be foremost in the queue to be served. This reduces the handover frequency and signaling overhead in the network. In service level, there are two types of users defined in this system,

- **unsatisfied users -** The unsatisfied users could be defined as those who do not have obtained the expected flow space but the minimum allowable flow capacity per flow.
- **angry users -** The angry users are those who do not get access to their traffic flows due to lack of resources.

We define fairness function $F(\theta, \Lambda)$ while θ and Λ are given in (7) and (8). There are two threshold levels for $F(\theta, \Lambda)$. When $F(\theta, \Lambda)$ is greater than upper threshold σ, new flows are directly admitted to the WLAN or mobile femtocell (MF) network. We attempt to reduce $F(\theta, \Lambda)$ in order to make the aggregated flow space as "flat" as possible among eNBs. When $F(\theta, \Lambda)$ is greater than the threshold σ for a given flow, such flows are assigned the minimum flow space (FS_{min}) which is 6 PRBs according to 3GPP standards or transferred to WLAN. As it is shown in Fig. 1, those users could be even transferred to MF network, given that $\phi_{mf} < \eta_{mf}^{lim}$ where ϕ_{mf} is the current utilization and η_{mf}^{lim} is the maximum flow space a cell can support. If there is no space for a new flow, it is assigned to WLAN given that $\phi_{wlan} < \eta_{wlan}^{lim}$. The parameters follow the same meaning as it is with the previous case.

When $F(\theta, \Lambda)$ in (9) is in-between the upper and lower bounds ($\sigma > F(\theta, \Lambda) > \varsigma$), the users can select either WLAN or cellular access depending on their demand for flow space. When the flow demand is above FS_{wlan}, controller assigns those flows into WLAN where FS_{wlan} is the lower bound of WLAN. Otherwise, the flow space will be allocated directly from the cellular network. If $F(\theta, \Lambda) < \varsigma$, flow space will be allocated from the cellular network. Assuming that cellular network is overloaded and the demand is equal or greater than FS_{min}, the incoming flows are assigned to the MF network. This approach offloads the cellular network from bulky flows.

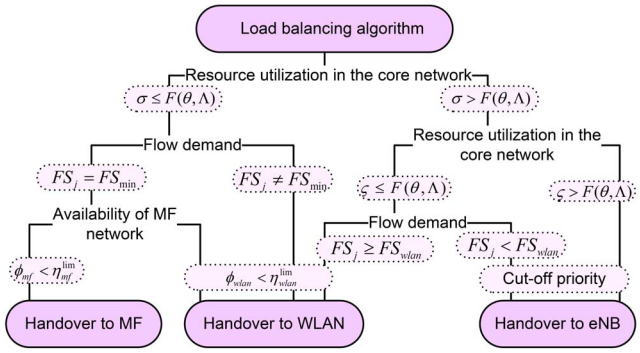


Fig. 1. Load-balancing algorithm.

B. Probabilistic Approach to the Model

The base stations have only a limited flow space. Flows are allocated space according to the facts below. 1) demand for flow space, 2) average waiting time in the queue, 3) current utilization at the base-station, 4) average utilization of the network and, 5) user preference and their priorities. If there are N_t users requesting FS_i flow space from the i^{th} base-station over a period of t_m,

$$FS_i = \int_0^{t_m} (N_t * FS_j)\mathrm{d}t. \tag{1}$$

We assume that a base-station can support K users at its best. We say a base station is overloaded when there are $0.8K$ associated users. We have defined utilization-index of i^{th} base-station as $\rho_i = (\lambda_i/\mu_i)$ where λ_i is the flow arrival rate and μ_i is the serving rate of i^{th} base station. For analysis purpose, we have modeled a base station as a M/M/1/K queuing system. Thus, the probability of the k^{th} state could be derived by (3).

$$P_0 = \frac{1-\rho_i}{1-\rho_i^{K+1}}. \tag{2}$$

$$P_k = \begin{cases} (K+1)^{-1} & \text{if } \rho_i = 1, \\ \\ \rho_i^k P_0 & \text{if } \rho_i \neq 1. \end{cases} \tag{3}$$

The above equation calculates the probability of occupying k flows at base-station i. In order to measure the expected waiting time (W_i), we need to know the average allocation of flow space, which is $N_i = (K/2)$ if $\rho_i = 1$, otherwise,

$$N_i = \sum_{k=1}^{k} kP_k = \frac{\rho_i}{1-\rho_i} - \frac{(K+1)\rho_i^{k+1}}{1-\rho_i^{k+1}}. \tag{4}$$

We have assumed that flow arrival follows a Poisson process. The average flow capacity $\overline{FS} = \sum_{i=1}^{M}(FS_i)/M$ where M is the number of base stations and FS_i is the instantaneous resource utilization of base station i. However,

FS_i is not uniform for all the base stations. According to $Little's\ law$ [5], we define waiting delay,

$$W_i = \frac{N_i}{(1 - P_k) * \lambda_i}. \tag{5}$$

We define $\gamma_i = e^{(FS_i/\overline{FS})}$, average flow intensity $F_{avg} = F_{tot}/M$, and the relative-intensity of station i, $A_i = F_i/F_{avg}$ where F_{tot} is the number of flows on the map. Thus, the average waiting time W_{avg} and load-index θ_i is given below.

$$W_{avg} = \frac{1}{M}\sum_{i=1}^{M}(W_i). \tag{6}$$

$$\theta_i = \gamma_i \ .\ exp\left(\frac{(A_i * W_i) - W_{avg}}{1 + \sqrt{W}_{avg}}\right). \tag{7}$$

The weighted delay (A_iW_i) normalizes the delay variance in the system when the difference is large. When queue has a large weighted delay more than the order of $\sqrt{W}_{avg}$, then, the exponent term in (7) becomes more significant and thus, overrides γ_i, since relative-intensity becomes more prominent. On the other hand, small weighted delay variance makes the exponent term close to unity and thus, γ_i dominates θ_i.

Let FS_i^j be the bandwidth demand of j^{th} flow and $FS_i^{j,max}$ be the maximum that it can demand. We define $Q_j = e^{(FS_i^j/FS_i^{j,max})}$ which is the flow demand compared to flow classification or prioritization. The flow contentment-index (Λ_j) counts both user classification and demand. The contentment-index is calculated according to the availability of network options. Firstly, (8-i) is used when they have only cellular and WLAN access and secondly, (8-ii) is used when they have only cellular and femtocell access, whereas finally, (8-iii) is used when they have access to all the networks. Flow-classification has a linear impact on the contentment-index while user-preference has an exponential relation. Thus, the user preference rules-out contentment-index when it is biased to a particular network. This turns-out that the flows with high priority are given the first chance to reconfigure. We define U_c^p, U_{wlan}^p and U_{mf}^p which respectively indicate the user-preference on cellular, WLAN and femtocell networks.

$$\Lambda_j = \begin{cases} Q_j \ .\ exp\left(\dfrac{U_c^p - U_{wlan}^p}{1 + \sqrt{U}_c^p}\right) & \text{(i)}, \\ Q_j \ .\ exp\left(\dfrac{U_c^p - U_{mf}^p}{1 + \sqrt{U}_c^p}\right) & \text{(ii)}, \\ Q_j \ .\ exp\left(\dfrac{U_c^p - U_{mf}^p - U_{wlan}^p}{1 + \sqrt{U}_c^p}\right) & \text{(iii)}. \end{cases} \tag{8}$$

Fitness function counts Λ_j to make decision for handling the j^{th} flow. In (9), we define the fairness function $F(\theta, \Lambda)$ with the upper threshold set to 0.5 and lower threshold set to 0.2. When contentment-index is small, the value of $F(\theta, \Lambda)$ increases. The same behaviour could be observed when load-index is high. Thus, we select the base station with minimum $F(\theta, \Lambda)$ to admit new flows and to transfer existing flows.

$$F(\theta, \Lambda) = argmin\left[\frac{\theta_i}{1 + \Lambda_j}\right]. \tag{9}$$

C. An analytical approach with birth-death process

In this section, we compare our solution with an analytical model to ensure correctness and effectiveness. We have adapted the model by Song at el. to measure the performance of this approach [6], [7]. Here, we compare the performance of both cellular only and cellular/WLAN integrated systems. Let's assume each cell supports C_i^c users, which is similar to K in our probabilistic model. The parameter, n_i denotes the number of calls connected with the i^{th} cell. The call arrival follows the Poisson process with the rates λ_i^c and λ_k^w which define the arrival rates of cellular and WLAN respectively. The channel holding time in each case is defined with $1/\mu_i^c$ and $1/\mu_i^w$. We define the super-scripts "c", "w", "r" and "h" for cellular, WLAN, new, and handover traffic respectively. The call admission process is described below. Here, we have used two admission control mechanisms, namely "cut-off priority" and "fractional guard channel" [8], [9]. The already connected users are given the first priority, since experiencing abrupt terminations are annoying. Cut-off priority is reserving a portion of channel for handover flows where the system accepts only the handover requests beyond a threshold T_i^c.

Fractional guard channeling is the technique of accepting new connections when $n_i > (C_i^c - Ti^c)$ with a probability of ω_i depend on the current channel occupancy (n_i). When $n_i \leq (C_i^c - T_i^c)$, new connections and the handover connections can be accepted under the same probability. This call connection could be seen as a birth-death process. Therefore, by considering the steady state probability, the following formulas could be derived for a cell i. In this subsection, ρ and α define the birth-rates.

$$P_i^c(n_i - 1)\rho_i^c = P_i^c(n_i).n_i\mu_i^c, \qquad 0 \leq n_i \leq (C_i^c - T_i^c). \tag{10}$$

$$P_i^c(n_i - 1)\alpha_i^c = P_i^c(n_i).n_i\mu_i^c. \qquad (C_i^c - T_i^c) < n_i \leq C_i^c. \tag{11}$$

$$\sum^{i} P(n_i) = 1 \qquad for\ all\ i. \tag{12}$$

In this model, the arrival of calls is regarded as one birth-death process and as mentioned above, we can learn that ρ_i is the birth-rate of state i whereas, $n_i\mu_i$ is the death rate at state i. $P_i(n_i)$ denotes the probability of states n_i and the sum of all status is equal to 1 (12). Thus, the connection blocking probability could be expressed as follows.

$$B_i^c = \sum_{n_i = C_i^c - T_i^c}^{C_i^c} P_i^c(n_i)(1 - \omega_i^c) + P_i^c(C_i^c). \tag{13}$$

The handover blocking probability is defined with,

$$B_{h_i}^c = P_i^c(C_i^c). \tag{14}$$

Considering the steady state probability of WLAN, we have derived (15) and (16). Equation (15) is derived when $0 \leq m_k \leq (C_k^w - T_k^w)$ while, (16) is derived when

$(C_k^w - T_k^w) < m_k \leq C_k^w$ where m_k is the number of users in WLAN k.

$$P_k^w(m_k-1)\rho_k^w = P_k^w(m_k).m_k\mu_k^w. \quad (15)$$

$$P_k^w(m_k-1)\alpha_k^w = P_k^w(m_k).m_k\mu_k^w. \quad (16)$$

Thus, new connection blocking probability with WLAN (B_i^w) and the new handover probability ($B_{h_k}^w$) are respectively defined in (17) and (18).

$$B_i^w = \sum_{m_k=(C_k^w-T_k^w)}^{C_k^w} P_k^w(m_k)(1-\omega_k^w) + P_k^w(C_k^w). \quad (17)$$

$$B_{h_k}^w = P_k^w(C_k^w). \quad (18)$$

Correspondingly, the birth rate of cellular cell i is respectively defined for $n_i \leq (C_i^c - T_i^c)$ and $n_i > (C_i^c - T_i^c)$ as follows,

$$\rho_i^c = \lambda_i^c + \sum_{j\in A_i^c} V_{ji}^{cc} + \sum_{k\in W_i^c} V_{ki}^{wc} + \sum_{l\in W_i^c} \gamma_{li}^w. \quad (19)$$

$$\alpha_i^c = \lambda_i^c\omega_i^c + \sum_{j\in A_i^c} V_{ji}^{cc} + \sum_{k\in W_i^c} V_{ki}^{wc} + \sum_{l\in W_i^c} \gamma_{li}^w. \quad (20)$$

The horizontal handover rate V_{ji}^{cc} defines the rate of cellular Cell j offered to Cell i.

$$V_{ji}^{cc} = \lambda_j^c(1-B_j^c)p_{ji}^{cc} + \sum_{x\in A_j^c} V_{xj}^c(1-B_{h_j}^c)p_{ji}^{cc} + \sum_{y\in W_j^c} V_{yj}^w(1-B_{h_j}^c)p_{ji}^{cc} + \sum_{z\in W_j^c} \gamma_{zj}^w(1-B_{h_j}^c)p_{ji}^{cc}. \quad (21)$$

$$\gamma_{zj}^w = V_{jz}^{cw}B_{h_z}^w + \sum_{l\in A_z^w} V_{lz}^{ww}B_{h_z}^w. \quad (22)$$

The vertical handover rate of WLAN access point k offered to overlay cellular cell i is defined as;

$$V_{ki}^{wc} = \lambda_k^w(1-B_k^w)p_{ki}^{wc} + \sum_{x\in A_k^w} V_{xk}^{ww}(1-B_{h_k}^w)p_{ki}^{wc} + \sum_{y\in D_k^w} V_{yk}^{cw}(1-B_{h_k}^w)p_{ki}^{wc} + \sum_{z\in D_k^w} \varsigma_{zk}^c(1-B_{h_k}^w)p_{ki}^{wc}. \quad (23)$$

$$\varsigma_{zk}^c = \left(V_{kz}^{wc}B_{h_z}^c + \sum_{l\in A_z^c} V_{lz}^{cc}B_{h_z}^c\right)R_{zk}. \quad (24)$$

In a similar manner, the occupancy of WLAN access point k with birth rates ρ_k^w and α_k^w based on the state m_k and death rate $m_k\mu_k^w$ are defined below. The birth rate in WLAN is defined in (25) and (26) for $m_k \leq (C_k^w - T_k^w)$ and $m_k > (C_k^w - T_k^w)$ respectively;

$$\rho_k^w = \alpha_k^w + \sum_{l\in A_k^w} V_{lk}^{ww} + \sum_{j\in D_k^w} V_{jk}^{cw} + \sum_{g\in D_k^w} \varsigma_{gk}^c. \quad (25)$$

$$\alpha_k^w = \alpha_k^w\omega_k^w + \sum_{l\in A_k^w} V_{lk}^{ww} + \sum_{j\in D_k^w} V_{jk}^{cw} + \sum_{g\in D_k^w} \varsigma_{gk}^c. \quad (26)$$

The vertical handover rate of cellular Cell j offered to WLAN k is given below:

$$V_{jk}^{cw} = \lambda_l^c(1-B_l^c)R_{jk} + \sum_{x\in A_j^c} V_{xj}^{cc}(1-B_{h_j}^c)R_{jk}q_{jk}^{cw} + \sum_{y\in W_j^c} V_{yj}^{wc}(1-B_{h_j}^c)R_{jk}q_{jk}^{cw} + \sum_{z\in W_j^c} \gamma_{zj}^w(1-B_{h_j}^c)q_{jk}^{cw} \quad (27)$$

In order to initiate the simulation, we have set the parameters $B_i^c = B_{h_i}^c = B_k^w = B_{h_k}^w = 0$. Using Erlang's fixed-point approximation we compute the blocking probability and resource allocation per-user. We have set $C_i^c = 30, C_k^w = 35, T_i^c = 5, T_k^w = 5$ and $R_{zk} = 0.75$. The parameters $\mu_i^c = 1, \mu_k^w = 0.5, \lambda_i^c$ and λ_k^w are set to comply with the probabilistic model. Then we analytically evaluate the system.

IV. Results and Analysis

Enhanced flexibility enabled with the flow tables consisting of match fields, counters, and instructions is the motivation behind SDN based load balancing. OpenFlow as a SDN enabler not only provides enough flexibility, but also aggregates network statistics required for load balancing. In Fig. 2, we have presented unsatisfied user percentage of the probabilistic approach for different flow intensities. The simulation model presents couple of scenarios with and without load balancing and many other services. In a nutshell, the unsatisfied user percentage without load balancing is almost 42% at highest intensity. With load balancing, the flows are transferred across

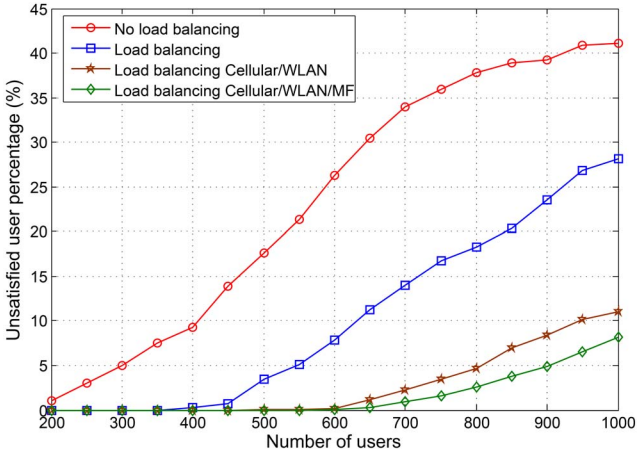


Fig. 2. Unsatisfied-user percentage: with increasing flow/user-intensity, flows would not acquire the capacity that they demanded, thus, allocate the minimum affordable flow space.

different network options while attempting to equalize utilized flow space. This is an improvement of 150% compared to no load balancing. With probabilistic model, flows are optimally configured across the networks while excess flows are simply dropped without reserving resources. Furthermore, we have also measured the unsatisfied-user percentage with different network options; WLAN and mobile femtocell (MF). The probabilistic model reduces the unsatisfied user percentage drastically which is almost five times less compared no load

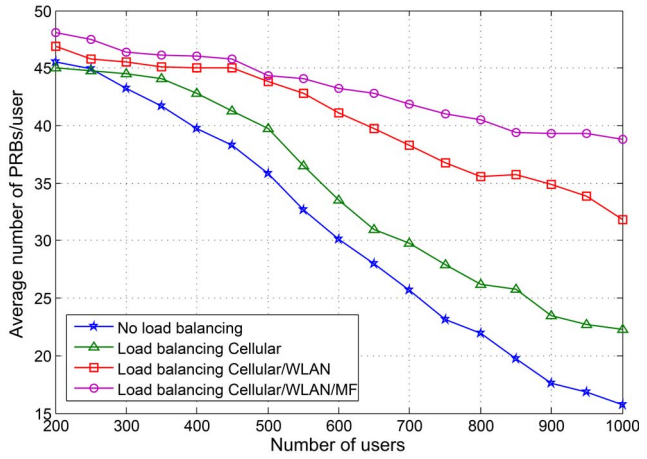


Fig. 3. Average resource allocation per-flow: while increasing flow intensity, a drastic reduction of allocated flow space is noticed. Our algorithm attempts to enhance per-flow space by transferring and carefully admitting new flows.

balancing at the highest intensity. In Fig. 3, we present the averaged resource allocation per-user. At low intensity, we have seen almost the similar level of performance where 45-50 PRBs per-flow. The relation between resource allocation and user intensity has more or less a linear combination. Without load balancing, the averaged resource allocation per-flow remains around 16 PRBs at the highest intensity. At this intensity there is a huge drop of flows due to non-organized flow admission. Resource allocation with the probabilistic approach is an improvement of 137% with only cellular network. In one hand, WLANs enhance coverage and capacity

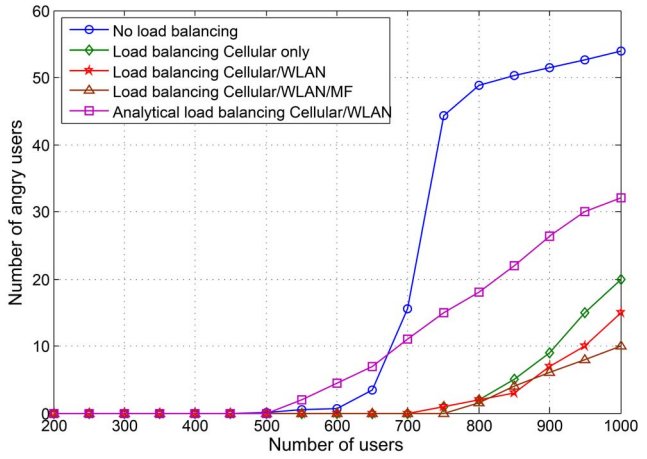


Fig. 4. Angry-user percentage: Even with load balancing, after some point, network can not offer resources to new & transferring flows due to mobility.

in the network as a whole. On the other hand, it improves per-flow allocation almost by 200% compared to no load balancing at its highest intensity. The algorithm proves per-flow allocation could be further improved by introducing the MF network. In a nutshell, reduced allocation turns out that the users gain access to low bandwidth services only. SDN as a common platform for inter-system communication enables seamless mobility between the systems and ensures that the users can seamlessly share flow space to enhance QoS. Fig. 4 illustrates angry-user percentage (drop-rate) correspond to different intensity levels. Without load balancing, we have noticed a drop of large number of users. According to Fig. 4, the maximum number of flows the network can afford without overloading is around $N_{user} = 700$. Both probabilistic and analytical approaches normalize the flow-space among flows, thus avoids overloading the network. In general, the level at which a network is utilized has a direct impact on QoS. The probabilistic approach reduces the drop-rate almost by 520% compared to no load-balancing and by 300% compared to the analytical model. The comparison of analytic and probabilistic approaches proves the theoretical efficiency and the correctness of our algorithm.

V. CONCLUSION

In this paper, we have proposed and evaluated a novel load balancing mechanism leveraged by flow admission control. Seamless connectivity enabled with SDN is the bottom-line of this work which ultimately offloads core network, maximizes the per-flow capacity, and enhances the end-user experience by means of reduced waiting time and drop-rate. Most strikingly, the results revealed that probabilistic approach has reduced unsatisfied-user percentage almost by five times. Our model reveals a 237% of improvement in terms of per-flow resource allocation. Furthermore, we have noticed a drastic reduction of drop-rate (300%) compared to the analytical model and almost 520% of reduction compared to no load-balancing. Overall, our findings in this paper have elaborated the ultimate gain of load balancing in the SDN context and verified the results based on an analytical model.

REFERENCES

[1] P. Lin, J. Bi, and H. Hu, "ASIC: an architecture for scalable intra-domain control in OpenFlow," in *Proceedings of the 7th International Conference on Future Internet Technologies*. ACM, 2012, pp. 21–26.

[2] T. Janevski, "5G mobile phone concept," in *Proceedings of the 6th Consumer Communications and Networking Conference*. IEEE, 2009, pp. 1–2.

[3] A. M. Mousa, "Prospective of Fifth Generation Mobile Communications," *International Journal of Next-Generation Networks*, vol. 4, no. 3, 2012.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[5] J. Keilson and L. Servi, "A distributional form of Little's law," *Operations Research Letters*, vol. 7, no. 5, pp. 223–227, 1988.

[6] L. Yang, G. Song, and W. Jigang, "A performance evaluation of cellular/wlan integrated networks," in *4th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)*. IEEE, 2011, pp. 131–135.

[7] G. Song, L. Yang, J. Wu, and J. Schormans, "Performance comparisons between cellular-only and cellular/wlan integrated systems based on analytical models," *Frontiers of Computer Science*, pp. 1–10, 2012.

[8] R. Ramjee, D. Towsley, and R. Nagarajan, "On optimal call admission control in cellular networks," *Wireless Networks*, vol. 3, no. 1, pp. 29–41, 1997.

[9] S. S. Rappaport, "The multiple-call hand-off problem in high-capacity cellular communications systems," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 3, pp. 546–557, 1991.

Enabling Secure Mobility with OpenFlow

Suneth Namal*, Ijaz Ahmad†, Andrei Gurtov‡ and Mika Ylianttila§
*†§Department of Communications, University of Oulu, Finland
‡Department of Computer Science and Engineering, Aalto University, Finland
Email: [*namal,†iahmad]@ee.oulu.fi,‡gurtov@hiit.fi,§mika.ylianttila@oulu.fi

***Abstract*—Software Defined Networking (SDN) and its one possible realization, OpenFlow, define the trends of future networks. However, the present OpenFlow architecture does not allow the switches to be mobile e.g., in a moving train as it would disrupt flow processing from network switches. We present OFHIP, an architecture that enables OpenFlow switches to change their IP addresses securely during mobility. OFHIP employs IPSec encapsulated security payload (ESP) in transport mode for protection against DoS, data origin authenticity, connectionless integrity, anti-replay protection, and limited traffic flow confidentiality. We demonstrate the benefits of OFHIP compared to present use of SSL in enabling mobility, reducing the connection latency and improving the resilience to known TCP-level attacks.**

I. Introduction

In SDN architecture, the control layer is logically centralized to a software-based controller which maintains the global view of the network. The controller allows the flexibility to configure, manage, secure, and to optimize network resources via automated and dynamic software programs. OpenFlow is one of the realizations of SDN that provides freedom to exploit new opportunities over the existing networking infrastructure. OpenFlow at the current stage does not support mobility [1]. This is a critical downfall that limits OpenFlow into fixed core-networks, clusters or data centers.

Robust multipath connectivity for fault tolerance is one of the critical issue inhabits in the current SDN architecture. This is even more crucial when the controller or the switch is mobile. In one hand, multiple channels between the switch and the controller can guarantee a certain level of assurance for seamless connectivity. On the other hand, multipath connectivity inspired by the efficient use of installed bandwidth and increased robustness by simultaneous use of potentially diversified paths that result to optimally utilize the network resources and increase redundancy.

Mobile base stations, mobile routers, switches, mobile clouds and server migration are some of the compelling reasons why mobility is insisted in software defined networks [2]. This would extend SDN benefits into mobile environments e.g., moving trains, buses, flights and other automobiles. The existing competing solutions with Mobile IP (MIP) still have problems related to "triangle routing" and drop of IP packets due to frequent handover when the host is away from Home Agent (HA). Mobility offers many dazzling opportunities that also bring with them some profound challenges related to security and privacy [3]. We argue that mobile IP has limitations against DoS, passive eavesdropping, insider attack, replay attack, tunnel spoofing and location privacy [4], [5]. Security in IP based networks is widely tackled with a common set of protocols composed of secure file transfer protocol (SFTP), secure socket layer (SSL), and transport layer security (TLS). OpenFlow enables an acceptable level of security with SSL or TLS though, it does not support mobility [6]. These limitations insist modifications to the current OpenFlow architecture. More specifically, below we describe the major problems of present OpenFlow version.

- **Flow processing :** Change of address would disrupt flow processing from network switches. Therefore, they require fast and regular updates to flow tables.
- **Secure session management :** Changing an IP address may also tear down active SSL/TCP sessions.
- **Secure handover :** Problem of mutual authentication and cannot support mobility and certificate exchange is not preferable for fast moving OpenFlow clients.
- **Flow rule management :** Change of IP address to solve latter issue causes additional overhead, since flow rules must be updated frequently.

This paper proposes a novel approach to handle OpenFlow based mobility with global identities introduced by the host identity protocol (HIP) layer. These identifiers are of the same format as IPv6 addresses. Furthermore, IPv6 is already proposed for the next version of OpenFlow and thus, applications could be easily adopted. In general, OFHIP addresses the security bottleneck of current OpenFlow version.

The rest of this paper is structured as follow. Section II, presents OpenFlow based connection establishment. Section III describes the proposed mobility architecture and scenario. Next, we present our implementation and results in Section IV. In Section V, we discuss the future research directions and some of the important findings. Finally, in Section VI, we conclude this paper.

II. OpenFlow based connection establishment

The present use of SSL over TCP introduces new security threats to known TCP-level attacks. There are several serious security flaws inherent in the protocol, regardless of the correctness of any implementation (e.g. SYN attacks, reset attack, sequence prediction attack, ICMP attacks and DoS attacks). The OpenFlow switches and routers connect to the remote control processor, namely the controller which handles the TCP requests. Therefore, an attacker will first attempt to break into the controller. Thus, any security weakness in the controller will allow attackers to penetrate into the network.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

Though, OpenFlow specification mentions the applicability of user datagram protocol (UDP) with datagram transport layer security (DTLS), there are no detailed discussions of how it could be used. Certain TLS implementations do not currently check client certificates. In case, if a similar approach is used with OpenFlow, it could be vulnerable to any kind of man-in-the-middle attack. The specification does not either provide the certificate format or indicate what fields in the certificate are used for naming. OpenFlow refers the use of "TLS" without specifying a reference or a version number. This would result in non-interoperable implementations if different OpenFlow implementations use different versions.

III. Proposed Mobile Architecture and Scenario

In Table I, we have compared the HIP layer options with the existing SSL/TLS solution. In a nutshell, OFHIP solution is an integration of diet version of HIPv1 [7] layer with the existing OpenFlow version to replace the SSL/TLS based mutual authentication. As a result, we would get almost the same level of security with enhanced mobility support. It provides end-to-end encryption, mutual authentication and secure key exchange. The HIP layer identifies a host either by a host identifier (HI) or a host identity tag (HIT) defined in [7]. HI is the public key of an asymmetric key-pair which could be used as a local identity. However, it is not suitable to serve as a packet identifier, since the length can vary [8].

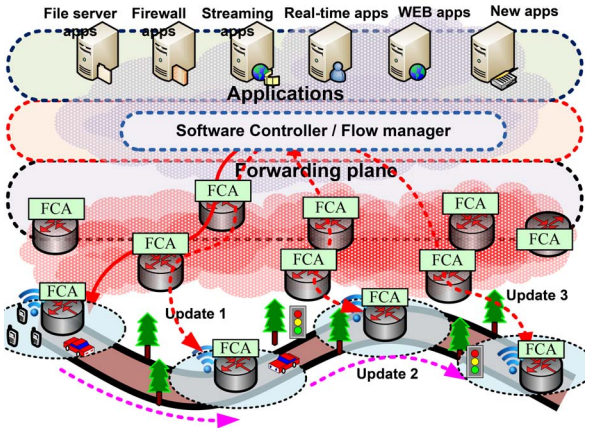


Fig. 1. SDN based mobile scenario.

In Fig. 1, we present SDN based mobility scenario where the switches or routers are configured on the fly. The flow control agents (FCA) on the mobile switches and routers are responsible for updating the software controller of the new location information for location based services. Besides that, the flow rules could be built on top the newly introduced cryptographic global identifiers which do not change over time. Thus, updating the flow rules due to mobility is no longer required. This would reduce the processing and control traffic overhead due to dynamic address configuration and thus, flow processing. Furthermore, it reduces overhead in the policy charging and rules function (PCRF) and improve the flexibility with QoS management and network configurations. Moreover, controller can keep records of HITs and authenticate them through controller signed ceertificates or using other techniques, such as DNSSEC and DHT-based verification.

The existing Mobile IP (MIP) solutions address the same problem in a different manner with the home address (HA) which always identifies the mobile node. The care-of address (CoA) associates the mobile node with its home address by providing information about the mobile node's current point of attachment. Mobile IP uses a registration mechanism to register the CoA with the HA. This process consumes a considerable amount of time as it is shown in [2]. Thus, we have proposed HIP layer "UPDATE" procedure to inform the peers of the new addresses. Fig. 2 presents our OFHIP solution which addresses mobility in OpenFlow.

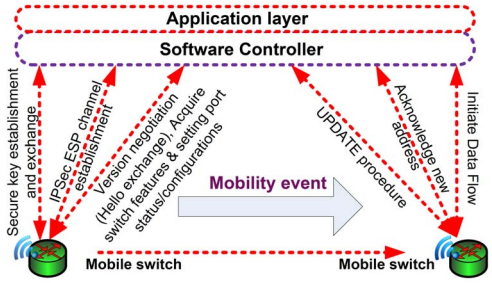


Fig. 2. OFHIP solution for secure connection establishment.

Mutual authentication in OpenFlow is a part of security assertion which is achieved with SSL or TLS. They operate at the transport layer in the OSI stack, and provide secured data transport for applications. It supports peer negotiation for algorithm selection, public key based exchange of secret session keys and X.509 certificates. However, they do not support mobility alone without the support of an underlying protocol (UDP, SCTP and etc). The modified version of TLS: DTLS can be used by applications directly or by tunneling to provide secure mobility [9]. However, with legacy IP protocols, the handover impact at the upper layers is still tight.

OFHIP uses IPSec encapsulating security payload (ESP) secure associations (SAs) that are bounded to HITs. Therefore, address reconfiguration would not have any impact on the higher-layer associations except the changes in network routing layer. The key-exchange in OFHIP is a cryptographic protocol that uses a randomly generated key encrypted by a Diffie-Hellman derived key in order to establish a pair of IPsec-ESP SAs between two entities: the initiator and the responder. The HIP layer in OFHIP solution maps arriving ESP packets to a HIT using the security parameter index (SPI) value in the packet and selects the source address and interface according to the SPI value set by ESP. After handover, data continues to flow inside of the ESP tunnel with the same SPI values but with a different IP address at the mobile node.

The initiator defines SPI value of the responder's outgoing SA, whereas the responder defines the SPI value of the initiators outgoing SA. To prevent replay attacks, we propose to use an incremental counter with a hash of the HIT. For secure mobility, rekeying may be necessary. Thus, new SAs

must be created by removing the old SAs. Once a host receives data on new SA, it can safely remove the old SA. The HIP layer uses AES-CBC for symmetric encryption and to provide CMAC for MACing functions while the session keys are encrypted with elliptic curve diffie-hellman (ECDH) keys.

The four-packet exchange makes OFHIP resilient to denial-of-service (DoS). The protocol transmits an EC Diffie-Hellman encrypted key in the 3^{rd} and 4^{th} packets, and authenticates the parties with those packets. The responder starts a puzzle exchange with the initiator in the 2^{nd} packet, and completes it in the 3^{rd} packet before the responder stores any state from the exchange. This model falls in the line of TLS and fairly equivalent to 802.11 master and pair-wise transient key, but handled in a single exchange.

IV. Implementation and Results

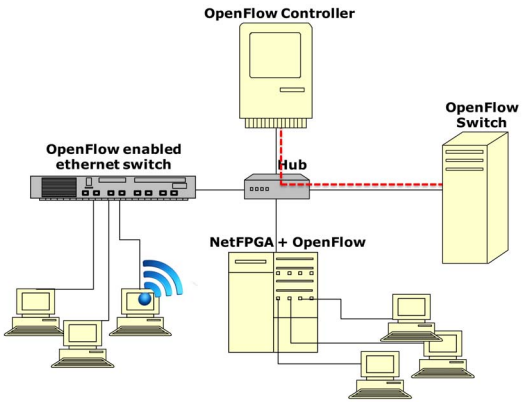


Fig. 3. OpenFlow experimental setup.

In a nutshell, the OFHIP architecture is an attempt to introduce secure mobility with OpenFlow. In this work, we have developed the OpenFlow test-bed presented in Fig. 3. The test-bed consists of two DELL PowerEdge-R320 servers, a DELL PowerEdge-2900 server, a HP-6600 Ethernet switch and two laptops. The PowerEdge-2900 server is installed a "NetFPGA open platform for gigabit network" card developed by Stanford university [10]. This platform has four physical gigabit interfaces, internal memory, user defined network processing logic, and field programmable gate array (FPGA) that allows designing of routers and other networking hardware. We have installed CentOS kernel 2.6.18 to support the NetFPGA platform. This platform is loaded a compiled OpenFlow bit-file. The HP-6600 Ethernet switch is configured with the latest OpenFlow-enabled software version K.15.06.5080 [1].

We have used a laptop with an i5 CPU of 2.67GHz as the controller and another laptop of CPU 2.16GHz as a switch (both running on Linux kernel 2.6.35). The OpenFlow version 1.1.0 defines both non-SSL and SSL support that must be configured at the built. As it is presented in Fig 1, the secure connection establishment in OpenFlow has several phases. The HIP layer in OFHIP is responsible for mutual authentication and secure key exchange. However, this solution does not literally change the flow of OpenFlow based connection establishment. The HIP layer replaces the SSL based mutual authentication and connection establishment in OpenFlow.

The OpenFlow "HELLO" messages include the version, type, length and transaction-ID associated with the packets. The "Feature" request/reply messages check and agree on the switch/controller compatibility upon establishment of an OpenFlow channel. Followed by it, the SET-CONFIG messages could be used by the controller to set configuration parameters appropriately. In case, if it happens to close the connection due to version incompatibility or any other issue between the switch and the controller, the "CLOSE" message type in the HIP layer could be used.

First, we have investigated the options for OpenFlow based connection establishment between the switch and the controller. Fig. 4 presents both secure and insecure connection establishment with SSL and non-SSL options. In order to use SSL with OpenFlow, it is required to set-up a public-key infrastructure (PKI) which includes a pair of certificate authorities (CAs) for the controller and the switch. We have used a script to generate the PKI. Thereby, we have established the private keys and certificate authorities' certificates for the switch and the controller, and root certificates for their CAs.

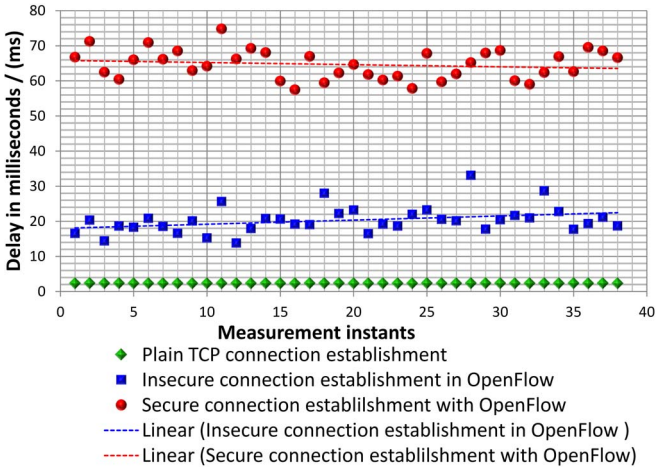


Fig. 4. Plain TCP and secure and insecure connection establishment delay with OpenFlow.

As it is depicted in Fig. 4, SSL exchange consumes a considerable amount of time compared to insecure connection establishment. The averaged delay for connection establishment with these options are measured to be 66 ms and 20 ms respectively. According to the results, the delay with SSL certificate option is almost as three times of the non-SSL option. The graph also presents linear regression of results. Plain TCP connection establishment phase is measured to be 2-3ms which could be almost neglected compared to that of the SSL-certificate exchange. The SSL based handshake in OpenFlow uses X.509 standard certificate type with RSA. Thus, this exchange is heavy and consumes relatively a long time compared to insecure session establishment over plain TCP. However, mobility as an essence in modern communi-

TABLE I
COMPARISON OF SECURITY AND MOBILITY FEATURES OF SSL/TLS, HIP-BEXV1, AND HIP-DEX.

	SSL/TLS	HIP-BEXv1	HIP-DEX
Mobility extendibility	Low (leverage mobility protocol)	High (Multihome extension)	High (Multihome extension)
Mutual authentication	High (SSL handshake protocol)	High (With four ways handshake)	High
Key exchange and encryption	High (RSA & CCM-AES)	High (Diffie-Hellman & RSA/DSA)	High (ECDH and AES encryption)
Signaling integrity	Medium (DSA)	High (HMAC - use HIP-gl or HIP-lg)	Low (external whitelisting)
Data integrity	Medium (symmetric-DES)	High (ESP-CBC)	High (AES-CCM)
Message authentication	HMAC (SHA-256)	HMAC (with SHA-1 or MD5)	CMAC
Replay attack resistance	High (with 3rd party)	High (with 3rd party)	High (with 3rd party)
Non-repudiation protection	Low (Standard TLS)	High (self certified)	High (self certified)
Man-in-the-Middle protection	Low	High (non-opportunistic)	Medium
Denial of Service attack	Medium	High (by packet design)	High (by packet design)
Forward secrecy	Low	High (with legacy applications)	Low (KeyWrap of random-keys)

cation does not perform well with connection oriented TCP. Therefore, these solutions that are already in OpenFlow would not meet the demanding high bandwidth and reduced latency.

In Fig. 5, we compare the delay in connection establishment with OpenFlow and OFHIP. As it is depicted from Fig. 5, IPSec connection establishment with OFHIP consumes around 44 ms whereas secure connection establishment with OpenFlow is about 66 ms. OFHIP delay consists of the delays associated with the wireless interface which encounters the wireless propagation delay and the 802.11 processing delay. These results here depict that the delay in secure connection establishment with OFHIP is almost close to that of the insecure connection establishment with OpenFlow which is measured through the wired interface. DTLS is proposed to

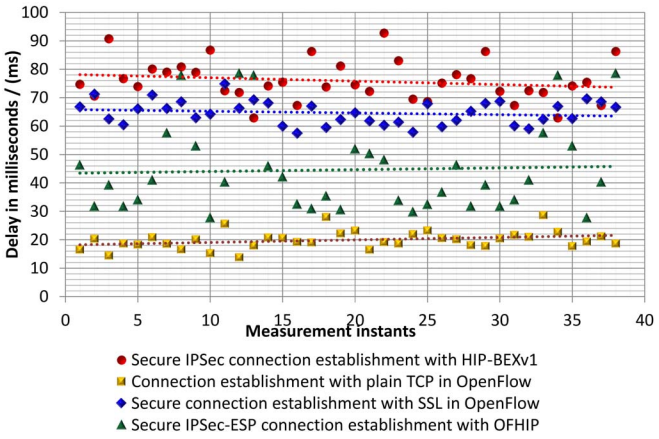


Fig. 5. Comparison of OFHIP with OpenFlow.

fill-up the mobility gap of OpenFlow. In one hand, it is deliberately designed to be similar to TLS as possible [9]. On the other hand, TLS cannot be directly used with datagram due to loss or reordering of packets. However, DTLS has the minimal changes to TLS to fix this problem. Since, DTLS is almost identical to TLS, we can expect same level of performance as it is with SSL/TLS. As a result, we believe the OFHIP solution has the high potential in terms of mobility, since the presented results are far below the requirements of widely used mobile applications. Next, we have measured the throughput characteristics of OFHIP. In order to analyze traffic characteristics, we install Jperf in the controller and the switch. Jperf is a graphical interface developed for Iperf. It allows easy configuration of parameters. Iperf is a testing tool which can create TCP and UDP traffic flows and measure throughput characteristics based on client/server mode [11]. We have investigated both TCP and UDP throughput of OFHIP. Fig. 6 presents throughput behavior of OpenFlow based plain TCP and secure OFHIP based ESP-IPSec channel over a 2Mbps limited bandwidth channel.

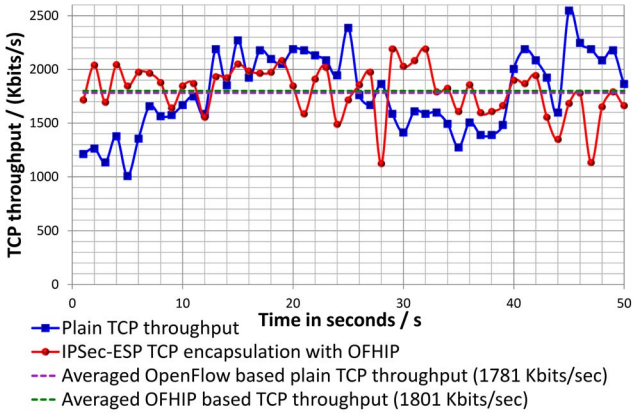


Fig. 6. TCP throughput (window size 83.5KBytes)

.

We have used local scope identifiers (LSIs) with Jperf for local communication in OFHIP. Therefore, sessions do not terminate even after IP renewal. However, the observations have verified that throughput drops to zero during handover since TCP is connection oriented. Thus, it is clear that TCP data transmission using ESP encapsulation over OFHIP cannot be used with mobile switches though; it has almost the same throughput as plain TCP in a fixed network.

In Fig 7, we have compared the plain UDP throughput and IP/ESP encapsulated UDP throughput of OFHIP. Using Ipref, the wireless bandwidth is limited to 1 Mbps. We have investigated the throughput characteristics of mobility events with a script that automatically changes the IP address of the device. The HIP layer handles address update and informs the recipient of the new address. After acquiring a new address,

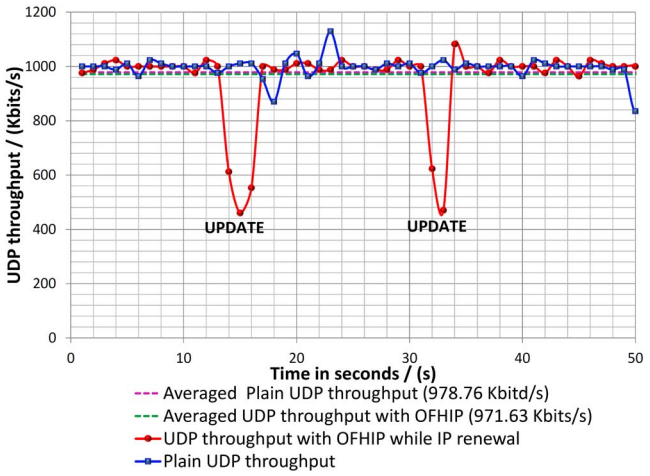


Fig. 7. UDP throughput (MTU-1470 byte, Buffer-160KBytes).

UDP traffic is redirected according to the address update procedure implemented in the HIP layer.

Therefore, throughput remains almost stable even with the mobility events which we have scripted. Fig. 7 illustrates two such instants where the throughput suddenly drops due to the scripted mobility events. The measured UDP encapsulated IPsec ESP throughput is almost similar to the unencrypted UDP throughput. Thus, OFHIP uses UDP encapsulation of IP traffic with mobile applications. It must be stated here that the actual packet loss during handover is the time of HIP layer "UPDATE" procedure call. Hence, multi-homing is a promising solution to avoid packet-loss during handover. Moreover, handover buffers are capable of avoiding any such packet loss that could be happening during handover. Since, HIP layer uses a separate identity other than the IP address, it can also enable multi-homing as a counterpart.

V. Discussion

With OFHIP, authentication is much faster compared to legacy cryptographic protocols due to elliptic curve cryptography (ECC) based key generation that offers the same level of security with relatively small key sizes. Despite that, the flow-tables with the new cryptographic identifiers can simplify mobility management, since it is not necessary to update the flow-tables when the flows are built on top of those identifiers.Therefore, the matching rules can be made globally unique and available for global mobility management. Furthermore, it allows redefining data-path identifiers by using these cryptographic identifiers. Moreover, we can prove concatenation of two globally unique identifiers can again produce unique identity which does not collide with another identifier. Thus, we propose that it could be passed through a hash function to generate the 64-bit cryptographic data-path identifiers. Otherwise, 128 bits lengthy identifiers could be used for global level flow processing and identification.

Commercialization of OpenFlow requires guaranteed connectivity between the switch and the controller which could be realized with robust multipath connectivity. Equal Cost Multipath (ECMP) or BGP Multi-Exit Discriminators (MEDs) addresses this problem though they are lacking the required flexibility and scalability. The HIP layer in OFHIP enables multihoming and thus allows to configure multiple identities. Inspired by multihoming, OFHIP could be an enabler for multipath connectivity to the OpenFlow controller. Ultimately, it will end-up utilizing the network resources optimally and enhance data-rate besides the improved capability for fault tolerance and redundancy management.

VI. Conclusion

This paper is an attempt to introduce mobility into legacy OpenFlow based switches and to improve resilience against the known TCP-level attacks. OFHIP introduces a cryptographic name space which is identical to the IPv6 address space. They are cryptographically globally unique and ideal for developing mobile applications, since they do not change with mobility events. The results depict that the handover latency in OFHIP is well below the limits of delay requirements of mobile applications. In comparison of the number of SAs, OpenFlow supports up to 15 secure associations per second, whereas OFHIP supports upto 22 secure associations per second. This is an improvement by 147% compared to legacy SSL connection latency. We have shown that OFHIP has better throughput compared to TCP backhaul throughput and stands very well against the scripted mobility events with UDP encapsulated IPSec-ESP. In a nutshell, the results prove benefits of OFHIP compared to present use of SSL in terms of mobility, reduced connection latency and improved security.

References

[1] ONF. (2013) Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/

[2] S. Namal, M. Liyanage, and A. Gurtov, "Realization of Mobile Femtocells: Operational and Protocol Requirements," *Wireless Personal Communications*, pp. 1–26, 2012.

[3] S. Namal, J. Pellikka, and A. Gurtov, "Secure and multihomed vehicular emtocells," in *75th Vehicular Technology Conference (VTC Spring)*. IEEE, 2012, pp. 1–5.

[4] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "OpenRoads: Empowering research in mobile networks," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 125–126, 2010.

[5] S. Namal, K. Georgantas, and A. Gurtov, "Lightweight authentication and key management on 802.11 with Elliptic Curve Cryptography," *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pp. 1830–1835, 2013.

[6] B. Boughzala, R. Ben Ali, M. Lemay, Y. Lemieux, and O. Cherkaoui, "OpenFlow supporting inter-domain virtual machine migration," in *8th International Conference on Wireless and Optical Communications Networks (WOCN)*. IEEE, 2011, pp. 1–7.

[7] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "RFC 5201: Host Identity Protocol," *Network Working Group*, 2008.

[8] A. Gurtov, "Host Identity Protocol (HIP): Towards the Secure Mobile Internet," *Wiley Publishing*, 2008.

[9] E. Rescorla and N. Modadugu, "RFC 4347: Datagram transport layer security," 2006.

[10] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ACM, 2008, pp. 1–9.

[11] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," *http://dast.nlanr.net/Projects*, 2005.

SDN for Inter Cloud Networking

Marouen Mechtri†, Ines Houidi‡[1], Wajdi Louati‡[1], Djamal Zeghlache†
†Institut Mines-Telecom, Telecom SudParis, UMR CNRS 5157, Evry, France
‡National Engineering School of Sfax, Tunisia
†Email: {marouen.mechtri, djamal.zeghlache}@it-sudparis.eu
‡Email:{ines.houidi, wajdi.louati}@gmail.com

***Abstract*—This paper presents a Software Defined Network (SDN) controller, called Cloud Networking Gateway (CNG) Manager, that enhances networking of distributed cloud resources and provides authorized customers with the ability to control and configure networks. The CNG Manager interconnects virtual machines acquired from distributed heterogeneous resources and services from multiple providers using a generic gateway. The cloud networking gateways are managed by the CNG Manager that handles allocation and configuration of the gateways according to connectivity requirements. Our implementation of the CNG Manager and the gateway is combined with an exact splitting algorithm and integrated in a cloud services provisioning system. The CNG Manager and the associated gateway extend the current state of the art by applying the SDN principle to connectivity control of distributed and networked cloud resources.**

***Index Terms*—Cloud Networking Gateway Manager; CNG; cloud broker; request splitting and provisioning**

I. Introduction

This contribution addresses the networking of distributed cloud resources and services from multiple providers. The networking of virtual resources dedicated to applications, end users or tenants needs more attention to facilitate and automate the establishment of inter providers links and more specifically the control, configuration and instantiation of connectivity to take full advantage of compute, storage and network virtualization.

The cloud community started to address networking requirements in OpenStack Neutron [1], for instance, but continues to put the priority on intra data center networking. Cloud users can control and manage their applications but have no control on the connectivity and networking of their dedicated and distributed cloud services. Network providers, users and tenants need more flexibility in deploying, configuring and instantiating cloud networking services to manage more easily and efficiently their resources.

To provide this desired control of connectivity, a Software Defined Networking (SDN) solution is proposed to handle inter cloud networking. SDN [2] is a network architecture and design that decouples data-plane forwarding from control and management plane functions. The SDN paradigm enables the transparent integration of applications provisioning in clouds and networks via programmable interfaces and automation.

[1]Ines Houidi and Wajdi Louati contributed initially to this work while at Telecom SudParis and more recently as faculty members of National Engineering School of Sfax.

The proposed networking architecture can be seen as an SDN controller built around two main components: a) an SDN controller called Cloud Networking Gateway Manager (CNG Manager) and b) a virtual and generic appliance acting as a gateway between user resources (named Cloud Networking Gateway, CNG). The CNG Manager deals with physical and virtual network resources to establish connectivity between virtual machines acquired from distributed cloud providers.

The CNG Manager establishes connectivity between user resources regardless of connectivity type (data centers connected via a dedicated link or using the Internet). The CNG Manager ensures the connectivity in a non-intrusive way that preserves the network configuration of cloud providers.

The CNG Manager and the gateway appliance, the CNG, are evaluated through an integration with a brokerage and provisioning system that decomposes user requests (expressed in the form of resource graphs) into sub-requests (or subgraphs) across selected candidate providers via a splitting algorithm.

The broker first splits the request and indicates how the subgraphs should be individually build and linked to the CNG Manager that takes care of the instantiation in the second step. The goal in this paper is to assess this second step, the instantiation phase of the networking of the selected distributed resources from the providers.

The providers process the portion of the request they receive by building the required VLANs corresponding to their respective subgraphs using existing cloud managers such as OpenStack [3], OpenNebula [4],... This is depicted in Figure 1 that also shows that the CNG Manager controls and configures CNGs to establish the inter data center connectivity to compose the overall graph corresponding to the original user request.

As previously mentioned, the focus of this paper is on the inter-domain, inter-provider or inter data center networking, especially on the required CNG appliances deployment and configuration and finally the instantiation needed to fulfil the inter-domain links establishments.

The organization of this paper follows the steps depicted in Figure 1, starting from the cloud broker framework description in section II. A short version of an exact request splitting algorithm originally presented in [5] is described in more details and evaluated in section III. This min-cut maximum flow algorithm splits the user requested graphs into smaller graphs to be hosted by the selected providers that use cloud managers to establish their corresponding VLANs (that achieve

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

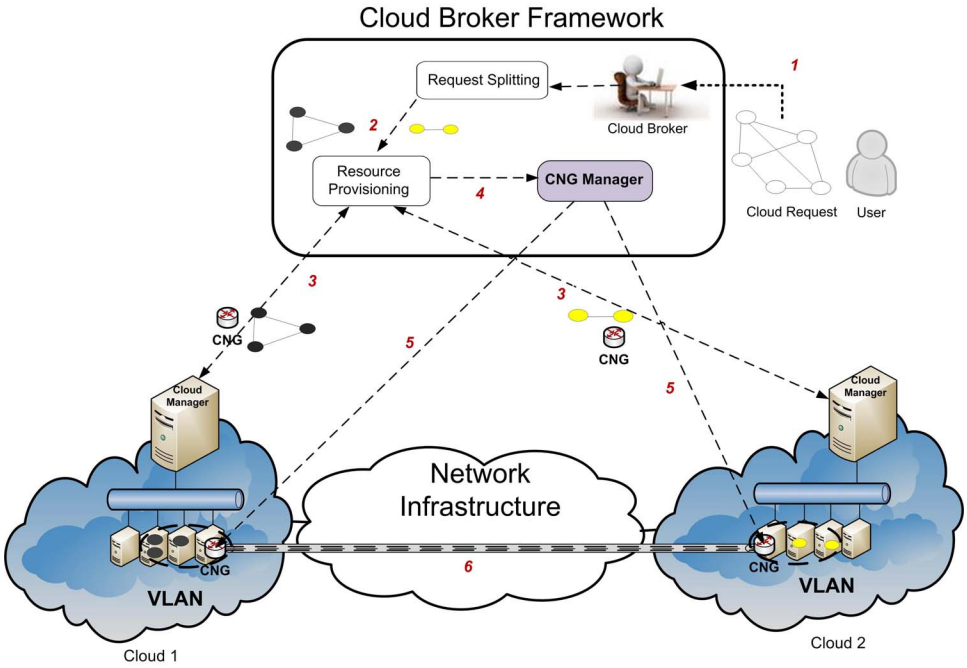


Fig. 1. Cloud Brokerage Architecture

interconnection of the subgraphs nodes or service components). This intra-domain networking is found in section IV while section V focuses on the cloud networking architecture proposed in this paper for deploying and interconnecting the user distributed service instances. Related work is discussed in section VI.

II. Cloud Broker Framework

Cloud service brokerage is a promising concept for enhancing service delivery over large scale heterogeneous cloud environments [6]. *Cloud Brokers* act as intermediaries between cloud providers and customers to negotiate and allocate resources among multiple data centers [7], [8], [9], [10], [11], [12]. This creates a marketplace where cloud providers can offer their services with different capabilities and pricing models. Customers (clients, enterprises, etc) allocate services from the marketplace using unified interfaces and APIs offered by the Broker. Since similar services may be offered by the same marketplace, the broker assists customers in selecting and choosing the cloud platform that best suits their requirements and needs. This is referred as the cloud service arbitrage [6] offered by the broker to provide flexibility and opportunistic choices to customers.

Figure 1 depicts the broker framework and describes a scenario where a Cloud Broker interacts with two Cloud Providers for cloud services provisioning. Each Cloud Provider uses a Cloud Manager (such as OpenStack [3], OpenNebula [4], etc) installed in its data center to allocate and manage their cloud resources. A Software Defined Networking approach using the CNG Manager is used to interconnect the cloud providers over network infrastructures.

The user, asking for infrastructure cloud services (or IaaS), formulates and sends requests to the Cloud Broker in step 1 in Figure 1. The Cloud Broker in co-operation with the Cloud providers responds to the cloud request. The request may have a complex topology composed of a set of *requested nodes* and their desired interconnections (i.e. *requested links*). The request is modeled in this work as a weighted undirected graph $G_r = (N_r, L_r)$, where N_r is the set of required nodes and L_r is the set of required links. Each node $n_r \in N_r$ is associated with a set of node requirements such as compute capacity, amount of memory, type of OS desired, imposed location, etc. Each requested link $l_r \in L_r$ is associated with a set of link requirements like minimum bandwidth, link type, QoS constraints, etc.

The Cloud Providers should first advertise, to the Cloud Broker, properties and characteristics of their offered resources including their prices, type, locations, availability, etc. The broker will match user queries with cloud resources published and offered by the cloud providers. The broker then splits the request across the cloud providers, acquires the required resources from each provider and sets up the inter-cloud networking via some networking system (the CNG Manager Framework in our case).

As shown in Figure 1, the brokerage framework is composed of three main components:

- Cloud Request Splitting: that splits the Cloud request graph into subgraphs to acquire from selected providers while satisfying performance, quality objectives and costs imposed by the initial user request. The optimization criteria may include prices, revenues, QoS and security aspects. Our contribution focuses on reducing cost for

customers, or equivalently minimizing prices proposed by the broker. The splitting is part of the Cloud Broker arbitrage function that decides where and how to distribute services related to the users' requests across providers.
- Resource Provisioning: that interacts with providers' Cloud Managers to compose their subgraphs.
- CNG Manager: that sets up and configures the inter-subgraph links over the network infrastructure. the CNG Manager establishes and controls the required inter-cloud networking to assist the Broker aggregation function .

The request splitting and the provisioning phases (steps 2 to 6 in Figure 1) are described in sections III and IV.

III. Cloud Request Splitting

We describe here one possible splitting algorithm, being aware that there is a plethora of methods of approaches to achieve splitting - a facet that is out of scope of this paper. We limit ourselves to our own algorithm to show how splitting can be performed. Recall, that the focus of this work is the configuration and instantiation of the network graphs.

Upon receiving the user's request, the Cloud broker finds for each requested node n_r several matching candidates in the multiple data centers. The same cloud service can be offered by several providers with competitive prices and the goal of the broker and the splitting is to select the most appropriate ones to compose the solution. The cloud broker thus decides to which cloud provider each requested node should be sent for matching and selection. The aim is to efficiently split the request graph into sub-graphs that will compose the request for each target cloud provider.

The Request Splitting module finds the best and optimal request splitting among the multiple cloud providers while minimizing the cost for the users. The price proposed for a given request is equal to the sum of the prices of the nodes and links provided by the selected cloud providers.

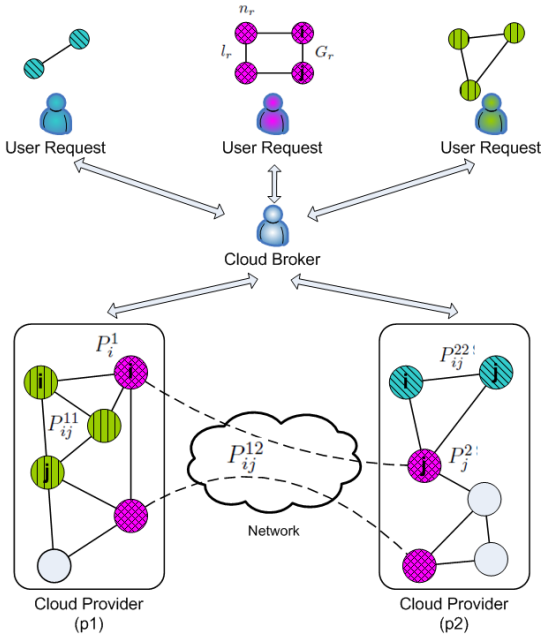
A. Mathematical Programming Formulation of the request splitting problem

Figure 2 provides an example of request splitting among two cloud providers ($p1$ and $p2$). The requested nodes *i* and *j* specified in the graph request G_r find dispersed candidate host nodes in two data centers. The goal of the splitting algorithm is to determine the subgraphs to be sent to the providers $p1$ and $p2$. Since the objective is to minimize the proposed prices to the user or to reduce their costs, the splitting problem considers only the attribute "price" defined and published by the cloud providers.

Let P_i^1 denote the price of allocating a requested node i from cloud provider $p1$. P_{ij}^{12} denotes the price of allocating a requested link between nodes $i, j \in N_r$ from an inter-provider path (i.e. peering path) where node *i* is in provider $p1$ and node *j* is selected from cloud provider $p2$. Likewise, P_{ij}^{11} denotes the price of allocating a requested link between nodes $i, j \in N_r$ from cloud provider $p1$.

The objective is to split the request among the two cloud providers (i.e. *p1* and *p2*). Finding the optimal request splitting

Fig. 2. Request Splitting



that satisfies the user requirements with a minimum price can be modeled using a 0-1 quadratic program where a variable x_i is considered for each requested node *i*. Variable x_i equals to 1 (or 0) if i is allocated by *p1* (or *p2*). The quadratic program for graph splitting is expressed as:

$$\begin{cases} \min \sum_{i \in N_r} (x_i P_i^1 + (1-x_i) P_i^2) + \sum_{\substack{(i,j) \in N_r \\ i<j}} (x_i x_j P_{ij}^{11} + (1-x_i) \\ \qquad (1-x_j) P_{ij}^{22} + x_i (1-x_j) P_{ij}^{12} + (1-x_i) x_j P_{ij}^{21}) \\ \\ x_i \in \{0,1\}; \quad \forall i \in N_r. \end{cases}$$

Even when just two cloud providers are involved, the request splitting problem is already NP-hard. The proof can be deduced from the well known NP-hard problem (see MAX-2-SAT [13], [5]).

When multiple cloud providers are involved (more than two), the splitting problem can be similarly modeled using the same price annotation. P_i^j then denotes the price resulting from allocating a requested node *i* from cloud provider j. $P_{ii'}^{jj'}$ denotes the price of allocating a requested link between nodes $i, i' \in N_r$ from an inter-provider path where i is allocated from cloud provider j and i' is allocated from cloud provider j'. Likewise, $P_{ii'}^{jj}$ denotes the price of allocating a requested link between nodes $i, i' \in N_r$ from cloud provider j.

Finding the optimal request splitting among multiple cloud providers can be formulated as a quadratic program. The following expression represents a generalization of the quadratic program formulation in the case of multiple cloud providers:

$$(A)\begin{cases} min(\sum_{i\in N_r}\sum_{j\in Pr} P_i^j x_i^j + \sum_{\substack{(i,i')\in N_r\\ i<i'}}\sum_{(j,j')\in P_r} P_{ii'}^{jj'} x_i^j x_{i'}^{j'}) \\ \sum_{j\in P_r} x_i^j = 1; \quad \forall i \in N_r; \quad x_i^j \in \{0,1\} \end{cases}$$

where x_i^j is equal to 1 (or 0) if i is allocated from cloud provider j (or not allocated from cloud provider j). Finding an optimal request splitting problem is also NP-hard (the proof can be deduced from the 3-multiway cut problem [13], [5]).

Despite NP-hardness, the problem can be solved exactly using a simple linearization. Indeed, by introducing $y_{ii'}^{jj'} = x_i^j x_{i'}^{j'}$, the quadratic optimization problem can be formulated as a simple linear integer program:

$$(A) \Leftrightarrow \begin{cases} min(\sum_{i\in N_r}\sum_{j\in Pr} P_i^j x_i^j + \sum_{\substack{(i,i')\in N_r\\ i<i'}}\sum_{(j,j')\in P_r} P_{ii'}^{jj'} y_{ii'}^{jj'}) \\ \sum_{j'\in P_r} y_{ii'}^{jj'} = x_i^j; \quad \forall i,i' \in N_r;\ \forall j \in Pr \\ x_i^j + x_{i'}^{j'} - y_{ii'}^{jj'} \leq 1 \\ \sum_{j\in P_r} x_i^j = 1; \quad \forall i \in N_r; \quad x_i^j \in \{0,1\}; \quad y_{ii'}^{jj'} \in \{0,1\} \end{cases}$$

This linear integer program can be solved using the branch and bound algorithm [14] to provide exact solutions for the request splitting problem. The cloud request splitting module in Figure 1 will use this program for each incoming request.

IV. Resource Provisioning

Once the Cloud request is split into subgraphs in step 2 of Figure 1, the Resource Provisioning component receives the splitting results including:

- the (subgraph, cloud provider) pairs
- and the virtual links for interconnecting the providers' subgraphs (i.e. a specification or description of the inter-subgraph links)

Resource Provisioning handles the composition of the subgraphs in the data centers by interacting with the cloud managers of each data center (data centers can belong to the same or different providers, in both cases the provisioning process is the same). This corresponds to step 3 in Figure 1. The Resource Provisioning relies on standard or proprietary APIs and interfaces (like OCCI, EC2 for computing and CDMI or Amazon S3 interfaces for storage) to invoke the Cloud Managers that use their schedulers and management framework to achieve optimal VM placement and allocation within the data center.

When the original request is split into 2 or more subgraphs, the Resource Provisioning deploys in each cloud provider the required Cloud Networking Gateway (CNG) appliance (running a VM) in the provider infrastructure (in a container or a compartment of physical resource) as depicted in step 3 of Figure 1. The deployed virtual nodes composing the subgraph including the CNG appliance belong to the same cloud provider VLAN.

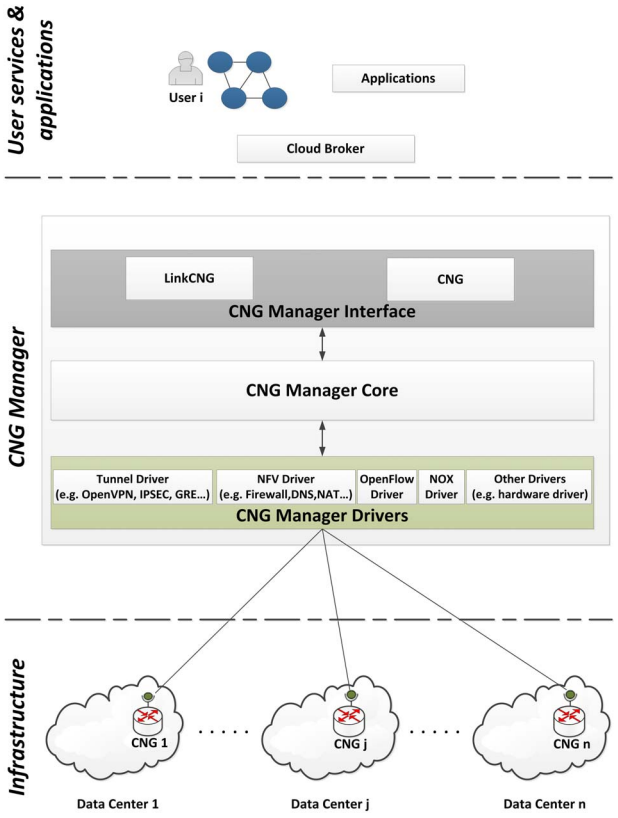


Fig. 3. Cloud Networking Gateway Manager

Once the subgraphs are created, the Cloud Managers return to Resource Provisioning the list of identifiers (names and/or addresses) of the instantiated virtual machines (matching the requested nodes). Using this list, the Resource Provisioning determines the endpoint identifiers of the inter-subgraph links to build the network topology. This topology is finally sent (step 4 in Figure 1) to the CNG Manager that sets up (or establishes) the inter-subgraph (or inter-data center) links.

V. Cloud Networking Gateway Manager

Once the Resource Provisioning component has instantiated the CNGs in the appropriate cloud providers, the CNG Manager configures the CNGs by activating the desired network function and by configuring the tunnels between the CNGs (as depicted in step 5 of Figure 1). Finally, in step 6 of Figure 1, the CNG Manager establishes the inter-subgraph links by injecting appropriate routing rules in the involved CNGs. The cloud broker can hopefully deliver the required resources and topology to the users so they deploy their services and applications.

The proposed cloud networking framework of Figure 1, presented in greater details in Figure 3: a) ensures connectivity between resources acquired from distributed and independent cloud providers irrespective of the used networking technologies and b) gives partial or complete control of connectivity to the users so they can handle the networking of their applications.

As depicted in Figure 3, the proposed solution is directly inspired by the emerging SDN concept. The CNG Manager is in essence an SDN controller for cloud infrastructures networking. The proposed architecture is composed of three (3) levels: a central component, the CNG Manager core, a northbound interface towards applications and a southbound interface interacting with transport technologies through drivers. These three levels are used to control, configure and program the cloud networking gateways deployed in the infrastructure.

The CNG Manager northbound interface presents to applications or users (application developers, customers or consumers) an extended Open Cloud Computing Interface (OCCI,[15]) RESTful interface. This interface is designed as an OCCI server that contains 2 OCCI categories responsible for the configuration of gateways ("CNG" category) and the establishment of links ("LinkCNG" category) between the CNGs.

For the CNG Manager core, we have developed the RESTful CRUD functions and the necessary actions of the exposed categories to handle networking demands from users (received via the northbound interface) and to select the appropriate drivers, in line with user expressed networking requirements, to interact with the desired networking technology.

The CNG Manager drivers, representing the southbound interfaces, hide the heterogeneity of the networking technologies. A specific driver is used per technology by this component to enable handling of multiple routing and switching protocols and systems. Each Driver is responsible for communicating with the underlying technology used in the gateway (OpenFlow [16], BGP, OSPF, MPLS, etc...). The CNG Manager interacts with the drivers to configure the connectivity between sites. The CNG Manager hides the heterogeneity of the underlying technologies used to establish the connectivity between the sites. The underlying networking framework can be software or hardware based (software switching and routing technologies or programmable hardware). As depicted in Figure 3, there are two main family of drivers. The first one provides the dynamic establishment of tunnels between CNGs (e.g. IPsec, GRE, OpenVPN, Capsulator...) and the second enables configuration of the network functions virtualization (NFV [17] like Firewall, NAT, DNS...).

Our proposed model is flexible and can be extended to support other network technologies by developing appropriate drivers like OpenFlow [16] and NOX [18] actually integrated in our solution. With the emergence of commercial programmable network equipment, we can even go further by supporting the configuration of the hardware by designing suitable drivers. The CNG Manager relies on the designed drivers to remotely configure the gateways.

In the infrastructure layer of the architecture, we propose an appliance compatible with multiple cloud computing platforms acting as a gateway (the CNG). This appliance exposes an OCCI interface so it can be configured remotely by the CNG Manager.

The CNG Manager, in Figure 3, exposes two (2) OCCI interfaces (at the dashed separation lines): one in the CNG Manager (for user cloud networking requests) and the second in the CNG appliance (for network configurations within the cloud and network infrastructures).

In our case both the CNG Manager and the CNG appliance expose a RESTful interface based on OCCI categories and hence can be invoked easily by a Cloud Broker and the CNG Manager respectively. This ensures that our proposed cloud networking operates in an OCCI compliant manner and is in line with current practices in the OpenStack cloud community for compute [3] and networking [1] services. This also considerably eases integration into cloud and network software architectures using the OCCI RESTful paradigms.

VI. Performance Evaluation

This section evaluates first of all the delay introduced by the CNG Manager architecture when it is used to establish connectivity between distributed VMs. This will provide us with the cost of instantiation when using the proposed architecture measured in terms of connectivity establishment delay penalty. Since our main objective is to assess only the additional delay penalty, we do not seek or conduct comparisons with similar approaches that may be available in the literature (in fact there are no vendor independent frameworks we can compare meaningfully our proposal to). Secondly, we evaluate our splitting algorithm in terms of delay required to split requests between multiple cloud providers as well as their average achievable prices to propose to end users.

TABLE I
Table of network configuration delay

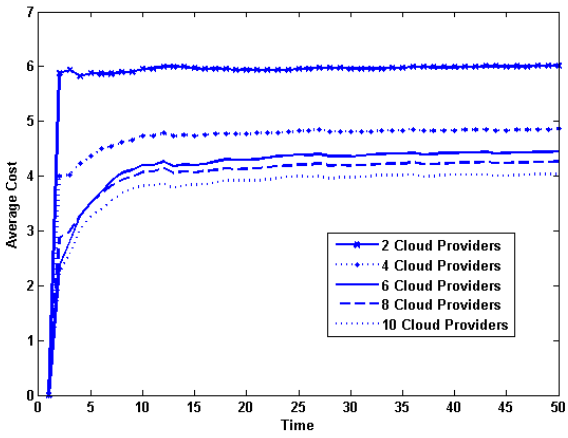
Cloud Providers	2	4	6	8	10
Delay (s)	6.77	18.86	27.29	44.77	59.135

In the first experimentation (table I), we evaluate the configuration delays of the network graph composed by the CNG nodes. Each node of the network graph represents a cloud provider. The focus is on the configuration delay induced by the CNG Manager. The evaluated network graph topologies are randomly generated using the GT-ITM tool [19]. The average network graph connectivity is fixed at 0.5. In this experiment the number of cloud providers varies between 2 and 10 providers. The configuration of the CNGs and their networking is achieved using the CNG interface.

Table I shows the time required to configure **sequentially** all the CNGs composing the network interconnecting user nodes. The configuration delay varies between 6 seconds when user nodes are deployed in 2 cloud providers and 1 minute when user nodes are deployed in 10 cloud providers. When the CNGs are configured in **parallel** (or quasi-simultaneously in all providers), the configuration delay will drop to 6 seconds irrespective of the number of the involved cloud providers.

The second evaluation focuses primarily on quantifying the advantage of involving multiple cloud providers to reduce costs to end users. For the purpose, we have implemented a discrete event simulator to evaluate our algorithm performance. The request graph topologies are randomly generated

Fig. 4. Average Price



using the GT-ITM tool [19]. The average request graph connectivity is fixed at 0.5. The number of nodes in the requests are randomly generated by a uniform distribution between 5 and 20 nodes. The resource prices proposed by the providers are uniformly distributed between 1 and 15 cost units. The mixed integer programming library CPLEX [20] was used to solve the associated linear program. To assess the performance of the request splitting algorithm, a 2.5 GHz duo-processor PC with 4 GBytes of available RAM has been used.

The first simulation evaluates the prices resulting from the application of our graph splitting algorithm when multiple cloud providers are involved. We assume that the requests arrive following a uniform distribution with an average rate of 5 requests per unit time. The same experiment is repeated with several providers (2, 4, 6, 8 and 10). As seen in Figure 4, the average offered price to the users decreases when the number of cloud providers increases. The upper curve represents the average price related to 2 cloud providers. The lower curve represents the price for 10 involved cloud providers. This shows that cloud brokers have increasing flexibility and opportunities to reduce end user costs (or provide more competitive prices) when more cloud providers are involved. However, this must be traded off with the increasing time splitting delays with increasing number of providers as shown in the next evaluation.

A second simulation evaluates the graph splitting delays for increasing graph sizes and number of providers (graphs of 5 to 20 nodes with 2 to 6 cloud providers). Table II shows a splitting delay in the order of a few seconds (4 seconds) when the number of nodes in the original request does not exceed 20 nodes and the number of involved cloud providers does not exceed 6 providers.

VII. Related work

Efficient solutions for inter-cloud networking are still lacking. A number of proposals have been put forward to address intra and inter networking problems for data centers [21], [22], [23], [24] but have not reported any explicit solutions for cloud networking at this stage.

TABLE II
Table of splitting delay of the Exact algorithm

Cloud Providers \ Request Size	5	10	15	17	20
2	0	0.01	0.02	0.02	0.09
3	0	0.03	0.08	0.05	0.36
4	0	0.05	0.33	0.47	0.67
5	0.01	0,08	0.55	0.81	1.61
6	0.01	0.23	0.94	2.38	4.05

Recently, OpenNebula [4] and OpenStack Neutron [1] proposed an appliance with networking services like DHCP, DNS, etc to handle networking in their cloud managers. OpenNebula and OpenStack Neutron refer to these appliances as Virtual Router appliance [25] and Provider Router [26] respectively. These solutions handle connectivity between virtual machines deployed in the same data center. CNG targets both intra and inter-cloud networking.

Meridian [27] proposes an SDN-based controller framework for cloud networking. Raghavendra et al. [28] propose the same for Cloud network management. These two approaches focus only on providing cloud networking between resources managed by a single cloud provider. Our solution provides cloud networking for distributed clouds along with an algorithm that solves the splitting of an initial cloud graph request using an exact mathematical formulation when a broker and multiple cloud providers are involved.

VIII. Conclusion

We have proposed an SDN controller (the CNG Manager) to achieve dynamic and on demand inter cloud networking that can handle any kind of underlying networking technology when multiple cloud providers are involved. The novelty of our solution consists in enabling the control of connectivity between distributed resources acquired from multiple cloud providers. The CNG Manager is conceived to be flexible enough (thanks to drivers) to easily support different network technologies (even programmable networking hardware). The CNG Manager was evaluated in a Cloud Broker Framework acquiring cloud resources from multiple data centers thus requiring inter-cloud or data center connectivity after partitioning the original requests across the providers. The partitioning is achieved using an exact splitting algorithm. Future work will focus on multi-objectives optimization as needed in the cloud context as well as on generalizing SDN principles to support distributed and connected clouds services.

References

[1] OpenStack Neutron, available on line: https://wiki.openstack.org/wiki/Neutron
[2] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", white paper, Apr. 2012.
[3] OpenStack. available on line: http://www.openstack.org/
[4] R. M.-Vozmediano, R. S. Montero and I. M. Llorente. "Elastic Management of Cluster-based Services in the Cloud". Proc. of 1st Workshop on Automated Control for Datacenters and Clouds, Barcelona, Spain, pp. 19-24, 2009.

[5] I. Houidi, et al., ”Virtual Network Provisioning Across Multiple Substrate Networks”, Computer Networks, Vol. 55, N. 4, Special Issue on Architectures and Protocols for the Future Internet, March 2011, pp. 1011-1023.
[6] Gartner report, July 2009, available on line: http://www.gartner.com/it/page.jsp?id=1064712
[7] S.G. Grivas, T. Uttam Kumar, H. Wache, ”Cloud Broker: Bringing Intelligence into the Cloud an Event-Based Approach”. IEEE 3rd International Conference on Cloud Computing (CLOUD2010), Miami, pp. 544, July 2010.
[8] H. Y. Huang, et al, ”Identity Federation Broker for Service Cloud,” icss, pp. 115-120, 2010 International Conference on Service Sciences, 2010
[9] S. K. Nair, et al. ”Towards Secure Cloud Bursting, Brokerage and Aggregation”, 8th European Conference on Web Services, Industry Track, Dec. 1-3, 2010, Ayia Napa, Cyprus.
[10] R. Buyya, R. Ranjan and R. N. Calheiros. ”InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services”. Lecture Notes in Computer Science, vol. 6081, pp. 13-31, 2010.
[11] R. Buyya, R. Ranjan, R. Calheiros, ”InterCloud: Scaling of Applications across multiple Cloud Computing Environments”, Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, LNCS, Springer, Germany, 2010.
[12] A. Celesti, F. Tusa, M. Villari, A. Puliafito, ”Security And Cloud Computing: Intercloud Identity Management Infrastructure”, Proceedings of The 19th IEEE International Workshops on Enabling Technologies, Tei of Larissa, Greece June 2010.
[13] M. R. Garey, D. S. Johnson, ”Computers and intractability. A guide to the theory of NP-completness”, San Francisco,W. H. Freeman, 1979
[14] L.A. Wolsey and D.L. Nemhauser, ”Integer and Combinatorial Optimization”. July 1999, John Wiley, ISBN-10: 0-471-35943-2
[15] ”Open Cloud Computing Interface OCCI”, available on line: http://occi-wg.org/about/specification/.
[16] N. McKeown, et al. ”Openflow: enabling innovation in campus networks”. SIGCOMM Comput. Commun. Rev., 38(2):69-74, 2008.
[17] ”NFV Network Functions Virtualisation.”, available on line: http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf
[18] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, and N. McKeown. ”NOX: Towards an Operating System for Networks”. In ACM SIGCOMM CCR, July 2008.
[19] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, ”How to model an internetwork”. In Proc. IEEE INFOCOM, 1996.
[20] ILOG CPLEX, ILOG CPLEX 11.1 user’s guide.
[21] A. Greenberg et al. ”The Cost of a Cloud: Research Problems in Data Center Networks”, ACM SIGCOMM Computer Communication Rev., Jan. 2009, pp. 68-73.
[22] Avetisyan et al., ”Open Cirrus: A Global Cloud Computing Testbed,” IEEE Computer, vol 43, no 4, April 2010, pp 42-50.
[23] SAIL project, www.sail-project.eu/
[24] Bernstein, el al. ”Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability,” In ICIW 2009.
[25] ”Virtual Router”. available on line: http://www.opennebula.org/documentation:rel3.8:router
[26] ”Provider Router”. available on line: http://docs.openstack.org/folsom/openstack-network/admin/content/use_cases_single_router.html
[27] M. Banikazemi et al., ”Meridian: an sdn platform for cloud network services.” IEEE Communications Magazine, vol. 51, no. 2, pp. 120127, 2013.
[28] R. Raghavendra et al., ”Dynamic graph query primitives for sdn-based cloudnetwork management,” ser. HotSDN 12, 2012, pp. 97102.

Virtual Links Mapping in Future SDN-enabled Networks

R. Trivisonno, I. Vaishnavi, R. Guerzoni, Z. Despotovic, A. Hecker, S. Beker, D. Soldani
Huawei European Research Center
Riesstrasse 25, Munich, Germany

***Abstract*—Software defined networking (SDN) has emerged as an efficient network technology for lowering operating costs through simplified hardware, software and management. Specific research focus has been placed to achieve a successful carrier grade network with SDN, in terms of scalability, reliability, QoS and service management. In the literature, very little material is currently available on traffic engineering (TE) using this technology. This paper presents a novel mixed integer linear programming (MILP) formulation for a centralised controller to calculate optimal end-to-end virtual paths over the underlying network infrastructure, considering multiple requests simultaneously. Extensive simulation results, over a wide range of underlying network topologies and input parameters, demonstrate that the proposed algorithm outperforms traditional shortest path first (SPF) approaches. In some cases, up to 30% more virtual connections were satisfactorily mapped onto the same substrate, independent of the number of physical nodes.**

I. Introduction

In this article, software defined networking (SDN) is based on the Open Networking Foundation (ONF) definition. The key features of SDN are: separation of control plane from data plane; a centralised controller and complete view of the underlying network; open interfaces between entities in the control plane (controllers) and data plane; and programmability of the network by external applications [1]. The controller can therefore optimise end-to-end flow management and, for example, allocate bandwidth dynamically into the data plane for traffic engineering (TE). In this work, the TE problem relates to the need for a network administrator to *efficiently* create a number of virtual connections from source to destination over a given physical infrastructure. In this regard, efficiency can have different meanings, depending on operators business processes: some carriers may want to use energy more efficiently; others may need to reduce the number of hops. In this paper, our TE algorithm aims at maximizing the number of source-destination *virtual link requests* that can be satisfactorily accommodated over a given physical infrastructure.

In the last couple of years, research focus has been placed on finding the optimal way for organizing controllers in a OpenFlow network (see [2] and [3]). Algorithms used within the OpenFlow controllers, which are the main intelligence of the data path formation mechanisms, have been less studied and most of the published works are based on heuristics. For instance, authors in [4] presented some algorithms that can be used for load balancing in HTTP servers. In [5], in order to address the path computation problem in wavelength switched optical networks, OpenFlow based solutions were considered as an alternative to GMPLS, exploiting the benefit of a centralised control element. In [6], authors introduced a new centrally managed algorithm to OpenFlow controller to achieve a differentiated QoS management for data and multimedia flows. In early 2000s, the benefits of a central computational element for TE were also investigated in MPLS and GMPLS network domains, see [7] and [8]. In those works, authors proposed source routing protocols where the route was recommended by a central path computation element having partial to almost complete network view. In [8], authors compared previous existing efforts in centralised path computation for MPLS network. Some other formulations tried to model the problem as a multi-commodity-flow problem [9], similarly to the approach proposed in this paper. However, our technique is not limited to a simple source destination pair connection, but it can be applied to groups of connections, representing more complex graph infrastructures.

In this paper, we formulate the path-computation problem as a mixed integer linear programming (MILP) problem, under the assumption that the controller can exploit complete knowledge of the network to optimise flow management and support service-user requirements of scalability and flexibility [1]. The proposed MILP formulation, denoted as Virtual Link Mapping (VLM), allows controllers to atomically map the entire virtual link graph structures onto the underlying network. Beyond this, the VLM makes it possible to deal with multiple virtual link requests simultaneously while keeping up to date knowledge of network status and to find solutions for subsets of the requests simultaneously handled.

It is worth mentioning the implementation of the proposed approach does not strictly require an SDN-based scenario; nevertheless, considering the widespread adoption this technology is expected to have in the next future, it is relevant to explore all features it might support.

The technical feasibility of the proposed MILP formulation is investigated by means of simulations and performance results are compared to the traditional shortest path first (SPF) based approaches. The comparison of performance is made in terms of number of successful requests and effectiveness and efficiency of network resources utilisation.

This paper is organised as follows. Section II introduces the TE problem formulation and the VLM algorithm. Section III presents our simulations assumptions and VLM performance quantitative evaluation. Conclusions are drawn in Section IV.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

II. Traffic Engineering Problem Formulation

This section presents the novel MILP formulation for virtual links mapping, as previously introduced. The formulation of the problem consists of five steps: *Physical Infrastructure model*, *Virtual Link Request model*, *Variables definition*, *Constraints definition* and, finally, *Objective function definition.*

A. Physical Infrastructure model

The physical infrastructure (P) is modeled as a non-reflexive physical graph $\mathfrak{G}^P = (V^P, E^P)$, where $V^P = \{n_1, n_2, \ldots, n_N\}$ is a set of N physical nodes and $E^P = \{e_{ij}\}$, with i and $j = 1, 2, \ldots, N$, is a set of $L \leq N^2$ physical links, with $\{e_{ij}\}$ connecting n_i to n_j. The physical infrastructure is described by a Connectivity Matrix and a Link Capacity Matrix. The Connectivity Matrix is defined by $B = [B_{ij}]$, with i and $j = 1, 2, \ldots, N$, where:

$$B_{ij} = \begin{cases} 1 & \text{if } n_i \text{ is connected to } n_j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and by definition $B_{ii} = 0, \forall i$.

The Link Capacity Matrix is defined by $B^c = [B^c_{ij}]$, i and $j = 1, 2, \ldots, N$, where B^c_{ij} defines the available capacity of the link connecting n_i to n_j.

B. Virtual Link Request model

Let G be the number of virtual link mapping requests that are handled simultaneously[1]. The g-th request, with $g = 1, 2, \ldots, G$, is defined by a set of K_g 2-node directed virtual graphs $\mathfrak{G}^{V_{kg}} = (V^{V_{kg}}, E^{V_{kg}})$, with $k = 1, 2, \ldots, K_g$. For each k, $V^{V_{kg}} = \{n^{kg}_s, n^{kg}_d\}$ is a set of 2 nodes identifying the source and destination node, and $E^{V_{kg}} = \{e^{kg}_{sd}\}$ is the virtual link connecting source to destination node with bandwidth equal to b_{kg}. ($K_g > 1$ corresponds to a complex traffic engineering case, where multiple source and destination nodes require to be connected.) The indexes of virtual nodes n^{kg}_s, n^{kg}_d identify the source and destination physical nodes to be connected, respectively.

C. Problem Variables definition

For each virtual link mapping request, g, associated to K_g 2-node directed virtual graphs, we define y_g and X^{kg} boolean variables as:

$$y_g = \begin{cases} 1 & \text{if the } g\text{-th request is accepted} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

and $X^{kg} = [x^{kg}_{ij}]$, where:

$$x^{kg}_{ij} = \begin{cases} 1 & \text{if } g\text{-th request uses } \{e_{ij}\} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

and by definition $x^{kg}_{ii} = 0$, $\forall i$.

In total, the problem requires the definition of $G + N^2 \cdot \sum_{g=1}^{G} K_g$ boolean variables (which is equal to $G \cdot (1 + N^2)$ in case $K_g = 1\ \forall g$).

[1]The total number of virtual link requests is typically much higher than G.

D. Problem Constraints definition

The virtual-to-physical link mapping problem is subject to the following constraints. The first topology constraint is given by:

$$x^{kg}_{ij} \leq B_{ij}, \forall (i,j). \quad (4)$$

The second topology constraint depends on the type of physical graphs considered. In the case of directed physical graphs it is defined by:

$$\sum_{g=1}^{G} \sum_{k=1}^{K_g} x^{kg}_{ij} \cdot b_{kg} \leq B^c_{ij} \forall (i,j). \quad (5)$$

For undirected graphs, where the bandwidth in the forwarding path consumes also bandwidth in the reverse direction, we have:

$$\sum_{g=1}^{G} \sum_{k=1}^{K_g} \left(x^{kg}_{ij} + x^{kg}_{ji} \right) \cdot b_{kg} \leq B^c_{ij} = B^c_{ji}, \forall (i,j). \quad (6)$$

Finally, the standard multi commodity flow problem imposes that:

$$\sum_{j=1}^{N} x^{kg}_{nj} - \sum_{i=1}^{N} x^{kg}_{in} = \rho(n) \cdot y_g \quad (7)$$

where:

$$\rho(n) = \begin{cases} 1 & n = n^{kg}_s \\ -1 & n = n^{kg}_d \\ 0 & otherwise. \end{cases} \quad (8)$$

Equation 4, which represents the fact that a physical link can be used only if it exists, results in $N^2 \cdot \sum_{g=1}^{G} K_g$ equations. Equations 5 (6), meaning that the bandwidth allocated on a physical link cannot exceed its capacity, result in N^2 ($N^2/2$) equations. Equation 7 imposes the outgoing flow from the source node to be equal to the incoming flow to the destination node, which, in turn, has to be equal to the virtual link bandwidth requirement, b_{kg}. For all other physical nodes the net flow is zero. This results in $G \cdot N$ equations. Considering all constraints, the solution to the mapping problem requires to solve a system of $N \cdot \left(\sum_{g=1}^{G} K_g + N \cdot \left[1 + \sum_{g=1}^{G} K_g \right] \right)$ linear equations (which is equal to $N \cdot (G + N \cdot [1 + G])$ in case $K_g = 1\ \forall g$).

E. Objective Function definition

The objective function aimed at maximizing the number of virtual links, while minimizing the utilisation of the physical infrastructure, is defined as:

$$\max \left\{ \sum_{g=1}^{G} \left[y_g - \sum_{k=1}^{K_g} \epsilon \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} x^{kg}_{ij} \right] \right\} \quad (9)$$

where:

$$\epsilon = \frac{1}{N \cdot (N-1)}. \quad (10)$$

The goal of (9) is to have as many virtual requests accepted as possible, i.e. as many y_g as possible set to 1, and, at the same time, minimise the usage of physical resources to accomodate the G virtual requests, i.e. minimising the number of variables x_{ij}^{kg} set to 1, as $x_{ij}^{kg} = 1$ implies that the g-th request is mapped onto the physical link $\{e_{ij}\}$.

F. VLM Algorithm Input

The VLM algorithm requires the following input data:

- G, defining the number of virtual link mapping requests to be simultaneously handled;
- A set of K_g 2-node directed virtual graphs $\mathfrak{G}^{V_{kg}} = (V^{V_{kg}}, E^{V_{kg}})$ for each virtual link request, defining topology and bandwidth requirements b_{kg} of the virtual links;
- A physical graph $\mathfrak{G}^P = (V^P, E^P)$, a connectivity matrix B and a link capacity matrix B^c, describing the physical infrastructure topology, characteristics and utilisation.

G. VLM Algorithm Output

Given the VLM formulation, the solver returns the set of variables (y_g, X^{kg}), indicating which requests were successful and physical links needed by the successful requests, where $X^{kg} = [x_{ij}^{kg}]$ with $g = 1, 2, \ldots, G$ and i and $j = 1, 2, \ldots, N$.

H. VLM Algorithm Execution Flow

The execution of VLM follows four main steps:

1) The controller buffers virtual link requests until G requests are collected.
2) The MILP formulation is applied to find solutions for the G requests. The higher G, the more information on the incoming requests the solver has, and the better the way to implement them. However, a larger G also implies more equations and, possibly, a longer time to solve them.
3) The MILP solver returns the solution in the form of a problem variable set (y_g, X^{kg}), with $X^{kg} = [x_{ij}^{kg}]$, for $g = 1, 2, \ldots, G$, i and $j = 1, 2, \ldots, N$, where y_g is the boolean variable indicating whether the g-th virtual link request is successfully mapped onto the physical substrate and x_{ij}^{kg} is a boolean variable indicating whether the k-th subgraph of the g-th request is mapped onto physical link $\{e_{ij}\}$ or not, as defined in Section II.C.
4) The Capacity Matrix is updated according to the resources utilised by the accepted requests.

III. VLM Performance Evaluation

A. Simulation Assumptions

VLM performance is evaluated by means of simulations with physical graph size ranging from 10 to 20 physical nodes (N). The physical graph topology is generated by assigning random node out-degrees, with an overall average of three out degrees per node. Then each node based on the number of node degrees selects another random node to connect to. If this link already exists, no attempts are made to find a new one. In addition, we ensure physical graphs are connected graphs and no forests are generated. The physical link bandwidth is uniformly randomly generated from $Bandwidth_{min}$ to $Bandwidth_{MAX}$. Random source destination pairs from the physical graph to emulate the *virtual link* requests are then generated. For the sake of simplicity, we assume $K_g = 1$ for all requests.

Bandwidth requirements of virtual links are also uniformly randomly generated over $Bandwidth_{min} \cdot LinkFactor$ and $Bandwidth_{MAX} \cdot LinkFactor$, where $LinkFactor \in (0, 1]$ is a simulation parameter that defines the ease in terms of bandwidth for incorporating a virtual link in a physical edge. In a single simulation run, the total number of virtual link requests is equal to $N \cdot NodeStress$.
Finally, for each generated physical graph the results are averaged over 10 simulation runs.
The algorithm performance is investigated for $G = 1, 5, 10$ and 20, and compared to the corresponding numbers attained using the shortest path first algorithm as a baseline. The analysed performance indicators are the percentage of successful virtual link mapping requests and physical link utilisation, which show the effectiveness and efficiency of the algorithms, respectively. Parameter settings are reported in Table I.

TABLE I
PARAMETER SETTINGS

Paramter	Values
N	10-20
$Bandiwidth_{min}$	1
$Bandwidth_{MAX}$	10^5
$NodeStress$	10, 20, 30, 40
$LinkFactor$	0.1, 0.2, 0.4
G	1, 5, 10, 20

B. Simulation Results

Simulations results are shown through Figure $1 - 7$. In particular, from Figure $1 - 3$ two important trends can be observed. First, the VLM effectiveness, in terms of percentage of successful mapping, is strongly dependent on the topology of the physical graph. Second, the characteristics of the attained results, using the parameter settings of Table I, are independent of the number of physical nodes (N) and simultaneous mapping requests (G) the MILP formulation handles.

Figure 1 depicts the percentage of accepted mapping requests for $G = 1$ and $G = 20$, with the number of physical nodes varying from 10 to 20. The test was performed with $LinkFactor = 0.1$ and $NodeStress = 20$. As illustrated in the Figure, the percentage of virtual link requests successfully mapped onto the substrate ranges from 20% to 80%. This remarkable variation, which is a common characteristic of all simulated physical network topologies, with N varying from 10 to 20, regardless the value of G, leads to two important conclusions. First, as the acceptance rate is very sensitive to the physical topology randomly generated, the comparison of

performance results is only significant on average, at a given number of physical nodes (N). Second, the more virtual link requests are simultaneously handled by the solver, i.e. the larger the value of G, the more requests can be mapped on average onto the underlying network, i.e. the better performance in terms of *Acceptance Rate*. It is of course intended that the most suitable value of G depends on the physical graph size: when G increases, the number of equations to be solved increases, leading to longer convergence time for the solver to find solutions. Beyond this, we may anticipate that, in the case of saturation of critical resources in the physical graph, G has little influence on the mean of the resulting acceptance rates. This effect is evident in the following figures, where a different values of the $NodeStress$ is used.

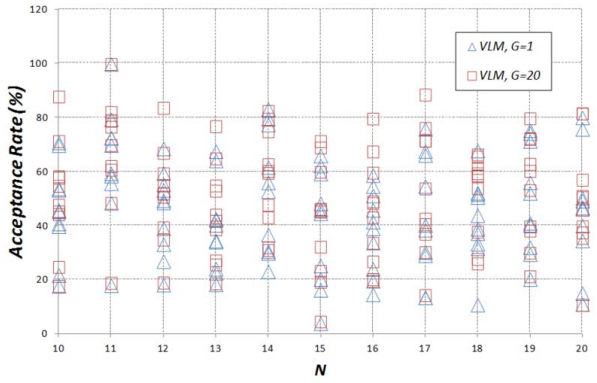


Fig. 1. Percentage of Successful Virtual Link mapping Requests vs. Number of Physical Nodes (N) with $LinkFactor = 0.1$ and $NodeStress = 20$.

VLM performance is evaluated and compared to current MPLS practice for traffic engineering: Shortest Path First algorithm (SPF). Figure 2 and Figure 3 show the average percentage of successful requests against the number of physical nodes, for both VLM and SPF, with $NodeStress = 20$ and $NodeStress = 40$, respectively. In both cases, $LinkFactor$ is set to 0.2.

Figure 2 shows that VLM outperforms SPF already for $G = 1$, being in that case the *Acceptance Rate* twice as much for any value of N. For $G = 20$, VLM percentage of success is five to seven times higher than the corresponding value attained using SPF.

The comparison between VLM and SPF leads to the same qualitative conclusions even for $NodeStress = 40$, as shown in Figure 3. In this case, as expected, the benefits given by a higher G are moderate.

An aggregate evaluation of VLM gain over SPF for different values of G is shown in Figure 4 . The gain-clipping effect for $NodeStress = 40$ is due to the system reaching its saturation. This is confirmed by the trend of the curves depicted in Figure 5, where the percentage of virtual link requests successfully mapped on the substrate and the percentage of resource utilisation, versus the number of total requests for a single simulation run, are shown. The curves show that beyond approximately 150 requests only a small number of virtual links can be further

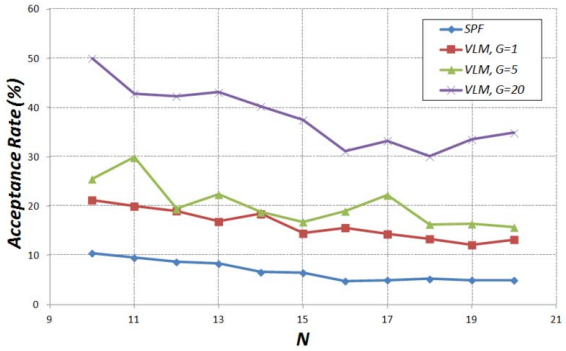


Fig. 2. SPF - VLM Comparison: Average Acceptance Rate with $LinkFactor = 0.2$ and $NodeStress = 20$.

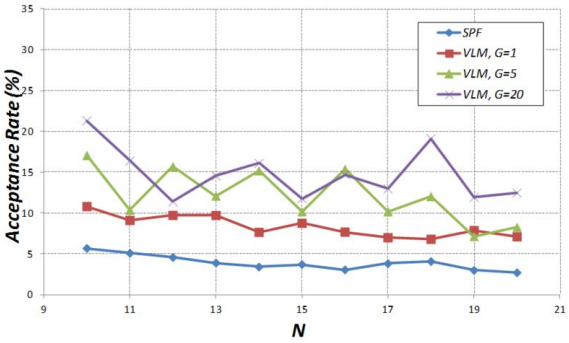


Fig. 3. SPF - VLM Comparison: Average Acceptance Rate with $LinkFactor = 0.2$ and $NodeStress = 40$.

accepted, due to the limit imposed by the resources utilisation. This explains why for high $NodeStress$ the gain provided by a higher value of G becomes negligible.

Figure 6 shows the cumulative distribution function (CDF) of link utilisation for both VLM and SPF. The graph refers to a single simulation run performed with $NodeStress$ set to 40. The figure explains why our algorithm outperforms traditional SPF approaches. This is because VLM better utilises physical resources compared to SPF. Furthermore, the higher the value of G, the higher the /textitAcceptance Rate, and the higher the average resource (link) utilisation. This is illustrated in Figure 7, where the average links utilisation is plotted against $NodeStress$, for G set to 5, 10 and 20. As expected, the maximal resource utilisation is for $G = 20$, as the controller has more degree of freedom in performing the link mappings.

IV. Conclusions

A novel MILP formulation for traffic engineering (TE) using software defined networking (SDN) technology was presented in this article. The algorithm maximises the number of virtual link requests that can be satisfactorily mapped onto a given substrate. In SDN, the proposed solution may be implemented in the controller to efficiently allocate bandwidth into the underlying forwarding plane, end to end. An extensive simulation campaign was run to evaluate the technical feasibility of the proposed approach. Simulation results showed a significant

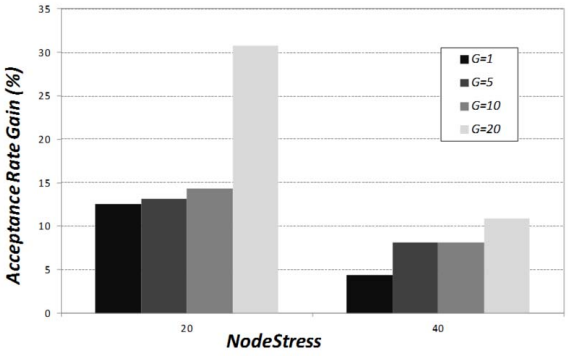


Fig. 4. Acceptance Rate Gain VLM vs. SPF; $LinkFactor = 0.2$.

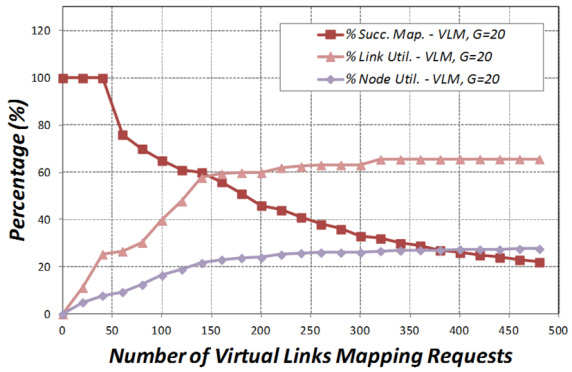


Fig. 5. Acceptance Rate and Resource Utilisation, with $LinkFactor = 0.2$, $N = 20$.

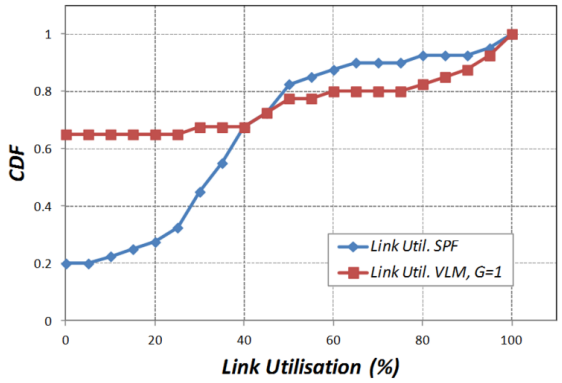


Fig. 6. VLM vs. SPF - Resource Utilisation CDFs; $NodeStress = 40$, Number Of Physical Nodes=20, $LinkFactor = 0.2$.

performance improvement compared to traditional SPF algorithms. Two digit gains were attained and, in some cases, using our formulation, up to 30% more virtual link requests were accepted, independent of the number of physically nodes. The substantial performance improvement is attributed to the more efficient physical resources utilisation, as our mapping algorithm, implemented in centralised controllers, can properly exploit the full knowledge of the network topology, and deal with (accommodate) multiple virtual link requests simultaneously. Evolutions of the presented work include performance

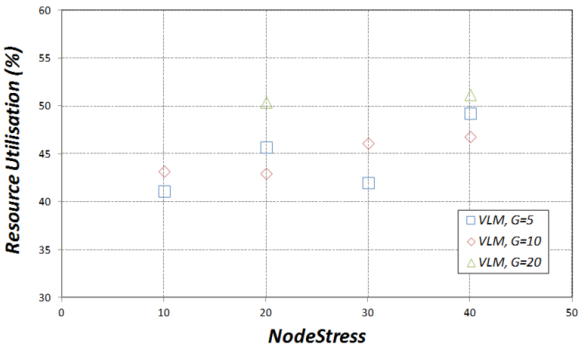


Fig. 7. VLM Average Links Utilisation with $LinkFactor = 0.2$. (Note that for $G = 20$, $NodeStress = 20$ and $NodeStress = 40$ have been considered only.)

comparisons to heuristic deviations of the SPF approach, as well as to other MILP formulations (e.g. [10]). Future developments of this work will also address implementation issues (e.g. scalability and real time path setup constraints) to introduce VLM into currently deployed as well as next generation networks.

Acknowledgements

The authors would like to extend a sincere appreciation to their steering committee, Dr. Wen Tong and Mr. Bingfu Wan, for giving them an opportunity to work on this important topic, and to our colleagues, Mr. Jianjun Wu and Mr. Yalin Liu, for their advise, guidance and support throughout the course of this investigation.

References

[1] S. Sezer et al., "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Communications Magazine*, July 2013.

[2] R. Kanagavelu et al., "Openflow based control for re-routing with differentiated flows in data center networks," *Networks (ICON), 2012 18th IEEE International Conference on*.

[3] H. Jin, D. Pan, J. Liu, and N. Pissinou, "Openflow-based flow-level bandwidth provisioning for CICQ switches," *Computers, IEEE Transactions on, Volume: 62 , Issue: 9*.

[4] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing web traffic using openflow," *ACM SIGCOMM Demo*, 2009.

[5] A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, "Openflow and PCE architectures in wavelength switched optical networks," *Optical Network Design and Modeling (ONDM), 2012 16th International Conference on*.

[6] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*.

[7] E. Oki, I. Inoue, and S. Kohei, "Path computation element (PCE)-based traffic engineering in MPLS and GMPLS networks," *Sarnoff Symposium, 2007 IEEE*.

[8] G. Banerjee and D. Sidhu, "Comparative analysis of path computation techniques for MPLS traffic engineering," *Computer Networks*, vol. 40, no. 1, pp. 149–165, 2002.

[9] S. Beker, N. Puech, and V. Friderikos, "A tabu search heuristic for the offline MPLS reduced complexity layout design problem," *NETWORKING 2004, Third International IFIP-TC6 Networking Conference, Athens, Greece, May 9-14*.

[10] J. Kempf et al., "Openflow MPLS and the open source label switched router," *Teletraffic Congress (ITC), 2011 23rd International*.

Scalable On-Demand Network Management Module for Software Defined Telecommunication Networks

Julius Mueller, Andreas Wierz

Institute for Telecommunication Systems,
Technische Universität Berlin

http://www.av.tu-berlin.de
julius.mueller@tu-berlin.de,
andreas.wierz@oms.rwth-aachen.de

Thomas Magedanz

FOKUS Fraunhofer-Institute for Open
Communication Systems
http://www.fokus.fraunhofer.de
http://www.openepc.net

thomas.magedanz@fokus.fraunhofer.de

ABSTRACT

The design of telecommunication networks and its provisioning is a challenging and complex task, which is constantly influenced by various factors. The disciplines of Traffic Engineering (TE) and Network Management (NM) have addressed these domains with static, semi-automatic and pure self-organizational approaches. Most of the existing approaches usually either relax the problem by taking strong assumptions on the problem instances or by only taking a small portion of the solution space into consideration, thus losing the chance of proving any global optimality gaps. The emerging cloud hosted flexible telecommunication system addressed by the industry nowadays puts new requirements on TE and ND. Current, telecommunication networks are often statically deployed and over-provisioned to cover predefined peak data rates, but are inflexible to adapt to dynamic network load situations. This paper presents novel dynamic Traffic Engineering (TE) and adaptive Network Management (NM) approaches for software defined telecommunication networks, which reduce Operational - and Capital Expenditures (OPEX/CAPEX), but also enhance the level of flexibility and elasticity at the same time.

Keywords

SDN, Traffic Engineering, Network, Design, Network Management, Cross-Layer, QoS

1. INTRODUCTION

The efficiency for the data transport within a network is defined through the Key Performance Indicator (KPI) 'costs per bit' from a mobile network operator's perspective. These costs have to be minimized in order to allow a growth of revenues. In the early years of Circuit Switched (CS) networks, voice calls and Short Message Services (SMS) generated large revenues for the network operator. The voice dominated area now fades out slowly and moves simultaneously into an AllIP era. The AllIP era is characterized through cheaper subscriber tariffs, (almost) free Over-The-Top (OTT) multimedia services offers and higher fixed and mobile data rates provided through new access- and core network technologies.

Several research aspects of the network have been analyzed to improve the performance significantly, of which on targets the combination of static telecommunication and flexible cloud technologies. The mobile network topology today consists of static hardware infrastructure and dynamic software service components. Hot research topics include the design of more efficient hardware with improved capacity, increase in number of network elements, strategic service placement, virtualization of the network infrastructure for enabling resource sharing, improved spectrum utilization, network optimization through definition of new network design and traffic engineering concepts, etc. At the same time, the up to now distinct domains of cloud technologies and telecommunication network started to merge. This process introduces other requirements on the telecommunication network, such as flexibility, reliability, elasticity, but also efficiency. Ongoing European research projects [1, 2, 3] are addressing these new challenges.

Our approach presented in this paper optimizes the underlying static network operator infrastructure and creates a fundament on top of which network services can be positioned, deployed and dimensioned flexible according to the resource demands. Challenges for improvements for today's telecommunication networks as well as for future network architectures of the fifth generation (5G) include disciplines of Traffic Engineering (TE) and Network Management (NM). TE and NM have been addressed in the literature with static, semi-automatic and pure self-organizational approaches. Most of the existing approaches usually either relax the problem by taking strong assumptions on the problem instances or by only taking a small portion of the solution space into consideration, thus losing the chance of proving any global optimality gaps.

The main contribution of this paper is a novel dynamic Traffic Engineering (TE) and adaptive Network Management (NM) approaches for software defined telecommunication networks, which reduce Operational - and Capital Expen-

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

ditures (OPEX/CAPEX), but also enhance the level of flexibility and elasticity at the same time. The rest of this paper is organized as follows. We present related work on routing in telecommunication networks in Section 2, mathematical preliminaries and definitions that are used throughout this paper in Section 3. The mathematical models and additional solving techniques are presented in Section 4. Section 5 outlines the implementation of the TE and NM concepts into a cross-layer optimization function for next generation mobile networks. The paper is concluded with experimental results and a summary.

2. RELATED WORK

We restrict ourselves to related work in the context of routing models for reduced power consumption which are at least slightly comparable to our approach. Along with many others, Chiaraviglio et. al [4], Bianzino et. al. [5], Amaldi et. al. [6] Huang et. al. [7] solve problems similar to ours. All of them determine a solution to a routing problem, usually a specific one, with minimal operational cost. All of them however relax the problem to a huge degree to be able to solve it. Chiaraviglio et. al, and Bianzino et. al. assume, that the physical network and the logical networks are very simple and coincide, consider exactly one traffic pattern and use traffic aggregation techniques to reduce the problem size, hence losing the ability to prove global optimality gaps. Amaldi et. al. restrict themselves to OSPF routing and realize, that their straight-forward problem formulation is not even able to solve small instances. Hence they present a two-stage model instead of addressing well-known results for OSPF routing. Huang et. al. model both, the logical and the physical network with a single traffic pattern, they however do not take wiring issues into account. Zhang et. al. [8] present a model that is very similar to ours, except that they only consider a single traffic pattern. Since their problem formulation is too big, they restrict the set of feasible routing paths, to cope with the problem, instead of using delayed column generation. Hence their results can not be ensured to be globally optimal unless the set of generated paths is sufficiently large. Heller et. al. [9] reduce the power consumption in data center networks by adaptively scaling the network resources. They use a model similar to our full formulation, with similar results and hence provide a simple greedy heuristic. Raack et. al. [10], Zhang et. al. [11], and Vasić et. al. [12] optimize routing of IP traffic via fiber optic cables in a very detailed manner, that is similar to ours. All three use a two-stage optimization approach to solve their individual problem. The first two authors achieve good optimality gaps. The results of Vasić et. al. can be shown to be arbitrarily bad due to their two-stage approach.

3. MATHEMATICAL PRELIMINARIES AND DEFINITIONS

The physical infrastructure of large scale telecommunication networks usually abstracts to a logical network that is used for routing. Each node consists of many line cards that are grouped together to form a logical node for routing. This comes from the fact that even the newest hardware does not provide sufficient capacity to support our largest nodes. Further it is much cheaper to extend a network node by adding new line cards instead of replacing the old hardware entirely. Since the logical nodes consist of several line cards, the logical links also collapse to a set of physical lines.

If not otherwise specified, we use the notation of Korte & Vygen [13]. However we'd like to present the following notations for the sake of consistency. Throughout this paper, we denote the logical network by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and its physical representation by $G = (V, E)$. The terms physical representation and physical network are used interchangeably. The physical network contains one vertex for each line card in the network and an edge connecting two nodes if and only if there is at least one physical line connecting the two line cards. Since line cards usually have several ports, it is often the case that several physical lines connect two line cards. However in our graph representation, we contract such edges, hence the graph G is simple. The logical network contains one vertex for each logical node and an edge $\{u, v\}$ connecting two nodes if and only if there is at least one physical line connecting a line card of u with a line card of v. The physical representation, i.e. the line cards, resp. physical lines, of a logical vertex $\bar{v} \in \mathcal{V}$, resp. logical edge $\bar{e} \in \mathcal{E}$ are denoted by the sets $V_{\bar{v}} \subseteq V$, resp $E_{\bar{e}} \subseteq E$. Note that $V = \dot{\bigcup}_{\bar{v} \in \mathcal{V}} V_{\bar{v}}$ and $E = \dot{\bigcup}_{\bar{e} \in \mathcal{E}} E_{\bar{e}}$ holds, i.e. the entire logical network decomposes into sets of disjoint physical objects. For the sake of readability we sometimes also exploit the notation $V_{\bar{e}}$ to denote the set of physical line cards incident to a logical link $\bar{e}$. Consequently, the logical network arises from the physical network by contraction of the sets $V_{\bar{v}}$. Usually, all line cards in a logical node are connected to each other via some internal structure. Hence we could assume that each of the sets $V_{\bar{v}}$ forms a clique of size $|V_{\bar{v}}|$. However for practical reasons we do not include these edges in G, thus the sets form a stable set in G. Each physical line card $v \in V$ has a cost $cost : V \to \mathbb{R}_{\geq 0}$ assigned, that occurs if it is activated. The bandwidth capacity of a physical line $e \in E$ is denoted by $cap : E \to \mathbb{R}_{\geq 0}$. The capacity contributes to the total logical bandwidth if and only if edge e is active. An edge is active if and only if both of its endpoints are active. Finally, we introduce a traffic demand $d : \mathcal{T} \times C \to \mathbb{R}_{\geq 0}$ for each traffic pattern and commodity.

4. ADAPTIVE FLOW PLACEMENT (FP)

We present a generic optimization scheme in which any, under mild assumptions, routing formulation $\mathcal{R}$, that is formulated as a mixed integer program (MIP), can be plugged in. The objective of the optimization scheme is to minimize the weighted operational cost arising in each of the traffic patterns that need to be supported. We assume that $\mathcal{R}$ induces capacity requirements $b : \mathcal{E} \times \mathcal{T} \to \mathbb{R}_{\geq 0}$ for each logical link $\bar{e} \in \mathcal{E}$ and traffic pattern T. Instead of formulat-

ing the operational cost arising from the physical network in a straight forward way, i.e. by introducing variables for each physical link and physical node, we use a Dantzig-Wolfe decomposition of blocks of these variables. For this, we state some observations in Section 4.1. Following in Section 4.2, we present the generic optimization scheme. Due to space limitations, we omit all proofs in this version of the paper.

4.1 Elastic Network Design (ND)

Consider a fixed set of routing paths and hence capacity allocations that need to be supported in a single traffic pattern T. Then the minimal operational cost required for this specific scenario can be evaluated by solving the problem (MCPEP1) below. We denote the problem of determining the minimal operational cost for a scenario by *The Minimal Cost Path Embedding Problem (MCPEP)* which is NP-hard as of Theorem 1. The reader can use a simple reduction from a special case of the partially ordered knapsack problem for the proof of the theorem.

Theorem 1. *MCPEP is NP-hard.*

$$\text{(MCPEP1)} \quad \min \sum_{v \in V} cost(v) \cdot x_v \quad s.t. \tag{1}$$

$$\sum_{e \in E_{\bar{e}}} x_e \cdot cap(e) \geq b_{\bar{e}} \qquad \forall \bar{e} \in \mathcal{E} \tag{2}$$

$$x_v \geq x_e \qquad \forall v \in V, \forall e \in \delta(v) \tag{3}$$

$$x_v \in \{0,1\} \qquad \forall v \in V \tag{4}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{5}$$

The problem formulation contains binary variables x_v and x_e for each physical line card and physical line. A variable is set to one if and only if the corresponding physical object is active. Constraint (3) makes sure that a physical line is active if and only if both end points are active. Constraint (2) ensures, that sufficient bandwidth is allocated on each logical link. Finally, the objective function minimizes the total operational cost arising from active objects. Due to the nature of the instances arising in our context, (MCPEP1) is usually highly block angular. Hence we can decompose the problem into several subproblems, which can be solved independently. The block structure comes from the following simple observation, that is formalized in Definition 1. If the physical hardware of a single logical link is independent of the physical hardware of all other logical links, then the variables in the optimization problem (MCPEP1) can be set independently of all other variables. This observation can also be generalized to bigger subgraphs for which the physical hardware remains independent. With help of Theorem 2, we can even find all diagonal blocks in linear time using a breath first search and the definition of independent subgraphs.

Definition 1. *A subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}') \subseteq \mathcal{G}$ is called independent if and only if*

1. $\forall \bar{e} \in \mathcal{E}', \forall \bar{f} \in \mathcal{E} \setminus \mathcal{E}' : \forall e \in E_{\bar{e}}, \forall f \in E_{\bar{f}} : e \cap f = \emptyset$,
2. *there is no subgraph $\mathcal{G}'' \subset \mathcal{G}'$ with the above property.*

Theorem 2. *For any instance $\mathcal{I}$ of MCPEP, there exists a unique decomposition into independent subgraphs $\mathcal{D}$. Further, any solution x^* of $\mathcal{I}$ can be expressed in terms of solutions to the subproblems in $\mathcal{D}$, i.e. for all $x^* \in sol(\mathcal{I})$ there exist solutions $x_D \in sol(D)$ for each $D \in \mathcal{D}$, such that $x^* = \sum_{D \in \mathcal{D}} x_D$.*

Now for each independent subgraph, we can generate a lookup-table which contains an optimal solution for each possible bandwidth requirement value. We denote the problem of determining a lookup-table for an instance of MCPEP1 by *The Partially Ordered Knapsack Lookup Table Generation Problem (POKP-LTG)*. Due to space restrictions, we give an idea for the generation of such a table for independent subgraphs which contain exactly one logical link $\bar{e}$. For such subgraphs we can easily see, that any assignment of bandwidth requirements in the interval $[0, \sum_{e \in E_{\bar{e}}} cap(e)]$ results in a feasible solution of the corresponding formulation (MCPEP1). Suppose that we found an optimal solution for a specific bandwidth requirement value $b^T_{\bar{e}}$. Then Theorem 3 states, that the solution remains optimal in a certain interval. Our proof uses a simple exchange argument. Similar results can be stated for bigger independent subgraphs. Note, that even if the lookup-table has polynomial size, generating such a table might still take an exponential amount of time. However, after the table was created, we can lookup an optimal solution to any bandwidth value b in $O(logn)$, where n denotes the total number of variables in (MCPEP1).

Theorem 3. *Let $\mathcal{I}_b$ be an instance of Min-POKP with item set $\mathcal{N} = [n]$, weights $w : \mathcal{N} \to \mathbb{R}_{\geq 0}$, cost $c : \mathcal{N} \to \mathbb{R}_{\geq 0}$, precedence constraints $P = (N, A)$ and bandwidth requirement b. If x^* is an optimal solution of $\mathcal{I}_b$ with cost c^*, then x^* is optimal for all instances $\mathcal{I}_{b'}$ with $b' \in [b_\ell, b_u]$, where $b_u := \sum_{i=1}^n w_i x_i^*$ and $b_\ell := \min\{b, \max\{\sum w_i x_i : \sum x_i c_i \leq c^* - 1, x_i \in \{0,1\}, x \text{ closed under } P\} + 1\}$.*

With help of the theorem, we can easily construct a lookup-table by iteratively solving the problem (MCPEP1) for rising values of b, the bandwidth requirement. After a solution x^* was found, the solution is optimal for the entire interval $[b_\ell, b_u]$. Hence the next bandwidth value to be evaluated is $b_u + 1$. Even though, there are instances for which the lookup-table has exponential size, this is not the case in our scenario. Theorem 5 states, that the number of intervals in a lookup-table is polynomially bounded, if the capacity and cost values are drawn from a finite set.

Theorem 4. *There are instances of POKP-LTG for which any partition has exponential size.*

Theorem 5. *Let $\mathcal{I}$ be an instance of POKP-LTG with weight and cost drawn from a finite set, i.e. $w_i \in \{u_1, ..., u_\ell\}$, $c_i \in \{c_1, ..., c_m\}$ and let ℓ_i, resp. m_i denote the number of elements with weight w_i, resp. cost c_i. then the optimal partition of $[0, \sum w_i]$ has at most $\min\{(\frac{n}{\ell} + 1)^\ell, (\frac{n}{m} + 1)^m\}$ intervals.*

4.2 Routing Formulation

With help of the lookup-tables from the previous section, we can express the operational cost arising from a fixed independent subgraph and traffic pattern by a convex combination of the solutions in the lookup-table. Instead of introducing a variable for each physical line card and physical line, we introduce pattern variables π.

Definition 2. *A pattern $\pi \subseteq V \cup E$ for $\mathcal{I}_b$ is a set of physical line cards and physical lines such that Q is closed under the partial order of $\mathcal{I}_b$.*

Definition 3. *Let π be a pattern for $\mathcal{I}_b$, then we denote the cost, resp. capacity of π by $cost(\pi) := \sum_{v \in Q} cost(v)$ and $cap(\pi) := \sum_{e \in Q} cap(e)$*

Definition 4. *If a complete set of pattern variables is given, the set $\Pi(D), D \in \mathcal{D}$, resp. $\Pi(\bar{e}), \bar{e} \in \mathcal{E}$ denotes the set of all pattern variables that contain an independent subgraph D, resp. contain a logical link $\bar{e}$.*

Consequently, each of the blocks of type (MCPEP1) can be formulated by an inner representation consisting of extreme points $\pi_i \in \Pi(D), i = 1, ..., |\Pi(D)|$.

$$\min \sum_{i \in \Pi} cost(\pi_i) \quad s.t. \tag{6}$$

(MCPEP-PAT)

$$\sum_{\substack{i \in \Pi: \\ \bar{e} \in \pi}} cap(\pi_i) \geq b_{\bar{e}} \qquad \forall \bar{e} \in \mathcal{E} \tag{7}$$

$$\sum_{i \in \Pi} \pi_i \leq 1 \tag{8}$$

$$\pi_i \in \{0, 1\} \qquad \forall i \in \Pi \tag{9}$$

With the formulation of (MCPEP-PAT), the generic optimization scheme can now be formulated as Generic Minimum Operating Costs Routing Problem with Dantzig-Wolfe (GMCRP-DW) reformulation. It consists of the routing formulation $\mathcal{R}$ and a block of type (MCPEP-PAT) for each independent subgraph D and traffic pattern T. The objective function minimizes the total weighted operational cost arising from all traffic patterns and independent subgraphs. The formulation basically corresponds to a Dantzig-Wolfe decomposition of a straight-forward formulation, where each network design block of type (MCPEP1) was replaced by an inner representation using the formulation (MCPEP-PAT).

We use SCIP to solve (GMCRP-DW) in conjunction with additional solving techniques, such as, but not limited to: 1) Primal rounding heuristics with problem specific ripup-and-reroute. 2) Domain propagation techniques that exploit the structure of (MCPEP-PAT). 3) A special variant of the well-known flow cover inequalities, see f.e. Padberg et. al. [14] for the general version, that works with (MCPEP-PAT). 4) Cutting planes that cut off feasible, however non-optimal, solutions in the branch-and-bound tree. We consider the set of fixed variables during branching to achieve this. 5) Pricing for pattern variables. 6) Pricing for path variables in $\mathcal{R}$.

$$\min \sum_{T \in \mathcal{T}} \alpha^T \cdot \sum_{D \in \mathcal{D}} \sum_{i \in \Pi(D)} {}^T\pi_i^D \cdot cost({}^T\pi_i^D) \quad s.t.$$

$$\begin{cases} \text{Routing formulation } \mathcal{R} \\ b_{\bar{e}}^T \text{ req. capacity in traffic pattern T} \end{cases}$$

(GMCRP-DW)

$$b_{\bar{e}}^T \in \mathbb{Z}_+ \qquad \forall T \in \mathcal{T}, \forall \bar{e} \in \mathcal{E}$$

$$\begin{matrix} \forall T \in \mathcal{T} \\ \forall D \in \mathcal{D} \end{matrix} \begin{cases} b_{\bar{e}}^T \leq \sum\limits_{\substack{i \in \Pi(D): \\ \bar{e} \in D}} {}^T\pi_i^D \cdot cap({}^T\pi_i^D) & \forall \bar{e} \in D \\ \sum\limits_{i \in \Pi(D)} {}^T\pi_i^D \leq 1 & \\ {}^T\pi_i^D \in \{0, 1\} & \forall i \in \Pi(D) \end{cases}$$

Due to space restrictions, we can not give any further details on these techniques in this version of the paper. For our experimental results, we want to briefly state the routing formulation $\mathcal{R}$, that was used to generate the results.

We introduce a simple multicommodity flow formulation, that allows the use of a fixed number k_C of paths x_C^P (14) for each commodity C. Constraint (11) ensures that limitation. For each traffic pattern T, each commodity can further choose which fraction ${}^T\lambda_C^P$ (15) of the traffic demand to send along each of the selected paths P. Constraint (10) ensures, that a threshold value is forced to zero, whenever the corresponding path is not selected, and Constraint (12) ensures, that the entire demand is routed. Finally, Constraint (13) specifies the bandwidth requirement for each logical link and traffic pattern. For the experimental results, we restrict ourselves to fat-tree topologies, i.e. topologies where we can assign a level to each logical node. Logical links only connect nodes in adjacent levels. Hence we denote the set of all feasible paths for a commodity C by $\mathcal{P}_C$.

$${}^T\lambda_C^P \leq x_C^P \qquad \forall C \in C, \forall P \in \mathcal{P}_C, \forall T \in \mathcal{T} \tag{10}$$

$$\sum_{P \in \mathcal{P}_C} x_C^P \leq k_C \qquad \forall C \in C \tag{11}$$

$$(\mathcal{R}) \sum_{P \in \mathcal{P}_C} {}^T\lambda_C^P = 1 \qquad \forall C \in C, \forall T \in \mathcal{T} \tag{12}$$

$$\sum_{C \in C} \sum_{\substack{P \in \mathcal{P}: \\ e \in P}} {}^T\lambda_C^P \cdot d_C^T \leq b_{\bar{e}}^T \qquad \forall \bar{e} \in \mathcal{E}, \forall T \in \mathcal{T} \tag{13}$$

$$x_C^P \in \{0, 1\} \qquad \forall C \in C, \forall P \in \mathcal{P}_C \tag{14}$$

$${}^T\lambda_C^P \in [0, 1] \qquad \forall C \in C, \forall P \in \mathcal{P}_C, \forall T \in \mathcal{T} \tag{15}$$

5. IMPLEMENTATION AND INTEGRATION

This section presents the prototypical implementation and integration of the dynamic Traffic Engineering (TE) and adaptive Network Management (NM) modules within a cross-layer framework namely Generic Adaptive Resource Control (GARC). In [15, 16] GARC has been described as a flexible cross layer control function, which bridges the gap between service and network layer and therefore enables novel system optimizations. GARC is an optional component for next generation mobile telecommunication core net-

works and enables value added services for telco networks. The three main references points of GARC are towards a) the underlying network, b) the end user device and c) the service or 3GPP Application Function (AF). The concept of Software Defined Network (SDN) and especially OpenFlow [17] have been used for our validation due to the open network control in comparison to other network technologies. The interaction between GARC and the SDN over the SDN northbound interface is depicted in Figure 1.

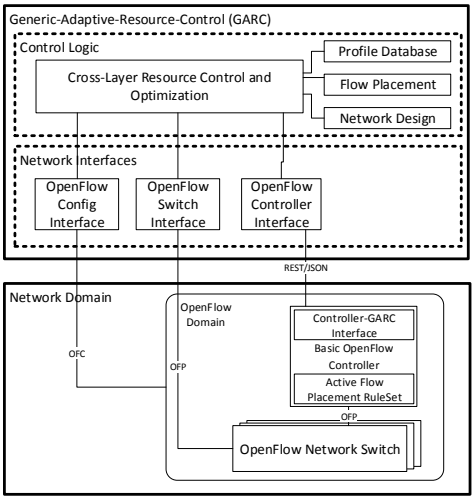


Figure 1: Elastic Network Design and Adaptive Flow Placement within GARC and SDN

GARC consists of functional components and an interface layer towards the underlying network technology. The main and central component of GARC is the Cross-Layer Resource Control and Optimization module. It controls and optimizes other modules as adaptive Flow Placement (FP) 4 and elastic Network Design (ND) 4.1 as well as network operator policies such as subscriber profiles further described in [15]. Real-time network statistics are queried by GARC from OpenFlow Controller and Switches and thus enables cross layer optimizations on higher layers in turn. Real-time network utilization and predefined thresholds influences changes in the routing scheme or network design through toggling activation or deactivation of network line cards. GARC enables in addition to radical modifications of the network design also application layer parameter adjustments of CODEC or Service Data Rate (SDR) changes. Therefore a cost model has to be defined, in which general network modification or individual re-routing or SDR adjustments are considered. Routing schemes outlined in Section 4.1 are precalculated and pushed towards the OpenFlow controller actively over the JSON-REST interface depicted in Figure 1. Scalability through FlowTable Entry push operation. Local evaluation per SDN domain.

6. EXPERIMENTAL VALIDATION

This section outlines the validation setup first and discusses the computational results finally.

6.1 Validation Setup

The model from Section 4 was implemented in C++ using SCIP linked to CPLEX as branch-and-price framework. All additional solving techniques were implemented as modules for SCIP in the usual fashion. For all computational results, we compare the models from this paper with a straightforward implementation that solves the same problem, i.e. using the same routing formulation $\mathcal{R}$, however without pattern variables and the additional solving techniques. All results were generated on a server with four Intel® Xeon® CPU E5-1620 processors with 3.60GHz and 16GB of RAM.

Name	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert V\rvert$	$\lvert E\rvert$	# tp	# comm.
T1-*	48	190	379	190	10	29
T2-*	19	51	308	490	5	8
T3-*	64	173	1,386	2,739	5	20
T4-*	64	170	1,249	1,377	5	20
T5-*	53	188	7,133	31,293	20	20

Table 1: Mean size, number of traffic patterns and number of commodities of the randomly generated fat-tree instances.

The test instances are randomly generated fat-tree instances with different properties. The set of instances describes a mix of sparse, as well as dense graphs and different amount of traffic patterns and bandwidth requirements. A table of statistics about the test instances is given in Table 1. For each type, five instances were considered, the statistics are mean values among each instance type. Instances of type T1 are connectivity tests, i.e. any assignment of routing paths yields a feasible solution. Instances of type T2 are rather small, however very dense networks. Instances of type T3 and T4 are very similar and differ mostly in the set of commodities which need to be routed. Instances in T3 have rather low bandwidth requirements, instances in T4 have low- as well as very high capacity requirements. Instance T5 is a medium-sized network, that needs to ensure 20 traffic patterns, consisting of both, low and high traffic demands.

6.2 Computational Results

A brief summary of the results is given in Table 2. A column with the sum of running times and average optimality gap is given for both models. The advanced model yields stronger results in every instance. On average, the solution quality is increased by a factor of 99.4. With the advanced model, we were able to solve 9 out of 25 instances to optimality, hence the difference in solving time. Six additional instances were solved to an optimality gap lower than 5%. The full formulation was not able to solve any instance to an optimality gap lower than 80%. On the big instances of type T5, not even the root LP relaxation was solved to optimality, due to the large number of variables. The worst optimality

Name	Full		Advanced	
	Time	Gap	Time	Gap
T1	05:00	819.81%	02:10	1.21%
T2	05:00	611.80%	01:01	0.38%
T3	05:00	2,423.56%	02:54	0.57%
T4	05:00	113.18%	05:00	35.57%
T5	05:00	∞	05:00	12.15%
Total	25:00	992.09%	16:06	9.98%

Table 2: Testbed results with running time limited to one hour. Results using the standard formulation on the left and the advanced model on the right. Time columns show the sum of solver times among the five instances in the format hh:mm, gap shows the mean optimality gap.

gap in the full formulation is 2695.91% in an instance of type T2. The advanced model is able to solve the same instance to optimality after 32 minutes, yielding both, better primal and dual bounds. The worst optimality gap for the advanced model does not exceed 50%.

7. CONCLUSIONS AND FUTURE WORK

We have seen both, a model for adaptive network design, as well as an optimization framework that can be used to guide any, under mild assumptions, MIP-based routing model to find optimal solutions with respect to minimizing the total weighted operational cost arising from a number of traffic patterns. We chose fat-tree network topologies aligned on realistic telecommunication networks to validate our concepts. The results using a straight-forward approach have proven to be outside of any reasonable bounds, on bigger instances, the model was not even able to solve the root relaxation. The advanced MIP model however is able to solve the presented randomly generated test instances either to optimality or to prove optimality gaps that were lower than 10% on average. In addition to theoretical results, we have outlined, how both modules can, independently of each other, be used in telecommunication networks.

8. REFERENCES

[1] FP7 IP MobileCloud-Networks (MCN), "MCN Consortium, Project Number: 318109." Website, 2012. http://www.http://mobile-cloud-networking.eu/.

[2] Mobile Networks Evolution for Individual Communications Experience (MEVICO), "MEVICO Consortium." Website, 2012. http://www.mevico.org/index.html.

[3] METIS 2020 Mobile and wireless communications Enablers for the Twenty-twenty Information Society 5G, "METIS2020 Consortium." Website, 2013. https://www.metis2020.com.

[4] L. Chiaraviglio, M. Mellia, and F. Neri, "Minimizing isp network energy cost: Formulation and solutions," *IEEE/ACM Transactions on Networking (TON)*, vol. 20, no. 2, pp. 463–476, 2012.

[5] A. Bianzino, C. Chaudet, F. Larroca, D. Rossi, and J. Rougier, "Energy-aware routing: a reality check," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pp. 1422–1427, IEEE, 2010.

[6] E. Amaldi, A. Capone, L. Gianoli, and L. Mascetti, "Energy management in ip traffic engineering with shortest path routing," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pp. 1–6, IEEE, 2011.

[7] S. Huang, D. Seshadri, and R. Dutta, "Traffic grooming: a changing role in green optical networks," in *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE*, pp. 1–6, IEEE, 2009.

[8] M. Zhang, C. Yi, B. Liu, and B. Zhang, "Greente: Power-aware traffic engineering," in *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pp. 21–30, IEEE, 2010.

[9] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks.," in *NSDI*, vol. 3, pp. 19–21, 2010.

[10] F. Idzikowski, S. Orlowski, C. Raack, H. Woesner, and A. Wolisz, "Dynamic routing at different layers in ip-over-wdm networks—maximizing energy savings," *Optical Switching and Networking*, vol. 8, no. 3, pp. 181–200, 2011.

[11] Y. Zhang, M. Tornatore, P. Chowdhury, and B. Mukherjee, "Time-aware energy conservation in ip-over-wdm networks," in *Photonics in Switching*, Optical Society of America, 2010.

[12] N. Vasić, P. Bhurat, D. Novaković, M. Canini, S. Shekhar, and D. Kostić, "Identifying and using energy-critical paths," in *Proceedings of the Seventh Conference on emerging Networking Experiments and Technologies*, p. 18, ACM, 2011.

[13] B. Korte and J. Vygen, "Combinatorial optimization: Theory and algorithms," 2010.

[14] M. Padberg, T. Van Roy, and L. Wolsey, "Valid linear inequalities for fixed charge problems," *Operations Research*, vol. 33, no. 4, pp. 842–861, 1985.

[15] J. Mueller, A. Wierz, V. D., and M. T, "Elastic Network Design and Adaptive Flow Placement in Software Defined Networks," in *Proceedings of the International Conference on Computer Communications and Networks ICCCN 2013*, IEEE, 2013.

[16] J. Mueller and T. Magedanz, "Towards a generic application aware network resource control function for Next-Generation-Networks and beyond," in *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, pp. 877 –882, oct. 2012.

[17] The Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," April 2012.

Live Migration of Virtualized Edge Networks: Analytical Modeling and Performance Evaluation

Franco Callegati, Walter Cerroni
D.E.I. "G. Marconi" - University of Bologna
via Venezia 52, 47521 Cesena - Italy
E-mail: name.surname@unibo.it

***Abstract*—Following the current evolution of virtualization techniques and software defined networking, edge networks might evolve towards a fully virtualized implementation by means of a number of virtual machines working cooperatively to perform the tasks of existing network middleboxes. In such a scenario the possibility to migrate groups of cooperating virtual machines as a whole set may be a very important feature, but what will be the performance issues of this solution? The live migration performance of multiple virtual machines working in some sort of correlated manner is a topic that has not been widely studied in the literature. This manuscript presents a model reasonably simple to implement that may be used to derive some performance indicators such as the whole service downtime and the total migration time. The model is used to compare some scheduling strategies for the migration and provide guidelines to such implementation.**

***Index Terms*—Virtualization, Software Defined Networks, Analytical modeling, Performance evaluation.**

I. Introduction

In current networks IP lost its simple and effective end-to-end delivery paradigm. Networking services require a number of additional and more complex functionalities, typically stateful, which cannot be supported by IP and must be performed by middleboxes operating at logical layers above routing and forwarding (L3). This has been outlined very clearly by some recent investigations. The results in [1] show that the amount of middleboxes and the burden related to running them is considered a major issue by network administrators, whereas [2] shows that at least 25% of the TCP flows analyzed suffered of some form of interference by middleboxes running tasks at or above layer 4.

In public networks, middleboxes management is an issue for operators [3], especially at the edge where the evolution of terminals is bringing a degree of complexity in the implementation of services never experienced before. Emerging technologies, such as Software Defined Networking (SDN) [4] and Network Function Virtualization (NFV) [5], offer a new opportunity to migrate the middleboxes in a virtualized environment, with possible very significant advantages.

In essence, these key enablers will allow the development of network nodes based on standard off-the-shelf hardware, cheap but powerful enough to run virtualized network functions and services. Edge networks will then be made of inexpensive nodes and user devices capable of collapsing the OSI layers (e.g., L2 to L7) on standard hardware solutions. In this scenario edge networks will become a distributed environment made of clouds of virtual resources (operated even by diverse players) interconnected by a simpler and less hierarchical core network [6].

Such a radical transformation will enable new roles and business opportunities, completely reshaping the value chains in the entire Telco-ICT world. One of the main challenges behind this vision is the capability of dynamically instantiating, orchestrating and migrating multiple Virtual Machines (VMs) running network services and applications across the providers transport networks. Ensembles of VMs will be strictly related and intertwined to implement sets of virtual resources and services that network operators, and even users, must be able to configure and program.

This important topic has still to be deeply investigated, since in the past mostly the migration of single VMs has been subject of research studies. In this work we intend to address this gap. In Section II we briefly review the issue of VM migration, outlining that the *pre-copy* migration strategy is the one that apparently better matches the specific application described above. Then in Section III we propose an extension to known analytical models for the migration of a single VM to the case of multiple VMs to be migrated with some form of coordination and discuss the two obvious alternatives of serial and parallel migration. In Section IV performance evaluation results obtained with the model are presented, comparing the two migration strategies and discussing some quantitative engineering guidelines. Finally in Section V some conclusions are drawn.

II. Live Migration of Virtual Machines

The problem of migrating virtual machines live, i.e. while the VM is running and end-user processes are active, is well known and has already been addressed in the literature. It is impossible to migrate the VM without stopping it even for a short time, therefore the migration impacts the running services. As a consequence a major performance parameter to be considered is the so called *downtime* (T_{down} in the following), i.e. the amount of time the VM will be stopped because of the migration. On the other hand the *total migration time* (T_{mig} in the following) is another important parameter that measures the impact that the migration of the VM has on the whole IT infrastructure, since the resources used during the migration phase cannot be used for other tasks.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

There are several strategies for the migration of VMs that aim at optimizing one performance indicator or another according to the overall system requirements. For the purpose of this study we will focus on a typical *pre-copy* strategy [7], which combines a *push* phase, during which the VM memory is transferred in subsequent rounds while the VM is still running, and a *stop-and-copy* phase, during which the VM is stopped and just a residual part of the memory is transferred.

This migration strategy can be implemented if the virtualization manager enables some form of shadow memory that makes possible to understand which part of the VM memory was modified in a given time window. This is the case in typical modern systems such as Xen and KVM [8].

The push phase consists of n memory transfers also called *rounds*. At round 0 the whole memory of the VM is copied to the new location. This will take time and, since the VM is running, part of the memory will be modified by the running processes. Assuming that the modified memory pages are less than the total, the transfer is iteratively repeated and during round i the memory pages that were modified during round $i-1$ are transferred. If memory pages *dirtied* during each round tends to decrease the duration of the rounds will also decrease and at some stage the estimated number of pages to be transferred, as well as the duration of round n, will be reasonably low. At this point round n is performed in the stop-and-copy phase: the VM is stopped, the modified memory pages are transferred and the VM can be consistently restarted at the new location.

A model for the migration of a single VM with pre-copy strategy is proposed in [9]. The model is presented for a single VM and using some simplifying conditions, such as for instance a constant memory page dirtying rate. The model is used to evaluate the downtime as well as the overall transfer time for the VM. We start from these results and extend them as explained in the following section.

III. Modeling Multiple VM Migration

In this paper we intend to tackle the more complex problem of migrating groups of correlated VMs, therefore the performance parameters to be considered will depend on what happens to the migration of several and not just one VM. The extension of the model in [9] to the case of a set of VMs is not trivial, since it depends on how the multiple VMs are scheduled for the live transfer. Furthermore, moving a set of mutually dependent VMs requires a new definition of service downtime, and this again is strictly related to the role and migration timing of each VM.

Consider the general case of a set of M VMs, where $V_{\mathrm{m},j}$ is the amount of memory allotted to the j-th VM in the set, $D_j(t)$ its memory page dirtying rate and P_j its memory page size. According to the pre-copy migration algorithm, when the j-th VM transfer starts at time $t_{0,j}$, the whole memory $V_{\mathrm{m},j}$ is copied to the destination host while the VM instance is still running at the source host. Then, the transfer of modified memory pages is iterated n_j times at $t_{1,j},\ldots,t_{n_j,j}$, until either the amount of dirty memory is below a given threshold V_{th} or the maximum number of iterations $n_{\max}$ is reached.

Let $R_j(t)$ be the bit rate used to transfer the j-th VM to the destination host and $V_{i,j}$ the amount of dirty memory to be copied during the i-th round. The following equations describe the iterative migration process:

$$V_{0,j} = V_{\mathrm{m},j} = \int_{t_{0,j}}^{t_{1,j}} R_j(t)dt \quad (1)$$

$$V_{i,j} = \int_{t_{i-1,j}}^{t_{i,j}} P_j D_j(t)dt = \int_{t_{i,j}}^{t_{i+1,j}} R_j(t)dt, \quad i=1,\ldots,n_j \quad (2)$$

As a first approximation, we can assume that both $R_j(t)$ and $D_j(t)$ are constant during the whole migration process. For the transfer rate this is a reasonable assumption, considering that some form of bandwidth sharing mechanisms with guaranteed bit rate must be enforced in order to keep the performance of the virtualized edge network under control. As for the memory page dirtying rate, its behavior strongly depends on the nature of the applications running on the VM: however, as a first approximation we can consider that the amount of pages being modified by the running processes is proportional to their execution time [9].

Then we also assume that the M VMs in the set are allotted the same amount of memory and that the applications running on them show the same page dirtying rate. We are aware that this may not be completely true in the virtualized edge network architecture we are considering, since the memory profile of each VM strongly depends on the specific network functions it has to perform. However, these assumptions allow to simplify the equations and to capture the macroscopic performance aspects of multiple VM live migration, which is the main purpose of this paper. A more refined model is left for future work.

Based on the previous assumptions, we have

$$D_j(t)=D, \quad V_{\mathrm{m},j}=V_{\mathrm{m}}, \quad P_j=P, \quad \forall j=1,\ldots,M$$

whereas $R_j(t)=R_j$ depends on how the M VMs are scheduled to be migrated. Calling $T_{i,j}=t_{i+1,j}-t_{i,j},\ i=0,\ldots,n_j$, equations (1) and (2) become

$$V_{0,j} = V_{\mathrm{m}} = R_j\,T_{0,j}$$
$$V_{i,j} = P\,D\,T_{i-1,j} = R_j\,T_{i,j}, \quad i=1,\ldots,n_j$$

from which we can compute the duration of the i-th iteration as

$$T_{i,j} = V_{\mathrm{m}}\,\frac{(P\,D)^i}{R_j^{i+1}}, \quad i=0,\ldots,n_j \quad (3)$$

The pre-copy migration algorithm is sustainable as long as the average memory dirtying rate is smaller than the transfer rate, i.e. when $\lambda_j=(P\,D)/R_j<1$. The last round (stop-and-copy phase) will happen either for the smallest i such that

$$V_{i,j}=\lambda_j^i\,V_{\mathrm{m}} \le V_{\mathrm{th}}$$

or when $n_{\max}$ iterations are reached. Therefore, the number of iterations and the time needed to migrate the j-th VM are

$$n_j = \min\left\{\left\lceil \log_{\lambda_j} V_{\rm th}/V_{\rm m} \right\rceil, n_{\max}\right\} \quad (4)$$

$$T_{{\rm mig},j} = \sum_{i=0}^{n_j} T_{i,j} = \frac{V_{\rm m}}{R_j}\,\frac{1-\lambda_j^{n_j+1}}{1-\lambda_j} \quad (5)$$

The computation of the total migration time and downtime of a set of VMs strictly depends on how and when the VMs are scheduled to be migrated. In the following we consider two simple cases: (i) when the M VMs are transferred one at a time (*serial migration*); (ii) when all the M VMs are transferred simultaneously (*parallel migration*). We then compare the performance of these two cases assuming that the migration is performed through a communication channel with total bit rate R. A useful parameter is the ratio between the dirtying rate and the transmission channel bit rate

$$\lambda = \frac{P\,D}{R}.$$

A. Serial Migration of Multiple VMs

When the M VMs are migrated one at a time, each transfer is performed at full channel bit rate, i.e. $R_j = R,\ \forall j$. In this case, $\lambda_j = \lambda$ and each VM is migrated in $n_j = n^{(s)}$ rounds, according to (4). Therefore, the serial migration time of the whole VM set is given by

$$T_{\rm mig}^{(s)} = \sum_{j=1}^{M} T_{{\rm mig},j} = M\,\frac{V_{\rm m}}{R}\,\frac{1-\lambda^{n^{(s)}+1}}{1-\lambda} \quad (6)$$

The downtime of the whole VM set starts when the first VM is stopped at the source host (i.e., when the last iteration of the first VM begins) and ends when the last VM is resumed at the destination host. If $T_{\rm res}$ is the fixed time required to resume a VM at the destination host, the serial migration downtime can be computed as

$$T_{\rm down}^{(s)} = \frac{V_{\rm m}}{R}\,\lambda^{n^{(s)}} + (M-1)\,\frac{V_{\rm m}}{R}\,\frac{1-\lambda^{n^{(s)}+1}}{1-\lambda} + T_{\rm res} \quad (7)$$

B. Parallel Migration of Multiple VMs

When the M VMs are migrated simultaneously, each one of them is transferred at a bit rate that depends on how the bandwidth is shared among the on-going connections. Assuming an equal share of the channel bandwidth and considering that all the VMs in the set have the same memory profile, all VMs start and end their migration at the same instants. Therefore, there are always M simultaneous transfers and the transfer rate seen by each VM is $R_j = R/M,\ \forall j$. In this case, $\lambda_j = M\lambda$ and each VM is migrated in $n_j = n^{(p)}$ rounds, according to (4). The parallel migration time of the whole VM set is equivalent to the migration time of any single VM and is given by

$$T_{\rm mig}^{(p)} = T_{{\rm mig},j} = M\,\frac{V_{\rm m}}{R}\,\frac{1-(M\lambda)^{n^{(p)}+1}}{1-M\lambda} \quad (8)$$

The parallel migration downtime of the whole VM set corresponds to the last iteration (stop-and-copy phase) of any single VM and is given by

$$T_{\rm down}^{(p)} = M\,\frac{V_{\rm m}}{R}\,(M\lambda)^{n^{(p)}} + T_{\rm res} \quad (9)$$

C. Comparing Serial and Parallel Migration

From the formulas obtained above we can draw some conclusions about how the serial migration of a set of VMs compares to the case of parallel migration. If we compare equations (6) and (8) we can immediately see that this is not a trivial case of serial vs. parallel data transfer. In fact, migrating a sequence of M VMs at rate R does not require the same amount of time as migrating M VMs in parallel each at rate R/M, because the iterative live migration process and the fixed memory page dirtying rate produce different amounts of data to be transferred in the two cases.

In particular, we can prove that the parallel migration time is larger than the serial migration time. If we subtract (6) from (8) when $n^{(s)} = \log_\lambda V_{\rm th}/V_{\rm m}$ and $n^{(p)} = \log_{M\lambda} V_{\rm th}/V_{\rm m}$, we obtain

$$T_{\rm mig}^{(p)} - T_{\rm mig}^{(s)} = \frac{M\lambda\,(M-1)\,(V_{\rm m}-V_{\rm th})}{R\,(1-\lambda)\,(1-M\lambda)} \quad (10)$$

which is always non-negative under our assumptions. Similarly, we can prove that the serial migration downtime is larger than the parallel migration downtime, obtaining from (7) and (9)

$$T_{\rm down}^{(s)} - T_{\rm down}^{(p)} = \frac{(M-1)\,(V_{\rm m}-V_{\rm th})}{R\,(1-\lambda)} \quad (11)$$

which, again, is always non-negative.

These results suggest a possible trade-off: on the one hand, parallel migration is better than serial migration in terms of service provisioning, since the downtime is smaller; on the other hand, serial migration shows a smaller total transfer time and thus is better than parallel migration in terms of communication resource usage and transmission overhead.

IV. Numerical Results

In this section we present some numerical results obtained with the multiple VM live migration model introduced above. In order to represent a realistic scenario for the virtualized edge network architecture, we assign the following reference values to the model parameters:

- $M = 8$;
- $R = 1\,\text{Gbps}$, a reasonable bit rate available for inter-data center communication within the network infrastructure that provides connectivity to the virtualized edge networks we are considering;
- $V_{\rm m} = 1\,\text{GB}$, a typical value in current virtualization technologies and large enough to allow the execution of the required network functions;
- $P = 4\,\text{KB}$, the most common memory page size adopted by modern operating systems;
- $D = 2500\,\text{pps}$, corresponding to $\lambda = 0.082$; this is a reasonable estimation of the dirtying rate based on benchmark measurements [9] and considering that: (i) the VMs performing network functions are expected to be

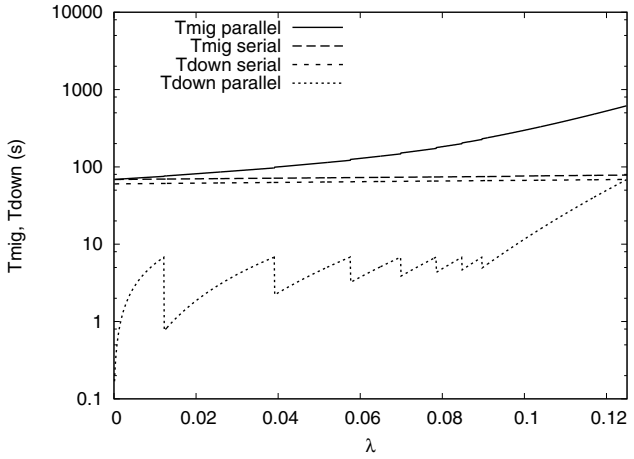


Fig. 1. Serial vs. parallel total migration time and downtime as a function of the ratio of dirtying rate over channel bit rate.

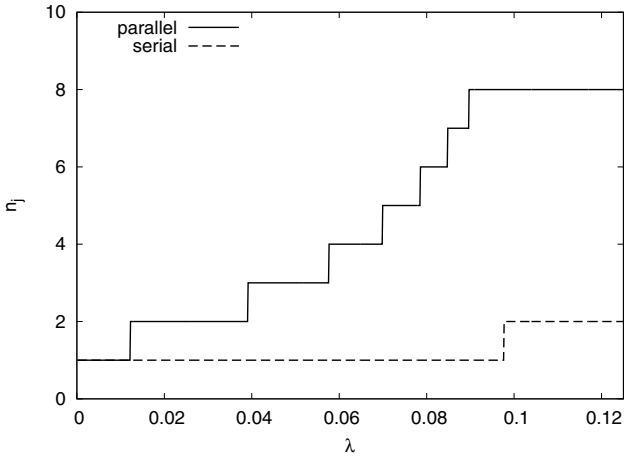


Fig. 2. Number of iterations required to migrate each VM as a function of the ratio of dirtying rate over channel bit rate.

not extremely demanding in terms of memory updates; (ii) the most frequently modified memory pages (the so-called *Writable Working Set*) are typically transferred only during the stop and copy phase, thus reducing the amount of dirty pages to be copied in each round [7];

- $V_{\mathrm{th}} = 100\,\mathrm{MB}$, a reasonable compromise considering the value of the other parameters;
- $n_{\max} = 8$, an estimated realistic value to limit the total transfer time of each VM;
- $T_{\mathrm{res}} = 100\,\mathrm{ms}$, a conservative estimation considering the current virtualization technologies.

The charts included in this section show the performance trends when one of the model parameters varies within its validity range, while all the other parameters are fixed to the reference values.

Figure 1 compares the serial and parallel migration strategies, in terms of both total migration time and downtime, as a function of the ratio of dirtying rate over channel bit rate. In this case the validity range $\lambda < 1/M$ is determined by the parallel migration. The figure allows to quantify the existing trade-off between serial and parallel migration already mentioned in subsection III-C. The logarithmic y-axes scale is used because the parallel migration downtime is extremely shorter than the serial migration downtime, while the total migration time is not too much larger as long as the dirtying rate is small. However, when λ goes beyond 0.04, $T_{\mathrm{mig}}^{(p)}$ grows significantly fast, whereas the serial migration shows much less sensitivity in this range.

This is mainly due to the different number of iterations needed to migrate each VM in the two cases. As shown in Fig. 2, the reduced transfer rate available to the single VM causes a quick increase in the number of iterations required in the parallel migration, whereas in the serial migration the full-rate transfer of each VM is completed in one or two iterations. The saw-tooth shape of the curves in Fig. 1 (quite visible for $T_{\mathrm{down}}^{(p)}$, but actually present in all the other curves) is caused

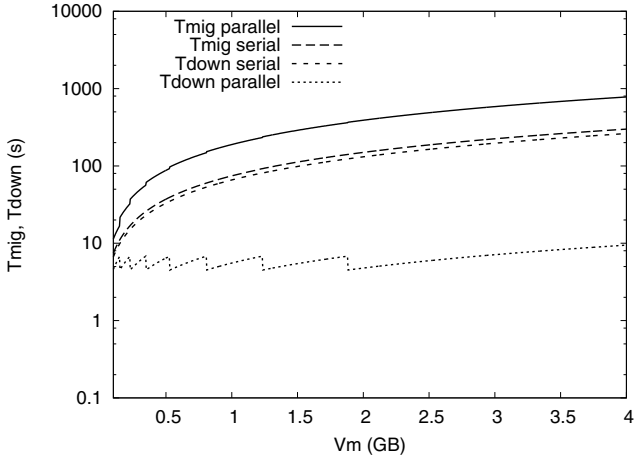


Fig. 3. Serial vs. parallel total migration time and downtime as a function of the VM memory size.

by the ceiling function present in (4), up to the point in which the number of iterations reaches $n_{\max}$, i.e., when $\lambda > 0.0897$ in the parallel migration case. Beyond that point, the total migration time grows more slowly with the dirtying rate, at the expenses of an increasing downtime.

Figure 3 compares the serial and parallel migration strategies as a function of the amount of memory allotted to each VM. In this case the validity range is $V_{\mathrm{m}} > V_{\mathrm{th}}$. Both serial and parallel total migration time as well as the serial downtime grow linearly with the VM memory size, as expected from (6), (7) and (8). The parallel downtime, instead, fluctuates around $5\,\mathrm{s}$ and starts to grow only when $n_{\max}$ is reached, i.e., when $V_{\mathrm{m}} > 1.87\,\mathrm{GB}$. This is due to the fact that, below that point, up to $V_{\mathrm{th}}\,\mathrm{MB}$ must be transferred during the stop & copy phase, thus keeping the downtime limited.

The role of the stop-and-copy phase threshold is displayed in Fig. 4. This parameter determines the required number of iterations as per (4). A large threshold reduces the value of n_j

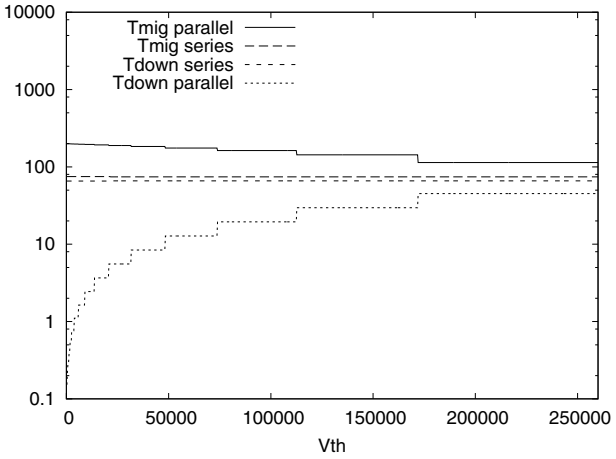


Fig. 4. Serial vs. parallel total migration time and downtime as a function of the stop-and-copy phase threshold.

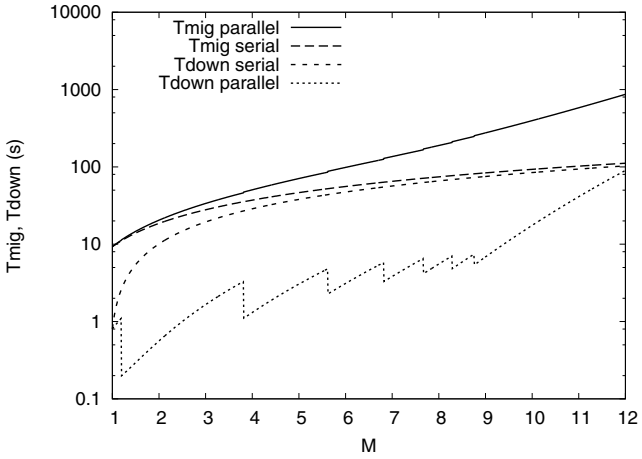


Fig. 6. Serial vs. parallel total migration time and downtime as a function of the number of VMs in the set.

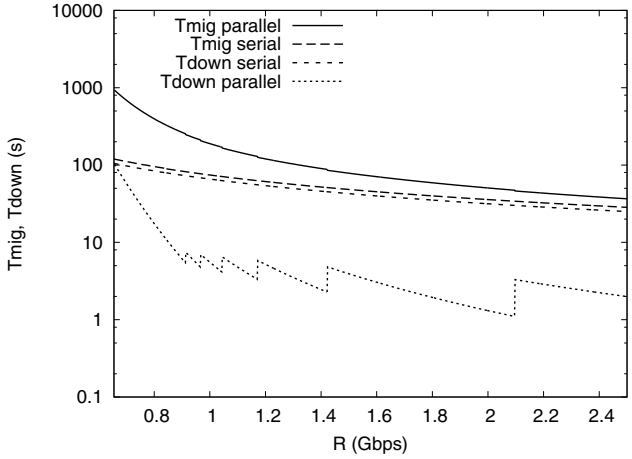


Fig. 5. Serial vs. parallel total migration time and downtime as a function of the transmission channel bit rate.

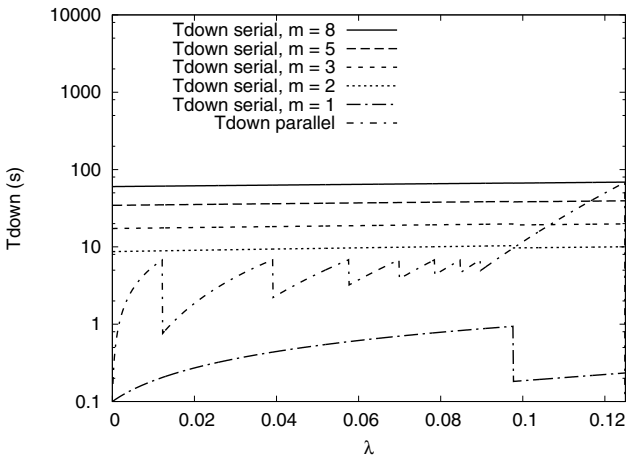


Fig. 7. Serial vs. parallel migration downtime as a function of the ratio of dirtying rate over channel bit rate for different sizes of the critical VM subset.

and then decreases the total migration time, but increases the downtime because more memory must be copied during the last iteration. With the reference values chosen, the parallel migration results much more sensitive to V_{th} than the serial migration due to the higher value of λ_j. The maximum number of iterations in the parallel migration is reached when $V_{\mathrm{th}} < 53.5\,\mathrm{MB}$.

The migration performance improves when the transmission channel bit rate increases, as shown in Fig. 5 where the validity range is $R > M\,P\,D$, as determined by the sustainability of the pre-copy algorithm in the parallel migration case. This figure is useful to dimension the inter-data center network bandwidth required to achieved a specified level of downtime and/or total migration time when the memory page dirtying rate is known. In this case, the parallel migration reaches the maximum number of iterations when $R < 913.71\,\mathrm{Mbps}$.

Figure 6 shows the impact of the size of the set of VMs to be migrated on the transfer performance. The parallel and serial migration strategies give the same result when $M = 1$, as obvious, and the difference becomes more significant when M increases, up to the validity limit of 12. As long as the number of VMs is kept limited, the parallel migration gives a very small downtime without exceeding too much with the total migration time. However, when M grows beyond 4, $T_{\mathrm{mig}}^{(p)}$ explodes, while $T_{\mathrm{mig}}^{(s)}$ keeps increasing linearly. The parallel migration reaches the maximum number of iterations when $M \geq 9$. When M approaches the validity limit the parallel migration downtime exponentially grows closer to the serial migration downtime, but with a much higher total migration time.

Finally, Fig. 7 shows how the serial migration downtime can be improved if we consider only a subset of the VMs as critical for the offered service. In fact, if we assume that only m out of M VMs are performing network functions that are strictly required to guarantee minimum connectivity service to the customer and if we give priority to these VMs in the migration process, we can redefine $T_{\mathrm{down}}^{(s)}$ as in (7) after

substituting M with m. The remaining $M - m$ VMs still to be migrated provide additional services that will be available to the customer with an additional delay. Figure 7 shows that the serial migration downtime could be very close to or even smaller than the parallel migration downtime if the critical VM subset is small, especially when the number of iterations in the parallel case has reached $n_{\max}$.

V. Conclusion

This manuscript presented a model to evaluate the performance of the live migration of a set of virtual machines cooperating to implement some sort of networking service. The service downtime due to the migration and the total migration time were derived for the two alternatives of serial and parallel migration strategies, showing that a trade-off exists. In particular, parallel migration provides a shorter downtime at the expense of a much longer migration time, meaning that it minimizes the service downtime at the cost of maximizing the communication resources occupancy and thus reducing their availability. The results presented also show that some interesting compromise could be achieved when just some of the virtual machines are strictly necessary to provide the basic connectivity service and therefore some of them may be temporarily switched off.

Acknowledgments

This work was partially supported by EINS, the Network of Excellence in Internet Science (http://www.internet-science.eu) through European Union's 7th Framework Programme under Communications Networks, Content and Technologies, Grant Agreement no. [288021].

References

[1] J. Sherry, et al., "Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service", Proceedings of the ACM SIGCOMM Conference, Helsinki, Finland, August 2012.

[2] M. Honda, et al., "Is it still possible to extend TCP?", Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC), Berlin, Germany, November 2011.

[3] A. Manzalini, "Middle-Boxes? No thanks, Stateless Core and Stateful Edges", Telecom Italia Future Center, February 2013, available at http://www.blog.telecomfuturecentre.it/author/manzalini/

[4] The Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks", ONF White Paper, April 2012, available at https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf

[5] "Network Functions Virtualisation", ETSI introductory white paper, October 2012, available at http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[6] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, A. Campi, "Clouds of Virtual Machines in Edge Networks", IEEE Communications Magazine, Vol. 51, No. 7, pp. 63-70 , July 2013.

[7] C. Clark, et al., "Live Migration of Virtual Machines", Proceedings of the 2nd USENIX Symposium on Networked Systems Design & Implementation (NSDI), Boston, MA, May 2005.

[8] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, E. Roman, "Optimized Pre-Copy Live Migration for Memory Intensive Applications", Proceedings of the International Conference of High Performance Computing, Networking, Storage and Analysis (SC), Seattle, WA, November 2011.

[9] H. Liu, H. Jin, C.-Z. Xu, X. Liao, "Performance and energy modeling for live migration of virtual machines", Cluster Computing, Springer, Vol. 16, No. 2, pp. 249-264, June 2013.

Evolution and Challenges of Software Defined Networking

Ángel Leonardo Valdivieso Caraguay, Lorena Isabel Barona López,
Luis Javier García Villalba, *Senior Member, IEEE*

***Abstract*—Software Defined Networking (SDN) proposes the separation of the control plane from the data plane in network nodes. Furthermore, Openflow architecture through a centralized control of the packet forwarding engines enables the network administrators to literally program the network behavior. The research and results of experiments show clear advantages over traditional network architectures. However, there are open questions to be solved in order to integrate SDN infrastructure and applications in production networks. This paper presents an analysis of the evolution of SDN in recent years. Additionally, this piece of work also describes some interesting SDN/Openflow research initiatives and applications. Finally, there is a disscussion on the main challenges of this new technology.**

***Index Terms*—Future Internet, Software Defined Networking, OpenFlow.**

I. Introduction

Network technologies have become in a critical element in almost all human activity. The number of devices as well as the amount of traffic moving in Internet is growing exponentially. However, the development of new protocols and services (*mobility, VoIP, QoS, video streaming*) has been limited by the hard union between specialized packet forwarding hardware and operating systems running hundreds of static protocols (some of them private) and only allowing the network administrator only the configuration of the network equipment. Additionally, the deployment time of a new idea from the design, simulation, testing, publication in a standard and finally the installation in network equipment can take some years. Clearly, the new generation networks need to change the rigidity of actual network architectures.

Software Defined Networking (SDN) is an architectural innovation that proposes the separation of *control plane* and the *data plane* enabling their independent evolution and development. Ethane [1] was one of the first SDN initiatives. This model was proposed for enterprise networks and uses a centralized control architecture. However, this implementation required the deployment of custom switches (OpenWrt, NetFPGA, Linux) to support Ethane Protocol.

Today, the principal crystallizing of SDN is Openflow [2], an open architecture initially developed as a way of allowing researchers to run experiments on heterogeneous networks without affecting the real users traffic. Openflow [2] aims at providing more configuration options in the switch dataplane.

The Openflow specification [3] establishes the rules of communication between *data plane* and a centralized *control plane* and allow the entire network to be controlled through users software applications (APIs). The Open Networking Foundation ONF [4] brings together about 90 companies and is dedicated to releasing, promoting and adopting of OpenFlow specification [3].

This implementation of Openflow [3] removes the limitation of rigidity of static protocols, opens the possibility of rapid innovation and led research community to develop new paradigms in network technologies. Some examples of this evolution are: Virtualization [5], [6], High Level Network Programming Languages [7], [8], optimization of Quality of Experience QoE applications [9], [10]. However, new challenges and unanswered questions appear when trying to implement these ideas in production networks: Modeling and Performance [11], Resilience and Recovery [12], Security [13], Convergence and Management [14], [15]. This paper presents an analysis of SDN/Openflow [2] technologies and discusses the challenges and open question in order to implement them in production networks.

The rest of the paper is outlined as follows: Section II defines the Openflow architecture. Then, Section III presents the actual research and experimentation. Next, Section IV analyzes the SDN challenges. Finally, Section V concludes with the discussion and conclusions.

II. OPENFLOW

The Openflow architecture [2] consists of three principal elements: an Openflow switch, an external controller and the Openflow Protocol [3] which establishes the communication between switch and controller through a secure channel.

An Openflow switch consist of one or more *flow tables* and a *group table* used for packet lookup and forwarding. Each *flow table* is composed of a set of match fields, counters and instructions. The packet fields to be compared with the match fields can be indistinct of the second, third or fourth layer in TCP /IP architecture. The number of the packet fields to be matched depends on the version of the Openflow protocol [4], [16]. The recent version of Openflow is v1.3 and specifies a number of 40 match fields including support to IPv6. However, the most widely used version is v1.0 with a number of 12 match fields.

The authors are with the Group of Analysis, Security and Systems (GASS), http://gass.ucm.es/en, Department of Software Engineering and Artificial Intelligence (DISIA), School of Computer Science, Office 431, Universidad Complutense de Madrid (UCM), Calle Profesor José García Santesmases s/n, Ciudad Universitaria, 28040 Madrid, Spain (e-mail: {angevald, lorebaro, javierv}@ucm.es).

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

If the header of an incoming packet is similar to a match field in a *flowtable*, the corresponding actions are taken. The Openflow Protocol [3], [4] specifies the set of mandatory and optional actions that a switch can take. The principal mandatory actions are: send the packet by a given port, send packet to the controller or drop the packet. In the case that incoming packet doesnt match with a match field, the switch can be configured to send it to the external controller or drop the packet. The external controller has the responsibility of analyzing the information received by switches and controlling their actions through the maintenance of the *flow tables* in switches.

III. RESEARCH AND EXPERIMENTATION

The direct manipulation of the forwarding plane on network devices allows researchers the possibility of designing new services, routing protocols, security models, and directly testing their functionality directly in the network. Next, there is a revision of important SDN research initiatives.

A. Virtualization

Similar to computer virtualization layer, where a hardware abstraction permits slicing and sharing the hardware resources with different Operating Systems OS in a host, the goal of network virtualization is to isolate multiple logical networks, each of them with completely different addressing and forwarding mechanism, but sharing the same physical infrastructure. SDN provides the opportunity to reach a network hardware abstraction layer.

Flowvisor [5] is a virtualization platform that uses Openflow [2] to sit logically between control and forwarding paths. Flowvisor [5] acts as a transparent proxy between controllers and switches. Then, it creates strong and transparent *virtual slices* according to administrators established policies and ensures isolation in terms of bandwidth, flowspace and switch CPU load. The user can only observe and control only their own slice. Additionally, it is possible to slice a *virtual slice* and create like hierarchies of virtualized networks (Fig.1).

In [6] there is a demonstration of four successful networking experiments (load balancer, wireless streaming video, traffic engineering and hardware prototyping experiment) using Flowvisor [5], each with its own slice. However, the experiments show some problems to be solved: the unexpected interaction with other deployed network devices, increase of broadcast traffic emitted from non-OpenFlow devices and some violations of the CPU isolation, especially when one slice inserts a forwarding rule that is handled via the switches slow path.

Another important aspect is the integration between network operations and computer virtualization. In computer virtualization, the different virtual machines VMs require a network access layer that provides inter- and intra-VM connectivity and provide similar network functions to the physical layer. The common model is to establish communication between virtual nodes and physical NIC implementing typical L2 switching or L3 routing. This makes the network management of virtual environments difficult, for example the migration of VMs between different physical servers. In this approach, SDN and network virtualization can help to achieve these goals.

Open vSwitch [17] is a software switch built for virtual environments. This switch exports an interface for fine-grained network control. Additionally, it has a logical partition of the forwarding plane through a flexible, table-based forwarding engine. The forwarding plane has an external interface and can be managed for example by Openflow [2]. With this abstraction, the controller can obtain a logical view of multiple Open vSwitches running on separate physical servers.

Another interesting application of virtualization is the *virtual network migration (VNM)*. In typical networks, the migration or change of a network node require the re-configuration and the re-synchronization of the routing protocol. This causes high delays and packet loss. Consequently, the use of virtual nodes can significantly reduce the downtime.

In the VNM system proposed in [18], the SDN controller creates new flow entries for the new switch and redirects the paths from the initial to the new node. Then, the controller deletes the flow entries of the old switch making it feasible to be removed with security. The results of experiments show a total migration time of 5ms without packet loss. Moreover, the system could be dynamically reconfigured to place virtual networks on different physical nodes according to the time of day or the traffic demand to save energy (green networks).

B. Simulation and Testbeds

The possibility of debugging and testing new designs in a real environment is difficult and expensive. The network simulation is the first step to evaluating the idea previous implementation in a real production network. However, the current network simulators are expensive and dont offer support to Openflow [2].

Mininet [19] is the first open source simulator for Openflow networks. It uses OS-level virtualization features, including processes and network namespaces and allows a scale to hundreds of nodes (switches, hosts, and controllers) in a simple laptop. The user can run commands on hosts, verify switch operation and even induce failures or adjust link connectivity. Another advantage is that the code used to program a controller in the simulator is the same as the one to be deployed into a real network. However, Mininet [19] presents some limitations. The performance is directly related to the number and topology of virtual nodes and the available resources in the host. For this reason, the results of experiments in terms of bandwidth or time can vary from one computer to another.

In case the industry or research community may need to test experiments in a scaled real network infrastructure, there is worldwide in development different large scale testbeds. Global Environment for Network Innovations GENI [20] is a research infrastructure sponsored by the US National Science Foundation. Currently GENI [20] is deploying a SDN/Openflow [2] technology at their infrastructure (8 campuses interconnected).

The project OpenFlow in Europe: Linking Infrastructure and Applications OFELIA [21] bring together different European research institutes within the European Commisions FP7 ICT Work Programme. This project interconnects an infrastructure of 8 Openflow [2] islands allowing experimentation on multi-layer and multi-technology networks and providing realistic test scenarios and diverse and scalable resources. The Extending and Deploying Ofelia in BRAzil EDOBRA proposal [22] has the purpose of extending the OFELIA network to Brazil as a new OFELIA island.

C. High Level Network Policy Languages

In computer systems, the Operating-Systems (OS) facilitate the development of programs through high level abstractions of hardware and information resources. In the SDN paradigm, the entire behavior of the network is managed by the control layer. In the Openflow architecture [2], the control layer is materialized in a centralized controller. Clearly, the controller needs a Network Operating System NOS that facilitates the user create software programs to control the behavior of the network. Currently, there are several NOS. The most used open source NOS are: NOX/POX [23], Maestro [24], Beacon [25], Floodlight [26] each with its particular characteristics. NOX and POX are similar but implemented in C++ and Python respectively. Additionally, they use a group of application programming interfaces (APIs) to communicate and interact with switches. Beacon [25] and Floodlight [26] are NOS based on Java platform. Maestro [24] use multithreading looking for better performance and scalability. In [27] these kinds of controllers are denominated as *southbound*, because they interact directly with the forwarding layer and manipulate the state of individual devices. However, programming High Level operational tasks using this kind of NOS is still difficult. The user need to mix different match fields, actions, prorities for each switch looking for the espected behavior of the network. Moreover, the synchronization between control and data messages requires high attention and complicates the creation of new applications. For this reason, the *northbound* interfaces automatically translates the network policies (policy layer) in coordinated rules to the switches and ensure a correct behaviour of the network.

Procera [8] is a control architecture that includes a declarative policy language based on the notion of *Functional Reactive Programing (FRP)* [28] which provides a declarative, expressive and compositional framework for describing reactive and temporal behaviors according to network conditions. It also has event comprehensions to manipulate event streams and signal functions of windowing and aggregation. In [27], [29] uCap project is presented using Procera [8]. uCap [29] enables home users to monitor and manage end-host devices data usage in their home network. When the device reaches the data capacity allocated by the administrator, the system can disable the connection of this device. This system is useful especially for subscribers of ISPs that enforce monthly bandwidth caps.

Frenetic [7] is a network programming language that comprises two integrated sublanguages: a *Declarative Network Query Language* and a *Network Policy Management Library*. The *Network Query Language* reads the state of the network by installing low-level rules on switches that are transparent to the programmer. The *Network Policy Management Library* manages the policy of the forwarding packets using a functional, reactive programming based on FRP [28]. A working prototype of Frenetic is available in [30].

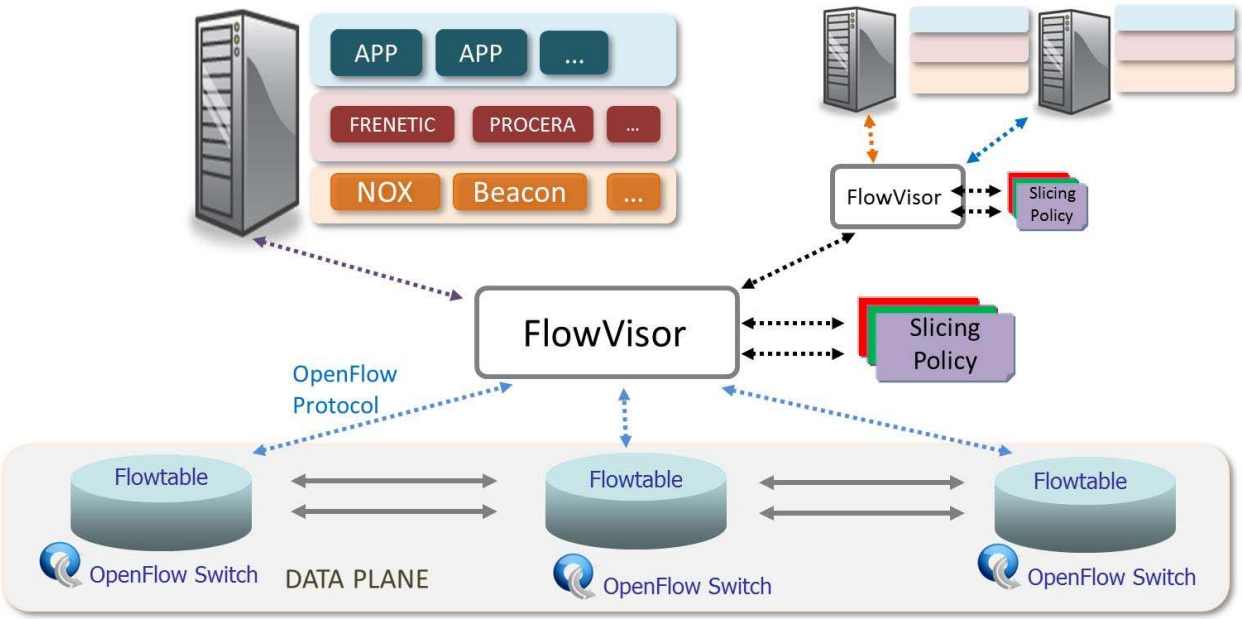


Fig. 1. Openflow Protocol, Virtualization and Network Operating Systems NOS

Another High Level Policy Language is Pyretic [31], which is designed to support modular programming. In other words, the programmer can write modules independently without interferance between them. Pyretic [31] establish different policy composition operators, including *parallel composition* and *sequential composition*. In *parallel composition* the policies are executed on the same packet and the results are combined. For example, send packet with header A to port 1 and packet with header B to port 2. In *sequential composition* the result of a policy A is the entry of the next policy B, such as the combination of balancing and switching policies (change the packet header and then routing based on the new header). The Fig.1 describes the location and function of the different abstraction levels (Openflow [2], Virtualization and NOS) within SDN paradigm.

D. Multimedia - Quality of Experience (QoE)

In multimedia systems, the notion of Quality of Experience QoE has growing since Quality of Service (QoS) is not powerful enough to express all features involved in a communication service. However, the term QoE doesnt yet have a solid, theoretical and practical concept.

The European Network on Quality of Experience in Multimedia Systems and Services (QUALINET [9]) emphasizes a working definition: *QoE is the degree of delight or annoyance of the user of an application or service. It results from the fulfillment of his or her expectations with respect to the utility and / or enjoyment of the application or service in the light of the users personality and current state*. Regardless, the use of this term QoE is increasing in the research community.

In [10], a SDN path optimization system for multimedia flows at the infrastructure layer improves QoE experience. The architecture consists of two main functions: QoS Matching and Optimization Function (QMOF) and the Path Assignment Function(PAF). QMOF reads the user multimedia parameters, determines whats feasible and looks for an optimal service configuration, while PAF maintains a network topology database. If the quality of the network is degraded, the system can react and dynamically modify the path parameters of the network prioritizing the users configuration.

Another important goal in multimedia communications and QoE is the capability to recovery a network path when a link fails (eg. *audio/video streaming*). In [32], a Openflow [2] based system is capable of recovering from a link failure using an alternative path. The system uses two well-known mechanisms: *restoration* and *protection*. In *restoration* method, when a path failure signal is received, the controller calculates and establishes an alternative path. In *protection* method, the controller anticipates a failure and calculates two disjoint paths (working and protected) before a failure occurs. The experiments show a recovery time of less than 50 ms for carrier-grade network.

IV. CHALLENGES

The deployment and experimentation of SDN/Openflow [2] initiatives have some important challenges at the moment to be successfully adopted at production networks. Below, there is a discussion of the most important aspects to considerate and the first proposed solutions.

A. Modeling and Performance

The implementation of SDN/Openflow requires the estimation of the number of controllers needed by a determinate topology and the localization of these controllers. This is a difficult question to answer. SDN establishes the separation of control and data planes, but doesnt necessarily order the existence of a single controller. SDN control plane may have single controller, a set of controllers in a hierarchy topology, or even any dynamic controllers topology. In [33], there is a complete study about this issue. The authors conclude that number and position of controllers depends on desired reaction bounds, metric choice and the network topology itself. The experiments show a decreased performance from each added controller, along with tradeoffs between metrics. However, the latency of a node connected with a single controller for medium sized network is similar to the response-time in actual production networks.

Another important factor is the selection of an appropriate NOS and the measure of its performance. In other words, how fast the controller responds to datapath request and how many datapath requests the controller can handle per second. However, it is also necessary to establish a balance between high performance and the productivity of developers (a developer friendly language). The experimentation [25], [34] show that the performance of the controller depends directly of programming language (C++, Java, Python) and the development environment.

The NOS previously named [23]–[26] are still in development and their features are constantly improved. For example, Fig. 2 and 3 show an evaluation of recently new Beacon improvements of performance in relation to other controllers. Tests were run on Amazons Elastic Computer Cloud using a Cluster Compute Eight Extra Large instance, containing 16 physical cores from 2 x Intel Xeon E5-2670 processors, 60.5GB of RAM, using a 64-bit Ubuntu 11.10 VM image (ami-4583572c) [25]. The experiment uses Cbench [35] utility for testing the throughput (number of transactions per second) and latency (time required for a packet to arrive at its destination through the network) of Openflow [2] controllers using a single thread.

Fig.2 shows the capacity of the different NOS handling constantly Packet In messages from 64 emulated switches as soon as possible. Fig.3 measures the average time between sending a single packet of an emulated switch and the reply from the controller.

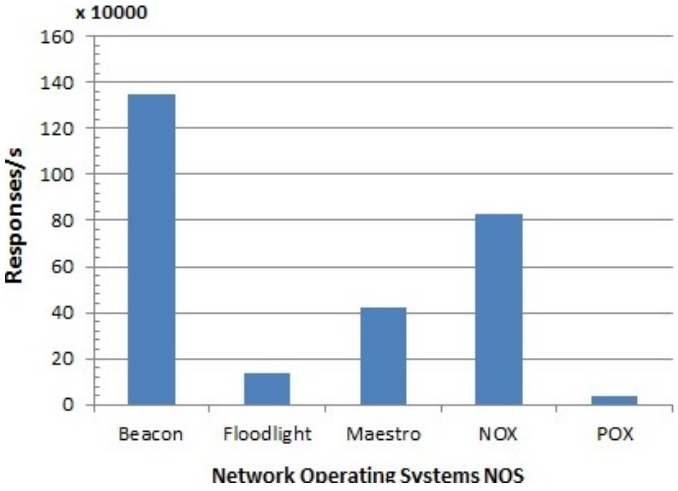


Fig. 2. Test of throughput of NOS [25]

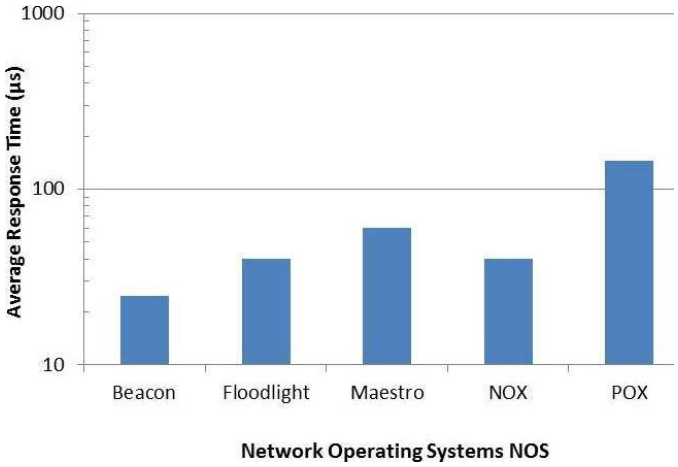


Fig. 3. Test of latency of NOS [25]

B. Resilience and Recovery

One of the problems related with the centralized control proposed by Openflow [2] is the vulnerability of the network. A failure of the controller can negatively compromise resilience of the whole network. Additionally, Openflow [2] establishes the configuration of one or more backup controllers, but doesnt provide a coordination mechanism between them.

CPRecovery component [12] is primary-backup mechanism that permits the replication between primary and secondary SDN controller in two phases: the *replication phase* and the *recovery phase*. The *replication phase* acts during normal functioning of the controller and send regular updates to the backup controller. The *recovery phase* acts during a failure state of the primary controller and starts the backup controller as a main controller. The failure state can be fired by abruptly aborting of a NOS process, an unexpected error of the application (API) or a DDoS attack. The results of experiments show a recovery time (from failure state to backup connection) of 800 ms.

C. Security

The current Internet architecture has huge problems to support, verify and enforce security properties. Typical Security Systems are based on securing hosts (e.g. antivirus) or installing specialized network devices (middleboxes) to detect anomalies into the network. However, these solutions become ineffective when the host is compromised (zero day vulnerabilities) or require constant updates and effort from the network administrator for example selecting the traffic to be filtered. In this point SDN can help to improve and develop new security models.

Pedigree [13] is a security system based on SDN for enterprise networks. This model uses Openflow [2] and a simple kernel OS-function on hosts to help the network controller to determine the provenance of the traffic network. Once the connection is validated, the controller enables and establishes the network path in switches. This system is designed with two components: the *tagger* and the *arbiter*. The *tagger* monitors all events in the OS and creates a tag for each process. When a process requires data sent through the network, a tagstream is sent to the controller. The controller has an *arbiter* function that checks the tags, verifies and orders an installment of the appropriate flowtables in switches. If the arbiter detects an unauthorized connection attempt, the controller orders a blockage of the flow of this link. Although, these processes generate additional load on the host and the network, the overhead is comparable to running an antivirus software.

D. Convergence and Management

SDN-Opeflow [2] architecture was originally developed for enterprise-campuses networks. However, to try and extend this architecture to large networks requires attend some issues, for example the problem of Interdomain Routing.

Today, the largest deployed protocol to communicate autonomous systems AS is the Border Gateway Protocol (BGP). BGP is a destination-based forwarding paradigm. In other words, the routing information between Autonomous Systems AS depends on the destination address in the IP packet header. An AS interchanges information and influence the traffic flow only with his direct neighboring AS. This makes it difficult to control of traffic flows along an entire path or in a remote part of the network. The task is to integrate SDN architecture to interconnect different AS.

In [15], an interdomain-routing solution based on BGP uses a NOX-Openflow [2] architecture. The new Inter-AS system proposes to extend two NOX components: *messenger* and the *switch*. *Messenger* creates a dedicated channel (port 2603) and enables the exchange of information between several NOX components. In other words, this *messenger channel* is used as a dependency of the Inter-AS component. The *Switch* adds a call to the Inter-AS component in case of an outside destination and selects the right datapath and port to forward the packet.

Another challenge to considerate is the integration with legacy networks. That is, networks with non-Openflow capacity (actual Internet infrastructure). It is noted that equipment could be upgraded or even replaced to support Openflow [2]. However, this change means costly network re-engineering. For this reason, the coordination between new SDN equipment and legacy infrastructure is necessary.

The project LegacyFlow [36] proposes a solution for coordination between Openflow [2] and traditional networks. The model introduces a new component between these architectures called *Legacy Datapath(LD)* that provides a bridge between different features of the legacy equipment. LD receives and interprets the messages sent by the Openflow controller and applies the corresponding actions in a real switch. Additionally, LegacyFlow [36] also proposes to add new actions to Openflow protocol [3] in order to improve the integration with traditional networks.

V. DISCUSSION AND CONCLUSIONS

This paper presents the state-of-the-art, the evolution and the challenges of Sofware Defined Networking in the last few years. Below, there is an exposition of some final comments and conclusions.

The separation of *control* and *data planes* opens the possibility to easily develop new services and network applications to industry and research community. Openflow [2] is a SDN technology based on the logic of match/action in *data plane* taking advantage of the actual network hardware capabilities. However, SDN can be expanded beyond match/action paradigm. For example, SDN can take into account several extensions, such as middleboxes or programmable custom packet processors.

This evolution could offer new services, for example on the fly encryption, transcoding, or traffic classification. Nevertheless, this requires a unifying control framework to allow coordination between the different types of network devices, as well as consensus and vendor support. At this point, the ONF [4] will take a decisive factor with the publication of new SDN protocols.

In *control plane*, the programming, debugging and testing are open issues in SDN. The High Level Languages facilitate the development of applications, but the composing and coupling of heterogeneous component are still difficult. For example, compose applications using NOX/POX [23] and Floodlight [26] or Beacon [25] simultaneously in the same controller is complicated today. The northbound policy languages also need to be constantly improved and tested to their implementation in production networks.

Finally, it is necessary to remark that Software Defined Networking is a tool. The research community can use this tool and develop new protocols, services, network applications taking advantage of the global view and the huge amount of information about the network.

ACKNOWLEDGMENT

This work was supported by the Agencia Española de Cooperación Internacional para el Desarrollo (AECID, Spain) through Acción Integrada MAEC-AECID MEDITERRÁNEO A1/037528/11.

Ángel Leonardo Valdivieso Caraguay and Lorena Isabel Barona López are supported by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT (Quito, Ecuador) under Convocatoria Abierta 2012 Scholarship Program N. 2543-2012. The authors would also like to thank Ana Lucila Sandoval Orozco for her valuable comments and suggestions to improve the quality of the paper.

REFERENCES

[1] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, (New York, NY, USA), pp. 1–12, ACM, August 2007.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 69–74, March 2008.

[3] "OpenFlow Switch Specification v.1.1.0," pp. 1–56, 2011.

[4] "Open Networking Foundation." https://www.opennetworking.org/.

[5] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A Network Virtualization Layer," tech. rep., OpenFlow Switch Consortium, October 2009.

[6] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the Production Network be the Testbed?," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, (Berkeley, CA, USA), pp. 1–6, USENIX Association, October 2010.

[7] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *Proceedings of the The 16th ACM SIGPLAN International Conference on Functional Programming*, (New York, NY, USA), pp. 279–291, ACM, September 2011.

[8] A. Voellmy, H. Kim, and N. Feamster, "Procera: A Language for High-Level Reactive Network Control," in *Proceedings of the First Workshop on Hot topics in Software Defined Networks*, (New York, NY, USA), pp. 43–48, ACM, August 2012.

[9] S. M. Patrick Le Callet and A. Perkis, "Qualinet White Paper on Definitions of Quality of Experience," *European Network on Quality of Experience in Multimedia Systems and Services (COST Action IC 1003)*, March 2012.

[10] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking," in *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks*, vol. 1, pp. 1–5, IEEE, September 2012.

[11] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *Proceedings of the 23rd International Teletraffic Congress*, pp. 1–7, ITCP, September 2011.

[12] P. Fonseca, R. Bennesby, E. Mota, and A. Passito, "A Replication Component for Resilient OpenFlow-based Networking," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 933–939, IEEE, April 2012.

[13] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, "Securing Enterprise Networks Using Traffic Tainting," tech. rep., August 2009.

[14] S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew, and L. Ong, "Packet and Circuit Network Convergence with OpenFlow," in *2010 Conference on Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference (OFC/NFOEC)*, pp. 1–3, IEEE, March 2010.

[15] R. Bennesby, P. Fonseca, E. Mota, and A. Passito, "An Inter-AS Routing Component for Software-Defined Networks," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 138–145, IEEE, April 2012.

[16] F. de Oliveira Silva, J. de Souza Pereira, P. Rosa, and S. Kofuji, "Enabling Future Internet Architecture Research and Experimentation by Using Software Defined Networking," *European Workshop on Software Defined Networking*, vol. 0, pp. 73–78, October 2012.

[17] B. Pfaff, J. Pettit, K. Amidon, and M. Casado, "Extending Networking into the Virtualization Layer," in *Proceedings of ACM SIGCOMM HotNets*, ACM, October 2009.

[18] P. S. Pisa, N. C. Fernandes, H. E. Carvalho, M. D. Moreira, M. E. M. Campista, L. H. M. Costa, and O. C. M. Duarte, "OpenFlow and Xen-Based Virtual Network Migration," in *Communications: Wireless in Developing Countries and Networks of the Future*, vol. 327, pp. 170–181, Springer Berlin Heidelberg, September 2010.

[19] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for. Software-Defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, (New York, NY, USA), pp. 19:1–19:6, ACM, October 2010.

[20] C. Elliott, "GENI: Opening Up New Classes of Experiments in Global Networking," *IEEE internet computing*, vol. 14, pp. 5–10, January 2010.

[21] "OFELIA." http://www.fp7-ofelia.eu/.

[22] C. Guimarães, "Extending and Deploying Ofelia in BRAzil (EDOBRA)," February 2013.

[23] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 105–110, July 2008.

[24] T. S. E. N. Zheng Cai, Alan L. Cox, "Maestro: A system for scalable openflow control," tech. rep., December 2010.

[25] D. Erickson, "The Beacon Openflow Controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot topics in Software Defined Networking*, (New York, NY, USA), pp. 13–18, ACM, August 2013.

[26] "Floodlight." http://www.projectfloodlight.org/.

[27] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.

[28] Z. Wan and P. Hudak, "Functional Reactive Programming from First Principles," in *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, (New York, NY, USA), pp. 242–252, ACM, May 2000.

[29] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards, "Communicating with caps: Managing Usage Caps in Home Networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, (New York, NY, USA), pp. 470–471, ACM, August 2011.

[30] "Frenetic." https://github.com/frenetic-lang/frenetic.

[31] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing Software Defined Networks," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, (Berkeley, CA, USA), pp. 1–14, USENIX Association, April 2013.

[32] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "A Demonstration of Fast Failure Recovery in Software Defined Networking," in *Testbeds and Research Infrastructure. Development of Networks and Communities*, vol. 44, pp. 411–414, Springer Berlin Heidelberg, June 2012.

[33] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement problem," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, (New York, NY, USA), pp. 7–12, ACM, August 2012.

[34] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, (Berkeley, CA, USA), pp. 10–10, USENIX Association, April 2012.

[35] "Cbench." http://archive.openflow.org/wk/index.php/Oflops.

[36] F. Farias, J. Salvatti, E. Cerqueira, and A. Abelem, "A Proposal Management of the Legacy Network Environment Using Openflow Control Plane," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium*, pp. 1143–1150, IEEE, April 2012.

Ángel Leonardo Valdivieso Caraguay was born in Loja, Ecuador, in 1985. He received a B.S. degree in Electronics and Telecommunications Engineering from the Escuela Politécnica Nacional, Quito, Ecuador in 2009 and a M.S. degree in Information Technology from the University of Applied Sciences Hochschule Mannheim, Germany in 2012. He is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. His research interests include computer networks and software-defined networking.

Lorena Isabel Barona López was born in Ambato, Ecuador, in 1985. She received a B.S. degree in Electronic and Information Networks Engineering from the Escuela Politécnica Nacional, Quito, Ecuador in 2010 and a M.S. degree in Telecommunications Engineering from the Universidad Politécnica de Madrid, Spain in 2013. She is currently a Ph.D. student of Computer Engineering in Universidad Complutense de Madrid, Spain. Her research interests include computer communication and software-defined networking.

Luis Javier García Villalba (SM04) was born in Madrid, Spain, in 1969. He received the Telecommunication Engineering degree from the Universidad de Málaga, Spain, in 1993, the M.Sc. degree in computer networks in 1996, and the Ph.D. degree in computer science in 1999, the last two from the Universidad Politécnica de Madrid, Spain. He was a Visiting Scholar at the Research Group Computer Security and Industrial Cryptography (COSIC), Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium, in 2000, and a Visiting Scientist at the IBM Research Division (IBM Almaden Research Center, San Jose, CA) in 2001 and in 2002. He is currently an Associate Professor in the Department of Software Engineering and Artificial Intelligence at the Universidad Complutense de Madrid (UCM) and Head of the Complutense Research Group GASS (Group of Analysis, Security and Systems), which is located at the School of Computer Science at the UCM Campus. His professional experience includes research projects with Hitachi, IBM, Nokia, Safelayer Secure Communications, and VISA. His main research interests are in information security, computer networks, and software-defined networking. Dr. Garca Villalba is an Associate Editor in Computing for IEEE Latin America Transactions since 2004.

SDN Security: A Survey

Sandra Scott-Hayward, Gemma O'Callaghan and Sakir Sezer
Centre for Secure Information Technology (CSIT)
Queen's University Belfast
Belfast, BT3 9DT, Northern Ireland

***ABSTRACT*—The pull of Software-Defined Networking (SDN) is magnetic. There are few in the networking community who have escaped its impact. As the benefits of network visibility and network device programmability are discussed, the question could be asked as to who exactly will benefit? Will it be the network operator or will it, in fact, be the network intruder? As SDN devices and systems hit the market, security in SDN must be raised on the agenda. This paper presents a comprehensive survey of the research relating to security in software-defined networking that has been carried out to date. Both the security enhancements to be derived from using the SDN framework and the security challenges introduced by the framework are discussed. By categorizing the existing work, a set of conclusions and proposals for future research directions are presented.**

I. Introduction

Software-defined networking (SDN) is rapidly moving from vision to reality with a host of SDN-enabled devices in development and production. The combination of separated control and data plane functionality and programmability in the network, which have long been discussed in the research world, have found their commercial application in cloud computing and virtualization technologies.

The advantages of SDN in various scenarios (e.g. the enterprise, the datacenter etc.) and across various backbone networks have already been proven e.g. Google B4 [1]. However, challenges exist for a full-scale carrier network implementation of SDN. A number of these challenges have been presented in [2]. One key area, which is only beginning to receive the attention it deserves, is that of security in SDN.

The SDN architecture can be exploited to enhance network security with the provision of a highly reactive security monitoring, analysis and response system. The central controller is key to this system. Traffic analysis or anomaly-detection methods deployed in the network generate security-related data, which can be regularly transferred to the central controller. Applications can be run at the controller to analyze and correlate this feedback from the complete network. Based on the analysis, new or updated security policy can be propagated across the network in the form of flow rules. This consolidated approach can efficiently speed up the control and containment of network security threats.

However, the same attributes of centralized control and programmability associated with the SDN platform introduce network security challenges. An increased potential for Denial-of-Service (DoS) attacks due to the centralized controller and flow-table limitation in network devices is a prime example. Another issue of concern based on open programmability of the network is trust; both between applications and controllers, and controllers and network devices.

A number of solutions to these SDN security challenges have been proposed in the literature. These range from controller replication schemes through policy conflict resolution to authentication mechanisms. Similarly, a number of proposals have been made to exploit the SDN framework for enhanced network security.

An analysis of the security challenges of SDN is presented in this paper. The individual security issues are categorized according to the SDN layer affected or targeted. The proposed and emerging solutions to these challenges are then discussed and categorized. The requirement for further work to establish a secure and robust SDN is clearly identified from the gap between the issues and the existing research. Without a significant increase in focus on security, it will not be possible for SDN to support the evolving capability associated with, for example, Network Functions Virtualization (NFV) [3].

II. Security Analyses of SDN

The basic properties of a secure communications network are: confidentiality, integrity, availability of information, authentication and non-repudiation [4]. In order to provide a network protected from malicious attack or unintentional damage, security professionals must secure the data, the network assets (e.g. devices) and the communication transactions across the network. The alterations to the network architecture introduced by SDN must be assessed to ensure that network security is sustained.

In an early iteration of what is known today as SDN, Casado et al. [5] specifically considered the security aspects of a separate control and forwarding framework. Their SANE architecture, proposed in 2006, centred on

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

a logically centralized controller responsible for authentication of hosts and policy enforcement. At the time of its proposal, this was considered to be an extreme approach that would require a radical change to the networking infrastructure and end-hosts, which could be too restrictive for some enterprises.

Ethane [6] extended the work of SANE but used an approach, which required less alteration to the original network. It controlled the network through the use of two components; a centralized controller responsible for enforcing global policy, and ethane switches, which simply forwarded packets based on rules in a flow table. This simplified network control allowed the data and control plane to be separated to allow for more programmability. Although the Ethane architecture gave us a closer look at what SDN and OpenFlow would become, it suffered from a number of drawbacks. One of these is the fact that application traffic could compromise network policy. In today's SDN architecture, applications are used to provide various services, as, for example, with Network Functions Virtualization (NFV). The compromise of applications could potentially breach the entire network.

Considering the specific issues with security in SDN from the perspective of the SDN framework (Fig. 1), we can identify challenges associated with each layer of the framework: application, control and data planes, and on the interfaces between these layers.

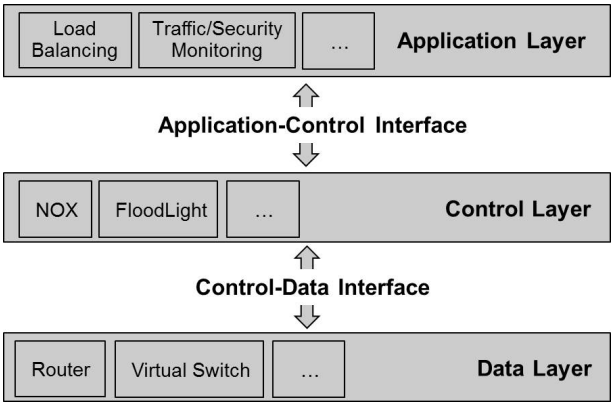


Fig. 1. SDN Functional Architecture illustrating the data, control and application layers and interfaces

A number of security analyses have recently been performed, which have found that the altered elements or relationship between elements in the SDN framework introduce new vulnerabilities, which were not present before SDN. One such paper [7] completes an analysis of the OpenFlow protocol using the STRIDE threat analysis methodology [8]. This paper focuses on the execution of Information Disclosure and DoS attacks, which the author established were possible to successfully execute. Although a number of mitigation techniques are proposed, these techniques are not proven in the work.

The OpenFlow switch specification [9] describes the use of transport layer security (TLS) with mutual authentication between the controllers and their switches. However, the security feature is optional, and the standard of TLS is not specified. The lack of TLS adoption by major vendors and the possibility of DoS attacks are the focus of an OpenFlow vulnerability assessment [10]. The authors found that the lack of TLS use could lead to fraudulent rule insertion and rule modification.

In [11] Kreutz et al. present a high-level analysis of the overall security of SDN. They conclude that due to the nature of the centralized controller and the programmability of the network, new threats are introduced requiring new responses. They propose a number of techniques in order to address the various threats, including replication, diversity and secure components.

Finally, the research network and testbed, ProtoGENI, has also been analyzed [12]. The authors discovered that numerous attacks between users of the testbed along with malicious propagation and flooding attacks to the wider internet were possible when using the ProtoGENI network.

The results of these analyses indicate the range of the security issues associated with the SDN framework. In Table I, a categorization of the SDN security issues is presented. A connection is drawn between the type of issue/attack (e.g. unauthorized access) and the SDN layer/interface affected by the issue/attack.

The control and data layers are identified in Table I as clear targets of attack. This reflects the main distinctions between the traditional network and the SDN; that of the centralized control element and the altered datapath elements to support programmability.

Although this analysis points towards security issues related to the control and data layers, there has been limited research in the field to tackle the challenges. In fact, as detailed in the next section, greater attention has been given to exploring the potential improvements in network security to be derived from the SDN framework.

III. Security Enhancement using SDN

The architecture of a software-defined network introduces potential for innovation in the use of the network. The combination of the global or network-wide view and the network programmability supports a process of harvesting intelligence from existing Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS), for example, followed by analysis and centralized reprogramming of the network. This approach can render the SDN more robust to malicious attack than traditional networks.

TABLE I
CATEGORIZATION OF THE SECURITY ISSUES ASSOCIATED WITH THE SDN FRAMEWORK BY LAYER/INTERFACE AFFECTED

Security Issue/Attack	SDN Layer Affected or Targeted				
	Application Layer	App-Ctl Interface	Control Layer	Ctl-Data Interface	Data Layer
Unauthorized Access e.g.					
Unauthorized Controller Access			✓	✓	✓
Unauthenticated Application	✓	✓	✓		
Data Leakage e.g.					
Flow Rule Discovery (Side Channel Attack on Input Buffer)					✓
Forwarding Policy Discovery (Packet Processing Timing Analysis)					✓
Data Modification e.g.					
Flow Rule Modification to Modify Packets			✓	✓	✓
Malicious Applications e.g.					
Fraudulent Rule Insertion	✓	✓	✓		
Controller Hijacking			✓	✓	✓
Denial of Service e.g.					
Controller-Switch Communication Flood			✓	✓	✓
Switch Flow Table Flooding					✓
Configuration Issues e.g.					
Lack of TLS (or other Authentication Technique) Adoption			✓	✓	✓
Policy Enforcement	✓	✓	✓		

A. The SDN Middle-box

Traditional networks use middle-boxes to provide network security functions. Recently, there has been discussion about the integration of security middle-boxes into SDN exploiting the benefit of programmability to redirect selected network traffic through the middle-box. For example, the Slick architecture [13] proposes a centralized controller, which is responsible for installing and migrating functions onto custom middle-boxes. Applications can then direct the Slick controller to install the necessary functions for routing particular flows based on security requirements.

The FlowTags architecture [14] proposes the use of minimally modified middle-boxes, which interact with a SDN controller through a FlowTags Application Programming Interface (API). FlowTags, consisting of traffic flow information, are embedded in packet headers to provide flow tracking and enable controlled routing of tagged packets. A clear disadvantage of this architecture is the fact that it works with only pre-defined policies and currently does not handle dynamic actions.

The SIMPLE policy enforcement layer [15] is an approach for using SDN to manage middlebox deployments. In contrast to [13], [14], it requires no modifications to SDN capabilities or middle-box functionality, which makes it suitable for legacy systems.

Based on these proposals, it would appear that a simple approach to network security provision would be to introduce an appropriate middle-box and programme the network to direct selected traffic through the middle-box. It is not, however, quite as straightforward as that. The appropriate placement and integration of SDN middle-boxes must be determined along with the performance penalty that can be tolerated when traffic is diverted through an additional link. Such questions have not yet been resolved.

However, as illustrated in Table I, the range of attacks that pose threats to the network is well understood. As such, beyond middle-boxes, a series of solutions have been proposed, which specifically exploit the SDN framework to provide network security solutions.

B. SDN = "Security Defined Networking"?

Attackers use various scanning techniques to discover vulnerable targets in the network. One defense presented to thwart these attacks is the use of random virtual Internet Protocol (IP) addresses using SDN [16]. This technique uses the OpenFlow controller to manage a pool of virtual IP addresses, which are assigned to hosts within the network, hiding the real IP addresses from the outside world. This presents moving target defense, which is a form of adaptive cybersecurity.

Monitoring Systems are essential in protecting the network from attack. In [17], the authors present a Distributed DoS (DDoS) detection method based on several traffic flow features. This system monitors NOX (C++ based OpenFlow Controller) switches at regular intervals and uses Self Organizing Maps to identify abnormal flows. In another approach, OpenSAFE [18] uses its ALARMS policy language to manage the routing of traffic through network monitoring devices. A similar idea focusing on SDN in the cloud was presented by Shin and Gu in [19]. CloudWatcher controls network flows to guarantee that all necessary network packets are inspected by some security devices. This framework automatically detours network packets to be inspected by pre-installed network security devices.

These solutions are based on a centralized network management scheme; however other work encourages the delegation of some control back to network devices and hosts. Resonance, for example, [20], provides dynamic access control enforced by network devices themselves based on higher-level security policies. Naous et al. [21] put forward the ident++ protocol to query end-hosts and users for additional information in order to make forwarding decisions; their argument being that the central controller could become a bottleneck. While retaining the programmability characteristic of SDN, these methods propose to involve the network devices in the control of the network, rather than relying on a single, centralized controller.

One specific form of monitoring system, the IDS, has been the focus of a number of SDN solutions. Skowyra et al. [22] propose a learning IDS, which utilizes the SDN architecture to both detect and respond to network attacks in embedded mobile devices. A hardware-accelerated NIDS (Network IDS) or NIPS (Network IPS) scheme, as described in [23], allows the network administrator to configure string patterns for use by a deep packet inspection (DPI) module. Finally, the value of using SDN to provide intrusion detection in a Home Office/Small Office environment is proposed in [24].

The possibility for improving and simplifying network security by means of the SDN architecture is evident from this body of research. This potential has also been recognised commercially with a range of SDN security products at various stages of development.

IV. Security Challenges with SDN

While security as an advantage of the SDN framework has been recognized, solutions to tackle the challenges of securing the SDN network are fewer in number.

SDNs provide us with the ability to easily program the network and to allow for the creation of dynamic flow policies. It is, in fact, this advantage that may also lead to security vulnerabilities. Within this dynamic environment, it is vital that network security policy is enforced. Model-checking becomes an important step in detecting inconsistencies in policies from multiple applications or installed across multiple devices. Model checking combined with symbolic execution may be used to test OpenFlow applications for correctness [25]. Binary Decision Diagrams can also be used to test for intra-switch misconfigurations within a single flow table [26]. FlowChecker exploits FlowVisor [27], which enables isolation by partitioning the network resources into slices. Son et al. propose Flover [28], which uses assertion sets and modulo theories to verify flow policies, while VeriFlow [29] studies the verification of invariants in real-time. An additional layer, which sits between the SDN controller and the network devices, intercepts flow rules before they reach the network. Although VeriFlow boasts low-latency of the checking process, it cannot handle multiple controllers. In [30], the authors propose the use of language-based security to enable flow-based policy enforcement along with network isolation. This solution is implemented as a NOX application and allows the integration of external authentication sources to provide access control. More recently, Splendid Isolation [31] has been proposed as a means of verifying the isolation of program traffic. This programming model supports the idea of network slices to provide the fundamental security concepts of confidentiality and integrity. There is a clear emphasis from the research community on this issue of policy conflict resolution.

However, proposals to aid in the design of secure SDNs are limited. Fresco [32] is one notable contribution; which presents an OpenFlow Security Application Development Framework incorporating FortNox [33]; a security enforcement kernel. The idea behind FRESCO is to allow the rapid design and development of security specific modules, which can be incorporated as an OpenFlow application. Porras et al. provide a library of reusable modules which can be used for the detection and mitigation of network threats. This system incorporates the FortNox enforcement engine, which handles possible conflicts with rule insertion. If a rule conflict arises as a result of a new OpenFlow rule enabling or disabling a prohibited/allowed existing rule, then the new rule is accepted or rejected depending on the level of security authorization of the author to the existing conflicting rule provider. Although FortNox provides numerous components, which are necessary for enforcing security, the authors feel that much work is still needed to offer a comprehensive suite of applications.

Moving from the design space to implementation, one of the key industry concerns with security in SDN is satisfaction of the audit process. For network compliance and operation, a controlled inventory of network devices is required. This involves knowledge of what devices

TABLE II
CATEGORIZATION OF THE RESEARCH ON SECURITY IN SDN

Research	Security			OpenFlow	SDN Layer/Interface				
Work	Analysis	Enhancement	Solution		App	App-Ctl	Ctl	Ctl-Data	Data
[7], [10], [12]	✓			✓			✓	✓	✓
[11]	✓				✓		✓	✓	✓
[5]							✓	✓	✓
[13], [14], [21]		✓		✓	✓	✓	✓	✓	✓
[15]		✓			✓		✓		✓
[16]		✓		✓			✓	✓	✓
[17], [24]		✓		✓	✓		✓	✓	
[18], [19]		✓		✓	✓	✓	✓	✓	
[20], [22]		✓		✓	✓		✓	✓	✓
[23]		✓			✓				✓
[25]			✓	✓	✓	✓		✓	
[26], [28]–[30], [32]			✓	✓	✓	✓	✓	✓	
[31]			✓			✓	✓		
[33], [34]			✓	✓		✓	✓	✓	
[35]			✓	✓	✓			✓	

are running, how they are bound to the network etc. This directly concerns the potential for virtualization of network elements and functions as supported by the SDN framework. Although there is an unresolved challenge regarding the feasibility of mapping network state across mobile and virtual functions, some related work regarding network verification is worth mentioning. In [34], the authors consider the problem of scalability and security of OpenFlow networks and their use in the cyber-physical space. Verificare allows for specification modeling and verification of network correctness, convergence and mobility-related properties. Hadigol et al. propose the use of a prototype network debugger [35], which could be used to allow SDN developers to reconstruct the chain of events which lead to a bug and identify its root cause.

As identified in Section II, the SDN architecture can be considered as a set of layers and interfaces. The layer/interface affected by some of the SDN-specific security issues was identified in Table I. In a similar manner, the SDN security research work is classified in Table II by the layer/interface, which the analysis, enhancement or solution targets. The results of this categorization are discussed in the next section. It can be noted that SANE [5] is included in Table II for categorization with respect to affected layers/interfaces. However, as a separate architecture, it is not identified as an SDN security enhancement or solution.

V. DISCUSSION

Considering the categorization of research work in Table II, it can be seen that there has been greater focus on exploiting SDN for enhanced network security than on generating solutions to the identifed security issues. The enhancement work has centred on the use of middle-boxes and monitoring systems for security service insertion to dynamically detect and/or prevent suspicious traffic during live network operation.

There is further potential in this area to exploit the dynamic and adaptive capabilities of the SDN framework using methods of moving target defense. The work presented in [16] is one such example where randomizing the virtual IP addresses makes it more difficult for an attacker to breach the network. Without a fixed system to observe and prepare to attack, the strength of the attacker is reduced.

New methods and techniques must be explored to expand on the programmability of the network enabling dynamic adjustments in security monitoring, detection and prevention capabilities.

A minor observation from the content of Table II is that the majority of the work references or implements OpenFlow for the control-data interface. Although any alternative to OpenFlow would have similar attributes, it is worth noting that OpenFlow may not be the only/definitive control-data interface protocol in SDNs. For example, several Internet Engineering Task Force (IETF) groups have defined protocols regarding separation of forwarding and control planes, network configuration and routing. These include IETF ForCES (Forwarding and Control Element Separation), PCE (Path Computation Element), Netconf (Network Configuration), LISP (Locator/ID Separation Protocol)

and I2RS (Interface to the Routing System). In addition, proprietary protocols are being developed by individual companies. The work to identify and correct security-related limitations of the OpenFlow protocol should be considered in the design and development of alternative protocols. This could apply both to the control-data plane interface and also to the higher-level abstractions at the application-control interface, which may present similar concerns.

The most significant element to highlight from the categorization of security-related SDN research is that there is an identifiable disconnect between the security analyses presented to date, which focus on the control-data plane issues, and the solutions to security issues, the majority of which focus on one application-control plane issue; that of policy conflict resolution.

Considering the breadth of potential security issues outlined in Table I, it is clear that a significant increase in effort is required to identify solutions to these challenges.

This requirement has been recognised in the past year in some areas of the networking community. Since the beginning of 2013, various working groups have been established in both the standardization industry and industry research groups. In the Open Networking Foundation (ONF) and the European Telecommunications Standards Institute (ETSI), groups focussed specifically on security in SDN and NFV, respectively, have been launched. In the Internet Research Task Force (IRTF) and the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), general SDN study groups have been launched in which security in SDN is an identified issue.

One of the recurring themes from these industry working groups is the importance of designing security in from the start. By this, it is meant that while SDN is in the early stages of development, the associated security issues should be identified and resolved. However, SDN-compliant hardware, software and services are already in production and in service. While some of these solutions are, in fact, SDN security products, many others have been developed with little or no consideration of the security implications of a wide area network deployment.

It is, therefore, essential, that techniques, methods and policies to overcome the SDN security challenges are explored and defined to enable robust and reliable wide area SDN deployments. An increased emphasis on this now could avoid a reduction in the performance and capability of future SDNs as a result of retrofit security solutions.

VI. Conclusion

There are two schools of thought on security in software-defined networking. The first is that significant improvements in network security can be achieved by simultaneously exploiting the programmability and the centralized network view introduced by SDN. The second is that these same two SDN attributes expose the network to a range of new attacks. In this article, we have categorized the SDN security challenges and presented a comprehensive review of the research work on security in SDN to date. Our analysis identifies that regardless of your school of thought, there is yet more to be done; more untapped potential and more unresolved challenges. A concerted effort in both directions could yield a truly secure and reliable Software-Defined Network.

References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, and M. Zhu, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 conference*. ACM, 2013, pp. 3–14.

[2] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, 2013.

[3] "Network Functions Virtualization - Introductory White Paper," October, 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[4] C. Douligeris and D. N. Serpanos, *Network security: current status and future directions*. Wiley. com, 2007.

[5] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks," in *USENIX Security Symposium*, 2006.

[6] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[7] R. Kloeti, "OpenFlow: A Security Analysis," April 2013. [Online]. Available: ftp://yosemite.ee.ethz.ch/pub/students/2012-HS/MA-2012-20_signed.pdf

[8] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Threat modeling-uncover security design flaws using the stride approach," *MSDN Magazine-Louisville*, pp. 68–75, 2006.

[9] "OpenFlow Switch Specification Version 1.3.2," Open Networking Foundation. [Online]. Available: https://www.opennetworking.org

[10] K. Benton, L. J. Camp, and C. Small, "OpenFlow Vulnerability Assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 151–152.

[11] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55–60.

[12] D. Li, X. Hong, and J. Bowman, "Evaluation of Security Vulnerabilities by Using ProtoGENI as a Launchpad," in *Global Telecommunications Conference (GLOBECOM 2011)*. IEEE, 2011, pp. 1–6.

[13] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A Slick Control Plane for Network Middleboxes," *Open Networking Summit*, 2013. [Online]. Available: http://nextstep-esolutions.com/Clients/ONS2.0/pdf/2013/research_track/poster_papers/final/ons2013-final51.pdf

[14] S. Fayazbakhsh, V. Sekar, M. Yu, and J. Mogul, "FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions," in *Proceedings of the second workshop on Hot topics in software defined networks*. ACM, 2013.

[15] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN." ACM SIGCOMM, August 2013.

[16] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 127–132.

[17] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *IEEE 35th Conference on Local Computer Networks (LCN)*. IEEE, 2010, pp. 408–415.

[18] J. R. Ballard, I. Rae, and A. Akella, "Extensible and scalable network monitoring using opensafe," *Proc.INM/WREN*, 2010.

[19] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *20th IEEE International Conference on Network Protocols (ICNP)*. IEEE, 2012, pp. 1–6.

[20] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 11–18.

[21] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich, "Delegating network security with more information," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 19–26.

[22] R. Skowyra, S. Bahargam, and A. Bestavros, "Software-Defined IDS for Securing Embedded Mobile Devices," 2013. [Online]. Available: http://www.cs.bu.edu/techreports/pdf/2013-005-software-defined-ids.pdf

[23] A. Goodney, S. Narayan, V. Bhandwalkar, and Y. H. Cho, "Pattern Based Packet Filtering using NetFPGA in DETER Infrastructure." [Online]. Available: http://fif.kr/AsiaNetFPGAws/paper/2-2.pdf

[24] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Recent Advances in Intrusion Detection*. Springer, 2011, pp. 161–180.

[25] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[26] E. Al-Shaer and S. Al-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*. ACM, 2010, pp. 37–44.

[27] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech.Rep*, 2009.

[28] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model Checking Invariant Security Properties in OpenFlow." [Online]. Available: http://faculty.cse.tamu.edu/guofei/paper/Flover-ICC13.pdf

[29] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.

[30] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker, "Expressing and enforcing flow-based network security policies," *University of Chicago, Tech.Rep*, 2008.

[31] C. Schlesinger, A. Story, S. Gutz, N. Foster, and D. Walker, "Splendid isolation: Language-based security for software-defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 79–84.

[32] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular composable security services for software-defined networks," in *Proceedings of Network and Distributed Security Symposium*, 2013.

[33] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 121–126.

[34] R. W. Skowyra, A. Lapets, A. Bestavros, and A. Kfoury, "Verifiably-safe software-defined networks for CPS," in *Proceedings of the 2nd ACM international conference on High confidence networked systems*. ACM, 2013, pp. 101–110.

[35] N. Handigol, B. Heller, V. Jeyakumar, D. Mazires, and N. McKeown, "Where is the debugger for my software-defined network?" in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 55–60.

Tearing down the Protocol Wall with Software Defined Networking

Yitzhak Bar-Geva (*Author*)

Independent
Jerusalem, Israel
Yitzhak.BarGeva.IL@ieee.org

Jon Crowcroft/Helen Oliver (*Authors*)
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
jon.crowcroft@cl.cam.ac.uk/helen.oliver@cl.cam.ac.uk

***Abstract*— The letter of the law of Software Defined Networking (SDN) is the OpenFlow specification, combined with appropriate controllers. The spirit of the law of SDN affords much more potential for innovation. We take the position that the future of communications is at stake unless the Protocol Wall is overcome. By "Protocol Wall" we mean the artificial barrier which, until recently, has been placed between the inflexible, protocol-bound communications space, and the application space in which far fewer constraints are placed on the programmer and in which far more innovative results can be achieved. Overcoming the Protocol Wall allows fully programmable, software-defined behaviour in both the data plane and the control plane, and opens the way for much-needed innovations in networking.**

In the stereotypical case, of which sockets are an example, the application functionality and the physical layer are separated between kernel and user respectively, with the kernel below the transport service layer, and the application above it. Archetypically, the router and host are separated between Internet Protocol (IP) and Transmission Control Protocol (TCP) respectively, where IP is responsible for hop-by-hop forwarding and TCP for end-to-end services. The Protocol Wall between application functionality and the physical layer constitutes a hegemony over innovation which is not being challenged sufficiently by current SDN approaches.

We assert that the realisation of unconstrained Software Defined Networking is closer at hand than anyone has yet acknowledged, and that networking can be evolved more quickly if we are guided by the spirit of the law of SDN. User mode network software, running on today's commodity and specialised communications hardware, can safely support satisfactory performance, while opening up potential for the same very high rate of innovation in networking as has been seen in appliances – for example, in smart mobile devices – in recent years.

Keywords—Software Defined Networking; protocol; networking; Future Networking Architectures; Future Internet Architectures

I. INTRODUCTION

Today, the state of the expanding Internet is reminiscent of the state of mainframe computing just before Personal Computing (PC) came to overshadow it. In our vision for the future, we see the Internet as a diminishing tax on specialised protocol-centric components, within a vastly larger, generally programmable networking/computing ecosystem.

We see protocols as artefacts of a bygone age, and as the wall blocking innovation. If networks were fully programmable, Future Internet Architectures (FIAs) would evolve naturally, and the formal process of universally accepted conventions would fall to entropy in the networking world, as they historically have in the software world.

Now, when community networks are beginning to be realised as a new model for the Internet [1], is the time to enable innovation by tearing down the Protocol Wall that stands in the way. Tearing down the Protocol Wall that divides programmable software from rigid, protocol-bound systems, would open the road to unlimited Future Networking Architectures (FNAs) without suffering "Clean Slate rupture" as Prof. Shenker envisioned in [2], ushering in an era of networking for the masses, comparable to the birth of computing for the masses of 40 years ago. The technology for truly unconstrained Software Defined Networking is already at hand. Implementation is conceptually simple, consisting (at least) of:

- Running protocols in programmable user-space memory on the software side of the Protocol Wall, in the form of networking programs within which the protocols can evolve as freely as any other application software.
- Merging networking gear and computers into a single unified appliance, eliminating the artificial rift which separates computing from networking.
- Deploying the appliance in networks to blur the distinction between core routers, edge routers and hosts.
- Incorporating data centre functionality in the network infrastructure by running shared application services atop the devices.
- Continuing the inspiration of mainframe computing's envelopment by PC, making it possible to evolve networks of programmable unified computing/networking appliances.

This work is enabled by generous funding by the EPSRC for the INTERNET project.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

In Section II we provide a roadmap for the evolution of programmable networks. In Section III, we explain why protocols are both a hindrance and a help, and introduce soft or flexible protocols as our concept for crossing the divide into full network programmability. In Section IV, we give examples of flexible protocols that have already been widely adopted. In Section V, we discuss related work and present a case for merging the data plane with applications. In Section VI we give two example models, and in Section VII we present the concept of the Inner Bubble as the realm of the Internet as we currently know it, and the Outer Envelope of programmable networks which we anticipate will evolve from and ultimately enclose the Inner Bubble and which will comprise multiple simultaneous networking regimes. In Section VIII we discuss what will be needed in order to accomplish the switch to the Outer Envelope, and in Section IX we consider the impact of the switch on performance as it relates to distance gained, consumers as contributing providers, streamlined protocols, Moore's Law, and optimisations. Finally, in Section X, we consider the reality of bootstrapping our vision for true Software Defined Networking.

In this section, we have stated our position on the necessity of tearing down the Protocol Wall in order to achieve the spirit of Software Defined Networking as it was originally intended. In the next section, we will provide a roadmap for the evolution of programmable networks.

II. EVOLUTION

In the previous section, we stated our position on the necessity of tearing down the Protocol Wall in order to achieve the spirit of Software Defined Networking as it was originally intended. Table 1 shows a roadmap for programmable networks to evolve alongside, and ultimately envelop, the Internet in the same way that PCs evolved alongside and enveloped mainframes.

A broader interpretation of Software Defined Networks can include appliances in the cloud and smart mobile devices. Further, the same interpretive framework can incorporate storage and traditional end system computation, which would allow coherent optimisation across all resources and components of a distributed service, using consistent techniques familiar to software engineers the world over, instead of stove-piped methods for each sector.

In the next section, we explain why protocols are both a hindrance and a help, and introduce soft or flexible protocols as our concept for crossing the divide into full network programmability.

III. OVERCOMING PROTOCOL RIGIDITY

In the previous section, we provided a roadmap for the evolution of programmable networks. In this section, we will address the question of when protocols are a hindrance to innovation and when they are a help. We will also introduce the concept of soft, or flexible, protocols as the way to cross the divide into full network programmability.

TABLE I. EVOLUTION

Evolution	
Mainframe/Internet	***PCs/Programmable Networks***
Long turnaround	Interactive
Restricted to professionals	Mass availability, free to change/adapt
Vendor lock-in, few suppliers	Open market, third party components
Prohibitively expensive	Low cost, frequent updates
Remote, little direct contact	Deployable anywhere
Centralised	Flexible

A. The Necessity and Hindrance of Protocols

In this paragraph we will distinguish between contexts where rigid protocols hinder and where they ease innovation, and we term protocols which ease innovation "*sProtocols"* – meaning "soft" or flexible protocols.

A protocol is a convention, a code of correct conduct, required to guarantee safety or interoperability, like driving on the right (or left) side of the road. Other protocols, such as the protocol that the butter knife should be placed diagonally on top of the butter plate, pose no danger and little risk of confusion if not followed. Where safety and interoperability can be guaranteed without protocols, protocols can be relaxed, becoming flexible "*sProtocols"* or done away with altogether.

Distributed communications programs require running the same program/protocol on all connected devices at any given instant. Protocols typically incorporate significant functionality, causing dependencies upon them and making them inflexible and difficult to evolve. New functionality is difficult to fit into the rigid structure of network stacks [3]. Indeed, this has led some researchers to propose other structures, such as heaps, graphs, or even protocol wheels, to regain some flexibility [4][5]. Protocol Layering is also being questioned as insufficiently flexible to implement FNA abstractions [6]. Protocols have traditionally been painfully "Etched in Stone" firmware, or, where the required performance cannot be achieved otherwise, in the Operating System (OS) kernel [7].

In the next paragraph we will introduce the concept of soft protocols, or *"sProtocols",* as the way to cross the divide into full network programmability.

B. Crossing The Divide to Full Programmability

In the previous paragraph, we distinguished between contexts where rigid protocols hinder and where they help innovation. In this paragraph, we introduce the concept of soft or flexible protocols as the way to cross the divide into full network programmability.

Because protocols have traditionally been "Etched in Stone" in firmware or the OS kernel, the Protocol Wall creates a division between:

- Protocol space, in which activity is dictated, usually at the hardware/software interface level, by rigid rules that are unlikely to change.

- Soft protocol or "sProtocol" space, in which activity is either free of constraints governed by the rules of protocol space, or in which the rules can evolve on their own to meet the challenges of changing constrictions.

Present-day SDN (for example, OpenFlow [8] and Network Functions Virtualization, among others) usurps portions of the controlled switches' protocol logic, such as the control plane, to an external, programmable controller device. Nevertheless, SDN software remains protocol-bound, subject to established universally-approved standards. SDN should strive to move network layer protocol stacks out of the stone in which they are etched, across the Protocol Wall, where they are accessible to developers. Granting unlimited accessibility to the code would remove the obstacles to free evolution, bringing a wealth of human resources to networking.

In the protocol-centric worldview, SDN only applies to flow-state as defined by the classic five-tuple, instantiated by a very limited repertoire of events, such as Transmission Control Protocol (TCP) *Synchronize sequence numbers* (SYN) and *No more data from sender* (FIN) packets, or other protocol packet flow level events. In a more general application-centred model, one would at least hope to incorporate many more general definitions of a flow (one-to-many, many-to-one, a forest of trees), and many triggers. However, this thinking is still constrained to what is possible within current *protocols*, rather than opening the concept space to what would be made possible by redefining communication.

In this section, we distinguished between the contexts in which rigid protocols hinder innovation and when they help. We also introduced the concept of sProtocols – soft or flexible protocols - as the way to cross the divide into full network programmability. In the next section we give examples of flexible protocols that have already been widely adopted.

IV. Existing Flexible Protocols

In the previous section, we introduced the concept of sProtocols, soft or flexible protocols, which we believe are the way to cross the divide into full network programmability. In this section we show that a number of flexible protocols are already in use.

There are abundant examples of flexible protocols that have already been widely adopted: instant messaging protocols, Twitter, online social networks, database access protocols, and media and file formats, to mention but a few.

Smooth operation of distributed applications, backhaul services, social networking communities, and synchronised virtual containers, among other things, clearly depend upon running uniform and dynamically evolvable sProtocols simultaneously on all nodes.

In this section, we gave examples of flexible protocols that have already been widely adopted. In the next section, we will discuss related work and present a case for merging the data plane with applications.

V. Discussion of Related Work

In the previous section, we showed that a number of flexible protocols have already been widely adopted. In this section, we discuss related work and present a case for merging the data plane with applications.

The bibliography of designs for introducing networking and dynamic protocols would fill an encyclopedia [9]. FIAs are not related per se as alternatives. Software-programmable switching supports any networking paradigm and, in fact, multiple architectures concurrently. Nevertheless, we might note that most FIAs are characterised by rigid definitions. They purport to replace today's outmoded protocols [10], but they do so by introducing new protocols.

The questions which should be raised regarding FIAs in general are: Are they evolvable? Can updated versions with enhancements, updates, and bug fixes be released at any time? Can the devices over which the network regime which adheres to the architecture flows load and run these updated versions? Can multiple network regimes of any architectures flow side by side simultaneously?

We now briefly discuss several examples from research groups whose ideation has some common ground with ours. These tend to be more constrained, but their conceptual insights were a valuable source of inspiration to us.

A. *Combining the Functions of Routing with an Embedded Server*

Reference [11] presents a way of combining the functions of routing with an embedded server. The holistic view of the network is presented in [12]. The Autonomic Service Management Framework (ASMF) is constructed based on the concept that everything is a service, and with Application-Oriented Networking (AON), the messaging system is built into the backbone [12]. Embedding the application messaging services within the network fabric, instead of into middleware stacks, where "A service component may act both as a service provider and a customer at the same time, which enables constructing applications over a hierarchy of service components" [12] is an important step towards transforming networks into distributed processing platforms. In that view, AON is an overlay astride Internet Protocol (IP), but there is the hint at its becoming part of lower strata.

Princeton's Serval architecture [13] realises the service abstraction as a new layer in the TCP/IP stack, the level three-and-a-half Service Access Layer (SAL). Serval's intuitive use of the Service Identifier as an anycast identifier, decoupled from any specific transport mechanism, further frees the service abstraction from lower layer rigidities while improving service flow efficiency.

While the potential of eliminating a protocol stack layer is not specific to Serval, Serval could be a testbed. Eliminating a protocol stack layer by placing Serval's level three-and-a-half SAL directly atop a layer two-and-a-half such as Transparent Interconnection of Lots of Links (TRILL) [14] or Seattle [15] would be an illustrative sample case of programmable network stack capability.

SwitchBlade [16] incorporates multiple custom data planes to ease implementing future protocols having one of the shared data plane logic on the same physical hardware, only one of

which is used in a given protocol implementation. To maximise performance, SwitchBlade is implemented in hardware, so cannot fully benefit from loadable networking code. The concept, however, is amply demonstrated in rapid deployment and in saving developers the added effort of implementing commonly used components.

References [17] and, in particular, [18], demonstrate the contribution of in-flow data-plane packet processing as a step towards realising network application programming, and as a cornerstone to many FIAs. See also [19] [10]. We shall see how this need not be limited to edge switches, nor require external devices, but we see room for non-IP networks. We emphatically agree that end users will be the ones that will contribute most to network evolution, just as end users were the ones that drove the innovation in the PC and smartphone markets with the creation of many unexpected applications. In contrast, recent news from industry suggests efforts in the opposite direction [20]: while the company Big Switch Networks is pushing virtual functions down in to the switch, the system is not decentralised, and the overlay aspect is downplayed.

In this paragraph, we have discussed related work. In the next paragraph, we will present a case for merging the data plane with applications.

B. A Case for Merging the Data Plane with Applications

In the previous paragraph, we discussed related work. In this paragraph, we present a case for merging the data plane with applications.

In-transit packet processing is a promising avenue for improving network throughput and for providing a foundation for new, innovative network architectures. Not the least of these are realisations of far reaching visions such as Clark's Knowledge Plane [21]. Data plane applications, however, are seen as an improvement to the network, on behalf of the network. We humbly believe there is vastly more potential in merging data plane processing with user applications to advance distributed applications and resource sharing to much higher levels.

Middleboxes [22] perform important auxiliary functions [7], such as:

- Static load balancing and possibly filtering;
- Dynamic request routing and application integration on behalf of application servers;
- Protocol acceleration by caching/compressing data and responding to requests directly from the caches.

Middlebox implementation in custom Application-Specific Integrated Circuits (ASICs), embedded processors, and software make them

- Rigid, bound to specific protocols;
- Limited in scalability and extensibility;
- Expensive.

The quest is to supplant the rigid layering models of protocols as we know them, with more flexible paradigms such as Role Based Architecture [6] and the Silo Framework [23, 3] in which the stack is composed dynamically as needed by the applications it supports. Here, too, the protocols remain entangled in their own web, unless all the devices in the network recognise and support the new stack structure.

We, and many others, view user-space packet processing is the key to programmable networking. Important advancements have been undertaken in:

- Software optimisations of user-space packet processing [24];
- Hardware boosting of user-space packet transfer;
- Improved utilisation of multicore Central Processing Units (CPUs) for networking;
- Integration of switches and servers;
- Programmable Ethernet chips;
- New distributed services paradigms;
- Commodity hardware in the network core.

Hardware packet transfer optimisations and new distributed services paradigms have made user-space packet processing into a viable option. Commodity PCs and servers now process packets in user-space at rates nearing those of kernel space software switches [5, 13, 17, 23, 24, 25]. Merged appliances incorporate middlebox functionality such that middleboxes become superfluous as separate devices. Nevertheless, all or nearly all of the important developments mentioned above, whether existing or newly evolving, are still grounded in rigid protocols. Networking has yet to liberate itself of protocols by moving them across the Protocol Wall where they turn into application code. The real, long-term benefit is in leveraging efficient low-level user-space packet handling on commodity PCs and servers, by moving the entire networking stack from rigid, protocol-bound kernel space and in-switch ASICs, over the Protocol Wall into programmable user-space, and merging switching and middlebox functionality into the computer.

Programmable user-space code networks are not just another proposal. In juxtaposition to the classical view in networking that something always has to be proposed, designed, and universally agreed upon before it can be implemented - the world of software just evolves. Our aim is to encourage opening the gates to allow software networking paradigms to evolve on their own.

An excellent example of evolution in software is the smartphone generation. Manufacturers did little more than provide development environments and then sat back and watched human ingenuity and creativity evolve. And evolve it did, on a massive scale. We see no reason why the future of networking should be different.

The winner-takes-all competitive concept has no place in the future architectures of programmable networks, where all the architectures will be implementable side-by-side. It is superfluous to play one architecture off against another where the deciding factors are user acceptance and proven reliability.

In this paragraph, we have presented a case for merging the data plane with applications. In the next section, we will provide two example models.

VI. Example Models

In the previous section, we discussed related work and presented a case for merging the data plane with applications. In this section, we provide two example models.

cations are on Development and deployment of web appli the verge of a quantum leap toward becoming vastly more lends to a [27][26] powerful distributed concurrent applications fundamental remodeling of the computing platform on which s cloud 'Remote computing resources such as today .they run a centers can become an organic part of the networking dat .infrastructures

The first example demonstrates the merging of computing and networking. The second example consists of an OpenFlow [8] Controller which is responsible not for routing, but for pushing code down to a number of switches which it controls.

A. First Example

The first example demonstrates the fusion of computing and networking platforms imlementable through but a few The description is broken into the following .feasible steps :sections

1) The Service Platform Abstraction

centric architecture such as -The abstraction is a Service with the distinction that deployed services run as [13]Serval but not necessarily on addressable ,concurrent processes .machine entities

2) ntThe Computing and Storage Compone

Processes run on the conglomerate of myriad cores and memory contributed by the distributed consumer[1]devices no longer distinguishable as discrete atomic ,themselves servers[2].

The network infrastructure component is also provided by the switching and routing capabilities of the participating Issues of .devices themselves which may or may not be static volatile computing and networking resource entering and re both application specific and a leaving the networks a challenge requiring redunancy of underlying software A .foundation components and networking resources alike prime example of the importance of enabling future protocols *."sProtocols"* to evolve freely as

3) Actual Service Platforms Deployment of the

The same conglomeration could perhaps be mapped over TCP/IP, but there would be no need to if a protocol were available with inherent routing, mapping, and maintenance of resource allocation and consumption at any given instance.

above can be sliced (3)and (1)The aggregated resources from amorphous clouds of networked ,*"nebulae"* called-into so is (s)conceptual succesor to the computer ,*"nebula"* The .cores analogous to ,the platform upon which distributed services d is that theiseThe point to be emphas .ms runcomputer progra addressable (anycast two preferably layer)is the *"nebula"*[3] .computing platform entity

servers and networking ,All devices are at once consumers The actual mix of service consumption and resource .apparatus ovision is obviously determined by live parameters of the pr applications and their dynamic allocation demands from neither in their ,Devices need not be equal .available resources static *"heavy"* Some may be .nor in their power ,functionality while ,by commercial suppliers (or supplied) devices deployed .others are wireless handheld devices

cores eSoftware switching components in the devices whos should support multiple networking /may ,make up the nebulus regimes[4].in parallel

B. Second Example

The second example consists of an OpenFlow [8] Controller controlling a number of switches, but instead of (or in addition to) doing routing, the controller is responsible for pushing code down to the switches, each of which can be running multi-instance software switches. The Controller has to sync the pushdown such that the network is able to load code while running without any problems.

The switches should have data processing functionality added, so that they can participate in offering distributed services or multi-virtual-machine cloud services.

As in the first example, the second system switches multiple networking regimes simultaneously: one regime per MAC address. The multiple protocol capability enabled by each physical MAC address exposing multiple MAC addresses, one for each protocol.

In this section, we have provided two example models. In the next section, we will present the concepts of the Inner Bubble as the realm of the Internet as we currently know it, and the Outer Envelope of programmable networks comprised of many simultaneously flowing networking regimes which we anticipate will evolve from and ultimately contain the Inner Bubble.

VII. The Inner Bubble and the Outer Envelope

In the previous section, we provided two example models. In this section, we explain the concepts of the Inner Bubble of the Internet as we currently know it, and the Outer Envelope of programmable networks which we anticipate will evolve from the present-day Internet, and ultimately contain it. We consider how this may be implemented with multiple simultaneously flowing networking regimes.

Design and implementation are conceived within the context of two concentric realms: the Inner Bubble, within the Outer Envelope. Functionality is identical in both realms, but with varying considerations and implications.

---

[1] *Consumer* in the sense of a client

[2] Whether physical or virtual

[3] By any present or future addressing scheme

[4] The entirety of the TCP/IP stack is an example of a networking regime, one of many others in the future.

The Inner Bubble is the Internet world as we know it. The term *Internet,* used generically, refers to protocol-bound networking. The term *Bubble* is the entirety of the Internet in the widest sense, including FIAs.

The Outer Envelope is the wider, more rapidly expanding universe of protocol-free programmable networks, which encompass the Internet, but are not detached from it.

We assume that rigid, protocol-bound networks will ultimately give way to the Outer Envelope of freely programmable networks, as mainframe computing gave way to PCs. The real meaning of this is popularisation of networking, of expanding the accessibility of networking to consumers and developers beyond the closed community of network specialists operating complex and expensive devices. It is about developing and using network applications with the same ease as desktop and smartphone applications. It is about compute/switch devices not as network gear, but as consumer electronic appliances.

The Outer Envelope will be:

- Hardware independent or with plugins, the user-safe equivalent of device drivers;
- Installable as a module or program;
- Protocol independent and evolvable;
- Multiple network regimes running in parallel.

In this paragraph, we have introduced the concepts of the Inner Bubble and the Outer Envelope. In the next paragraph, we will consider the Outer Envelope in which multiple networking regimes may flow simultaneously.

A. Multiple Simultaneous Networking Régimes?

In the previous paragraph, we introduced the concepts of the Inner Bubble and the Outer Envelope. In this paragraph, we consider the Outer Envelope in which multiple regimes may flow simultaneously.

SDN is often predicated on virtualised switches and links. Once switching and communication resources are partitioned, and isolation is provided, why limit oneself to given *protocols?* Just as we no longer need memory access control checking between processes using virtual memory, we no longer need two levels of multiplexing going on in communications systems once the underlying hardware supports virtualisation. Instead, we envisage a communications substrate, much as is present in the Mirage Cloud library OS (libOS) world [25]. In Mirage, cloud applications are built to run in a network slice [15] and/or virtualised, and are specialised to the requirements of the given application. In that cloud world, the hypervisor and the type system provide all the isolation that is needed. This approach is even simpler in a network context than in a data centre environment. Performance, as in Netslice [28], is not compromised.

The appliance can be viewed as comprising (at least) three functional domains:

- Main Machine Domain
    - Native on-the-metal host operation
- Network Regimes Domain
    - Switching and in-transit packet handling;
    - Multiple software switches handling separate, distinct network regimes in parallel;
    - Runs on the metal, but functionally, as a separate domain.
- Services Domain
    - Containerised/Virtualised (along the lines of Virtual Machine cloud hosting) services;
    - Provided to or operating cooperatively with the outside;
    - Components of distributed applications.

The design is for an in-network net/compute appliance utilising recent Network Interface Controller (NIC) enhancements such as Single Root Input-Output Virtualization (SR-IOV), Receive Side Scaling (RSS), and so on. Hand-held wireless devices could also be used to install the networking functionality described here. Networking hardware enhancements are not essential. Many applications could run smoothly without them. Custom chips enhancing switching performance in smaller devices become a likely prospect as use increases. Network/compute appliances employing newer NIC enhancements, SR-IOV, RSS, and so on, could be attractive alternatives to edge-switches and their adjacent servers, and enable their placement in the network core. This would open the field to smaller network operators. This design is based on SR-IOVs: single root, pertaining to only one physical machine. Multi-Root Input-Output Virtualization (MR-IOV) can be used for aggregating multiple servers on common NICs in heavier networking environments.

Each external port on the NIC exposes a distinct MAC address to the physical network for each networking regime running in the box. Each embodies multiple virtual switches (vSwitches) running in parallel. These are not hypervisors implementing switching on the internal Virtual Machine (VM) networks. They are first-class user-space vSwitch instantiations, such as OpenVSwitch; the same software switches as would be used in a standalone switch. Each vSwitch realises a distinct networking regime, isolated from and independent of other regimes flowing through the box. The switch has one or more custom logic controllers and data plane processors for regimes such as MobilityFirst [19], which perform in-transit packet processing.

One or more of the regimes can, of course, be classical protocol-bound regimes, such as TCP/IP with the full suite of protocols and services available on present-day stand-alone switches for Clean Slate-Evolutionary [2] coexistence.

It should be possible to splice streams through differing regimes while in-transit, implementing hybrid regime mixing when needed.

The data centre component of the appliance, be it the currently accepted hypervisor, VM-based model or true distributed applications, runs in parallel. It is isolated, with direct NIC-to virtual NIC (vNIC) transfer on hardware capable

units. vSwitches which discover frames directed to MAC addresses internal to the data centre component can be hairpin forwarded - not as a requirement, but as a performance enhancement. In any case, discovery of such frames by vSwitches should not occur, as the NIC directs frames to the internal vNICs by their MAC addresses.

Flows from any of the networks can all terminate at the same shared applications running in the service domain of the appliance. The physical transport medium can either be shared by exposing multiple MAC addresses on a common NIC port, or dedicated by allocating separate physical NIC ports to each network.

In either case, every switch in the domain belongs to a separate network, implementing the regime of that network. The networking regimes can be content-oriented, service-oriented, or location-oriented. They can run under any addressing, location or naming scheme, and provide in-transit packet handling or direct dispatching.

So long as the implementation logic of each switch is installable software, each switch is entirely independent and unaware of parallel running instantiations. Conceptually, it is no different from running multiple Instant Messaging schemes on the same machine.

In this section, we have introduced the concept of the Inner Bubble of today's Internet which we envisage will be enclosed within the Outer Envelope of programmable networks which will evolve from today's Internet. We have considered the Outer Envelope with any number of differing networking regimes abiding side by side. In the next section, we will consider how multiple simultaneous networking regimes might give rise to a greatly expanded Outer Envelope.

VIII. What Is Required In Order To Facilitate Adoption and Growth?

In the previous section, we introduced the concept of the Inner Bubble of the Internet as we know it today, and the Outer Envelope of multiple simultaneous networking regimes which we envision will evolve from and ultimately contain the Inner Bubble of today's Internet. In this section, we consider what will be required in order to encourage adoption and growth of the multiple simultaneous networking regime Outer Envelope.

The concept of involving the network edge in the core was raised by Clark, Partridge et al. in their work on the Knowledge Plane [21]. The distinction between client/server end nodes and the network blurs entirely in programmable networks. Any device, including hand-held wireless devices with installed user-space networking code, can be both a traffic switching network node and a peer processor service provider and consumer. The mix of services provided or consumed by each node, and the volume of network traffic it handles, is in accordance with the demands and requirements of running applications, device capabilities, and operational policies. All devices capable of running the same networking regime are capable of creating and tearing down networks among themselves. Furthermore, let us not forget the formidable computing power of even the tiniest future devices.

In this section, we have considered the requirements for encouraging the adoption and growth of the Outer Envelope consisting of multiple simultaneous networking regimes. In the next section, we will consider the impact on performance as it relates to distance gained, consumers acting additionally as contributing providers, streamlined protocols, Moore's Law, and optimisations.

IX. Performance Considerations

In the previous section, we discussed what was needed in order to encourage migration from the Inner Bubble of the Internet as we know it today, to the Outer Envelope of multiple simultaneous networking regimes evolving from the Inner Bubble and ultimately enclose it. In this section, we consider the impact of the switch on performance as it relates to distance gained, consumers as contributing providers, streamlined protocols, Moore's Law, and optimisations.

A. Distance Gained

Packets no longer need traverse large distances to reach services at remote edge locations, as they do in today's monolithic client/server data centre environments. Instead, the data centre functionality may be embedded in the network core. This will make it easier to deploy regional and local services, by smaller network operators. The resulting shorter routes ease transport demands.

B. Consumers as Contributing Providers

Proliferation of software switching capable devices will turn consumers into providers as well, increasing network resources. Distributed services can be amorphous, residing anywhere, reachable at the nearest points of contact through anycast addressing.

C. Streamlined Protocols

Unhindered programmability gives rise to continually improving networking paradigms. An example implementable using today's technology might do away with IP where it is superfluous in networks with Ethernet routing.

D. Performance Hit

Performance is not paramount in the Outer Envelope, just as it was not paramount in the evolution of personal computers or in smartphones today. Even if performance were to take a hit, the combination of Moore's Law, more efficient multicore processing, and the scale-out property of networks in which computation is parallel should amply overcome the performance issue within a short time.

E. Optimisations

The following review of some leading optimisation techniques demonstrates how genuinely implementable user-space packet processing and switching has become on commodity computers.

Significant strides have been made in core partitioning, NIC enhancements, kernel optimisation, and remodelled hypervisors, to name but a few.

Bypassing of certain kernel functions to maximise NIC-to-user-space transfer rates has become a common technique. Implementing entire protocol stacks in kernel space should be

superfluous for user-space protocols, where the protocols can freely evolve into enhanced or completely new networking regimes, divested of their rigid *protocolity,* unfettered of tortuous acceptance processes by standards bodies.

This optimisation process may be seen as akin to favouring high level programming languages over assembly coding. Meticulous coding of individual machine instructions can produce faster running programs. However, the price in time and effort is so high relative to the marginal efficiency, that it is seldom considered.

Newer model NICs are capable of vaulting packets over the Protocol Wall by steering them directly to the hypervisor vSwitches connecting to the VMs, thus. The result is considerable reduction of unnecessary context switches, CPU processing, and cache memory pollution. Newer techniques deliver raw packets directly to the target virtual NIC ports in the form of quasi-virtualised connections, or, optionally, directly point-to-point to the virtual NIC port.

Payment Card Industry (PCI) Standards for either proprietary SR-IOV or MR-IOV or other multi-vNIC technologies, create many PCI functions that share a physical device and uplink, such that they appear as multiple devices to the Operating System (OS) providing each VM with access to its own buffer. SR-IOV NICs have separate buffers for each vNIC function, enabling vNICs to implement up to 64KB different virtual functions. 802.1 Enhanced Transmission Selection (ETS) queues are often implemented at the shared link.

RSS is a NIC feature utilising multiple CPU cores concurrently for packet processing in high-speed networks. RSS steers same-flow incoming IP packets to the same hardware Receive (RX) queue of a modern NIC, granting exclusive access to the queue only to the CPU core handling the flow. This eliminates lock contention between multiple CPU cores, improving scalable processing of incoming IP packets across all flows.

In this section, we have considered the impact of performance on evolving to the Outer Envelope as it relates to distance gained, consumers acting also as contributing providers, streamlined protocols, Moore's Law, and optimisations. In the next, and final, section, we will consider the reality of bootstrapping our vision for true Software Defined Networking.

X. BOOTSTRAPPING THE VISION

In the previous section, we considered the impact of performance on migration to the Outer Envelope as it relates to distance gained, consumers acting also as contributing providers, streamlined protocols, Moore's Law, and optimisations. In this final section, we will discuss the reality of bootstrapping our vision for true Software Defined Networking.

The idea of networking becoming popularised as computing did in the past is a whole new realm of thought. The shift from heavyweight, highly professionalised networks to consumer appliances, will open a multitude of new and exciting business opportunities.

The matter of active networks is last, but not least, in our discussion. Research into active networks has been divided into two main strands. The strong version of active networks proposed data packets as programs, potentially changing the behaviour of switches radically and frequently. The weaker version of active networks restricted itself to the modification of the control plane, and represents one of the main ancestors of today's mainstream model for SDN. Our proposal is not like either of these, in that we do not expect frequent and radical reprogramming of communication between any specific entities, but we do envisage having many different communications patterns between many different entities. Nor do we make such a clear distinction between control and data as in current networks. An analogy with the effect of the cloud on the smartphone era might make this clear: early smartphone services were web based and the client side ran in a browser, the server side ran on the data centre, and everything ran on top of the Hypertext Transfer Protocol (HTTP) Wall. Now, many programmers write millions of applications which may have multiple components running anywhere (such as third party advertisement servers, analytics, and so on) and can devise their own communication patterns. This has led to rapid innovation and successful new businesses, and we believe that applying this type of restructuring to the underlying Internet would have an equivalently beneficial effect.

Protocols will not disappear from networking. They will always be needed, whether in the form of hard protocols as we know them today, or soft protocols (sProtocols) in programmable user-space memory. Moving much of the etched-in-stone, hard protocol code into user-memory space will lower the Protocol Wall to the height necessary for today's technology. In practice, that seems to be below the Ethernet packet line, the lowest common denominator in today's networking world.

ACKNOWLEDGMENTS

Thanks are due to Richard Mortier and Marwan Fayed for helpful discussion.

REFERENCES

[1] A. Neumann, I. Vilata, X. Leon, P.E. Garcia, L. Navarro, and E. Lopez, "Community-Lab: architecture of a community networking testbed for the future Internet," In Proceedings of the IEEE International Conference on Peer-to-Peer Computing. Tarragona, Spain, September 2012.

[2] S. Shenker, "The future of networking, and the past of protocols," guest keynote given at Ericsson Research, USA, 2011. Available online at http://www.youtube.com/watch?v=WVs7Pc99S7w

[3] R. Dutta, G. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The silo architecture for services integration, control, and optimization for the future internet," In Communications, 2007. ICC '07. IEEE International Conference on, pp. 1899–1904, 2007.

[4] J. Su, J. Scott, P. Hui, J. Crowcroft, E. De Lara, C. Diot, A. Goel, M. H. Lim, and E. Upton, "Haggle: seamless networking for mobile applications," In Proceedings of the 9th international conference on Ubiquitous computing, UbiComp '07, pp. 391–408, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield, "Plutarch: an argument for network pluralism," SIGCOMM Comput. Commun. Rev., vol. 33, no. 4, pp. 258–266, Aug. 2003.

[6] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap: role-based architecture," SIGCOMM Comput. Commun. Rev., vol. 33, no. 1, pp. 17–22, Jan. 2003.

[7] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: extensible open middleboxes with commodity servers," In Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, pp. 49–60, 2012.

[8] Open Networking Foundation, "OpenFlow Switch Specification," Version 1.4.0 (Wire Protocol 0x05), Oct. 2013. Available online at https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[9] V. Gazis, E. Patouni, N. Alonistioti, and L. Merakos, "A survey of dynamically adaptable protocol stacks," Communications Surveys Tutorials, IEEE, vol. 12, no. 1, pp. 3–23, 2010.

[10] J. Pan, S. Paul, and R. Jain, "A survey of the research on future internet architectures," Communications Magazine, IEEE, vol. 49, no. 7, pp. 26–36, 2011.

[11] V. Tanyingyong, M. Hidell, P. Sjödin, "Offloading packet processing in a combined router/server," in Proceedings of the 7th Swedish National Computer Networking Workshop, SNCNW 2011, pp. 34-37, Linköping, Sweden, 2011.

[12] Y. Cheng, A. Leon-Garcia, and I. Foster, "Toward an autonomic service management framework: A holistic vision of soa, aon, and autonomic computing," Communications Magazine, IEEE, vol. 46, no. 5, pp. 138–146, 2008.

[13] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, "Serval: An end-host stack for service-centric networking," In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, pp. 7–7, Berkeley, CA, USA, 2012. USENIX Association.

[14] J. Touch and R. Perlman, "Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement," RFC 5556 (Informational), May 2009.

[15] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: a scalable ethernet architecture for large enterprises," In ACM SIGCOMM Computer Communication Review, vol. 38, pp. 3–14, 2008.

[16] M. B. Anwer, M. Motiwala, M. B. Tariq, and N. Feamster, "Switchblade: a platform for rapid deployment of network protocols on programmable hardware," ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 183–194, 2010.

[17] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," In Proceedings of the 11th ACM Workshop on Hot Topics in Networks, pp. 43–48, 2012.

[18] F. Risso and I. Cerrato, "Customizing data-plane processing in edge routers," In Software Defined Networking (EWSDN), 2012 European Workshop on, pp. 114–120, 2012.

[19] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "MobilityFirst: a robust and trustworthy mobility-centric architecture for the future internet," ACM SIGMOBILE Mobile Computing and Communications Review, vol. 16, no. 3, pp. 2–13, 2012

[20] .C. Matsumoto, "Big Switch Regroups: Overlays Go, OpenFlow Stays," SDNCentral, Sep. 2013. Available online at http://www.sdncentral.com/news/big-switch/2013/09/

[21] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 3–10, 2003.

[22] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," In Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM '12, pp. 13–24, New York, NY, USA, 2012. ACM.

[23] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "RouteBricks: exploiting parallelism to scale software routers," In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 15–28, 2009.

[24] N. Egi, G. Iannaccone, M. Manesh, L. Mathy, and S. Ratnasamy, "Improved parallelism and scheduling in multi-core software routers," The Journal of Supercomputing, vol. 63, no. 1, pp. 294–322, 2013.

[25] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 195–206, 2010.

[26] Elixir home page, available online at http://elixir-lang.org

[27] M. Camba, "Tutorial: Build a web app using Elixir and Dynamo with streaming and concurrency," April 2013. Available online at http://miguelcamba.com/blog/2013/04/29/tutorial-build-a-web-app-using-elixir-and-dynamo-with-streaming-and-concurrency/

[28] T. Marian, K. S. Lee, and H. Weatherspoon, "Netslices: scalable multi-core packet processing in user-space," In Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, ANCS '12, pp. 27–38, New York, NY, USA, 2012. ACM.

Session Border Controller virtualization towards "service-defined" networks based on NFV and SDN

Giuseppe Monteleone and Pietro Paglierani, *Senior Member, IEEE*

***Abstract*—The promises of Network Functions Virtualization (NFV) complemented with Software Defined Networking (SDN) can be simply stated as: get much more, pay much less. Adopting such paradigms, Service Operators expect to achieve high reductions in capital investments and greater operational agility. However, applying these concepts to existing networks will require challenging architectural updates, as well as deep changes in service models and operating procedures. To analyze such aspects, in this paper we provide an overview of the experience gathered at Italtel R&D labs in this field, and discuss state-of-the-art and future perspectives of these technologies. In particular, we describe how NFV was applied to a specific network function, the Session Border Controller, solving some related technical issues of general interest, and providing scalability and flexibility. Finally, we discuss the evolution of this virtualized network function, and its integration into an SDN-based network.**

***Index Terms*— Hypervisor; NFV; SBC; SDN; virtual machine; virtualization.**

I. Introduction

Network Functions Virtualization (NFV) and Software Defined Networking (SDN) are rapidly emerging concepts, which have the potential to radically modify the telecommunication scenario in the next few years [1], [2]. The promises of NFV complemented with SDN can be simply stated as: *get much more, pay much less*. Adopting NFV, Communication Services Providers (CSP) expect to achieve consistent cost reductions with respect to the current networking scenario, in which network equipments consist of closed boxes containing a bundle of proprietary SW and customized HW, provided by a single Telecom Equipment Manufacturer [1].

The adoption of the NFV concept is just the starting point to introduce in the Telco world the benefits that virtualization has brought about in the IT sector. Besides significant cost reductions, NFV and SDN also raise great expectations on the possibility to achieve a never experienced network flexibility and service agility; to introduce automation in all the life-cycle of network functions, from deployment, installation and commissioning, to the operational phases; to adapt the network to different traffic loads thanks to a novel "cloud elasticity"; to develop new models for improving network resilience, etc [3]-[5].

On the other hand, applying the NVF and the SDN paradigms to the existing Telco infrastructures can require challenging architectural upgrades, as well as radical changes in service models and operating procedures [5]. Moreover, the typical services provided by telecommunication networks can greatly differ from the standard IT applications running in the cloud, in terms of "carrier grade availability" and "high processing throughputs", which are usually achieved by specialized hardware devices, such as Network Processor Units (NPU) and/or Digital Signal Processors (DSP) [3]. Thus, in summary, many open problems must still be addressed and overcome, before NFV and SDN can be considered well-established and widely adopted technologies in the Telco world.

In this paper, as a contribution to the on-going research in this field, we provide an overview of the experience gathered in Italtel R&D labs about NFV and SDN, and discuss state-of-the-art and future perspectives of these technologies.

Firstly, we describe how NFV has been applied to a specific network function, the Session Border Controller (SBC). Deployed at the border between different network domains, an SBC must concurrently process a high number of media flows transmitted by the users, with strict real-time constraints [6]; hence, it represents a typical example of Telco equipment implemented on specialized hardware using *ad hoc* devices, such as DSP's and NPU's. For this reason, developing a fully virtualized SBC, which can represent an effective and attractive alternative to HW-based equipment, is a very challenging task.

The performance analysis of a fully virtualized SBC running on a standard industrial server can be found in [6].

In this paper, we proceed in the analysis by presenting and discussing the overall software architecture adopted for the virtual SBC, in comparison to the standard architecture that is being developed by the NFV ETSI industry study

Giuseppe Monteleone is with Italtel S.p.A, Smart Network Products Unit, Carini, Palermo, Italy (e-mail: giuseppe.monteleone@italtel.com).

Pietro Paglierani is with Italtel S.p.A, Smart Network Products Unit, Castelletto di Settimo Milanese, Milano, Italy (e-mail: pietro.paglierani@italtel.com).

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

group [1]. Also, we describe the virtual SBC deployment model, and address two problems of primary importance in the Telco world, that is achieving service "High Availability", and providing flexibility and scalability to the system. Finally, we discuss the evolution of the virtualized SBC, and its integration into an SDN-based network.

The paper structure is the following. In Par. II we introduce the NFV concept, summarizing the potential benefits of this approach both for Telco CSP and end-users. In Par. III, we briefly introduce the SBC. Par. IV deals with the virtualization of Network Functions, by applying some principles that anticipate the results of the work in progress of the NFV ETSI Industry Study Group [1].

In Par. V we discuss the SBC deployment model and the mechanisms for achieving high availability, while in Par. VI we shortly describe how we are extending these implementations, by applying a model compliant to ETSI NFV complemented with an SDN approach.

Finally we describe our efforts in providing new use cases based on the XaaS (everything as a Service) concept by implementing a marketplace from which Network Functions can be chosen and deployed, jointly applying NFV and SDN concepts (specifically focusing on the medium and large enterprise market) and taking into account real time communication use cases.

II. NFV: A SHORT OVERVIEW

NFV refers to the virtualization of network functions, and their migration from stand-alone boxes based on bespoke hardware to software appliances on top of IT standard infrastructures [1]. On the basis of this very simple definition, the NFV concept can be straightforwardly applied to any data plane and control plane function, both in fixed and in mobile network infrastructures. By adopting NFV, Telecom Operators aim at the appealing possibility of consolidating many network equipment types onto standard high volume servers, switches and storage, in Data Centers, Network Nodes, and End User premises.

Abstracting Software Applications and Operating System from the underlying HW platform allows taking advantage of the latest available compute and storage technologies. The evolution of HW components can originate significant improvements in processing and storage performance, as well as relevant power consumption savings, through the reduction of the number of server nodes needed to deploy a service. It is also widely acknowledged by the NFV community that the use of virtual infrastructures and the development of common interfaces and protocols will allow integrating multiple virtual appliances from different vendors with different HW/hypervisors, using a "mix & match" approach, and thus avoiding the so-called lock-in effect, i.e. the locking of a CSP to a specific equipment vendor [1].

Through NFV, network operators can therefore achieve many benefits; we can summarize them as follows:

Flexibility/optimization:

- Consolidate multiple SW applications onto one HW platform;
- Share resources across different services and tenants, with a better us age of available resources;
- Introduce targeted service based on geography or customer sets, and rapidly scale services;
- Reduce Time To Market, running production and tests on the same infrastructure.

Enhance innovation:

- New services quickly originating revenues at much lower risk;
- Virtual appliance market opening to pure software entrants, small players, academia;
- Wide variety of eco-systems encouraging openness.

The application of the NFV paradigm to the existing telecommunication infrastructures will also originate many new technological and architectural challenges. Among these, an evolutionary path of particular interest for CSPs and TEMs (Telco Equipment Manufacturers) leads to the implementation of high performance virtual functions, portable on different hypervisors and different HW platforms [1], [6]. In the following paragraph, we will focus on this aspect by introducing a specific network function, the Session Border Controller.

III. THE SESSION BORDER CONTROLLER

While the IT world has rapidly moved to the generalized adoption of Virtualized Data Centre solutions, today considered the "state-of-the-art" technology, there is not a widespread deployment of virtual functions in the Telco environment. This is due to many reasons, such as the different level of service availability required for communication services, or the bandwidth throughput and latency values required to achieve a satisfying quality of service for real-time voice and video services. In fact, to achieve such requirements, specialized HW platform are usually adopted by Telco equipment manufacturers.

In this context, the case of the SBC (Session Border Controller), namely Netmatch-S in Italtel implementation, is particularly significant because of the specificity of this network function [6]. In the following, before entering in details, we provide a short description of the SBC concept.

The SBC operates at the edge of two separate IP networks, and controls the interconnection between [7]:

- A core network and an access network; in this case we have a UNI (User to Network Interconnection);
- Two core networks belonging to two different administrations; in this case we have a NNI (Network to Network Interconnection). The two networks are the inner network, managed by the service provider, and the outer network, managed by an interconnected

operator.

For simplicity we consider here only the NNI case. The reference architecture is hereafter shown in fig.1.

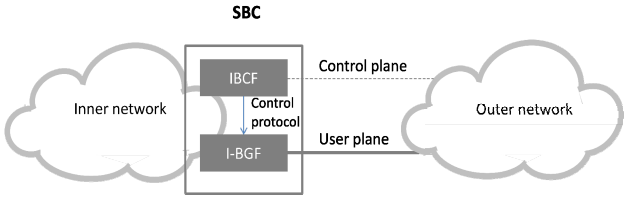


Fig. 1. Session Border Controller simplified scheme

According to ETSI and 3GPP specifications, the IP-IP NNI interconnection is provided through the interaction between the following functional elements [7]-[9]:

- Interconnection Border Control Function (IBCF): on the control plane, the IBCF performs call and session control, establishing communication sessions between different users and/or network applications;
- Interconnection Border Gateway Function (I-BGF) or Transition Gateway (TrGW): on the media plane, the I-BGF provides media processing functions under the control of the IBCF, allowing establishing media streams on the basis of the analysis of the signaling protocol.

A list of function performed for interconnection control and media processing is the following [7]-[9]:

- Border control functions between two core networks subsystem;
- Pinholing: i.e. opening gates and allowing the media streams to be exchanged;
- IPv4-IPv6 interworking;
- NA(P)T-PT functionality;
- Media adaptation (transcoding/transrating), adjusting in real time the coding format of the media transmitted between end users;
- QoS marking (for outgoing traffic);
- Resource allocation and bandwidth reservation (traffic policing);
- Traffic usage metering;
- Multi VLAN support;
- Other: DTMF detection and/or generation; FAX/T.38 interworking.

IV. THE SBC AND NFV

According to the NFV network model, the issues related to virtualization of network functions can be addressed by considering the architecture shown in fig. 2 [1].

The included domains are:

- NFV Infrastructure: hypervisor, compute, storage and networking delivered to the Service Domain according to the IaaS approach [4];
- NFV Service Domain, implemented by virtualized network functions;
- Management and Orchestration domain, for deployment, installation, configuration and general control of network functions;
- Not Virtualized Functions, that in the early adoption of virtualization will be anyway included in any network architecture;
- The legacy OSS/BSS domain that will undergo a deep transformation to be aligned with the new management and orchestration requirements.

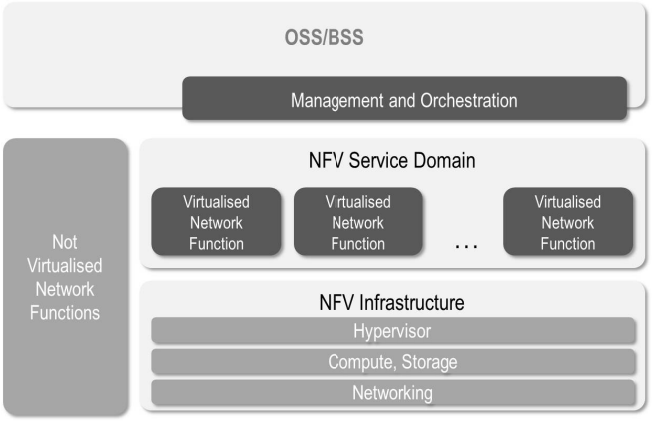


Fig. 2. Domains in the NFV general architecture.

Considering the architecture shown in fig.2, SBCs and many other network functions are now typically included in the "Not Virtualized Functions" domain. In fact, they are typically based on highly specialized HW platforms, mainly because of the following (not-functional) requirements:

- Reliability and availability;
- Performances and scalability;
- Security.

The importance of these aspects is better highlighted if we consider the main differences between Telco services and the generic IT applications running in the cloud.

In the IT world, service outages in the area of seconds are tolerable; conversely, more severe requirements are applied in the Telco world, where the expectation for service outages shall be below recognizable levels (i.e. milliseconds). Moreover, in the IT world the "service user" typically initiates retries after a down of the service, while service recovery shall be automatically performed by the Telco networks (e.g.: active calls shall be maintained in case of server fault). Also, security issues play a major role in the Telco world, because Telecommunication Networks are considered "critical infrastructures" of national relevance. To fulfill such requirements, in many cases the most appropriate and straightforward solution has so far been the design of ad hoc platforms.

As regards performance and scalability, we can safely

state that implementing virtual functions with equivalent performances to those achievable with specialized equipment is very challenging, because *ad hoc* hardware components, such as NPU (Network Processor Units) or DSP (Digital Signalling Processors), currently provide premium performance for tasks concerning security, voice and video transcoding, deep packet inspection, etc.

In the following, we will describe how we addressed the above issues, starting from the deployment model. Through the description of the virtual SBC, we will analyse some technical problems of general interest, such as developing an effective, modular and scalable solution, implementing High Availability, and obtaining acceptable performance levels. It should also be mentioned that the described solution still is the object of research activities and studies, both in view of further improvements, and to include security features, which are outside the scope of this paper.

V. THE VIRTUALIZED SBC SOLUTION

A. *The deployment model*

The implemented solution takes advantage from a "building block" approach, which gives the possibility to combine different basic components, and to obtain - with great flexibility - different configurations, in particular by reusing basic components that are common to different network functions (e.g. protocol stacks). The adoption of a layered architecture allows deploying proprietary and third party "off the shelf" applications including the following "blocks":

- The Software Platform, composed by the Carrier Grade Linux Operating System, the High Availability Middleware and the Italtel Application Platform;
- Proprietary and third party applications including O&M framework and database;
- Proprietary Software Applications.

One of the most relevant aspect of the new "carrier grade" platform is the replacement of proprietary hardware and proprietary Telco operating systems with a virtual infrastructure based on Commercial Off The Shelf HW, making use of hypervisors and virtual machines.

Software applications are bundled in virtual machines (VM) installed over hypervisors. A VM is described as a deployment unit and hosts a self-consistent software module that implements a well-defined set of features.

Referring to fig.3, we can identity how the physical resources (computing modules, memory, storage and networking interfaces) are virtualized in support of VMs hosting the Software Applications. Scalability for a specific function can be achieved by installing a suitable number of VM.

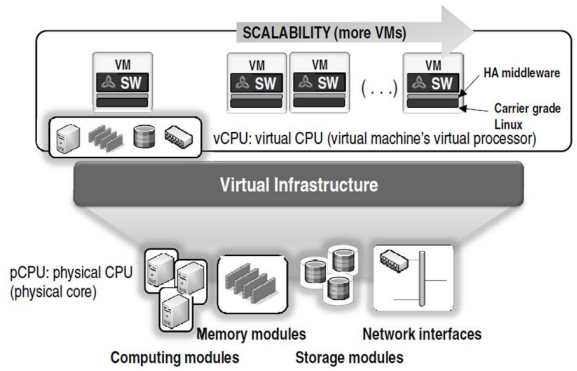


Fig. 3. Deployment of virtual machines over the virtual infrastructure

For the SBC, within the application domain, the following deployment units are used:

a) LBF: operating on the control plane, implements the Front End and Load Balancing Function (LBF)
b) BCF: operating on the control plane, implements the Border Control Function (IBCF);
c) BGF: operating on user plane implements the Border Gateway Function (I-BGF) performing control of media streams;
d) OAM: dedicated to providing the SBC Operation, Administration & Management (OA&M) functions.

The LBF unit has the role of distributing signaling messages towards different instances of the BCF unit (and, for optimization, also between different processes internal to a single BCF unit). Load balancing is not required for the BGF unit, because the choice of a specific instance is performed by the BCF, according to specific resource management algorithms.

In the entry-level configuration and when no scalability is requested, the LBF and BCF units can be jointly deployed. The following table shows the requirements of each unit.

TABLE I: VIRTUAL SBC UNIT REQUIREMENTS

Unit	VCPUs	HA
LBF+BCF	2	Active-standby model
BGF	4	Active-standby model
OAM	4	Active-Active model

In fig.4, we show a simplified scheme of the virtual SBC implemented by Italtel to handle up to two thousands contemporary communication sessions (i.e., 2k Erlang).

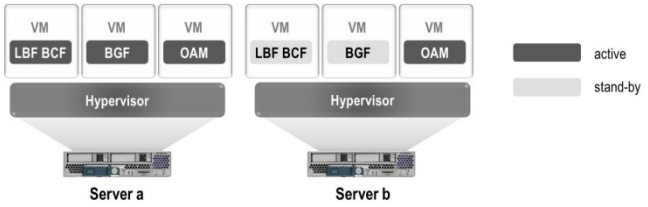


Fig. 4. Simplified deployment of virtual SBC.

The used hypervisor is VMWare vSphere Hypervisor 5.1, the VM are based on the Linux operating system. The virtual SBC runs on two CISCO UCS B200 servers, with hyper-threading enabled. Each server runs three Virtual Machines (VM) implementing three different functions.

The number of physical resources needed for a system configuration can be determined by following this method:

- Analyze, forecast, and plan the capacity needs identifying traffic figures as input data: it is possible to identify all the source of signaling and determinate a model based on CAPS/SAPS (Call/Sessions Attempts per second), Erlang, Call Holding Time [6];
- Media control capabilities requested: i.e. pinholing, NAT, trancoding/transrating [6];
- Identify total capacity needs for each component deployment unit (e.g. LBF, BCF, BGF, etc) leaving space for "spikes in demand": i.e. starting from the traffic figures it is possible to individuate the number of BCF and BGF units;
- Identify the number of VM to be added (for each deployment unit) for redundancy: e.g. we need to deploy two VM for each unit (active and standby one);
- Sum the total needs (for all VMs) obtaining total number of physical CPUs/cores, RAM, disk;
- Allocate VM on physical HW using different HW for redundant groups.

B. *A new paradigm*

First efforts in virtualization were simply devoted to the emulation of the previous HW-based environments. Therefore, blades in a system were substituted by "virtual blades", i.e. virtual machines with equivalent functions, and an implementation of a network function was based on a predefined set of compute, storage and networking resources. Scalability was achieved by adding more computational resources by configuring additional "virtual blades" with a Load Balancing function providing messages distribution among a farm of "real servers". Also, operation and maintenance functions were designed in order to provide management for all the software and hardware components in a system because it was necessary to provide to the network operator a functional behavior fully corresponding to a "not virtualized" function. Hardware resources were anyway explicitly allocated to a network function and also their management was performed specifically in association with this function.

Now we are crossing the chasm, fully experiencing the value of the virtualization of network functions. The new architectural framework for NFV, recently described in [10], introduces a Management and Orchestration function and specific VNF managers. Considering the possibility to use shared resources in a "data center" it is possible to implement a "cloud elasticity" concept by extending or reducing the capacity of a system according to the network traffic figures. A "proof of concept" was implemented in our test bed. Resources in the NFV infrastructure allocated for the VMs of the network function are continuously monitored and whenever specific thresholds are crossed an alert is sent to the element manager, giving the possibility to start a system configuration that is the optimal one. This operation is usually performed under the control of an operator in charge of authorizing the reconfiguration processes.

C. *The virtualized SBC High Availability*

Carried grade availability has been achieved by using a High Availability Middleware (HA) making available: application redundancy; overload control; fault detection and automatic switchover.

The HA middleware implements the Availability Management Framework (AMF) concept introduced by SA Forum [9], referring to the software entity that manages the resources in a system. The objective is to deliver availability by coordinating redundant resources organized in service groups. A service group (SG) is a logical entity corresponding to the ISO term "protected group", identifying a group of managed objects cooperating to provide a redundancy model. A service group contains one or more service units (SU), all of them containing the same components and providing service availability for one or more service instances. The redundancy model of the service group defines how the service units in the service group are used to providing service availability. Different models can be implemented being the 2N (or active/standby) the most common. At most one service unit will have the active HA state for all service instances and at most one service unit will have the standby HA state for all service instances. A use case in shown in fig.5.

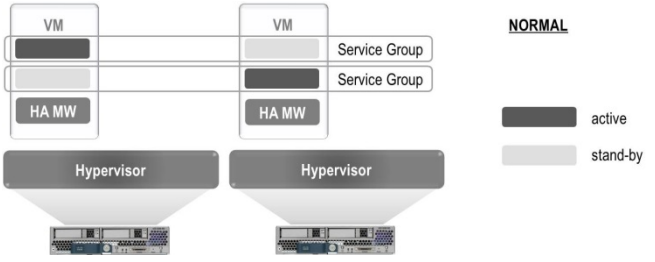


Fig. 5. Service Groups and Service Units

Fig.5 shows two virtual machines each hosting two Service Units (in this case one is active and the other in

standby, to distribute the load on two VMs). The components in the active service unit execute the service, while the components in the standby service unit are prepared to take over the active role, if the active service unit fails. Check-pointing is the continuous update of the log of active calls/sessions and checkpoint data are aligned when a call, after the setup phase, becomes an active call.

Switchover is the operation that occurs when an active service unit is replaced by its designated standby SU. Switchover can happen for failure of the active service unit or for operator-initiated outage. When a fault condition arises, switchover automatically happens putting in service a standby service units so to take in charge, without service interruptions, the role of the faulted one (see fig. 6).

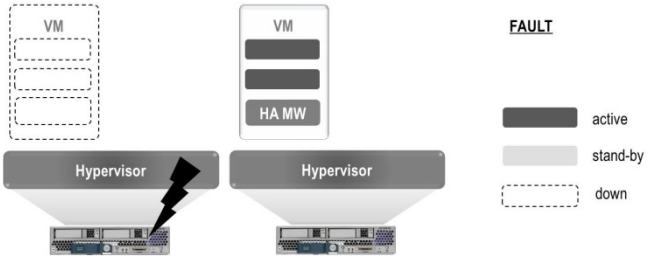


Fig. 6. Fault condition

The main scenarios that Redundancy and Failover mechanisms cover to assure the High Availability are:

- Hardware failures: the failure detection triggers the system reconfiguration, taking advantage of protection groups of redundant resources. The reconfiguration is transparent for internal and external applications;
- Software failures: downtime due to internal software failures is minimized trough check-pointing, application-specific recovery policies, and live reconfigurations within service protection groups;
- Network failures: fast system reaction to external faults, triggered by link and network faults detection. Network interfaces are always duplicated;
- Disaster recovery: full service availability even in case of unavoidable disasters supporting multi-site solutions over extended (geographical) distance and preventing potential "split-brain" (or double master) conditions;
- Hardware upgrade: smooth expansion without interrupting the services handling continuous increase of network throughput.
- Software upgrade: in-service software upgrade offering the possibility to test new services during the software upgrade operations.

D. Upgrades in performances

An important objective for NFV will be to fulfill the performance requirements for virtualized functions. The activities performed towards performance optimization are the following:

- Analysis of functional blocks, identifying workloads and corresponding processing bottlenecks in architecture (computing, networking, etc).
- Design of a methodology for overcoming current performance limitations of user-data processing network functions, or of the component basic functional blocks (e.g. for an IP-IP border gateway function).
- Identification of the best practices for optimizing the performance of the different workloads.

A related objective is to understand what performance parameters are relevant, which functional blocks in the architecture give limitations in computing and/or communication, how to measure performances and carry out benchmarking with existing implementations. The media processing functional entities require a specific approach based on a hybrid of general-purpose cloud Hypervisor architecture, and direct HW access extensions. Specific high performance hypervisor features are shown in fig. 7, such as the exclusive allocation of whole CPU cores to VMs; direct memory mapped polled drivers for VMs, to directly access the physical NICs; direct memory mapped polled drivers for inter-VM communications.

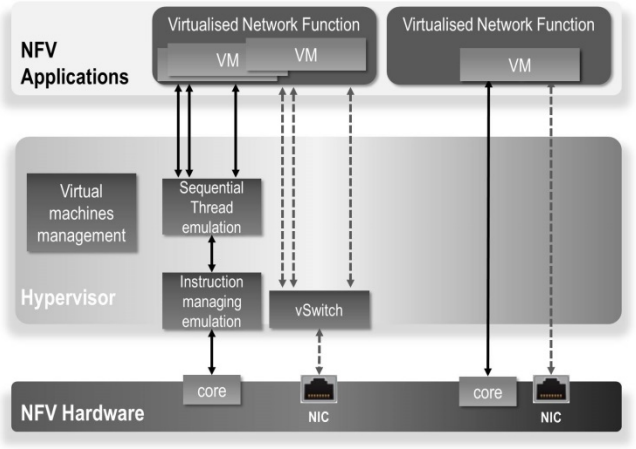


Fig. 7. Resource Allocation Optimization scheme.

VI. COMPLEMENTING NFV WITH SDN

NFV can be implemented without SDN being required; though, the two concepts are highly complementary, and can be effectively combined to (potentially) accrue greater value. Currently, various activities are being carried out at Italtel labs to complement NFV with SDN. Aimed at the optimization of the network infrastructure, a first objective is to provide traffic and address space isolation between VNFs (each requiring its own virtual network). Another objective is the development and implementation of the novel "service-defined networking" concept, i.e. the possibility to dynamically configure the data network according to the requests originated from the "service layer". This concept is exemplified in fig. 8.

The virtualized SBCs network is complemented with a centralized node: i-RPS (Routing and Policy Server), providing centralized control functions. Real Time Communication applications are handled by i-RPS whenever a specific QoE is required. In this case, i-RPS receives from Netmatch, the Italtel virtualized SBC, information on the media sessions to establish along with the QoE requirements. i-RPS maintains a description of the network topology and, by means of information based on, for instance, *ad hoc* measurements on network links, policy rules provided from the external, etc., determines the forwarding rules for media sessions, providing instructions to the underlying network through an SDN controller. i-RPS can maintain "aggregate virtual QoE bearers" or "bandwidth pipes" authorizing access to the "virtual bearer". The capacity of these pipes is varied over time by the network controller according to: Network topology; Link measurements; Incoming signaling requests Established policies. Whenever necessary i-RPS provides new forwarding rules to the underlying network.

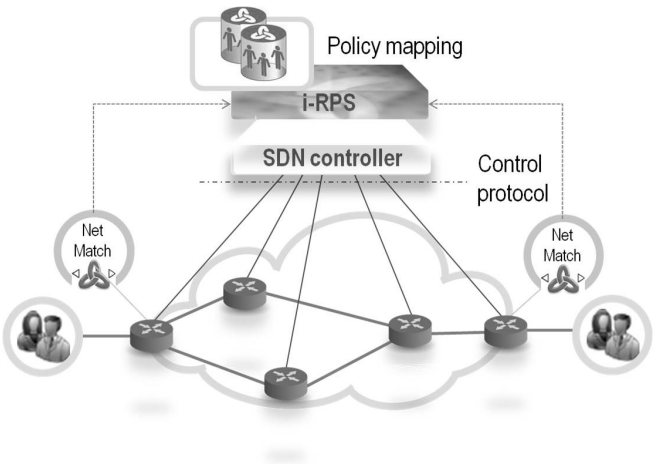


Fig. 8. Service-defined network concept in practice.

This concept is under development in the labs by implementing a "topology independent routing" mechanism that is able to determine the path with lowest latency between two end points. This path is determined by considering the latency measurements between nodes.

VII. THE T-NOVA PROJECT

T-Nova (the acronym of: Network functions as-a-service Over Virtualized infrAstructures) is an integrating project financed by the European Commission (its activities will officially start in November, 2013). With the aim of promoting the NFV concept, T-NOVA introduces a novel, enabling framework, allowing operators not only to deploy virtualized Network Functions (NFs) for their own needs, but also to offer them to their customers, as value-added services. Virtual network appliances (gateways, proxies, firewalls, transcoders, analyzers etc.) can be provided on-demand as-a-Service, eliminating the need to acquire, install and maintain specialized hardware at customers' premises.

For these purposes, T-NOVA will design and implement a management/orchestration platform for the automated provision, configuration, monitoring and optimization of Network Functions-as-a-Service (NFaaS) over virtualized Network/IT infrastructures. An equivalent concept (VNFaaS) was recently introduced in the list of use cases individuated by the ETSI NFV ISG [11].

T-NOVA leverages and enhances cloud management architectures for the elastic provision and (re-) allocation of IT resources assigned to the hosting of Network Functions. It also exploits and extends Software Defined Networking platforms for efficient management of the network infrastructure.

Furthermore, to facilitate the involvement of diverse actors in the NFV scene and attract new market entrants, T-NOVA establishes a "NFV Marketplace", in which network services and Functions by several developers can be published and brokered/traded. Via the Marketplace, customers can browse òand select the services and virtual appliances that best match their needs, as well as negotiate the associated SLAs and be charged under various billing models. A novel business case for NFV is thus introduced and promoted.

REFERENCES

[1] M. Chiosi, *et al.*, "Network Functions Virtualization," presented at the "SDN and OpenFlow World Congress", October 22-24,2012, Darmastadt (http://portal.etsi.org/NFV/NFV_White_Paper.pdf).

[2] ONF, "Software-Defined Networking: The New Norm for Networks," white paper, https://www.opennetworking.org

[3] Bakshi, K., "Considerations for Software Defined Networking (SDN): Approaches and use cases," *Aerospace Conference, 2013 IEEE* , vol., no., pp.1,9, 2-9 March 2013.

[4] Kim-Khoa Nguyen; Cheriet, M.; Lemay, M., "Enabling infrastructure as a service (IaaS) on IP networks: from distributed to virtualized control plane," *Communications Magazine, IEEE* , vol.51, no.1, pp.136,144, January 2013

[5] Sezer, S.; Scott-Hayward, S.; Chouhan, P.K.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M.; Rao, N., "Are we ready for SDN? Implementation challenges for software-defined networks," *Communications Magazine, IEEE* , vol.51, no.7, pp.36,43, July 2013

[6] S. Montagna and P. Paglierani, "On the Analytical Characterization of a Real Life Virtual Network Function: The Italtel Virtual Session Border Control", IARIA CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, 27 May - 1 June 2013, Valencia, Spain

[7] 3GPP TS 23.228: "IP Multimedia Subsystem (IMS); Stage 2".

[8] 3GPP TS 24.229: "IP Multimedia Call Control based on SIP and SDP; Stage 3".

[9] 3GPP TS 29.165: "Inter-IMS Network to Network Interface (NNI)".

[10] *ETSI GS NFV 002: "Architectural* Framework".

[11] *ETSI GS NFV 001: "Use cases*".

[12] Service Availability Forum Availability Management Framework, *SAI-AIS-AMF-B.04.0.*

Software Service Defined Network: Centralized Network Information Service

Jiafeng Zhu[1], Weisheng Xie[1,2], Li Li[1], Min Luo[1], Wu Chou[1]
[1]Futurewei Technologies, 2330 Central Expressway, Santa Clara, CA 95050
[2]Erik Jonsson School of Engineering and Computer Science, University of Texas at Dallas, Richardson, TX 75252
{Jiafeng.zhu, weisheng.xie, li.nj.li, min.ch.luo, wu.chou}@huawei.com

Abstract—Software-defined networking (SDN) is fast evolving as a new paradigm for next generation data networking. In SDN, controller is a central piece of its networks, which provides a logically centralized control of the network. Different from the previous studies on Northbound APIs and Southbound APIs of SDN, in this paper, we propose a service oriented network architecture for east-west network expansions in SDN. In particular, we present the software service defined network (SSDN), a network infrastructure to support east-west network expansion and network services federation across multiple domains. SSDN is based on a highly extensible service computing framework, which differs from the traditional BGP/PCE based inter-domain approaches. We describe the network software-service layer (NSSL) in SSDN, its use cases, its controller and service backup mechanisms, and the new business opportunities under the context of SSDN. It is shown that the proposed approach enhances the development efficiency, enables controller agility, improves scalability, achieves separate life-cycle management, enables inter- and intra-domain communication between different controllers, and enhances the network resiliency.

Keywords—SDN; controller; service oriented architecture; enterprise service bus

I. INTRODUCTION

The emergence of cloud-based services together with big data and server virtualization technologies makes it more important for a dynamic computing and networking infrastructure that can support a wide range of high-bandwidth heterogeneous applications. In contrast to the static and hierarchical traditional network architecture, software defined networking (SDN) [1-2] decouples network control plane from data forwarding plane. It provides the network programmability for users, as well as the underlying network infrastructures for service providers to improve the network automation and management.

Fig. 1 depicts the three layer model of SDN, in which SDN controllers are deployed in the middle of the network control layer. In the SDN architecture, the network intelligence is logically centralized in SDN controllers that maintain a global view of the data network. Modules in the controller can be divided into two categories: *core modules* and *network modules*. The core modules are those components for interfacing with the network and providing resources for the network modules. An example of the core module is the messenger module, which provides TCP/SSL server sockets for communications with the data plane devices. On the other hand, the network modules perform the functionalities of network control and management, e.g., network topology discovery, authenticator, routing, and network monitoring, etc. In addition to the southbound API, e.g. OpenFlow, that a controller uses to communicate and control the data forwarding devices (e.g. switches and routers), the SDN controller also provides a northbound API to the application layer for creating and customizing various network applications, such as traffic engineering (TE), network virtualization, and all other applications customized to users' needs. OpenFlow [1], an industry standard from ONF, has become increasingly popular as the standard southbound API for SDN. The SDN data plane layer contains all the physical or virtual switches, routers, etc, for forwarding the data to various endpoints.

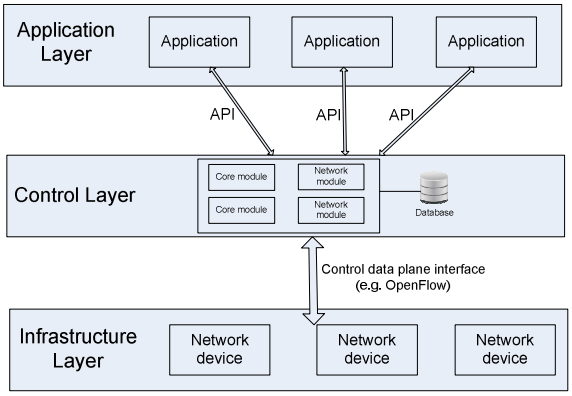


Fig. 1. SDN architecture

However, despite the rapid development on the standardization of the southbound API through ONF, there is no common standard for the northbound API of SDN being adopted. In addition, network vendors are bringing their own SDN controllers to the market, and different vendors' controllers may lack of interoperability. Moreover, it is impossible to control the entire network with a single SDN controller, thus the controllers need to be interconnected to expand the network in the east-west directions. This is mainly for two reasons. First, a SDN controller has certain capability limits, e.g., flow table size, so it can only control a certain number of devices. Secondly, a single controller for the entire network is a single point of failure. Even though this situation

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

can be addressed through multi-controller architecture, where the logically centralized controller is in fact a cluster in the Cloud, there is a strong demand for network expansions cross multiple domains to support multi-tenant or multiple service providers and enterprises. As a consequence, the whole network would unavoidably be divided based on network domains, in which one or a set of controllers would be used for each SDN domain. And, it is perceivable that those controllers could come from different vendors with various types of implementations.

The situation with heterogeneous controllers and multiple SDN domains give rise to a list of challenges:

- Low interoperability and development efficiency: the controller modules and applications may be tightly coupled with specific controller implementation, making it difficult to transplant a module or an application from one controller to another. For example, the NOX controller [3] exposes a set of C++ APIs while the Floodlight [4] controller exposes a different set of JAVA APIs. The term software-defined used in SDN is not a well-defined software design pattern in software engineering, and it should be elevated and enhanced to enforce the required interoperability disciplines to separate the network service applications from lower level and vendor dependent concrete implementations.
- Less flexibility: traditionally, SDN controllers are designed and produced by different vendors. The controller usually has fixed architecture with unchangeable modules and features. The users cannot easily customize the controller modules to meet different requirements.
- Poor scalability: the resources (computation, storage, and network) of a controller may become insufficient to support the fast expansion of the network topology and data forwarding plane. This requires adding new controller resources. For example, a new controller of the same modules and resources as the existing controller needs to be added to share the load. Additional interfaces are needed for the newly added controllers. Also, it could be challenging to develop a synchronization mechanism among the collaborating controllers. Thus, it would be difficult to meet the scale-out requirement.
- Inefficient manageability: when adding or upgrading a controller's module, we may have to terminate other modules, or even shut down the whole controller until the adding or upgrading is finished, causing interruption of the controller's network services.
- Lack of communication between controllers: the controller for a domain usually maintains some domain-specific information, such as the domain's topology and the flow tables of switches under its control, which is typically stored in the controller's own database and not shared with other controllers. This separation makes it difficult to obtain a global optimization across multiple domains.
- Insufficient resiliency: the working controller and the backup controllers of a domain may fail at the same time. There is no mechanism for additional new controllers or other domain's controllers to substitute the failed controllers, since the additional new controllers or other domain's controllers are not aware of the domain-specific information stored in the failed controllers. The domain under the failed controllers is then out of control.

To address these challenges, we propose the software service defined network (SSDN). It provides a service oriented architecture (SOA) [5] utilizing an enterprise service bus (ESB) [6] and a dedicated network software service layer (NSSL), as shown in Fig. 3, to allow network resources to be federated cross multiple network domains. In SSDN, we firstly describe the *flexible network software module*, which is a SDN controller module with two sets of APIs. One set of API is RESTful, which is used to register the module on the ESB and serves multiple controllers. The other set of Internal API is used to integrate the module in the SDN controller. Thus, in the context of *flexible network service architecture*, a controller can be customized by integrating the required flexible network software modules, and subscribing the desired flexible network software service modules on the ESB. Likewise, a controller application can also be registered to the ESB. Moreover, we centralize the network information from different domain controllers in a central network information base (CNIB). We implement the controller modules on the ESB as *network services*, and the applications on the ESB as *network applications*. Network service exposes a set of APIs to network applications or controllers, receives the requests, accomplishes a specific function, and responses to the requests. Network application is usually managed by users, and it doesn't expose any APIs. Network application may consume one or several network services to accomplish a certain task. An example of network application is the traffic engineering, which consumes network services such as topology service, traffic information service, and configuration service. Each service component, e.g., controller, network service, network application, the CNIB, etc., is wrapped in a container and registered on the ESB as a network service element. Each network service element communicates with pre-defined protocols.

The main benefits of the SSDN are as follows:

- Improving interoperability and development efficiency: the network services can be accessed by different controllers, and agnostic to the lower level platforms, operating systems, or even programming languages that are being used to embody a particular controller. The network applications are based on the service description in NSSL, and therefore, they can also be used for any controller in SSDN either directly or generating a matching client from the service description in NSSL efficiently, as long as it is registered to the ESB of the NSSL. A network application or a network service can also share use or call the other network services registered on the ESB, reducing the need to implement separate or to duplicate controller module or application on different platforms.
- Enabling controller agility: with the flexible software network modules, the users can customize their own controllers by integrating modules or subscribing services on the ESB. The new architecture enables users to customize controllers with different requirements.

- Increasing scalability: when scaling out, we only need to increase the resources for those network service resources on the ESB that are in shortage. There is no need to duplicate the controller with a complete set of modules and resources. Instead, resources on ESB can be federated with a well-defined software engineering discipline for network expansion which greatly improves the scalability.
- Achieving separate life-cycle management: for the controller modules and applications that are decoupled from the controllers and registered to the ESB, any modification or upgrading of these modules or applications will not interrupt the controller. Besides, new application or module can be registered to the ESB and serve for other controllers, without the need to install it on each controller separately.
- Enabling inter-controller communication: with SSDN, a controller may send request to other controllers with the pre-defined protocols. Also, a controller may access the information of other controllers in the CNIB, creating a global view of the network and achieving cross domain optimization through the ESB of NSSL.
- Enhancing network resiliency: even if all the working and backup controllers fail, there is still backup information in the CNIB. The additional new controllers or the controllers of other domains may take control of the failed domain by retrieving the domain-specific information in the CNIB.

The rest of the paper is organized as follows. In Section II, we present the background and related work. In Section III, we discuss the architecture, the use cases, the backup and recovery mechanisms, and the new business opportunities of the SSDN. The conclusion is given in Section IV.

II. Background and Related Work

Recently, a draft regarding the inter-domain message exchange protocol for SDN was submitted [7]. This protocol, named SDNi, defines the messages exchanged between SDN controllers of different domains. The message exchange heavily relies on the prevailing border gateway protocol (BGP). BGP makes routing decisions among domains in a decentralized manner, based on the reachability information. However, the BGP has no knowledge of the end-to-end route. It only knows about the next hop along the route. Hence, the path along which data is forwarded is based on a comparison of all available next hops, which may not be a global optimal path. Also, the SDNi is just a protocol for inter-controller communication. The controller modules or applications are still tightly coupled with the specific types of controllers.

The current complex software environments with challenges as distributed software, various devices, diverse platforms and protocols has led to the emergence of SOA. A simplified SOA architecture is shown in Fig. 2. SOA is based on a set of loosely coupled interacting software components that providing their functionalities services. A service accomplishes certain functions and is made available by a service provider to a service client. A service consumer sends a service request to a service provider, and the service provider returns a response with the expected results to the service client. To realize SOA, two constraints have to be satisfied: 1) interfaces are defined for all participating services, which are available for all providers and clients; 2) messages are described and constrained by pre-defined protocols. SOA provides a simple scalable architecture to organize a large number of computers running discrete software modules.

There are various middleware technologies that can connect the service clients and the providers in the SOA. One of the most common middleware is the enterprise service bus (ESB). The ESB is responsible for translating the messages according to pre-defined protocols between service clients and providers. Also, ESB needs to route the messages and guarantee the arrivals of the messages.

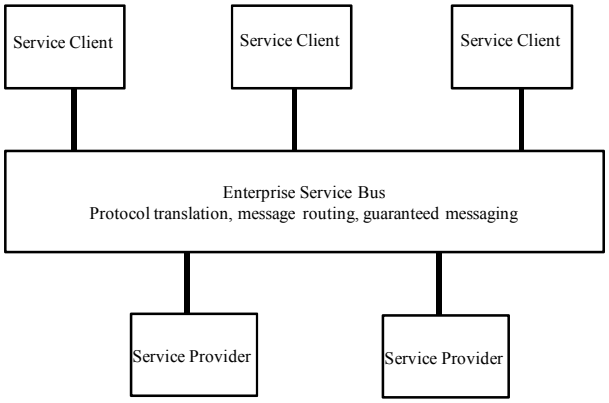


Fig. 2. An example of SOA based on ESB

III. Software Service Defined Network

A. Architecture of SSDN

As shown in Fig. 3, SSDN consists of a network software service layer NSSL, formed by network applications, network services, ESB, and CNIB, which is different from the traditional SDN architecture shown in Fig. 1. With the flexible network software service modules, some controller modules are registered to the ESB as network services. The network services and network applications can serve multiple controllers. The information stored in each SDN controller is replicated into the CNIB.

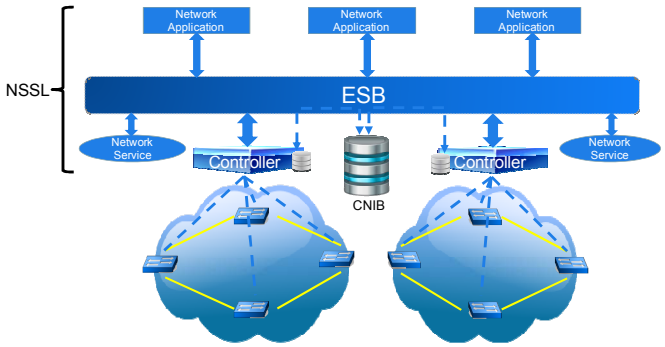
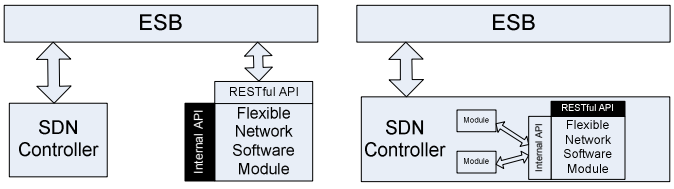


Fig. 3. Architecture of SSDN

1) Flexible network software module: the flexible network software module is a controller module which can either register to the ESB or integrate in the controller. As shown in Fig. 4, the flexible network software module has two sets of APIs. One is the RESTful API, which is used to interact with other services on the ESB, and the other set of APIs is the Internal API, which is used to connect with other modules of the SDN controller. Note that only one set of the APIs is used at a time, as shown in Fig. 4(a), or integrated in the SDN controller as an ordinary controller module, as shown in Fig. 4(b).



(a) Flexible module registers to the ESB

(b) Flexible module integrates in the controller

Fig. 4. Flexible network software module

2) Flexible Network Service Architecture: based on the flexible network software module, a SDN controller can be customized according to users' needs. When assembling a SDN controller, the new controller may begin with the core modules, then select desired flexible network software internal modules to form a customized SDN controller. The customized SDN controller can also subscribe other network services on the ESB. Thus, different SDN controllers may have different features, depending on what modules and what services they choose. Multiple vendors can contribute to the same flexible network software module or network service. Users have the choices of different vendors' internal modules or external network services.

As the example shown in Fig. 5, the customized SDN controller has the basic core modules, and adding the firewall module and the routing module from a list of the vendors. Meanwhile, the SDN controller subscribes the network services on the ESB.

In SSDN, what functions should be decoupled from the controllers is a system design choice. In this paper, we provide a general guideline for such design in SSDN. Keeping the module inside the controller may reduce the traffic overhead and latency, while registering the module on the ESB enables higher efficiency and scalability as discussed in Section I. Thus, the core modules used to establish TCP connections and maintain these connections with the programmable switches should be integrated in the controller. Some modules with offline computation, such as the network diagnose, may be good candidates to be decoupled and registered on the ESB.

The flexible network service architecture enables the realization of service orchestration and service chaining. Service orchestration is the coordination and arrangement of multiple services exposed as a single aggregated service. The

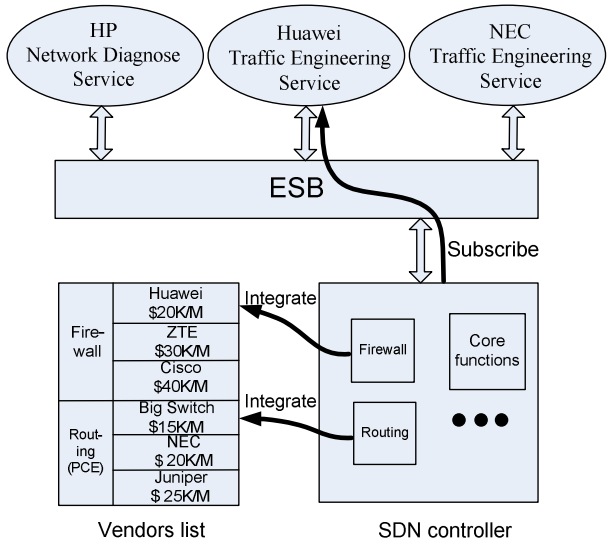


Fig. 5. Flexible network service architecture

logic is specified from one single participant, called the orchestrator. The control logic is usually described in a machine-readable language. An example of the service orchestration is traffic engineering, in which the traffic engineering application is the orchestrator. We will show this example in detail in Section III. B. In contrast, the control logic of service chaining is not centralized in an orchestrator, but realized through the message exchanges that occur between services. Also, the output of a service is the input of the next service. An example of the service chaining is the routing service including the services of controller as a service, path computation service, and the topology service. The details will be shown in Section III. B.

3) Network applications: the network application can also register to the ESB. It may consume one or more services to accomplish a certain task. It is invoked or modifed by the users or administrators, without exposing APIs on the ESB.

4) CNIB: the CNIB is an independent, high performance database wrapped in an OSGi container and registered to the ESB. The CNIB provides a set of APIs for basic operations including querying, inserting, updating, and deleting data.

When a new controller is registered to the ESB, its database is replicated into the CNIB. Whenever the controller's database is updated, either through the information update from the network or through the input of administrators, its corresponding part in the CNIB is updated as well. CNIB can serve different network services, applications, and controllers from same or different domains. When a controller fails, its backup controller or the controllers from other domains may retrieve the information from the CNIB and take place of the failed controller immediately. In our approach, this failover mechnism can be realized through the service discovery mechnism of the ESB based on the registered service specifications.

Network information security is a big concern, especially in a multi-tenant environment. The controller's information replicated into the CNIB should be kept private to the

controller itself. When a controller needs to retrieve the information of other controller, it needs to firstly communicate with the controller through the ESB for access permission, then the requesting controller can only retrieve the information from the CNIB after the permission is granted. To provide the network information security, there are a number of steps that can take. Some steps include providing unique authentication credentials for each controller, preventing the theft of authentication credentials and protecting the database or the information it contains. In our approach, the service registration and invokation process of the ESB provides a framework for authenticated service access and data sharing.

A unique CNIB service can be deployed in a cluster and support multiple network domains. We can synchronize multiple CNIBs via the ESB federation, if it is required.

With the CNIB, a controller may retrieve the information of other controllers to create a global view of the network and achieve better network optimization. Besides, the high availability and high performance of the CNIB provides a reliable storage for backup and recovery. A controller can retrieve the information from the CNIB upon the failure of the working or backup controller and take control of the network domain managed by the failed controller.

5) ESB: ESB in our approach serves as the middleware connecting the network applications, services, CNIB, and controllers in the NSSL layer of the SSDN. It is responsible for translating, routing, service registration, authentication, service discovery, and messages passing between different service components. The ESB may support two messaging modes: one is Point to Point (PTP), in which the message is sent to only one receiver. The other is the Publish/Subscribe (Pub/Sub), in which the message is sent to all the receivers subscribed the same service. For example, multiple network applications may subscribe the same topology service. The updates in topology are sent to all the subscribers.

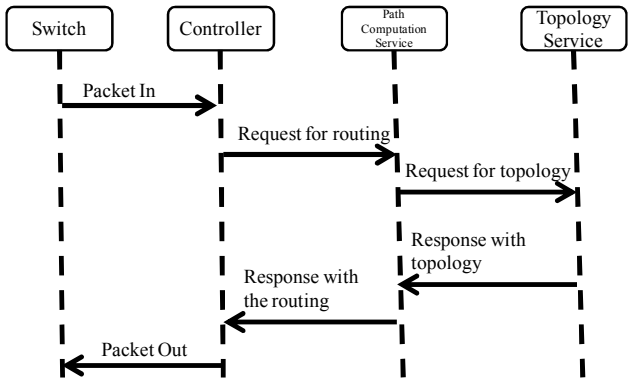


Fig. 6. Routing service process

B. Use Cases

1) Network Service

We use routing service as an example to demonstrate how the network service works. The routing service calculates paths between end points according to certain routing requirements.

The routing service process is shown in Fig. 6. When the first packet of a flow arrives at the switch, the switch sends the packet to the controller if no flow entry in the routing table of the switch matches the incoming data flow. Assuming the path computation service is now decoupled from the controller and registered on the ESB, the controller then needs to send a request to the path computation service on the ESB of NSSL for the routing decision of the new flow. The request contains the domain ID of the controller. The path computation service then retrieves the topology information of the associated domain from the topology service, and calculates a path for the new flow. The calculated path is sent back to the controller. The controller initializes a new flow in the flow table, sends the packet back to the switch, and finalizes this routing service. In this example, the controller, path computation service, and the topology service form a service chain.

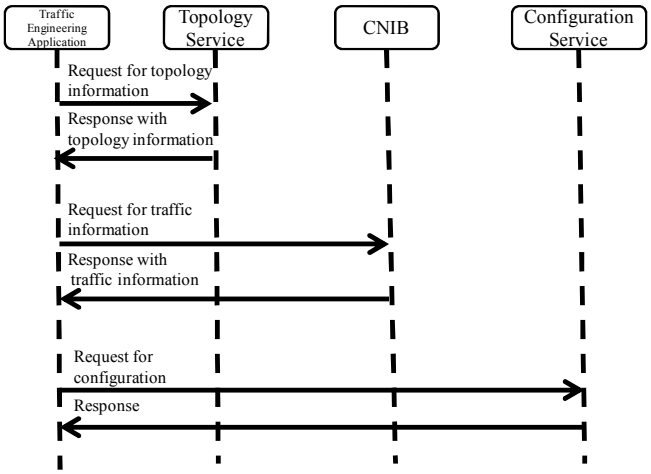


Fig. 7. Traffic engineering process

The routing service may be intensively used. For example, the frequent and rapid virtual machine migration may cause the fast changes of routing. In traditional controller, the routing is integrated in the controller as a module. When the routing service is intensively used, additional controllers are needed to share the computation load. However, there is no yet a synchronization mechanism between the controllers to share the load. Also, new interfaces are needed for the new controllers. Our new architecture solves this problem, by decoupling the routing module from the controller and registering it on the ESB as a service. When the routing service is intensively needed, additional new instances of the routing service can be created easily on the ESB. When the need for the routing service decreases, we can simply reduce the routing service instances. Thus, our architecture increases the scalability. Also, the upgrading or modifying the routing service does not interrupt the controller, as long as the interface of the routing service on the ESB does not change. The separate life cycle management is thus achieved.

2) Network Application

We use traffic engineering as an example to demonstrate the running of a network application. The traffic engineering application takes the network topology and the traffic condition as input, runs its optimization algorithm, and configures the network accordingly.

Fig. 7 illustrates the steps. The topology service builds the topology from the node and link information collected from the network. The configuration service communicates with the SDN controllers and configures their settings. The traffic engineering application firstly collects the topology information from the topology service, and then requests the traffic information from the CNIB. Using the topology and the traffic information as input, the traffic engineering application runs its optimization algorithm, derives the calculation results, and sends them to the configuration service. The configuration service then sends requests to the corresponding SDN controllers to modify their flow tables accordingly. This is an example of service orchestration, in which the topology service, the CNIB, and the configuration service are orchestrated by the traffic engineering application.

In traditional SDN controller, the traffic engineering is integrated in the controller as a controller application. Each controller has its own traffic engineering application. By decoupling the traffic engineering application from the individual controller and register it on the ESB as a network application, multiple controllers can share the traffic engineering application on the ESB. The users of the controllers also share the development and maintenance cost of the application, which is much lower than keeping the application in every controller. Besides, decoupling the traffic engineering application from individual controller has the advantages in scalability and separate life-cycle management as mentioned above.

C. *Backup and Recovery*

The SSDN provides backup and recovery mechanisms different from the traditional SDN controller backup mechanism. Currently, each SDN working controller has several backup controllers, and maintain separate database. The backup controller replicates the database of the working controller periodically. If both the working controller and the backup controllers fail, the domain will be out of control. In SSDN, the CNIB is deployed on the ESB as a service, therefore, the additional new controllers or the controllers of other domains with matching service description may take control of the domain in which both the working and backup controllers fail. This mechanism improves the network resiliency, and moreover, such enhancement for failure recovery and repair can come from the service registration and discovery mechanism of the ESB used in our approach.

There are two possible backup and recovery mechanisms: proactive backup and passive recovery. In proactive backup, the backup controller registers to the ESB even if the working controller is up. The register ID of the backup controller is different from the working controller. When the computation load of the working controller is too high, the backup controller may share the computation load and work at the same time. When the working controller is down, the backup controller can quickly retrieve the information of the working controller from the CNIB and replace it. In passive recovery, the backup controller registers to the ESB only when the working controller is down, and the backup controller has the same ID as the working controller. Similarly, the backup controller retrieves the information from the CNIB after the failure of the working controller. The passive recovery does not have the load balanced function as the proactive backup; however, it enables the working and backup controller to share the same ID on the ESB. If both the working controller and the backup controllers fail, the additional new controllers or the controllers from other domains may retrieve the information of the failed domain from the CNIB and replace the failed controllers.

D. *New Buisness Opportunities and Ecosystem*

The SSDN introduces new business opportunities and a new possible ecosystem. Nowadays, different vendors are designing and producing their own SDN controllers. Application developers usually have difficulties, if not impossible, to modify the modules inside the proprietary controllers, and the development of controller applications has to be based on the specific APIs exposed by the vendor.

With our SSDN approach, different vendors may bring to the market their controller modules, network services or applications. The modules or applications registered on the ESB can be used by any types of controllers, since the flexible network service architecture enables the registered services to communicate with each other, regardless of their platforms.

From users' perspective, SSDN enables them to customize their own SDN controllers to meet their requirements. Users can choose modules they want to integrate in the controller and subscribe the additional network services on the ESB.

The SSDN creates a new ecosystem. Multiple vendors can contribute to the same or different network elements, while users can choose their favorite network elements according to their requirements from the vendor list. The SSDN greatly increases the number of controller modules and applications that each controller can choose. It opens the market to small and medium companies to produce their network elements instead of producing the whole controllers, which the small and medium companies can hardly afford. Therefore, our open ecosystem introduces new business opportunities to the market.

IV. CONCLUSION

In this paper, we presented the software service defined network (SSDN), a network infrastructure to support east-west network expansion and network services federation across multiple domains. SSDN is based on a highly extensible service computing framework, which differs from the traditional BGP/PCE based inter-domain approaches. We described the network software-service layer (NSSL) in SSDN, its use cases, the controller and service backup mechanisms, and the new business opportunities under the context of SSDN. It is shown that the proposed approach enhances the development efficiency, enables controller agility, improves scalability, achieves separate life-cycle management, enables inter- and intra-domain communication between different controllers, and enhances the network resiliency.

REFERENCES

[1] Software-Defined Networking: The New Norm for Networks, Apr. 2012. White paper. [Online]. Available: http://www.bigswitch.com/sites/default/files/sdn_resources/onf-whitepaper.pdf

[2] S. Jain, A. Kumar, S. Mandal, J. Ong. L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," to appear, ACM SIGCOMM, 2013.

[3] N. Gude, T. Koponen, J. Petit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards and operating system for networks," ACM SIGCOMM, vol. 38, issue 3, July 2008.

[4] Project Floodlight, http://www.projectfloodlight.org/floodlight/.

[5] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, "A brief survery of software architecture concepts and service oriented architecture, " IEEE International Conference on Computer Science and Information Technology (ICCSIT), pp. 34-38, 2009.

[6] S. Ortiz, "Getting on board the enterprise service bus," Computer, vol. 40, issue 4, pp. 15-17, 2007.

[7] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains, " submitted to IETF Internet-draft, Dec. 2012. [Online]. Available: http://tools.ietf.org/html/draft-yin-sdn-sdni-00.

An OpenNaaS based SDN Framework for Dynamic QoS control

Iris Bueno, José Ignacio Aznar, Eduard Escalona, Jordi Ferrer, Joan Antoni García-Espín

Distributed Applications and Networks Area (DANA), i2cat Foundation, Barcelona, Spain

Email:{iris.bueno, jose.aznar, eduard.escalona, jordi.ferrer, joan.antoni.garcia}@i2cat.net

***Abstract*— Network Service Providers should offer provisioning services with guaranteed Quality of Service (QoS), specifically adapted to the characteristics of the applications running on their network. In this paper we propose the Network Control Layer (NCL), a software framework solution based on Software Defined Networks (SDN), OpenFlow and Network as a Service (NaaS) paradigms. It addresses a major innovation area in the field of network control and management providing on-demand end-to-end network provisioning services with guaranteed QoS, based on the specific requirements of on-top running interactive applications. The NCL implementation is based on the OpenNaaS framework, and it includes mechanisms for network status monitoring and SDN switches configuration based on the interactive applications´ QoS network requirements. We demonstrate NCL´s utility in the context of control plane models making use of a practical use case.**

Keywords—QoS, Control Layer, SDN, OpenFlow, NaaS, OpenNaaS

I. INTRODUCTION

The sheer increase of applications that require strict and diverse levels of QoS such as on-line gaming, VoIP or video steaming are experiencing tremendous growth. This growth, together with the need for user customization pushes the boundaries of what current best-effort networks with unstable Quality of Service (QoS) can support. Besides, the use of Over-The-Top (OTT) services such as content based applications has increased causing an abusive usage of the shared resources that might be needed by other applications.

Such applications demand dynamic QoS provisioning. Nevertheless, network operators and ISPs just offer high quality tailored services to premium users and services. The rest of services use best-effort policies leading to unpredictable services' behaviour and a lack of reliability on network providers. Also, the number of users accessing applications varies over time and entails highly variable loads impacting network scalability and elasticity. Therefore, it is necessary to provide the network with smart real-time reconfiguration of the forwarding plane resources to optimize network performance. Moreover, users do not have control over specific network configuration parameters whose control delegation might enable certain management options to them and thus enhancing the service delivery.

Offering higher end-to-end (E2E) QoS as a Service entails, on one hand a source of revenues for network service providers and, in the other hand, an attractive added-value for the user community. Nevertheless, QoS poses considerable challenges for applications as network infrastructures are tight to complex non-programmable models whereas users´ applications demand new levels of optimization to request network services adapted to their requirements.

In this paper we address applications QoS limitations while provisioning services. To this end, we propose a Network Control Layer (NCL) which provides with a mechanism for the dynamic, on-demand E2E provisioning of network resources upon changes in the application requirements. The NCL supports mechanisms to manage different kinds of traffic depending on their QoS parameters and allows accommodating them by dynamically reconfiguring the network.

The NCL is based on SDN [1] and Network as a Service (NaaS) paradigms providing an abstraction layer to use a homogeneous control over the heterogeneous underlying infrastructure. SDN technology enables the efficient configuration of the forwarding plane network resources to achieve QoS requirements. To this target, we make use of the standardized OpenFlow [2] interface. On the other hand, the NaaS approach allows dynamic and scalable network service management. It provides users secure and isolated tenant access to network infrastructures, delegating management permissions and control functions and thus, enabling the

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

possibility to easily deploy and operate customized advanced network services with predictable QoS performance according to on-top application needs.

We also propose an NCL implementation based on the OpenNaaS framework [3] created within the FP7 Mantychore project [4]. OpenNaaS confers to the NCL a sophisticated manner to supply applications with on-demand self-service, resource manipulation rights, resource pooling, flexibility, elasticity and dynamic service management. We provide with a description of the NCL design, interfaces, modules and OpenNaaS abstracted resources. Finally, we propose a use case example that demonstrates NCL mechanisms utility, leveraging SDN, OpenFlow and OpenNaaS technologies.

The reminder of this paper is organized as follows. Section II describes the state of art related to QoS provisioning services; in section III we present the NCL architecture and functionalities. Section IV describes the NCL OpenNaaS implementation and presents a relevant use case. Finally, Section V concludes the paper.

II. STATE OF THE ART

Current Internet Best-effort model does not guarantee the demanding performance requirements of current inelastic applications and services, since it does not exists a complete solution for providing with dynamic QoS configuration. From the user perspective, best-effort services result in an unpredictable behaviour and often in a poor quality of the services. There have been several attempts and initiatives in support of QoS while provisioning services. Integrated Services model (IntServ) [5] focuses on per-flow parameters and uses RSVP (Resource Reservation Protocol) to make the resources reservation. In order for IntServ to work, all routers along the traffic path must support it and cache all the information related to each of the services. Consequently, many states must be stored in each router leading to significant scalability issues. Due to such scalability limitations, Differentiated Services model (DiffServ) [6], [7] was developed. DiffServ makes use of the Type of Service (ToS) field for traffic classification. Per-flow states are maintained only at the edge routers of the domain so that guaranteed QoS is only delivered based on aggregated traffic. Besides, another limitation of this model relies on the need of a previous mapping process between applications and service classes. Other mechanisms use DiffServ over Multi-Protocol Label Switching (MPLS) which is based on aggregated broad flows. Traffic Engineering extensions to MPLS (MPLS-TE) allow directing data from one network node to the next based on short path labels, rather than long network addresses [8]. Also, Call Admission Control (CAC) systems [9] have been developed in the past. CAC systems accept or reject new incoming calls depending on the network behaviour in order to keep QoS levels. Nevertheless, their use has been limited to networks dedicated to VoIP applications and both centralized and distributed approaches present scalability impairments. Finally, Flow-aware architectures are based on implicit admission control and per flow scheduling [10]. However, some new techniques are emerging in order to solve the inter-network QoS issue. None is completely integrated neither supports dynamic on-demand QoS delivery.

Thus, an alternative while providing E2E service-guaranteed QoS consists of enabling in enterprises´ switches the capability of taking decisions autonomously. However, this requires switches to be equipped with sophisticated and expensive integrated circuits. The SDN model is an approach for building networks that separates and abstracts the control and data planes. SDN is designed to rely on external controllers for all network decisions, which can be implemented in specific software and run on standalone servers. Easier Data Center (DC) management and new network capabilities are also seen as eventual benefits of SDN.

On the other hand, the Network as a Service (NaaS) paradigm seems to be a key paradigm for future network infrastructures. An optimal solution to the lack of QoS in provisioning services would be the deployment of software and tools to provide Network as a Service (NaaS) capabilities to applications providers. NaaS is a management model related to network infrastructure servicing based on resources and capabilities. The resources are the logical representation of a physical or virtual device and the capabilities are the features associated to a specific resource according to the functionalities.

The NCL is based on these two concepts: SDN and NaaS. Both paradigms provide to the NCL with the flexibility, reliability and adaptability to the requirements of applications and services. The following sections detail the NCL fundamentals and implementation framework.

III. NCL ARCHITECTURE

The main goal of the NCL model is to provide the network intelligence and keep an updated network state, enabling the abstraction of the underlying network infrastructure by decoupling the control and forwarding planes, while providing End-to-End QoS. The NCL is composed by three main components: the provisioning intelligence module in form of SDNapp, the SDN Controller and the Monitor module. In addition, the NCL implements three main interfaces: Northbound, Southbound and Inter-Domain interfaces. These interfaces confer to the NCL flexibility and elasticity and enable network configuration based on applications´ requirements and the state of the network. Figure 1 shows the NCL architecture, depicting the different modules and interfaces.

A. NCL modules

QoS SDN Application (SDNapp): the QoS SDN Application (SDNapp in Figure 1) consists of a set of generic network modular functionalities running on top of a SDN/OpenFlow network. It enables to adapt the control plane to fit service providers' requirements in terms of control and management and configure the forwarding plane on demand. Thus, SDN applications' functionalities provide with unprecedented programmability and enable business to rapidly adapt according to new needs and requirements. The NCL implements the following SDNapp functionalities in order to match application requirements:

a) Resource allocator module: implements the intelligence to take decisions over the forwarding plane (e.g. routing decisions or load balancing) to match application QoS requirements.

b) Provisioning module: the NCL architecture makes use of it to allocate the required network resources demanded by the applications and guarantee the expected QoS.

c) QoS tracker module: this functionality enables the SDN application to customize QoS to its different types of traffic in order to suitably accommodate them in the network forwarding plane.

d) Application request engine: it filters the requests coming from the application layer. In case a certain application requests for certain QoS that exceeds service level agreements or there are no available network resources to accommodate it, the request engine denies the request.

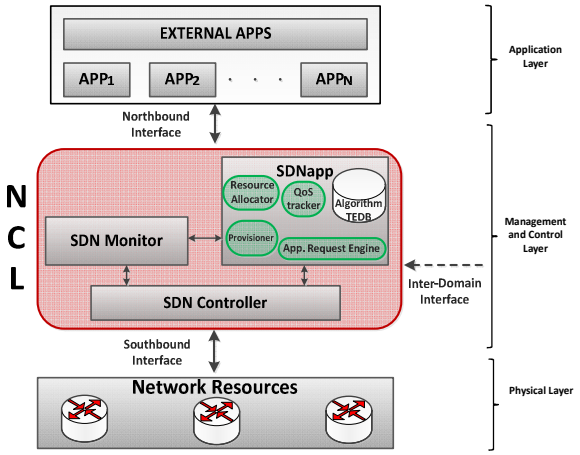


Figure 1. NCL architecture

e) Algorithm TEDB: the SDNapp also incorporates a Traffic Engineering Data Base (TEDB) where different algorithms are stored. The algorithms calculate the customized routes to guarantee the QoS required by the applications. Depending on the desired network performance, the NCL administrator is able to select which one offers better reliability for provisioning E2E services.

Thanks to these modular functionalities, the QoS SDNapp builds an updated topology map which reflects status of the network resources and the QoS levels of the running applications. From this information and the chosen TE algorithm, the NCL configures the forwarding plane by assigning different types of traffic to different paths depending on the features of each route (bandwidth, delay, jitter), applying queues, priorities, etc.

SDN_Controller: it is the responsible for the configuration of the physical resources. Besides, it maintains the network rules and distributes the appropriate instructions to the network resources. In essence, the controller centralizes the network intelligence, while the network maintains a distributed forwarding plane through the resources. The controller is OpenFlow-based; therefore it uses the OpenFlow protocol to communicate with the resources (OpenFlow switches). The controller is responsible for determining how OpenFlow switches handle packets: if an incoming packet matches a flow entry in a flow table, its associated action is executed. On the other hand, if the packet does not match a valid flow entry the packet is sent to the controller. The default action is to send the packet to the controller over the control channel and wait till the NCL makes a decision on how to handle that packet. If no rule can be applied matching the required QoS

the packet can be dropped. Otherwise the controller will add a flow entry specifying how to forward all the packets that match the particular rule.

The QoS SDNapp module interacts with the controller by triggering updated QoS application requirements. This provides to the controller with the dynamicity to rapidly respond to changes in applications´ QoS. Static paths similar to end-to-end VLAN tagging present limitations in the control of the QoS offered because of the lack of adaption capability in real time. The controller adds, removes and updates flows during the application lifetime allowing the network to reconfigure on-demand and overcome this limitation. The main controller functionalities include:

- Map data flow rules to the switches. Add, update or delete flows.
- Discover network resources that are available in the network.
- Discover switches capabilities: OpenFlow protocol supported ports, queues, etc.
- Retrieve network statistics: deploy an interface to retrieve network statistics via the OpenFlow protocol.
- Acquire the rule to map into the switches. The SDNapp sends the OpenFlow rules to the controller.
- Deliver the retrieved statistic to the monitor module.
- Abstract the heterogeneous underlying infrastructure.
- Decouple the control and the data planes.

SDN_Monitor: monitoring tools measure the state of the network by collecting statistics using OpenFlow counters features and other monitoring tools. The Monitor module allows an interface to receive monitoring information from external monitoring tools or applications. The SDN_Monitor main functionalities are:

- Retrieve network state information. The statistics provide information of the state of the network to accommodate new traffic or verify if the QoS are being complied.
- Deliver the network statistics to the SDNapp and other external modules. The statistics are sent to the SDNapp to verify if the QoS requirements are satisfied. Also, this information could be sent to external modules for their knowledge.

Besides the NCL modules, there have also been defined a set of internal and external interfaces. Internal interfaces enrich the NCL performance while external interfaces are critical to retrieve E2E QoS application layer requirements and configure the network layer according to them.

B. NCL Internal Interfaces

a) QoS SDNapp - SDN_Controller interface: the controller periodically updates to the Resource allocator about the available network resources, so that the latter is aware of the updated network topology and the usage of the resources. Besides, the SDN controller sends the OpenFlow rules to the switches to dynamically accommodate applications´ requirements.

b) SDN_Monitor interface – QoS SDNapp: the monitor sends statistics to the QoS SDNapp to inform about the network state.

*c) SDN*_Controller - SDN_Monitor interface: the controller retrieves network statistics through the OpenFlow protocol and sends it to the monitor.

C. NCL External Interfaces

Northbound interface:

The northbound interface is formed by two different interfaces: The

Northbound_SDN_Application_Interface (NSDN_App_Iface) and the Northbound_SDN_ Monitor_ Interface (NSDN_Mon_Iface)

The NSDN_App_Iface enables the application layer to request connectivity services. The Service Level Agreements (SLAs) signed between the service consumer and the providers are mapped into QoS network parameters, specifically bandwidth, delay and jitter. QoS requirements are then forwarded to the NCL by means of this interface.

Additionally, end-users may need to be able to monitor the provisioned service while running their applications. The SDN_Monitor makes user of the SDN_Monitor_Iface to request for network statistics to the SDN network resources. This interface also pushes the retrieved statistics from the network to the SDN_Monitor. Thus, the NCL pulls the statistic counters of the OpenFlow switches, to obtain QoS metric values. Besides, the SDN_Controller enables periodically “per flow” measurements so that each end-user receives the monitoring information that

attains to its own service, yet preserving the system scalability.

Southbound interface:

The Southbound interfaces are the Southbound_SDN Application_Interface (SSDN_App_Iface) and Southbound SDN_Monitor_Interface (SSDN_Mon_Iface).

Through the SSDN_App_Iface, the dynamic network configuration between the NCL and the network devices is enabled. This interface pushes the routing decisions taken at the SDN_Controller to each OpenFlow switch by means of the OpenFlow rules. Indeed, since OpenFlow protocol is used, the secure channel is the interface that connects each OpenFlow switch to a controller allowing switch configuration, switch events and packets reception.

The SSDN_Mon_Iface pushes to the switches the OpenFlow statistics request and retrieves monitoring information from the network switches to send it to upper layers.

Inter-Domain interface:

It is intended that NCL components can be deployed, in a multi-domain environment, at each organization's site. This implies that there are a number of inter-domain actions that the components need to support. The SDN_Monitor enables customers to retrieve QoS measurements related to their use of up-stream resources. Similarly, the QoS SDNapp module must allow customers to reconfigure or change their use of up-stream resources allowed under the terms of the supplier QoS (e.g., by registering new flow types and/or changing flow priorities).

The coordination, control and application information exchange across multiple SDN domains is currently being investigated at the IETF in the context of the SDN interface protocol (SDNi) [11]. Other initiatives are investigating the usage of the OGF NSI protocol for SDN controllers inter-domain communication. The NCL Inter-Domain interface protocol used can be considered an "east-west" protocol between SDN controllers. The main functionalities that provides include the exchange of the inter-SDN routing information and the coordination of the flow setup across multiple SDN domains. It is worth underlying that, since SDN architectures are evolving and changing rapidly, inter-domain SDN implementations need to be open and generic enough to support not only the capabilities offered by different types of the existing controllers but also the potential future ones that could be required by new data types or controllers. This important requirement is met by using meta-data exchange for all types of information including also unknown capabilities. In particular, the message types exchanged via this interface include:

- Reachability and Capability updates (including bandwidth, Quality of Service (QoS), system and software capabilities available inside the domain).
- Flow setup/teardown/update request.

IV. NCL OPENNAAS IMPLEMENTATION

The integration platform that have been used for implementing the different modules of the NCL is the OpenNaaS framework. In this section we provide with a description of such implementation.

A. OpenNaaS

OpenNaaS (Open Network as a Service) is an open software platform for resource management (see Figure 2). It consists of a modular architecture distributed in three main layers: Platform, Resource Layer, and Remote applications. The OpenNaaS software is released with a dual Lesser General Public License/Apache Software Foundation (L-GPL/ASF) licensing schema assuring not only that the platform will remain open and consistent but also the possibility to build commercial derivatives on top. Some of the most remarkable features include:

- A sophisticated manner of on demand provisioning of network resources.
- Recursive delegation of access right over managed resources.
- Abstraction of resources from the underlying infrastructure hiding the vendor technological details.
- Embedding instantiation of virtualized resources in the regular Business Support Systems (BSS) workflows.

OpenNaaS enables on-demand provisioning of network resources by means of device abstraction and enables the use of web services to expose such resources to upper architecture layers. The software is implemented based on two key components: resources and capabilities. The resources contain the information regarding the logical model of the device. The capabilities represent its features.

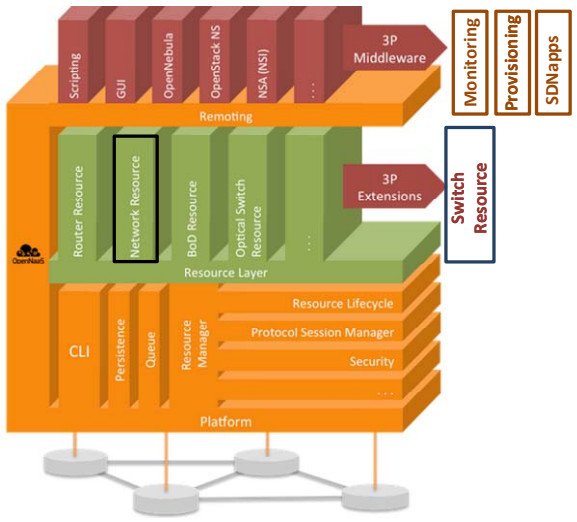


Figure 2. OpenNaaS architecture with new added resources.

Currently, OpenNaaS supports different types of resources such as router, BoD (Bandwidth on Demand), optical switch and mac-bridge. The modular architecture of OpenNaaS allows the implementation of new types of resources and capabilities. Thus, NCL enhances the OpenNaaS framework as the NCL extends it with SDN capabilities.

B. *NCL software architecture*

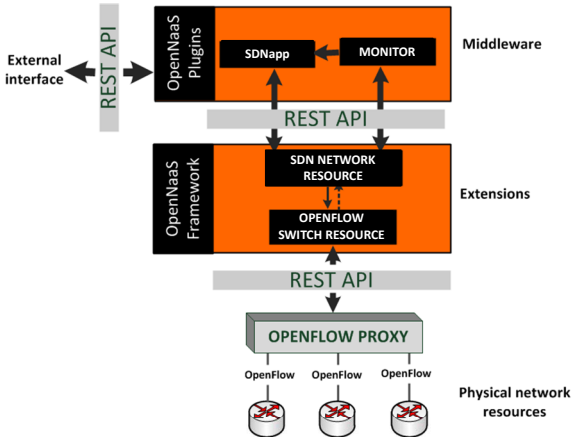


Figure 3. OpenNaaS NCL implementation

Figure 3 shows the preliminary OpenNaaS NCL software architecture inserted in a multilayer generic framework model: the lower layer comprises the network physical resources. On top of it, an open source OpenFlow controller is used to communicate with the OpenFlow resources and acts as a proxy. For this design, the FloodLight controller [13]implementation is used to provide the OpenFlow communication protocol as a capability, so it is implemented as a proxy in the NCL architecture. It sends the OpenFlow rules to the physical resources. Floodlight has been chosen since it implements the OpenFlow protocol and it has been developed in Java making it easier to integrate with OpenNaaS and extend if necessary. Finally, OpenNaaS control and management framework implements upper and lower bound interfaces in form of REST APIs to expose resources to the application layer and trigger actions to the underlying resources.

C. *NCL use case*

In this section we present a use case built on top of previous described framework. This use case is the starting point of a provisioning service that can be offered by service providers, obtaining the benefits mentioned in previous sections. The use case main goal is to enable on demand QoS- guaranteed service provisioning by connecting using SDN technology based on OpenNaaS framework.

By means of the REST API, a certain external application requests a QoS-guaranteed provisioning service between two end points (1); it is assumed that such external application previously performs a secured access to the system. The Application request engine function of the QoS SDNapp checks that the requested QoS matches the service level agreements and accepts the service provisioning request. The Provisioning module allocates the required network resources demanded by the applications to guarantee the expected QoS and forwards the decision to the Resource allocator module that performs the intelligence on top of the forwarding plane (2). Then, the SDN Network resource calculates the best route based on the knowledge of the network (topology and status) and previous provisioning module information (3).

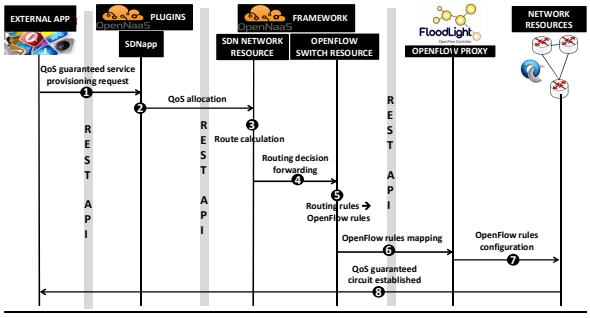


Figure 4. QoS guaranteed provisioning service workflow.

The network resource pushes the routing decision to each OpenFlow switch resource (4) and the switch resource SDN capabilities translate the routing rules in OpenFlow rules (5). The switch resource maps the OpenFlow rules into the controller through the REST API interface (6) and eventually, the OpenFlow proxy uses OpenFlow protocol to send the OF rules to the physical network resources (7) in order to provision and configure the circuit with the QoS agreed levels (8).

After the provisioning mechanism finishes, the monitoring process is periodically performed, so that in case the QoS of the provisioned service is degraded below certain service level agreement threshold, the system triggers a re-provisioning mechanism starting again from step (1) until the QoS levels are again restored. Figure 4 shows the workflow operation.

V. CONCLUSIONS

In this paper, we have presented a prototype implementation of an SDN based network controller that aims to solve QoS limitations while provisioning E2E services. The NCL architecture solution relies on SDN and OpenNaaS technologies and allows exploiting the current deployed network with a more efficient usage of the resources avoiding network over-provisioning. Moreover, through an SDN enabled network, network operators could offer to the network applications development and providers' community a way to leverage the network intelligence and help deliver differentiated services. The programmability of the network increases its efficiency and makes it more effective for customer necessities. Also, the NCL enables network elasticity functionalities allowing to perform the dynamic provisioning of the network on-demand, to quickly scale out and rapidly release to quickly scale in. SDN capabilities will allow an automatic provision process which dynamically adapts to application requirements on-demand during run-time. The work presented in this paper represents a milestone towards demonstrating the feasibility and viability of implementing SDN applications tailored to the network operator service portfolio.

Future work includes NCL performance testing and framework enhancement. The initial tests have been deployed in a Mininet-based scenario and future evaluation will be performed in a real test-bed provided by the OFELIA project [14]. These experiments may gather feedback for future improvements in the NCL software. Among others, enhancements will include the addition of specific TE algorithms to the TEDB and the exploration of how the IPv6 Flow Label may offer advantages to flow management for IP traffic. Also, we will investigate novel multicast mechanisms within SDN-based networks for interactive applications and the requirements of inter-SDN controller communication for inter-domain management.

ACKNOWLEDGMENTS

Our research has received funding from the EC's 7th Framework Programme under grant agreements 261527 (Mantychore) and 318665 (OFERTIE).

REFERENCES

[1] https://www.opennetworking.org/sdn-resources/sdn-definition

[2] OpenFlow consortium website: http://www.openflow.org/

[3] http://www.opennaas.org/

[4] http://www.mantychore.eu/

[5] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an Overview," RFC 1633, Internet Engineering Task Force, June 1994.

[6] S. Blake,M. Carlson, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", RFC2475, Internet Engineering Task Force, December 1998.

[7] Blake, S., et al.: An Architecture for Differentiated Services, RFC 2475 (1998)

[8] R. Rosen, Cisco Systems, Inc. R. Callon, Juniper Networks, Inc. "Multiprotocol Label Switching Architecture. ," RFC 1633, IETF January 2001.

[9] Yu, J., Al-Ajarmeh, I.: Call Admission Control and Traffic Engineering of VoIP. In: Second Intenational Conference on Digital Telecommunications, IEEE ICDT 2007 (2007)

[10] S. Oueslati and J. Roberts, France Telecom R&D. A new direction for quality of service: Flow-aware networking.

[11] http://tools.ietf.org/html/draft-yin-sdn-sdni-00. SDNi: A Message Exchange Protocol for Software Define Networks (SDNS) across Multiple Domains draft-yin-sdn-sdni-00.txt

[12] J. Yu and I. Al Ajarmeh, An Empirical Study of the NETCONF Protocol," in Sixth International Conference on Networking and Services (ICNS), 2010, pp. 253-258, Mar. 2010.

[13] Floodlight OpenFlow Controller http://floodlight.openflowhub.org/

[14] http://www.fp7-ofelia.eu/

PolicyCop: An Autonomic QoS Policy Enforcement Framework for Software Defined Networks

Md. Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo
[mfbari| sr2chowdhury | r5ahmed | rboutaba]@uwaterloo.ca

***Abstract*—Network management is becoming increasingly challenging with the relentless growth in network size, traffic volume, and the diversity in QoS requirements. Traditionally, the concept of predefined Service Level Agreements (SLAs) has been utilized to establish QoS parameters. However, state-of-the-art technologies in this area are both proprietary and inflexible. To this end, Software Defined Networking (SDN) has the potential to make network management tasks flexible and scalable, and to provide an open platform to encourage innovation. In this paper, we present *PolicyCop* – an open, flexible, and vendor agnostic QoS policy management framework targeted towards OpenFlow based SDN. PolicyCop provides an interface for specifying QoS-based SLAs and enforces them using the OpenFlow API. It monitors the network and autonomically readjusts network parameters to satisfy customer SLAs. We present experimental results to demonstrate PolicyCop's effectiveness in ensuring throughput, latency, and reliability guarantees.**

I. Introduction

Network management systems are becoming more complex and sophisticated in order to support continuously evolving online applications and their QoS requirements. A wide variety of online and real–time applications, like video streaming, video–on–demand, online interactive gaming, video conferencing, virtual collaborative environments *etc*. have emerged over the last decade. These applications impose diverse QoS requirements on latency, throughput, error-rate, jitter, and redundancy. In addition, the relentless growth in network size, sophistication and geographic span have lifted the network management challenges to the next level. The convergence of these services impose strict performance isolation with stringent and fine-grain constraints on the underlying switching fabric. As a direct consequence, network management systems are continuously challenged to satisfy these ever increasing requirements while adhering to the resource constraints.

Traditionally, the concept of predefined Service Level Agreements (SLAs) has been utilized to establish QoS parameters for traffic management. Usually, each SLA consists of a set of Service Level Objectives (SLOs). Network managers use the SLOs to derive network level policies, which are in turn translated into device level primitives (*e.g.*, forwarding rules, queue configurations, packet dropping rate, traffic shaping policies, *etc*.). Policy-based QoS management is a well investigated topic in network management. A good number of research works have proposed autonomic policy based management systems [12, 24, 28, 29, 32]. However, most of these approaches are based on either DiffServ [1] or MPLS-DiffServ [25], which come with a number of problems. Firstly, they offer static traffic classes with a coarse granularity of QoS levels. Secondly, they require installation of specialized software or hardware components in the network. Automation approaches based on specialized policy specification languages like Ponder [28, 29] can be applied to other QoS techniques like MPLS–TE. However, this kind of methods require proprietary policy servers and policy configuration agents to be deployed on network devices.

Recently Software Defined Networking (SDN) has emerged as a promising approach for providing flexible network programmability. It facilitates dynamic configuration, operation and monitoring of a network. SDN proposes to separate the network's control plane from the data plane and provide a single system image of the control plane to the data plane [13, 15]. However, the controller itself can be implemented as a distributed [11, 17, 18, 22, 30] system. This separation enables rapid network application development. OpenFlow [9] has become the *de facto* standard of communication between the control and data planes. The control plane provides a rich northbound API and a global view of the network, which provides the network operator a programmatic and elegant way of dynamically implementing a wide-range of network policies (*e.g.*, fault-tolerance, accounting, routing, security, *etc*.) and rapidly deploy new services.

In this paper, we present the design and implementation of *PolicyCop*, an autonomic QoS policy enforcement framework based on SDN. PolicyCop provides an interface for specifying QoS policies and exploits the northbound API of SDN controller to enforce

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

them. PolicyCop also takes advantage of the control applications to monitor the compliance of the policies and autonomically adapts the control plane rules with changing traffic conditions.

The rest of the paper is organized as follows. First we discuss our motivation in Section II, then we present related work on policy based QoS management in Section III. Next we describe the architecture of PolicyCop in Section IV. We present experimental evaluation results in Section V to demonstrate the effectiveness of PolicyCop and finally, we conclude with some future research directions in Section VII.

II. Motivation

Providing appropriate QoS to user traffic is primarily related to managing routing mechanisms, metric, and protocol updates. It may seem that the management of routing has been completely automated. But in reality, network routing is not automated despite a plethora of protocol and software. Establishing and maintaining routes still remains one of the most challenging aspects of network management [31]. Typical routing protocols running on a collection of distributed switches or routers can present obstacles when a manager needs to override the choices made by the protocol. PolicyCop coupled with the flexible programmability features offered by SDN, enables the network manager to describe his requirements as high level network-wide policies, which are implemented, monitored, and enforced by PolicyCop. Now, instead of running a distributed routing protocol that is hard to control, the switch forwarding table can be populated by the SDN controller based on the high level policies defined by the network manager. The advantages offered by PolicyCop over traditional autonomic QoS frameworks are described below:

- *Per flow control and dynamic flow aggregation:* Traditional QoS architectures aggregate traffic based on the Type of Service (TOS) field in IPv4 packets, or the Class of Traffic field in IPv6 packets. However, OpenFlow permits us to provide per flow QoS control in a scalable and flexible manner.
- *Flexible programming model:* PolicyCop has a layered architecture and the APIs between the layers are defined using JSON based RESTful API. This enables the option for using different programming languages in different layers in the framework. A network administrator can introduce new modules at any layer in PolicyCop to deploy new services as long as these modules implement the standard APIs defined by PolicyCop.
- *Dynamically configurable traffic classes:* DiffServ and MPLS–DiffServ offer a predefined number of traffic classes. In contrast, PolicyCop can define new traffic classes at runtime without requiring to bring down any network device or service.
- *Reduced operational overhead:* Traditional QoS services require a number of concurrent protocols to run for performing tasks like routing, MPLS label exchange, resource reservation, *etc*. This problem is well known as the *Protocol Clutter* problem. PolicyCop avoids this issue by mediating all communication between network services and network devices over the OpenFlow protocol. PolicyCop exploits this feature to bring a new genre of services for a network operator, *e.g.*, an operator can use PolicyCop to offer a combination of guaranteed QoS for VoIP traffic and best effort service for HTTP traffic to the same customer.
- *Ease of deployment:* PolicyCop can be deployed in a network consisting of switches from different vendors as long as all switches support the same OpenFlow version. It does not require any software or hardware modifications. Its components can be deployed on the same or different physical machines based on scalability requirements.

III. Related Work

Various policy enforcement frameworks have been proposed for adaptive and autonomic service management in QoS enabled networks [10, 28, 29]. Most of the existing works target inflexible QoS architectures like DiffServ or MPLS–DiffServ, which lack broader network picture, reconfigurability, and adaptivity [23]. Our work aims at bringing together the flexible programmability and monitoring capabilities of SDN with autonomic policy-based service management to enable a network operator to provide better service offerings.

The IETF policy working group has proposed a QoS management framework using the X.500 directory where policies are represented as *if–then–else* rules [10]. However, the network level policies cannot be directly mapped to devices. This mapping functionality has to be done by relay nodes. Moreover, the interaction between the application and the network policy have been ignored. In [28, 29] the authors propose a policy automation framework based on Ponder (a policy specification language). However, it requires specialized policy servers and installation of additional software modules in the network devices.

DiffServ based policy adaptation frameworks are introduced in [12, 24, 32]. DiffServ provides a fixed number of pre–configured traffic classes, which cannot be used in practice due to the ever changing nature of traffic. Moreover they require custom scripts to be downloaded to network devices to adapt to traffic conditions. During this download time, devices cannot provide regular traffic forwarding functionality.

There has been also some work in the SDN realm related to policy management. However, most of these works focus on either static [20] or dynamic [16, 19, 21] checking for policy rule installation in network devices. To the best of our knowledge this is the first work that explores autonomic QoS policy enforcement framework in the realm of SDN.

IV. Proposed Framework

As depicted in Figure 1(a), PolicyCop's architecture is organized in three planes: data plane, control plane, and management plane. The data plane (Section IV-A) consists of OpenFlow enabled switches. The control plane (Section IV-B) contains one or more OpenFlow controller(s). We have developed a few *control applications* that provide different control functions to the management plane. The management plane (Section IV-C) is divided into two components: (i) Policy Validator, and (ii) Policy Enforcer. The former monitors the network to detect policy violations and the later adapts control plane rules based on current network conditions and policies.

A. Data Plane

In this work, we are assuming that the underlying network is built from OpenFlow enabled switches. In this section we provide an overview of the QoS related features in OpenFlow protocol that we have used.

OpenFlow API: OpenFlow defines a traffic flow as a sequence of packets having the same 12-tuple containing switch ingress port, Ethernet MAC addresses (source and destination), Ethernet type, VLAN id, VLAN priority, IP addresses (source and destination), IP protocol, IP Type of Service (ToS) bits, and TCP/UDP ports (source and destination). These fields either contain exact values or wildcards to match a set of values. The OpenFlow specification also provides standardized APIs for the controller to manage the data plane switching fabric. Each switch connects to the controller over a secure TCP channel. The controller uses the OpenFlow API to install traffic forwarding rules in the switches' flow tables, discover network topology, monitor flow statistics, and track device up/down status.

QoS features: Starting from OpenFlow Specification 1.0 [6], packets belonging to a flow can be enqueued in a particular queue of an output port. The queue can be configured through standard protocols like SNMP, CLI and NetConf [3]. Controllers can query configuration and statistics parameters from the existing queues. These switches can rewrite the IP ToS field in the IP header, which can be used to implement QoS mechanisms like DiffServ [1]. Support for rewriting the Explicit Congestion Notification (ECN) bits has been incorporated since OpenFlow version 1.1.0 [7].

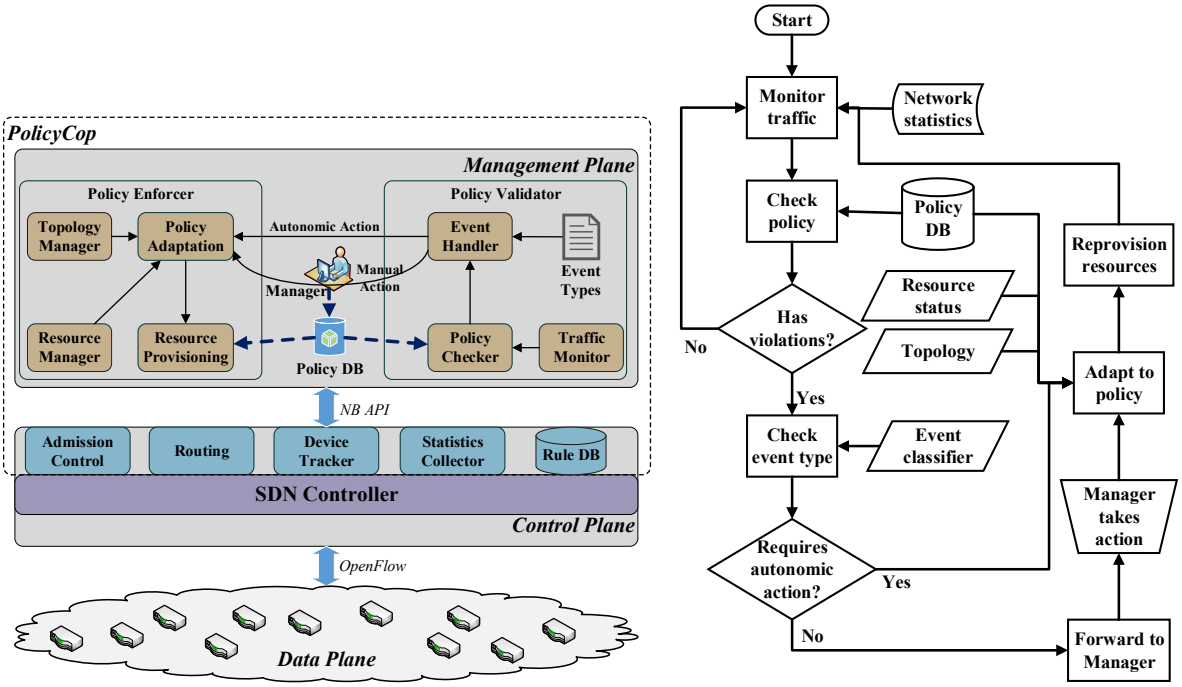
Open Networking Foundation (ONF) has created an auxiliary protocol called OF-Config [5] to support configuration of various features of an OpenFlow switch. OF-Config can be used to configure the minimum and maximum transmission rates of a queue in an OpenFlow switch. According to OpenFlow specification 1.2 [8], an OpenFlow controller can also read these rates from a switch. The most recent QoS related additions in OpenFlow are *meter tables* and *meter bands*, which can be used to limit the transmission rate of an output port. A meter band specifies a transmission rate and an actions set to be performed once the specified rate has been exceeded. Meter tables are used to store a collection of meter bands. Multiple meter bands can be associated with a single flow entry. Based of the transmission rate of the flow(s) matching this entry one or more meter band action(s) can be triggered.

In terms of monitoring, an OpenFlow controller can query a switch for statistics at different aggregation levels: table, flow, port, and queue. *Table level* statistics provide information regarding an entire flow table. *Flow level* statistics provide information about a particular flow, *e.g.*, how many bytes were matched against this flow, how many packets were forwarded, how many packets were dropped, how many errors occurred, duration for which this flow entry was active *etc*. *Port level* statistics provide more specific information about a particular port. *Queue level* statistics provide information about how many bytes and packets were enqueued at a particular queue attached to a particular output port, how many packets were dropped, duration for which this queue was active *etc*. In OpenFlow specification 1.3.0, support for querying meter level statistics was also added. *Meter level* statistics contain similar information *e.g.*, how many bytes and packets were forwarded, duration of this meter *etc*. To summarize, OpenFlow provides extensive support for configuring and monitoring QoS related features that can be used to overcome un-necessary complexities introduced by distributed routing and traffic engineering mechanisms.

B. Control Plane

The control plane consists of one or more OpenFlow controller(s) and a set of controller applications that implement different network functions, *e.g.*, admission control, device tracking, statistics collection, routing *etc*. These functions are implemented as pluggable software modules on-top of the OpenFlow controller, which typically provides either a RESTful or language specific API (henceforth referred as North Bound API or NB-API). The management plane uses these control applications to implement policy validation and enforcement. The management plane translates high-level network-wide policies to low-level rules, called *control rules* in the rest of this paper. Control rules can be used by an SDN controller to compute the FIB entries (or flow entries) for each network device.

PolicyCop requires four control applications and a database for storing control rules. These components are explained in detail below:



(a) Architecture (b) Workflow

Fig. 1. PolicyCop architecture and workflow

Admission Control: This application receives resource provisioning requests from the management plane and decides whether to accept or reject the request. It uses the SDN controller's NB-API to provision the requested resources in network devices. The NB-API can be used to reserve network resources like queues, flow-table entries, bandwidth, *etc*. If the network devices have adequate resources then the resources are provisioned and the application accepts the request from the management plane, otherwise the request is rejected.

Routing: The routing application determines path availability. It calculates route(s) based on the control rules in *Rule DB*. Suitability of a route to serve a request is determined by network topology and a collection of performance metrics like latency, throughput, error-rate, jitter and redundancy. The management plane collects these data using the *Statistics Collector* and *Device Tracker* applications.

Device Tracker: This application tracks the up/down status of network switches and their ports by listening to the asynchronous status messages exchanged between the OpenFlow controller and switches. The data collected by this application helps the management plane to maintain a global view of the network.

Statistics Collector: This application uses a mix of passive and active monitoring techniques to measure different network metrics, like bandwidth usage, residual capacity and number of dropped packets, at different aggregation levels, *e.g.*, per flow, per switch port/link, per user, *etc*. It also measures per flow latency, error–rate and jitter by inserting packet probes [26] in the network. We have developed a monitoring framework for SDN, which provides a clean an flexible way of writing monitoring applications that require such statistics in different aggregation levels [14].

Rule DB: The management plane translates high-level network-wide policies to control rules and stores them in the rule DB. The controller and other control applications (*e.g.*, routing) use these rules to compute the flow table entries for each switch.

C. Management Plane

PolicyCop's principle functionality is delivered by the management plane. PolicyCop consists of two components: (i) Policy Validator, and (ii) Policy Enforcer. These two components comprise a feedback loop for enforcing SLAs. The Policy Validator monitors the network to detect policy violations, while the Policy Enforcer adapts control plane rules based on network conditions and high-level policies. A network manager can specify network-wide policies that are stored in the *Policy DB*, which is a general purpose database for storing policies. It can be implemented using an LDAP server, a NoSQL key-value store, or a relational database. Management plane components are explained in detail below:

1) **Policy Validator:** The policy validator component periodically collects network traffic data and detects policy violations. In case of a violation, it forwards an action request to either the autonomic policy

adaptation module or the network manager based on the violation type. This component consists of three modules, which are explained below:

Traffic Monitor: This module collects the active policies from *Policy DB*, and determines appropriate monitoring interval, network segments and metrics to be monitored. Based on traffic characteristics, monitoring data might be collected more frequently from some switches compared to the others. This module utilizes the statistics collector application to collect data.

Policy Checker: The objective of this module is to identify policy violations. This module collects data from both the *Policy DB* and *Traffic Monitor*. It then analyzes the collected data to identify policy violations and forwards them to the *Event Handler*.

Event Handler: This module examines the violation events. It uses a pre-specified list of "Event Types" to determine the severity of a violation event. Depending on the event type, an action request is either forwarded to the network manager for manual action or to the policy adaptation module for autonomic action.

2) **Policy Enforcer:** The objective of this component is to re-provision network resources to adhere to the network-wide policies once the policy validator component detects a policy violation. The policy enforcer consists of the following four modules:

Topology Manager: This module collects data from the device tracker application in the control plane. It maintains a complete view of the network, which is used by the policy adaptation modules to make resource re-provisioning decisions.

Resource Manager: This module keeps track of currently allocated resources in the network using the admission control and statistics collector control applications. The data collected by this module can be used by other modules to make informed decisions when reconfiguring the network.

Policy Adaptation: The policy adaptation module consists of a set of Policy Adaptation Actions (PAAs). PAAs are distinguished by the type of metric that has been violated. A separate PAA is designed for handling each type of policy violation. For example, latency violation and throughput violation events are handled by separate PAAs. Table I shows the functionality of some example PAAs. The PAAs are pluggable components and the network manager can specify them though PolicyCop's programming interface.

Resource Provisioning: This module re-provisions network resources when a policy violation occurs. The policy adaptation module invokes this module, and provides necessary details for provisioning. The resource provisioning module either allocates more resources or releases existing ones or both based on the violation event. The policy adaptation module also provides data regarding the QoS nobs to be tuned, and where these changes should be applied.

The process workflow in PolicyCop is shown in Figure 1(b). The traffic monitoring module collects network statistics through the statistics collector application in the control plane. This data is used by the policy checker module to detect policy violations. If no violation is detected then the policy validator just keeps monitoring the network without taking any action. If a violation is detected then the event is forward to the event handler module. The event handler examines the violation event and forwards it either to the network manager or to the policy adaptation module. If the event requires manual intervention, then the network manager chooses appropriate actions based on the event, its corresponding data, and current network condition. On the other hand, if the event can be handled by the autonomic handler in the policy adaptation module, the violation event is directly forwarded to the policy adaptation module. This module determines the appropriate action based on the event type, current network topology, resource allocation, traffic condition and informs the resource provisioning module to reallocate network resources. The resource provisioning module makes the appropriate changes in the network devices to enforce the contracted policy.

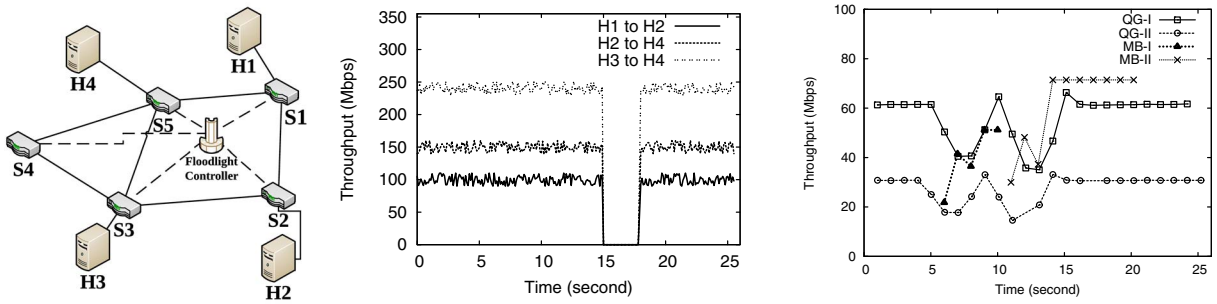
V. Experimental Evaluation

We deployed a test network consisting of five Open vSwitches [4] (OVS) and four hosts. Each OVS runs on a separate physical machine. OVS switches are inter-connected using GRE tunnels. Link bandwidth and delay are simulated using the Linux `tc` command. All OVSs are controlled by a Floodlight controller [2] as shown in Figure 2(a). The hosts run *iperf* servers and clients to generate traffic. We performed two experiments for demonstrating how PolicyCop reacts to link failure and policy violation related to throughput. In this section, we present the obtained results.

Link failure: For this experiment, we started six TCP flows between each pair of physical hosts: H1 to H2, H1 to H3, H1 to H4, H2 to H3, H2 to H4, and H3 to H4. Three flows passed through the link S3-S5: H1-S1-S5-S3-H3, H2-S2-S3-S5-H4, and H3-S3-S5-H4. Bandwidth requirements of these flows are 100 Mbps, 150 Mbps, and 240 Mbps, respectively. Now, after 15 seconds we physically disconnected the link S3-S5, which dropped the throughput to zero (see Figure 2(b)). PolicyCop detected this event as the three flows mentioned above triggered violations. These events were handled by the policy adaptation module and the three flows were re-routed though alternative links. H1-S1-S5-S3-H3 was re-routed through H1-S1-S2-S3-H3, H2-S2-S3-S5-H4 was re-routed through H2-S2-S3-S4-H4, and finally H3-S3-S5-H4 was re-routed through H3-S3-S4-S5-H4.

SLA Parameter	PAA Functionality
Packet loss	Modify queue configuration or reroute to a better path
Throughput	Modify rate limiters to throttle misbehaving flows
Latency	Schedule flow though a new path with less congestion and suitable delay
Jitter	Reroute flow though a less congested path
Device Failure	Reroute flows though a different path to bypass the failure

TABLE I. FUNCTIONALITY OF SOME EXAMPLE POLICY ADAPTATION ACTIONS (PAAS)



(a) Experimental setup (b) Effect of link failure (c) Flow throughput

Fig. 2. Throughput and bandwidth usage

Throughput: In this experiment, we started two flows with soft QoS guarantee (QG-I and QG-II) from H1 to H3 along path H1–S1–S5–S3–H3 at epoch 1s (Figure 2(c)). The maximum capacity of each link is capped at 100 Mbps using the Linux *tc* command. QG-I and QG-II were transmitting at 60 Mpbs and 30 Mbps, respectively. To introduce interference we started two misbehaving flows: MB-I from H2 to H4 at time 5s along path H2–S2–S1–S5–H4 with a duration of 6s and MB-II from H4 to H2 along the same path at time 10s with a duration of 10s. MB-I and MB-II transmit at 50 Mbps and 70 Mbps, respectively. The throughput of these four flows are shown in Figure 2(c). When MB-I started at 5 ms, the throughput of both QG-I and QG-II degraded. This was detected by PolicyCop at the 10th second (as the monitoring interval was set to 5s) and MB-I was rerouted though S2–S3–S4–S5 and QG-I and QG-II returned to their expected throughput. However, MB-II started at 10s, again reducing the throughout of QG-I and QG-II. This event was detected by PolicyCop and MB-II was rerouted to restore the throughput of QG-I and QG-II.

For our current experiments we have considered simple policy adaptation algorithms as outlined in Table I. From the results presented above, we can see that PolicyCop can detect policy violations and quickly react to overcome the problem.

VI. ACKNOWLEDGEMENT

This work was supported by the Natural Science and Engineering Council of Canada (NSERC) in part by the NSERC Discovery program and in part under the Smart Application on Virtualized Infrastructures (SAVI) NSERC Strategic Network.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented the design of PolicyCop, an autonomic QoS policy enforcement framework for SDN. To the best of our knowledge, this is the first attempt towards an autonomic QoS management framework for SDN. We have demonstrated the effectiveness of PolicyCop through a working implementation and a set of representative experiments in real network. PolicyCop's capability in autonomic adaptation of simple QoS specifications is well reflected in the experimental results.

The next step in our work is to complete the design and implementation of all the components of PolicyCop, streamline the interfaces between different components, provide a full-fledged collection of controller applications on top of an SDN controller, and experiment on an OpenFlow testbed we have developed [27]. We envision that our work will produce a generic collection of control applications while creating an abstraction layer on top of the control platform for rapid development of network applications that rely on high level primitives.

REFERENCES

[1] DiffServ: RFC 4594. http://tools.ietf.org/html/rfc4594.

[2] Floodlight openflow controller. http://www.projectfloodlight.org/floodlight/.

[3] Network Configuration (NetConf). http://datatracker.ietf.org/wg/netconf/.

[4] Open vSwitch, An Open Virtual Switch. http://openvswitch.org/.

[5] OpenFlow Management and Configuration Protocol (OF-Config) 1.1.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf.

[6] OpenFlow Specification 1.0. http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[7] OpenFlow Specification 1.1.0. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[8] OpenFlow Specification 1.2. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.2.pdf.

[9] OpenFlow Specification 1.3.0. https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.

[10] N. Badr, A. Taleb-Bendiab, and D. Reilly. Policy-based autonomic control service. In *POLICY 2004*, pages 99–102, 2004.

[11] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic Controller Provisioning in Software Defined Networks. In *9th IEEE/ACM/IFIP International Conference on Network and Service Management 2013 (CNSM 2013)*, pages 18–25, Oct 2013.

[12] S. Cabuk, C. I. Dalton, K. Eriksson, D. Kuhlmann, H. V. Ramasamy, G. Ramunno, A.-R. Sadeghi, M. Schunter, and C. Stüble. Towards automated security policy enforcement in multi-tenant virtual data centers. *Journal of Computer Security*, 18(1):89–121, 2010.

[13] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[14] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba. PayLess: A Low Cost Netowrk Monitoring Framework for Software Defined Networks. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014) (To appear)*.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[16] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the debugger for my software-defined network? In *HotSDN 2012*, pages 55–60, 2012.

[17] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 19–24. ACM, 2012.

[18] B. Heller, R. Sherwood, and N. McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, 2012.

[19] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *NSDI 2013*, pages 99–112, 2013.

[20] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: static checking for networks. In *NSDI 2012*, pages 9–9, 2012.

[21] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: verifying network-wide invariants in real time. *SIGCOMM CCR*, 42(4):467–472, Sept. 2012.

[22] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 351–364, 2010.

[23] F. Le Faucheur, W. Lai, et al. Requirements for support of differentiated services-aware mpls traffic engineering. Technical report, RFC 3564, July, 2003.

[24] L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy-based framework for network services management. *Journal of Network and systems Management*, 11(3):277–303, 2003.

[25] I. Minei. MPLS DiffServ-aware traffic engineering. *Juniper Networks*, 2004.

[26] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk Jiri Navratil, and L. Cottrell. pathChirp: Efficient Available Bandwidth Estimation for Network Paths. In *PAM 2003*, Apr 2003.

[27] A. R. Roy, M. F. Bari, M. F. Zhani, R. Ahmed, and R. Boutaba. Design and Management of DOT: A Distributed OpenFlow Testbed. In *14th IEEE/IFIP Network Operations and Management Symposium (NOMS 2014) (To appear)*.

[28] N. Samaan and A. Karmouch. An automated policy-based management framework for differentiated communication systems. *IEEE JSAC*, 23(12):2236–2247, 2005.

[29] S. Shanbhag and T. Wolf. Automated composition of data-path functionality in the future internet. *Network, IEEE*, 25(6):8–14, 2011.

[30] A. Tootoonchian and Y. Ganjali. HyperFlow: A distributed control plane for OpenFlow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010.

[31] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on*, 14(7):1228–1234, 1996.

[32] K. Yoshihara, M. Isomura, and H. Horiuchi. Distributed Policy-based Management Enabling Policy Adaptation on Monitoring using Active Network Technology. In *DSOM*, pages 265–277, 2001.

Towards a distributed SDN control: Inter-platform signaling among flow processing platforms

Francesco Salvestrini, Gino Carrozzo, Nicola Ciulli
Nextwork s.r.l., via Livornese 1027, Pisa Italy
Email: f.salvestrini@nextworks.it, g.carrozzo@nextworks.it, n.ciulli@nextworks.it

Abstract— **Today's Internet is a concatenation of IP networks interconnected by many distributed functions integrated into a plethora of highly specialized middleboxes. These elements implement complex network functions like firewalls, NATs, DPI, traffic scrubbing, etc. The product is a quite complex and rigid internetworking system in which network administrators and users cannot easily determine what is happening to traffic flows as they go toward destinations.**

SDN research and programmable network functions for flow processing and virtualization are unlocking the current scenario, though most of the COTS products generally support network functions only for virtual L2 switching over IP networks (e.g. VXLAN, GRENV, STT) and LAN based flow pinpointing.

This paper presents a different perspective for implementing flow processing via distributed SDN controllers and inter-platform signaling. The distributed end-to-end service provisioning among adjacent flow processing platforms is implemented via a signaling framework in which the different actions/functions to be executed by each platform are described in a generic Flow Processing Route (FPR) object. The FPR is exchanged among the SDN controllers over the end-to-end network service path and contains information on routing rules and local flow processing actions to be instantiated at the different platforms.

The proposed signaling architecture has been designed and implemented in the FP7-ICT CHANGE project. This paper reports on the key signaling architectural aspects and the developed signaling prototype.

Index Terms—Middleboxes, Next generation networking, SDN

I. INTRODUCTION

The Internet has grown to a point that it interconnects billions of users, who run a wide range of networking application and exchange petabytes of data through the network. It is quite unanimous opinion that the Internet – and the packet network in general – innovated lifestyles and made obsolete the communications made possible by the traditional circuit switched telephone and television networks.

In the last decade networks, servers, storage technologies, and applications have all undergone significant changes with the introduction of virtualization, network overlays, and orchestration. Such technologies have allowed network operators and service providers to easily introduce a variety of (proprietary) hardware-based appliances in order to improve their network manageability as well as rapidly launch new services, keeping up with the pace of their users demand. Therefore, the current Internet looks like a concatenation of networks with many distributed functions, implemented via a plethora of highly specialized middleboxes [1]. Examples of such functions are firewalls, which are used to implement the traffic authorization policies in a network perimeter. Other examples are deep-packet inspection (DPI) boxes by which it is possible to identify, classify and possibly rate-limit traffic from certain applications. Many other cases of network flow processing functions exist to implement e.g., Network Address Translation (NAT), traffic scrubbing, etc. .

On the one hand, the deployment in various forms and locations of these flow-aware boxes has improved the overall network manageability; but, on the other hand, it has also resulted in an increasingly complex and rigid network system in which the different functions are bound to specialized and vendor-locked hardware gears. This state of the art does not allow a flexible and dynamic network re-configuration after the service deployment, and it is difficult to customize network architectures and functions based on user-demands changing over time. Since services are typically implemented by the ordered combination of a number of service functions that are deployed at different points within the network, their delivery is currently both a technical and a management challenge that involves significant modification to the network configuration.

Software Defined Networking (SDN), Network Functions Virtualization (NFV) [2], Service Function Chaining (SFC) [3] and programmable network flow processing platforms, possibly based on commodity hardware [4], are emerging with the promise to unlock the current scenario. These technologies are not yet standardized and most of the off-the-shelves products support only network functions for virtual L2 switching over IP networks (e.g. VXLAN, GRE-NV, STT) and LAN based flow pinpointing. The allocation of L3/L4 traffic processing functions (e.g. for security services) is still not fully covered. Also, most of the SDN products adopt a centralized SDN controller to manage the overall programmable network, which further stimulate the never-ending debates on scalability, system resiliency, resource and network ownership, etc.

Based on these assumptions, this paper focuses on a distributed end-to-end service provisioning among adjacent flow processing platforms. In our proposal, the reservation process not only occurs to the routing and switching rules needed for the proper forwarding of the traffic through the programmable network. Instead, it applies also to the allocation of the local flow processing actions that implement

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

the desired network functions in software and distributed among different platforms (e.g. distributed firewall, NAT, DPI, traffic scrubbing, etc.). Key concept behind the proposed signaling framework is the Flow Processing Route (FPR), which is a generic description of the routing rules and local flow processing actions to be exchanged among the flow-processing (SDN) controllers over the end-to-end network service path.

In order to illustrate the inter-platform signaling framework and its capabilities, this paper is organized as follows. In Section II, the basic concepts of flow processing platform and the FPR are defined. In Section III, the high level inter-platform signaling architecture is described with emphasis on the key functional blocks it includes and network interfaces among them. In Section IV, some specific prototype implementation details are provided as a more low-level reference for the reader on the signaling framework developed by the authors. Most of the proposed work originated during the execution of the FP7-ICT CHANGE project [6], and the inter-platform signaling prototype described in this paper is currently under validation in the CHANGE testbed. In Section V, conclusions and hints to future work are given.

II. THE FLOW PROCESSING PLATFORM

In general terms, a traffic flow can be defined as a sequence of network packets sharing common characteristics, often derived from the packet headers. For example, a single TCP connection identifies a flow (5-tuple <srcIP, srcPort, dstIP, dstPort, proto>), as well as an aggregate of TCP connections between two routers is a macro flow, or all the VoIP traffic processed by a router, etc. A network flow can be mono-directional, i.e. directed from the upstream to the downstream platform, or bi-directional, i.e. with traffic and processing required in both directions.

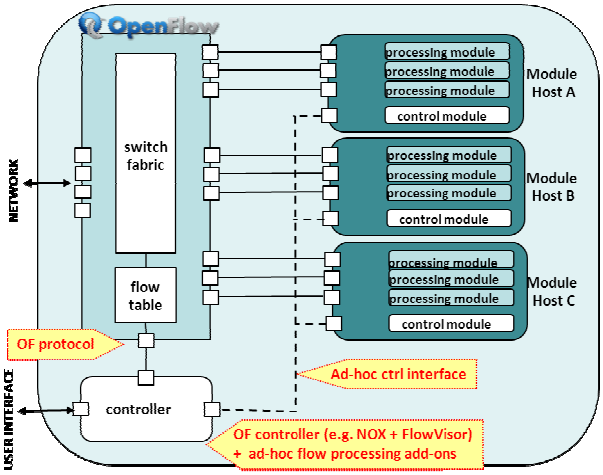


Fig. 1. The Flow Processing Platform as defined in FP7-ICT CHANGE.

A Flow Processing Platform [11, 12] is a key SDN element capable of mangling the data traffic related to a network flow. To properly process the flows, the platform needs to maintain some measure of flow state in the network. For example, flow processing might consist of tuning the routing and QoS to provide low-latency forwarding; similarly, another flow processing might consist of parsing of application-level contents to implement functions like network intrusion detection or traffic/application scrubbing, etc.

Based on the common SDN architecture concepts and the specific processing architecture defined in the FP7-ICT CHANGE project [4], a Flow Processing Platform can be summarized as composed of the functional components presented in Figure 1. Basic elements of the platform are the Processing Modules (PMs), which are either virtual machines[1] or specialized hardware devices (e.g. a DPI box), both able to perform programmable flow processing functions. The PMs run in Module Hosts, which are the hosting servers for the VMs and provide backend primitives for the fast interaction among adjacent PMs and access to the network. The different Module Hosts are interconnected through an OpenFlow (OF) switch [13], which can maintain flow entries to create a chain of PMs for a specific flow processing function. The Platform Controller implements the control logic of the whole platform, managing the phases of commissioning and provisioning CPUs, disks and network links locally in the platform.

A. *Platform setup concepts*

The platform setup process could be logically summarized as the ordered sequence of the following phases: a) Platform commissioning; b) PMs provisioning; c) PMs configuration.

In the commissioning phase, the bare-bone platform is configured. No PMs are allocated to run applications and the startup configurations for the Platform Controller, the OF switch and the module hosts are created and loaded. This commissioning phase is meant to be performed by the platform management authority (e.g. the ISP, the platform owner, etc.). At the end of the commissioning phase the platform is ready to be provisioned with one or more processing modules.

In the provisioning phase the network manager (or the platform itself upon signaling request) "installs" a set of processing modules images. Each PM image contains all the information required to run its flow processing application (i.e. code and data). From the signaling perspective, the PMs provisioning could be activated during the flow processing setup procedures. In that case, only synchronization events have to be exchanged between the Platform Controller (implementing the installation) and the local signaling entity (requesting the instantiation). At the end of the provisioning phase the involved set of processing modules are ready to be configured.

The management functions instantiate a set of processing modules with their specific flow processing configuration (e.g. the filtering rules for a Firewall PM). At the end of the configuration phase, the involved processing modules run the flow processing application.

B. *Platform, flow and service identification*

The end-to-end network service instantiation related to a traffic flow is generally distributed in multiple platforms due to multiple factors, like network geography, load sharing among platforms, etc. Therefore, an end-to-end network service generally requires the configuration of one or more

[1] For example, in the FP7-ICT CHANGE architecture the PM virtual machines are based on Click modular routers.

flow processing platforms depending on the type of service, its features and the available resources on each involved platform. It is critical to have a signaling mechanism that enables the cooperation among platforms for the management of the full network service lifecycle (i.e. service provisioning, modification and deletion). Key to the implementation of this signaling mechanism is the possibility to univocally identify the end-to-end service to be installed, the involved platforms, and the flows in each platform for that service.

The service identifier (SID) is a globally unique identifier for the flow-processing service that is used to univocally refer to and act on the service. The SID is defined within a service provider domain (the first handling the service request in case of multiple domains) and exists once the service has been instantiated.

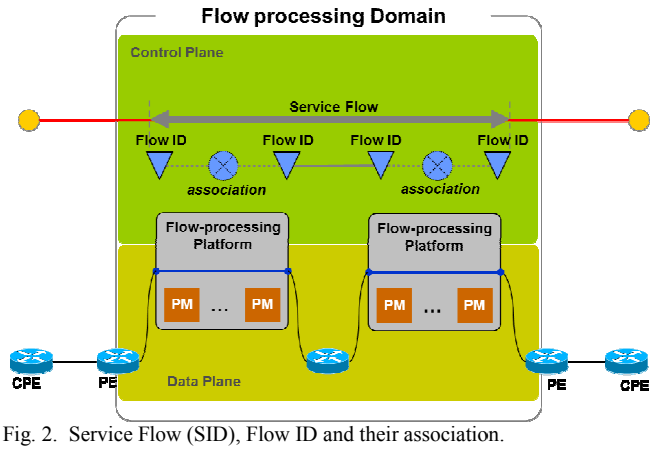


Fig. 2. Service Flow (SID), Flow ID and their association.

To address the issue of platform identification, a unique identifier (Platform Identifier, PID) is assigned to each platform. The PID is conceived to be a domain-local identifier assigned to the platform by the domain administrator, during the platform commissioning phase.

To identify the exact instantiation of the traffic flow at the data plane, a Flow Identifier (FID) can be defined. It consists of the n-tuple used to identify a flow entry in the OF switch. The logical association at a given platform of the flow instantiation at ingress (FID-in) with that at egress (FID-out) of either the OF switch or of a PM allows to fully characterize the flow through a flow-processing platform for the aims of inter-platform signaling (ref. Figure 2).

C. *The flow processing route*

The configuration of a service into a flow-processing domain implies different actions.

At the service layer, the proper PMs for each platform need to be instantiated and configured to implement a specific service (e.g. filtering, monitoring, etc.). In each platform, many PMs could be instantiated and chained together to implement a service part.

Then, at the platform data plane, the flows need to be configured on the platform' OpenFlow switch in order to enable the proper classification and forwarding rules for the traffic. There are at least two flows to configure: one for packets coming in from input port towards the PM (the first in the chain in case of multiple PMs deployed), one from the PM (the last in the chain) towards the platform output port.

Finally, at the network control plane, the mechanisms to perform the proper (ingress) flow attraction and (egress) flow redirection need to be configured, to divert the IP traffic related to that flow towards the first processing platform, and subsequently to guarantee the routing of the flow traffic along the chain of platforms configured. The network control/management plane must coordinate all these actions through control plane entities. The provisioning of a service flow is the result of a composition of different processing actions on different planes/layers.

A generic mechanism to describe the network, the flow application and inter-platform actions for signaling may consist of describing the service flow in the form of a Flow Processing Route (FPR). The FPR is a list of objects summarized in RBNF [16] as in the following:

```
<FPR>      ::= <SID> ( <FPRO> | <FPR> <FPRO> )
<FPRO>     ::= <PID> <ACTIONS>
<ACTIONS>  ::= ( <ACTION> | <ACTION> <ACTIONS> )
<ACTION>   ::= ( ATTRACT-FLOW | REDIRECT-FLOW |
                 PMS-CFG | OF-CONFIG |
                 TUNNEL-INGR | TUNNEL-EGR | … )
<PMS-CFG> ::= ( PM-CFG> | <PM-CFG> <PMS-CFG> )
```

The FPR encapsulates a concatenation of hops (FPR objects, FPRO) which constitutes an explicitly routed path. Using the FPR, the paths taken by flows can be administratively pre-determined or automatically computed by a decision entity. Explicit routing can be used to optimize the resources utilization taking into consideration the current network state and enhance traffic oriented performance characteristics. The ACTIONS term in the grammar, represents a list of processing actions to be performed for the specific service flow on the platform, e.g. BGP/DNS based ingress flows attraction, egress flows redirection, configurations of the OF switch, the PMs involved in the service and their interconnections, tunnels between adjacent platforms [17], etc. Further details on the protocols and mechanisms necessary to combine flow processing platforms, the FPR contents, its complete RBNF grammar and the resulting messages objects are available in [15, 5].

III. INTER-PLATFORM SIGNALING ARCHITECTURE

The overall inter-platform signaling refers to the interactions and procedures occurring for the provisioning of a service across a number of reference points and interfaces. With reference to Figure 3, the following reference points exist in our signaling architecture [5]:

- A Service User to Network Interface (Service-UNI)
- An Internal Node to Node Interface (Internal-NNI)
- An Inter Autonomous System NNI (Inter-AS NNI)

At the Service-UNI, the provisioning request from an end-user is originated. It could be expressed in a protocol independent abstract description, in order not to depend on the underlying data plane implementation. For example, a service like: "*a NetFlow v9 traffic originating from host A has to be counted both in its IPv4 packets and NetFlow records, and finally must be diverted to host C before reaching its final destination host B*" could be expressed as follows:

```
(define-end-point SRC (ip-addr(A), src-port(2055)
                       ip-addr(B), dst-port(9555)
                       ip-proto(UDP),
                       pdu-field16(offset(0),
                                   value(0x0009)))
(define-end-point DST1 (ip-addr(B))
(define-end-point DST2 (ip-addr(C))
(service-setup (SRC (apply-action count-ip-packets)
                    (apply-action count-nf-records)
                    (apply-action divert-to(DST2))
                DST1))
```

Once received, this high-level description is processed and an FPR derived. The resulting FPR will be used to carry the service provisioning information across the domain platforms.

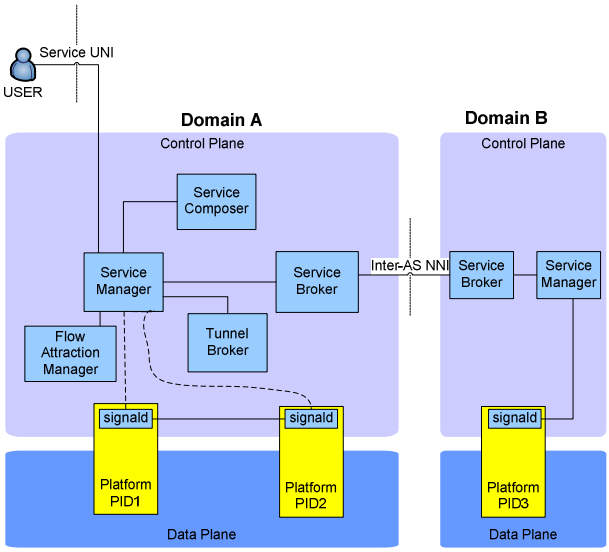


Fig. 3. High-level inter-platform signaling architecture.

When the signaling messages reach the domain borders, the Inter-AS NNI comes into play: each ISP involved in the inter-domain dialogue might want to hide its internal network information and/or perform access-control on the passing requests (either if acting as a "passing-through" or as a "destination" domain). Peering ISPs may then adopt the Internal-NNI protocol and messages to exchange the service provisioning related information. These messages are therefore subject to modifications (i.e. mostly mangling, filtering and injection of protocol objects) that largely depend on the requested service and the ISP policies. As soon as the signaling reaches the last platform in the destination domain, the E2E path might be considered complete.

A. *Signaling Control Network*

The signaling messages across the interfaces mentioned above do need to be exchanged over a communication network that guarantee the reachability of the Platforms in spite of the existence of any network service along the data path. To enable the control plane communication between platforms, it is sufficient to have the platform control modules connected to the signaling control network via a control interface. Through this control interface signaling adjacencies are established. These adjacencies are used by two peering platforms to directly exchange signaling messages. The existence of a signaling adjacency does not imply the existence of a direct physical link between the two adjacent platforms: logical links can be created via IP tunneling or the standard hop-by-hop routing in the signaling control network can guarantee the correct delivery of signaling messages between the two ends.

B. *Inter-platform signaling messages*

Focusing on the Internal-NNI interface which is a core aspect of our flow-processing signaling architecture, the activities supported across this interface can be summarized in the following list:

- **Setup**, to provision a service flow end-to-end
- **Deletion**, to remove a service flow end-to-end
- **Modification**, to modify some features (e.g. QoS profiles) of a provisioned service flow
- **Delegation**, to delegate some flow processing from a platform to another platform for a provisioned service flow end-to-end (e.g. load balancing, distributed firewall, etc.)
- **Notification**, to synchronously retrieve or asynchronously receive status information about a provisioned service flow end-to-end

TABLE I
INTER-PLATFORM SIGNALING ABSTRACT MESSAGES

Activity	Message	Up/Downstream Direction
Setup	Service Setup Request	Up → Down
	Service Setup Allocation	Down→ Up
	Service Setup Confirmation	Up → Down
Deletion	Service Deletion Request	Both
	Service Deletion Response	Both
Modification	Service Modification Request	Both
	Service Modification Response	Both
Notification	Notify	Both
	Notify Acknowledge	Both

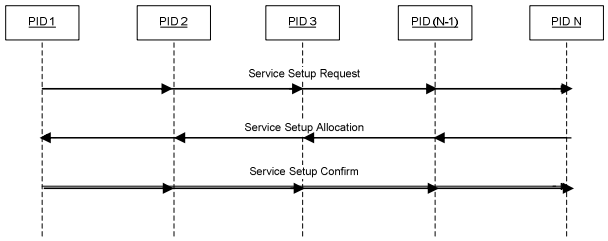


Fig. 4. Setup messages for Inter-platform signaling.

Various procedures could be implemented by combining these basic activities, e.g. migration schemes as in a make-before-break approach, discovery propagation along the path, etc. The specific abstract messages related to each of the above activities are summarized in Table I and Figure 4 for the messages exchanged among peering platforms during the setup phase. Further details on the procedures, the messages syntax and their contents (messages objects) are available in [5].

C. *Functional entities of the inter-platform signaling*

This section briefly describes the key functional entities of the inter-platform signaling architecture presented in Figure 3.

1) Service Manager

The Service Manager is the flow processing domain orchestrator. It processes the service requests received from the Service-UNI clients and coordinates their composition, instantiation and maintenance into the involved platforms. Depending on the service type and composition it may start attracting or redirecting traffic and propagate management and signaling to other network domains through the Service Broker via the Inter-AS NNI. Further details are available in [5, 14].

2) Service Composer

A generic flow processing service could consist of one or more processing modules instantiated into one or more platforms. The service request needs at first to be mapped into specific low-level service profiles indicating the number of components required to perform the whole service (e.g. the types of PMs, their requirements in terms of CPU and RAM requirements). The distribution of these slices of components across the available platforms along the flow processing domain is performed by the Service Composer. This entity slices the service in contiguous parts, generates the corresponding FPROs, assigns them to the domain platforms, inject into the FPR eventual actions required to ensure the data-plane connectivity (e.g. tunnels between adjacent platforms). In order to determine the optimum path, it should take into account the network topology and the capabilities and availability of resource on each platform in the domain.

Note that the composition of the requested service in the inter-domain/AS case must be performed in the visiting domains by their local Service Composer since a domain could perform its slice of the requested end-to-end service, demanding to the downstream domains the processing of what still remains in order to fulfill the user request. Further details are available in [5, 14, 17].

3) Attraction Manager

The Attraction Manager is the entity that manages the traffic attraction for a network domain and mainly operates at the routing layer. This entity performs the attraction from the redirection points to the platform(s), coordinating its actions with the entity that handles the tunnels setup required for the flow delivery from platforms to the grafting points. The coordination of these functionalities is necessary to avoid traffic loops and to ensure that the traffic attraction is started only at the end of the path setup phase.

Different techniques could be used to attract flow to a PM as described in [15] (e.g. FlowSpec-based, BGP-based and DNS-based). Among these, the FlowSpec-based solution seems be the best appropriate for the flow traffic attraction because it provides a finer flow granularity matching any field of the layer 3 and 4 unlike the other two solutions.

4) Tunnel Broker

The composition of a service could span PMs located in different platforms not directly connected. Depending on the network configuration a tunnel could be required in order to create a logical wire between the two platforms and ensure the end-to-end traffic flow as expected. In case the full knowledge of the network topology is available, a platform-to-platform tunnel could be managed directly by the involved platforms. When those complete network information are missing, the platforms local tunnel management should "delegate" the required setup to a Tunnel Broker that could act as a proxy [18].

The Tunnel Broker automatically manages the tunnel creation, modification and deletion requests coming from the platforms (or from the Service Manager, depending on the service and its composition).

5) Service Broker

Platforms belonging to different AS-es do not directly exchange signaling messages, but rather proxy them to an entity, the Service Broker, which manages the inter-AS communications and propagates signaling and management functions between domains. The message types exchanged between two Service Brokers may be almost the same as those exchanged through the Internal-NNI across platforms belonging to the same domain. Nevertheless, there could exist restricted policies for exporting some information (e.g. topology, flow processing route, etc.) towards other AS-es, due to privacy and/or scalability issues. For these reasons, the presence of a Service Broker is required in order to opportunely mangle the messages contents (e.g. masquerade, filter-out FPROs). Further details are available in [5, 14].

6) Signaling Manager

The Signaling Manager [5, 14] is the platform local entity that implements the I-NNI signaling with peering platforms and/or control plane entities. It handles the signaling protocol and only relates to the Platform Controller, depending on the incoming messages contents or internally generated events (e.g. the Signaling Manager autonomously generates *Notification* and *Deletion* messages towards its peers, upon changes in the service parameters or unrecoverable errors notified by the Platform Controller).

Its signaling application is mainly composed by the following logical blocks (ref. Figure 5):

- **Adaptation logic**: this block adapts the communications between the Protocol FSM and the Platform Controller
- **Protocol FSM**: it maintains protocol states and manages transitions, depending on the events (i.e. incoming messages, Platform Controller events such as PMs and MHs provisioning acknowledgements, errors or internal optimizations [15, 17])
- **FPR Management**: this block mainly provides the FPR processing functionalities [7], e.g. it detects if the current platform is the first/last one in a FPR, detects if the current platform is the first/last in the domain, provides message-objects filtering and injection functionalities.
- **Service Management**: it manages the data-model, mainly maintaining the information related to the various (in-progress or already instantiated) services
- **Routing/Service helpers**: The Routing/Service Helpers block handles the FPR completion, routing and service composition for the signaling application logic

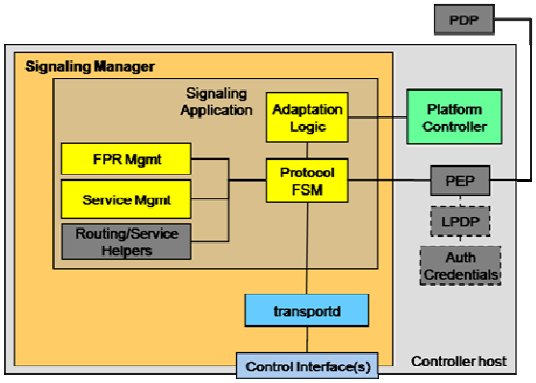


Fig. 5. The Signaling Manager internal architecture

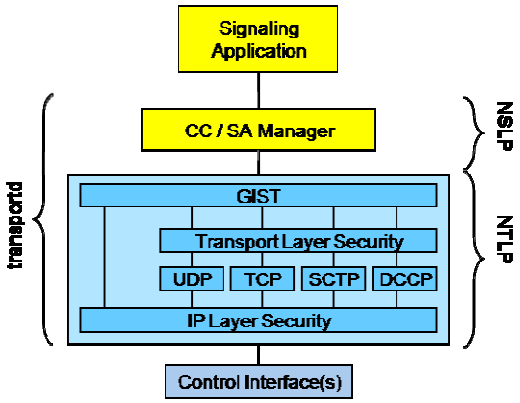


Fig. 6. The transportd, the lower half of all signaling entities

Platform local optimizations (e.g. load balancing the PMs already installed) can autonomously happen if the Platform Controller transparently remaps the PMs without changing the ingress/egress traffic endpoints [17].

IV. Inter-Platform Signaling Prototype

When it turns to the implementation of the signaling architecture described in the previous sections, a software design approach that could be adopted consists of splitting the different functions in each module in two parts: a) those that represent the business logic of the application layer; b) those that implement the message transport layer. Therefore, the application layer handles all the signaling related tasks (e.g. the protocol FSM and the dialogues with the Platform Controllers), while the transport layer manages the low level mechanics related to the protocol messages exchanges among different signaling entities.

We adopted the split approach mentioned above for the implementation of the inter-platform signaling prototype. This allowed to develop once several common aspects of these signaling modules, and to make them available as common communication primitives to the protocol application layer. In our prototype implementation, this split results in the interoperation of two daemons: a transportd for the bare transport of messages (common to Signaling Manager, Service Manager and Service Broker) and a set of signaling applications specific for the different functionalities. To implement the backbone of the transportd we chose the NSIS framework [8, 9, 10] since it natively support the requested split through the NSIS Transport Layer Protocol (NTLP) and the NSIS Signaling Layer Protocols (NSLPs). In particular, we adopted the NSIS-KA framework (http://nsis-ka.org), which provides a fully functional and updated NTLP layer as well as a set of NSLP related libraries which we used to implement the transport layer (transportd), as shown in Figure 6.

The inter-platform signaling prototype has been mostly coded in C++ and Python. Its source code will be released as open source software, available for download from [6].

The prototype is currently under verification in the FP7-ICT CHANGE testbed to validate the signaling scenarios among platforms and the integration of the signaling application with the Platform Controller for the dynamic and distributed configurations of the Processing Modules. Results and reports on the experimentation activities will be publicly available with the final deliverables of the project expected for the end of Q4-2013.

V. Conclusions

This paper has presented an inter-platform signaling architecture to be used for the distributed end-to-end service provisioning among adjacent flow processing platforms. Key aspect of the work is the definition of a Flow Processing Route (FPR) as a way to convey information about routing and local flow processing actions to be instantiated at the different platforms.

The proposed signaling architecture has been designed and prototyped in the FP7-ICT CHANGE project and leverages on the layered NSIS framework to maximize the reuse of the primitive message transport functions.

In a framework of SDN applications mostly centralized and minimal (down to none) interactions among controllers, the proposed signaling framework aims at providing an important background for an east-west SDN interface, to be used for chaining flow processing services among different controllers. A possible evolution of the signaling framework described in this paper will consist of integrating the signaling modules and the FPR concept in the OpenDaylight project (http://www.opendaylight.org).

Acknowledgment

This work has been partly funded by the European Commission through the Project #257422 “CHANGE: Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions”.

REFERENCES

[1] "Is it still possible to extend TCP?". Honda et al. (2011). Internet Measurement Conference. [Online]. Available http://conferences.sigcomm.org/imc/2011/docs/p181.pdf

[2] "Network Functions Virtualisation— Introductory White Paper". ETSI. 22 October 2012 [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[3] Service Function Chaining (sfc) Working Group [Online]. Available https://datatracker.ietf.org/wg/sfc/charter

[4] CHANGE Deliverable D3.3 (revised) - Flow Processing Platform: Main implementation [Online]. Available http://change-project.eu/fileadmin/publications/Deliverables/CHANGE_Deliverable_D3-3_Revised.pdf

[5] CHANGE Deliverable D4.2 (revised) – Inter-platform signaling [Online]. Available http://change-project.eu/fileadmin/publications/Deliverables/CHANGE_Deliverable_D4-2_Revised_01.pdf

[6] FP7 ICT project CHANGE, Enabling Innovation in the Internet Architecture through Flexible Flow-Processing Extensions, www.change-project.eu

[7] RSVP-TE: Extensions to RSVP for LSP Tunnels, *RFC 3209*, *December 2001*

[8] Next Steps in Signaling (NSIS): Framework, *RFC 4080, June 2005*

[9] GIST: General Internet Signaling Transport, *RFC 5971*, *October 2010*

[10] NSIS Signaling Layer Protocol (NSLP) for Quality-of-Service Signaling, *RFC 5974*, *October 2010*

[11] "Flow Processing and the Rise of Commodity Network Hardware", Adam Greenhalgh, Felipe Huici, Mickael Hoerdt, Panagiotis Papadimitriou, Mark Handley, and Laurent Mathy. ACM SIGCOMM Computer Communication Review, Volume 39 issue 2, April 2009.

[12] CHANGE Deliverable D2.4 – Flow processing architecture specification – final version [Online]. Available http://change-project.eu/fileadmin/publications/Deliverables/CHANGE_Deliverable_D2_4.pdf

[13] OpenFlow Switch Specification [Online]. Available https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

[14] CHANGE Deliverable D4.4 – Network architecture: inter-platform communication software [Online]. Available http://change-project.eu/fileadmin/publications/Deliverables/CHANGE_Deliverable_D4-4.pdf

[15] CHANGE Deliverable D4.3 (revised) – Protocols and mechanisms to combine flow processing platforms [Online]. Available http://change-project.eu/fileadmin/publications/Deliverables/CHANGE_Deliverable_D4-3_Revised.pdf

[16] Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications, *RFC 5511*, *April 2009*

[17] CHANGE Deliverable D4.5 – Network Architecture [Online]. http://www.change-project.eu/publications/deliverables.html

[18] IPv6 Tunnel Broker, *RFC 3053*, *January 2001*

OpenFlow rules interactions: definition and detection

Roberto Bifulco, Fabian Schneider
NEC Laboratories Europe
{firstname.lastname}@neclab.eu

***Abstract*—Software Defined Networking (SDN) is a promising architecture for computer networks that allows the development of complex and revolutionary applications, without breaking the backward compatibility with legacy networks. Programmability of the *control-plane* is one of the most interesting features of SDN, since it provides a higher degree of flexibility in network management: network operations are driven by ad-hoc written programs that substitute the classical combination of firewalls, routers and switches configurations performed in traditional networks. A successful SDN implementation is provided by the OpenFlow standard, which defines a rule-based programming model for the network. The development process of OpenFlow applications is currently a low-level, error prone programming exercise, mainly performed manually in both the implementation and verification phases. In this paper we provide a first formal classification of OpenFlow rules interactions into a single OpenFlow switch, and an algorithm to detect such interactions in order to aid the OpenFlow applications development. Moreover, we briefly present a performance evaluation of our prototype and how it has been used in a real-word application.**

I. Introduction

Software Defined Networking (SDN) introduces a new architecture that separates the network *control-plane* from the network *data-plane*. Network equipment, i.e., network switches, is not involved in control-plane operations anymore: the control-plane is implemented in a separated device, usually called *Controller*, which communicates with network switches through well-established interfaces and protocols, introducing greater flexibility in the network, e.g., allowing the programmability of the control-plane. Programmability is actually one of the most important features of SDN: the control-plane behavior can be defined writing "network programs" that manage a set of switches, providing rich network applications and features. Programming the network, instead of configuring it, on the one hand gives a powerful tool to develop revolutionary applications, but on the other, it also increases the difficulty of providing efficient and reliable networks: the flexibility of a program, like in the case of classical computer software, can lead to errors that are even more complex to handle, given the ineherently distributed and asynchronous nature of a network. In some way, SDN is bringing into computer networks the same shift that in past decades electronic devices had from special purpose machines, to general purpose ones. With SDN, the current special purpose network can become a general purpose, hence programmable, network. Programmability, in a new context like computer networks, calls for new programming models, tools and languages. In this sense, to continue with the general purpose computer metaphor, SDN is still in the "machine language" phase, where the programming languages are rudimentary, strictly connected to the hardware, and the main part of network programming is still performed manually, without the aid of any tool. In last years, OpenFlow [8] has been a successful SDN implementation. The Open Networking Foudation [1] (ONF), that is responsible for the OpenFlow specification, currently involves a number of academic and industrial partners, making OpenFlow a promising technology, with a good number of industrial players already implementing the specification in their products. OpenFlow uses a flow abstraction to describe the network behavior, with switches configuration described through the use of flow forwarding rules: control-plane programs are written in a traditional high-level programming language, e.g., Python or Java, to provide a set of entries to be installed at different switches' flow tables (FT) as output. In this paper we present a set of definitions to characterize the interactions of entries installed in an OpenFlow switch, and an algorithm to automatically detect such interactions, with the aim of aiding the development and debugging of OpenFlow network applications. The paper is organized as follows. In section II we present related work. Section III introduces the OpenFlow switches programming model, while in section IV we present our contribution on the OpenFlow rules interaction definitions and in section V we introduce the algorithm we designed to detect such interactions with its evaluation. In section VI some possible application scenarios and a concrete use case are presented. In section VII we conclude and present future work.

II. Related work

OpenFlow network programming is a problem addressed in some other works. Frenetic [4] is a high-level language based on the functional programming paradigm, that provides the programmer with an omniscient, centralized view of the network. A run-time system, linked to the language, "translates" the high-level instructions in a set of low-level packet processing rules, and manages them interacting with network equipments. NetCore [7] is an evolution of Frenetic, that extends the high-level language and provides some improvement in the compilation algorithms and run-time system, trying to

[1] https://www.opennetworking.org/

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

speed up the network performance. To test the correctness of OpenFlow applications, NICE was proposed in [3]. NICE is a tool for automatic OpenFlow applications testing, that combines model checking and concolic execution to explore the state space of OpenFlow programs written for the NOX [5] controller platform. FlowChecker [1] uses manually built binary decision diagrams to encode OpenFlow rules and then applies model checking in order to detect OpenFlow switches misconfigurations. Header space analysis [6] is a technique that can be applied to OpenFlow networks to check if a given property, e.g., reachability between two nodes, is provided by the network: it requires the modeling, either performed manually or automatically, of transform functions for the box composing the network, in order to map the transformation to which a packet undergo when traversing such a network.

The definitions and the algorithms we present in this paper are targeted at identifying and automatically detect OpenFlow table entries interaction in a single switch, hence, our work can be seen as a basis for new OpenFlow tools targeted at applications development and debugging, that, differently from, e.g., [1], requires no representation conversion of the flow table entries (FTE). Moreover, our work aims at providing a semantic interpretation of the interactions among FTEs in the switch, in order to help the developer in discovering bugs or checking the behavior of its application at runtime. Differently from the reported verification techniques our approach is limited to a single switch, nevertheless, our tool points at the exact flow table entries in the switch that are interacting among them and provides a clear definition of the discovered interactions.

III. Rule-based programming

OpenFlow-enabled Switches (OFS) behavior is configured using FTEs, installed by means of the OpenFlow (OF) protocol [8]. A FTE is defined by the *match set*, that defines to which network flows the entry is applied, the *action set*, that defines the elaborations and the forwarding decision that must be applied to the matched flows, a *priority*, to relatively order among them the entries installed in a switch's FT, and an *expiration time*, specified through the use of timeouts. According to the OF specification [9], only the highest priority FTE that matches a packet is applied to that packet.

Because of this definition, the switch behavior is dictated by the combination of all the installed FTEs. In fact, looking at a single FTE does not suffice to understand the behavior of the switch in respect to the flow identified by the match set of such FTE, since other FTEs, with higher priorities, can introduce a different behavior.

The problem is usually raised up in the process of developing an OF application, that is, basically the process of defining when, where and what FTEs have to be installed at managed OFSes. This issue can be even more problematic if we are trying to extend an already developed OpenFlow application, or if we are combining several applications at the same time.

IV. Interactions definition

To characterize the behavior of an OFS, we define FTEs interactions extending the work presented in [2]. An interaction is a particular relation between two FTEs. An interaction may be expected, i.e., the network programmer is aware that FTEs are interacting, or it may be raised by unexpected relations between FTEs, e.g., as result of a programming error. To define the possible interactions that can occur between two FTEs, we firstly define the relations that can be in place among match sets and among action sets. Then, based on such relations combination, we define FTEs interaction types.

A. Match sets relations

A match set is composed by a number of match fields. Typical match fields are *l2 source [destination] address*, *l2 protocol type*, *l3 protocol type*, *l3 source [destination] address*, etc. All match fields can have a wildcard as value, that means *any value*. Some match fields can have partially wildcarded values, e.g., an l3 address can be associated with a bitmask to specify which bits are wildcard. Because of the presence of wildcards, there are four different relations among two match fields of the same type. The relation between match field f_0 and match field f_1 can be one of the following: *disjoint*, when match fields have different values ($f_0 \neq f_1$); *equal*, if f_0 value is the same of f_1 ($f_0 = f_1$); *subset*, if f_0 value is a subset of the value of f_1 ($f_0 \subset f_1$), e.g, f_0 has a defined value, while f_1 value is a wildcard; *superset*, when f_0 value is a superset of the value of f_1 ($f_0 \supset f_1$), e.g., f_0 value is an IP address in the form *192.168.0.0/16*, while f_1 value is *192.168.1.0/24*.

Using the defined match fields relations, we are now able to define the relations between two match sets. These relations can actually be described as classical relations among sets, anyway, we provide a formal definition of them based on the mutual relations of the match fields composing the match sets. We adopt this formal definition since it is the basis of our algorithm implementation. The relation between match set M_0 and match set M_1 can be one of the following:

Disjoint: M_0 and M_1 are *disjoint* if every field *i* in M_0 is *disjoint* with the correspondent field in M_1 ($M_0 \neq M_1$);

$$M_0 \neq M_1 \text{ if } \forall \text{ i } f_i^0 \neq f_i^1,\ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Exactly matching: M_0 and M_1 are *exactly matching* if every field *i* in M_0 is *equal* to the correspondent field in M_1 ($M_0 = M_1$);

$$M_0 = M_1 \text{ if } \forall \text{ i } f_i^0 = f_i^1,\ f_i^0 \in M_0 \wedge f_i^1 \in M_1$$

Subset: M_0 is a *subset* of M_1 if one field *j* of M_0 is *subset* of the correspondent field of M_1 and any other field *i* in M_0 is *equal* or *subset* of the correspondent field in M_1 ($M_0 \subset M_1$);

$$M_0 \subset M_1 \text{ if } (\exists j | f_j^0 \subset f_j^1) \wedge [\forall \text{ i } (f_i^0 = f_i^1) \vee (f_i^0 \subset f_i^1)],\ i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

Superset: M_0 is a *superset* of M_1 if one field *j* of M_0 is *superset* of the correspondent field of M_1 and any other field *i* in M_0 is *equal* or *superset* of the correspondent field in M_1 ($M_0 \supset M_1$);

$$M_0 \supset M_1 \text{ if } (\exists j | f_j^0 \supset f_j^1) \wedge [\forall \text{ i } (f_i^0 = f_i^1) \vee (f_i^0 \supset f_i^1)] \ , \\ i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

Correlated: M_0 is *correlated* with M_1 at least one field *j* of M_0 is *superset* of the correspondent field of M_1 and any other field *i* in M_0 is *equal* or *subset* of the correspondent field in M_1 ($M_0 \sim M_1$);

$$M_0 \sim M_1 \text{ if } \exists j | f_j^0 \supset f_j^1 \wedge \exists i | f_i^0 \subset f_i^1 \ , \\ i \neq j \wedge f_i^0, f_j^0 \in M_0 \wedge f_i^1, f_j^1 \in M_1$$

B. Action sets relations

An action set contains zero or more actions. An action has a type and a value, typical action types are *forward to port X*, *rewrite network source [destination] address*, *pop [push] VLAN tag*, etc. An action can be *equal*, *related* or *disjoint* in respect to another action. An action a_0 is *equal* to an action a_1 ($a_0 = a_1$) only if they have the same types and values, if the types are equal but values are different, the actions are *related* ($a_0 \sim a_1$). An action a_0 is disjoint from action a_1 ($a_0 \neq a_1$), if their types are different. Depending on the relations of the contained actions, the relation between action set A_0 and action set A_1 can be one of the following:

Disjoint: A_0 is *disjoint* from A_1 if for any action in A_0, such an action is disjoint from any action in A_1 ($A_0 \neq A_1$);

$$A_0 \neq A_1 \text{ if } \forall \text{ i,j } a_i^0 \neq a_j^1, \ a_i^0 \in A_0 \wedge a_j^1 \in A_1$$

Related: A_0 is *related* to A_1 if there is at least one action from A_0 that is related to an action of A_1 ($A_0 \sim A_1$);

$$A_0 \sim A_1 \text{ if } \exists i, j | a_i^0 \sim a_j^1, \ a_i^0 \in A_0 \wedge a_j^1 \in A_1$$

Subset: A_0 is a *subset* of A_1 if all the actions contained in A_0 are equal to actions contained in A_1, and A_1 contains more action than A_0 ($A_0 \subset A_1$);

$$A_0 \subset A_1 \text{ if } \forall \text{ i,j } a_i^0 = a_j^1 \wedge |A_0| < |A_1|, \ a_i^0 \in A_0 \wedge a_j^1 \in A_1$$

Superset: A_0 is a *superset* of A_1 if all the actions contained in A_0 are equal to actions contained in A_1, and A_0 contains more action than A_1 ($A_0 \supset A_1$);

$$A_0 \supset A_1 \text{ if } \forall \text{ i,j } a_i^0 = a_j^1 \wedge |A_0| > |A_1|, \ a_i^0 \in A_0 \wedge a_j^1 \in A_1$$

Equal: A_0 is *equal* to A_1 if all the actions contained in A_0 are equal to actions contained in A_1, and A_1 and A_0 contains the same number of actions ($A_0 = A_1$);

$$A_0 = A_1 \text{ if } \forall \text{ i,j } a_i^0 = a_j^1 \wedge |A_0| = |A_1|, \ a_i^0 \in A_0 \wedge a_j^1 \in A_1$$

C. Interaction types

To define interactions between two FTEs we look at entries' priorities, match sets relations and action sets relations. Considering a FTE R_x, with match set M_x and action set A_x, and a FTE R_y, with match set M_y and action set A_y, assuming that R_x priority is always smaller than R_y priority (If it is not the case we will explicitly point it out), The interaction between R_x and R_y can be of one of the types listed in table I. Here we provide a brief description of them:

TABLE I: OpenFlow Rules interactions

MATCH SET	ACTION SET	PRIORITY
Duplication		
$M_x = M_y$	$A_x = A_y$	$prio(R_x) = prio(R_y)$
Redundancy		
$M_x \subset M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x = A_y$	$prio(R_x) < prio(R_y)$
Generalization		
$M_x \supset M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \supset M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
Shadowing		
$M_x \subset M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x = M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
Correlation		
$M_x \sim M_y$	$A_x \neq A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \sim A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
$M_x \sim M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$
Inclusion		
$M_x = M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
$M_x \subset M_y$	$A_x \subset A_y$	$prio(R_x) < prio(R_y)$
Extension		
$M_x \supset M_y$	$A_x \supset A_y$	$prio(R_x) < prio(R_y)$

Duplication: assuming that the priorities of two entries are equal, they are duplicated if they have the same match and action sets.

Redundancy: redundant FTEs have the same effect on the subset of flows matched by both entries, hence, in some conditions (e.g., no interactions with third rules), depending on the entries priorities, one of the entries could be deleted without affecting the datapath behavior, or the entries could be aggregated.

Generalization: entries have different actions, but R_x matches a superset of the flows matched by R_y. So, action set A_y will be applied to flows matched by $M_x \cap M_y$, while to the flows matched by $M_x - M_y$ the action set A_x will be applied.

Shadowing: if R_x is shadowed by R_y, then R_x is never applied, since all the flows are matched by R_y before that R_x is examined.

Correlation: the FTEs have different match sets, but the intersection of these match sets is not void, so, to flows that are in the intersection only higher priority entry's action set (A_y) will be applied. Note that this interaction is different from the *shadowing* interaction, since for some flows the lower priority entry (R_x) is still applied.

Inclusion: *inclusion* interaction is similar to *shadowing*. The lower priority entry is never applied "as is", but its actions are still applied in combination with the actions of another entry (of higher priority). I.e., R_x is never applied, but, since the

Algorithm 1 $matchset\_relation(R_x, R_y)$

```
relation ← undetermined
field_relations ← compare_fields(R_x, R_y)

for field in match_fields do

    if field_relations[field] = equal then
        if relation = undetermined then
            relation ← exact
        end if

    else if field_relations[field] = superset then

        if relation = subset or relation = correlated then
            relation ← correlated
        else if relation ≠ disjoint then
            relation ← superset
        end if

    else if field_relations[field] = subset then

        if relation = superset or relation = correlated then
            relation ← correlated
        else if relation ≠ disjoint then
            relation ← subset
        end if

    else
        relation ← disjoint
    end if

end for

return relation
```

Algorithm 2 $interactions\_detection(R_x, R_y)$

```
interaction ← None

ms_relation ← matchset_relation(R_x, R_y)
as_relation ← actionset_relation(R_x, R_y)

if priority(R_x) = priority(R_y) and ms_relation = exact and as_relation = equal
then
    interaction ← duplication

else if ms_relation ≠ disjoint then

    if ms_relation = correlated then
        if as_relation = equal then
            interaction ← redundancy
        else
            interaction ← correlation
        end if

    else if ms_relation = superset then
        if as_relation = equal then
            interaction ← redundancy
        else if as_relation = superset then
            interaction ← extension
        else
            interaction ← generalization
        end if

    else if ms_relation = exact then
        if as_relation = equal then
            interaction ← redundancy
        else if as_relation = subset then
            interaction ← inclusion
        else
            interaction ← shadowing
        end if

    else if ms_relation = subset then
        if as_relation = equal then
            interaction ← redundancy
        else if as_relation = subset then
            interaction ← inclusion
        else
            interaction ← shadowing
        end if

    end if
end if

return interaction
```

action set A_x is a subset of the action set A_y, the actions of A_x are still applied, but only in combination with the actions of A_y.

Extension: *extension* interaction is similar to *generalization*. An entry with lower priority is extending the action set applied by another entry, adding more actions. Only to the flows matched by $M_x - M_y$ the extended actions are applied.

V. Interactions detection

To detect the interactions presented in previous sections, we designed an *interactions detection algorithm* (IDA). The algorithm takes two entries, R_x and R_y, assuming $prio(R_x) \leq prio(R_y)$, and detects the interaction generated by the composition of the entries. We assume that R_x and R_y are data structures containing all the information we need regarding a FTE, i.e., match set, action set and priority. The algorithm uses two auxiliary algorithms to find match sets and action sets relations, respectively, these algorithms are represented by the functions $matchset\_relation(R_x, R_y)$ and $actionset\_relation(R_x, R_y)$.

Algorithm $matchset\_relation(R_x, R_y)$ finds the relation between match sets, by firstly comparing match fields one by one, using the $compare\_fields(R_x, R_y)$ function. Then it cycles among the fields' relations to evaluate the match sets relation, by applying a state machine where each state corresponds to one of the match sets relation (plus "undetermined") and transitions corresponds to match fields relations [2].

Algorithm $actionset\_relation(R_x, R_y)$ is not presented in these pages, since its implementation is simpler and can be easily derived by the definitions of action sets relations presented in previous sections.

Algorithm $interaction\_detection(R_x, R_y)$ uses the just defined functions to apply the interaction types definitions in order to find if the FTEs generate an interaction and, in that case, which type of interaction. We implemented a prototype of our algorithm in Python, integrating it into the NOX controller platform [5]. We performed a first evaluation of our implementation using randomly generated FTEs sets. We generated FTE sets defining three parameters: (i) number of entries in the set, (ii) number of non-wildcard match fields, (iii) probability of generating the same value for match fields belonging to different entries. This last parameter provides a mean to set the number of interactions in the FTEs set, e.g., a low probability corresponds to fewer interactions in the set. Action sets were also generated randomly, selecting from 1 to 3 actions per action set. For each FTEs set, we run the algorithm several times to extract a mean of the running times. The testbed machine is an Ubuntu Linux virtual machine, equipped with 2 GB of RAM and running on a dedicated cpu-core of an Intel CPU E7600 @ 3.06GHz. The execution times are shown in figures 1 and 2: our first implementation is providing a linear increasing of the execution time with the growing of the FTEs set dimension. Moreover, the greater is the number of wildcard match fields, the lower is the execution time. Interestingly, the current implementation provides better performance when the number of interactions in the rule set is bigger (Figure 1).

[2]We are using a "foreach" notation in this *for* cycle, putting in the $field$ variable, one by one, any value found in the $match\_fields$ variable, that contains a list of the all possible fields name

VI. Use cases

The algorithm and the interactions definition we provided in this paper can be used as a basis for OpenFlow network

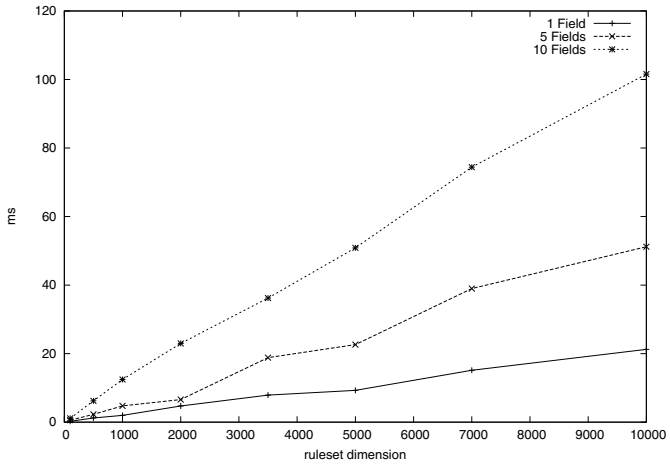


Fig. 1: Interactions detection algorithm performance with low match sets overlapping values probability. Entries in FTEs set have from 1 to 10 non-widlcard fields in the corresponding match sets.

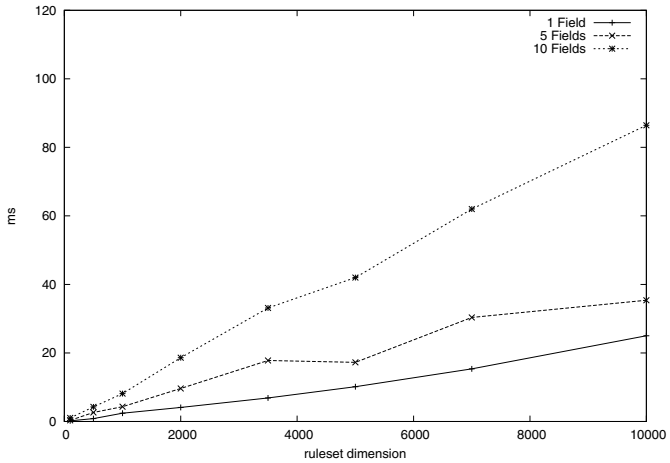


Fig. 2: Interactions detection algorithm performance with high match sets overlapping values probability. Entries in FTEs set have from 1 to 10 non-widlcard fields in the corresponding match sets.

development methodologies and for tools targeted at aiding OpenFlow network programming, analysis, switches management and optimization, etc.. In this section we briefly describe some possible applications, then, we present a concrete use case. During the development of an OpenFlow application, the IDA can be used as debug tool to verify the interactions among entries installed in a switch. In the simplest cases, it can point out entries duplications and redundancies, reducing any overhead in the developed application, or it can detect unexpected FTEs interactions that would lead to the wrong handling of some traffic flows. IDA could be also integrated in advanced controller platforms as a mean to analyze FTEs in order to provide some forms of automation in FTEs management. E.g., rejecting duplicated entries, reordering entries' priorities to avoid shadowing, FTE splitting to avoid redundancy and correlation, etc.. An advanced controller would require more work on the semantic part of the entries management, but the interactions detection algorithm is the enabling technology to analyze FTEs interactions. A different application would be the use of IDA to compare a FTE against a set of FTEs that is used as an admission control policy. Specifying the allowed interactions with the given policy FTEs set, a FTE can be checked to be admitted or not. Complex policies can specify which operations are allowed on which flows, using properly specified FTEs set and allowed interactions. As last application example, the IDA can be used to optimize the FTEs installed in a switch. FTEs can be checked against other FTEs to find interactions, and, in case, they can be rewritten to split or aggregate them for a better use of the switch hardware resources (E.g., some switches have multiple flow tables with different properties). Clearly, we presented only a few examples of possible IDA exploitations. In the following part of this section we briefly introduce a concrete application for the development of a real OpenFlow network application.

A. Extending an OpenFlow application

Follow-Me Cloud (FMC) is a technology, developed at NEC Laboratories Europe, that provides mobility features in a TCP/IP network for both users and services, maintaining all the ongoing network connections active. FMC is applied to a TCP/IP network in which L2 access networks are connected to a "core" network, that provides connectivity among them, through OpenFlow-enabled switches (OFS). To provide mobility to a mobile node (MN) that is changing its access network from an "home" to a "foreign" network, FMC requires that a new IP address, belonging to the foreign network, is assigned to MN to work as "locator". The original IP address of MN, that belongs to the home network, is still used by MN itself and by any node that is communicating with MN, since it works as "identifier". When a network node (we call such a node *correspondent node* or CN in short) sends a packet to MN, it uses the *identifier* address as destination address. The OFS connecting the CN's network (CNet) to the core network performs an address translation, to substitute the *identifier* with the *locator* address. When the packet reaches the foreign network, the OFS at the edge of such network performs a new translation, substituting the *locator* with the *identifier*, in order to deliver the original packet to the MN.

The current FMC implementation uses NOX [5] as controller platform. To make deployment of FMC in a TCP/IP network as easy as the placement of OFSes at the edge of L2 access networks, we decided to extend an OpenFlow learning switch application with FMC functionalities. A learning switch (LS) application provides Ethernet Switch functionalities, by learning MAC addresses and associating them with switch ports. LS installs proper FTEs to forward to the correct port a packet with a given destination MAC address. Rules 3 and 4 from table II are a typical example of two rules installed by the LS application, to provide connectivity among a node *X* and the gateway of the *X*'s network (*routerA*). OFSes are controlled

Using SDN for cloud services provisioning: the XIFI use-case

Eduard Escalona[†], José Ignacio Aznar Baranda[†], Luis Miguel Contreras Murillo[‡], Oscar González de Dios[‡], Giuseppe Cossu[*], Elio Salvadori[*], Federico M. Facca[*]
[*]CREATE-NET, Via Alla Cascata 56/D, 38122 Trento, Italy
Email: gcossu@create-net.org, esalvadori@create-net.org, ffacca@create-net.org
[†]Distributed Applications and Networks Area (DANA), i2CAT Foundation, Barcelona, Spain
Email: eduard.escalona@i2cat.net, jose.aznar@i2cat.net
[‡]Telefónica S.A., Ronda de la Comunicacion, 28050 Madrid, Spain
Email: lmcm@tid.es, ogondio@tid.es

***Abstract*—Cloud-based infrastructures are becoming the de facto standard for service hosting through the innovation of traditional Data Centres (DCs). The availability of advanced multi-site service provisioning solutions (e.g., availability zones in Amazon EC2), is enabling the creation of advanced high-availability architectures for services and applications. These architectures, being highly distributed across cloud infrastructures, largely rely on the connectivity between DCs. While in the last years, the adoption of Software-defined Networking (SDN) solutions within a single cloud site is becoming a consolidated practice, their application across geographically separated data centre sites is still largely unexplored. In this paper we discuss how we plan to tackle this issue within the XIFI project, an EU FI-PPP program project that aims at creating a multi-site community cloud across Europe. We propose a backbone connectivity network service connecting XIFI DC sites that will leverage pan-European Network Research and Education (NREN)'s infrastructures, and advanced SDN solutions to ensure VM-to-VM connectivity across the sites while guaranteeing the Quality of Service (QoS) Service Level Agreements (SLAs) required by the distributed services.**

I. Introduction

Data Centres (DCs) have been facing deep advances in the last few years: from traditional storage and computing power oriented systems to host websites and store data, DCs have become the basic building block of cloud computing architectures. Through DCs, a plethora of novel services have been created, from online storage systems like Dropbox to music stream services like Spotify up to the massive offer of Google which includes email, storage, Office apps, etc. Furthermore, DCs are becoming central in the management of the huge amount of data both users and interconnected machines are producing and that are used to build either highly personalised services or to extract important information that may be used in many of the raising so called "Smart-City" scenarios. Therefore, DCs and cloud computing are becoming a major component for the services of the so-called Future Internet.

The FI-PPP [1] is an ambitious programme by the European Commission part of the 7th Framework Programme aiming to explore the potential of a common platform for Future Internet technologies to establish new business ecosystems. Within this framework, XIFI [2] aims to be the community cloud for Future Internet services developers enabled by advanced experimental infrastructures in Europe. By being strongly focused on cloud computing based services and architectures, XIFI leverages on a set of data centres facilities distributed geographically all over Europe and interconnected by means of the network services provided by the National Research and Education Networks (NRENs) part of GÉANT.

Enabling FI services developers to deploy satisfying end user compliant Quality of Service (QoS) pushes the need to provide a tight orchestration of network resources both at intra-DC level and at inter-DC level; from this viewpoint, the strongest challenge comes from the last scenario, where network resources are usually placed across several network domains generally belonging to different network owners. While in the last years, the adoption of Software-defined Networking (SDN) solutions within a single DC site is becoming a consolidated practice, their application across geographically separated data centre sites is still in its infancy even though several companies are proposing their own (usually highly customized) SDN-based solutions. Probably one of the most famous use case is the one recently presented by Google [3], a geographically wide area network (but single-domain) used by Google to interconnect its own DCs and managed through OpenFlow [4], the most common implementation of SDN principles.

Despite these preliminary attempts to handle inter-DCs connectivity via SDN, there are not many consolidated solutions on the market and very few discussed in literature, especially when taking into account the increased complexity of dealing with a multi-domain network scenario. In this paper we will present the architecture that XIFI plans to setup and manage several DC facilities across Europe, by describing its key components and the major challenges behind it. One of the key points of the proposed solution is the combination of OpenNaaS and ABNO, two recently proposed frameworks whose combination with the network services provided by European NRENs and GÉANT, are enabling an effective orchestration of network resources to facilitate the deployment of several Future Internet application scenarios.

The rest of the paper is organized as follows: Section II presents the state of the art on the inter-DCs solutions available in literature and discuss about the challenges of SDN for distributed cloud while Section III presents XIFI

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

TABLE II: OF-Switch installed rules

#	Matching criteria				Priority	Actions
	DL_DST	DL_TYPE	NW_SRC	NW_DST		
1	$MAC_{routerA}$	IP	*	$identifier$	200	set NW_DST: $locator$; out: $PORT_{routerA}$
2	MAC_x	IP	$locator$	*	200	set NW_SRC: $identifier$; out: $PORT_x$
3	$MAC_{routerA}$	*	*	*	100	out: $PORT_{routerA}$
4	MAC_x	*	*	*	100	out: $PORT_x$

by the learning switch application, ensuring traditional TCP/IP operations, while only the flows directed to mobile nodes are handled by FMC-related FTEs. As an example, the addition of FMC to a CNet's OFS requires that a destination address translation is performed on any flow destined to an *identifier* address. At the same time, a source address translation must be performed on any flow with *locator* as source network address (See rules 1 and 2 from table II). We have to ensure that FMC-related FTEs are not shadowed by LS-related FTEs. Our FMC implementation uses the interactions detection algorithm to guarantee that newly installed rules are always involved in generalization interactions with LS-related FTEs, i.e., LS-related FTEs are generalizing FMC-related FTEs. The algorithm has been integrated into the FMC OpenFlow Controller and it is used as a runtime tool, to define the required priority value to assign to newly generated FTEs, and as debug and validation tool, to check the absence of shadowing interactions into switches.

VII. Conclusion and future work

In this paper we presented a formal definition for the interactions of entries installed in a OpenFlow Switch's flow table, an algorithm for the automatic detection of such interactions and we showed how the algorithm has been used to develop a real OpenFlow application. Furthermore, we evaluated our prototype implementation of the proposed algorithm to understand the actual applicability in other real-word scenarios. Our evaluation shows that the dimension of the managed entries set can limit the applicability of the algorithm as a runtime tool for the definition of a complex management policy, since the execution times, when the number of entries is in the order of thousands, can negatively affects the network performance. Anyway, the algorithm is well suited for the development phase of OpenFlow applications, e.g., as debug tool, or in applications where the number of managed FTEs per switch is no more than a few hundreds, which is a dimension currently in line with most hardware OpenFlow switches' flow table size. In future work we plan to improve the algorithm execution times by exploiting, e.g., smart ordering of entries or FTEs set reduction strategies. Moreover, we are extending its applicability to the detection of network-wide FTEs interactions.

References

[1] Al-Shaer, E., and Al-Haj, S. Flowchecker: configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration* (New York, NY, USA, 2010), SafeConfig '10, ACM, pp. 37–44.

[2] Al-Shaer, E., and Hamed, H. Modeling and management of firewall policies. *Network and Service Management, IEEE Transactions on 1*, 1 (april 2004), 2 –10.

[3] Canini, M., Venzano, D., Peresini, P., Kostic, D., and Rexford, J. A nice way to test openflow applications. *EPFL Technical Report* (Oct. 2011).

[4] Foster, N., Harrison, R., Freedman, M. J., Monsanto, C., Rexford, J., Story, A., and Walker, D. Frenetic: a network programming language. *SIGPLAN Not. 46*, 9 (Sept. 2011), 279–291.

[5] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev. 38*, 3 (2008), 105–110.

[6] Kazemian, P., Varghese, G., and McKeown, N. Header space analysis: static checking for networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 9–9.

[7] Monsanto, C., Foster, N., Harrison, R., and Walker, D. A compiler and run-time system for network programming languages. *SIGPLAN Not. 47*, 1 (Jan. 2012), 217–230.

[8] Openflow - http://www.openflow.org/.

[9] Openflow specification 1.1.0 - http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

project and the two network connectivity and management scenarios (intra-DCs and inter-DCs), and a scenario from the developer perspective. Section IV discusses the SDN-based network connectivity enabling service provisioning in XIFI; finally, Section V discusses next steps in the project and draws the conclusions.

II. SDN Challenges for distributed Clouds

Cloud-based infrastructures allow end-users to deploy applications and services in a distributed way and to make use of computing resources allocated in distributed data centres. The communication among DCs is performed by circuits traversing Operators and Service Providers (SPs) networks which usually present heterogeneous technologies and topologies and typically comprise different administrative domains. Additionally, network resources can be dynamically re-assigned within and between data centres changing on-the-fly both the overlay service topology and the corresponding traffic load. On one hand, the delivery of distributed cloud services implies the inter-domain coordination in terms of forwarding, routing protocols, QoS levels and Security. On the other hand, intra-domain aspects, such as failure recovery or congestion control are also important functionalities to be considered.

The main requirements of data centre services, scalability, multi-tenancy, VM migration, and ease of configuration had led to a several solution to realize virtual networks. The traditional approach is to use VLANs [5] to implement isolation between tenants. On one side this solution overcomes the layer 2 broadcast problem providing broadcast domain isolation, from the other it does not scale since with the 12-bit VLAN ID, the segments are limited to 4096. Furthermore, VLAN is too tied to the physical infrastructure and its configuration is difficult to manage. To solve the VLANs problems several solutions has been proposed with a similar approach. IEEE 802.1ad Provider Bridges (Q-in-Q) [6] extends the VLAN allowing multiple VLAN tags in an Ethernet frame (e.g., customer VLANs inside SP VLANs), whereas IEEE 802.1ah Provider Backbone Bridges (MAC-IN-MAC) [7] encapsulates the packet into another Ethernet header allowing the possibility to interconnect multiple Provider Bridge networks. These standards while solving some problems, does not represent suitable solution for Data Centre Networks (DCNs). Recently the direction to solve issues related to providing multi-tenancy in large DCNs are the overlay network [8] . The main overlay solutions are: VxLAN (VMware, Cisco), NVGRE (Microsoft,Intel, HP) STT (Nicira),. This approach enables the separation of logical function from the physical network by means of a tunnelling protocol, virtualizing layer 2 networks over layer 3 networks. VXLAN [9] oovercome the scalability problems of VLAN with a 24-bit header that expands the number of virtual segments to 16M. Similarly NVGRE [10] u use Generic Route Encapsulation (GRE) tunnels to isolate traffic with the ability to encapsulate an arbitrary protocol over IP.

The SDN [11], [12], [13] paradigm allows the management of such a complex and heterogeneous environment in a more simple, scalable and flexible way. SDN-based architectures are characterized for decoupling the network intelligence from the forwarding plane, so that network control and state can be logically externalized (by means of a so-called "SDN-Controller"), which allows the network infrastructure to match the dynamism of the cloud computing services. A programmatic interface is defined to interact with the data plane functions of the network nodes. The OpenFlow protocol is the most extended realization of such interface. Through OpenFlow the SDN controller can define rules for individual packet flows according to specific fields on the packet header at either layer 2 or layer 3. These rules are populated in the form of flow table entries that result on the forwarding configuration of the nodes. Sophisticated flow handling can be achieved by cascading a number of flow tables before the flows are delivered to the egress port. This fine grained control provides high flexibility. OpenFlow implements a vendor-agnostic, abstract interface that helps to integrate heterogeneous multi-vendor networking environments. At the same time it facilitates the orchestration of network resources since the representation of the node behaviour becomes generic and independent of the particular hardware implementations and command syntax. In highly distributed environments with a large number of administrative domains and resources, the deployment of a single SDN controller to manage all the required connections may not scale. One potential solution could consist of the federation of non overlapping SDN-based domains. However, the existence of multiple domains raises the problem of controller coordination for the provision of an end-to-end connectivity service. SDN controllers interconnection has been proposed in [14] as a way of exchanging information between them. Nevertheless, in such a federated environment it is yet required a logically centralized view and an orchestrated provisioning of all the network resources to set up End to End (E2E) connectivity services among DCs. These two requirements need to be addressed for a consistent delivery of a connectivity service between SDN domains.

III. XIFI project

XIFI aims to be the community cloud for European FI-PPP developers enabled by advanced Future Internet (FI) infrastructures in Europe. The FI-PPP is an ambitious programme by the European Commission part of the 7th Framework Programme aiming to explore the potential of a common platform for Future Internet technologies to establish new business ecosystems. XIFI will provide a market place to access:

- the web-based services offered by FI-PPP, i.e., the Generic Enablers developed by FI-WARE [15] and the specific Enablers provided by Use Case Trials;
- advanced Future Internet infrastructures that provide capacities, i.e., sensor networks or other FI facilities like smart factories;
- data to empower the applications developed by early adopters of FI-PPP technologies.

XIFI will provide with a community cloud by federating a number of Future Internet infrastructures in which OpenStack [15] has been chosen as the DCs service management solution. The overall architecture is based on a central portal (Master Node) that controls the different nodes (Slave Nodes) that belong to the federated infrastructure and provide cloud and other resources to the federation. The architecture is based on high-availability principles, so that single nodes can run even

if there is an outage on the main node. A detailed description of XIFI architecture is available on XIFI wiki [16].

XIFI aims at supporting developers deploy distributed applications across the different nodes (or DCs) part of the federation (or community cloud). Distributed applications may comprise several services and are located at different premises. This entails several implications for both the network within the single data centre and the network that interconnects multiple data centres.

It is crucial for developers to be able to design applications that satisfy end-user compliant Quality of Service. QoS depends, among others, on software, servers and network configuration. Developers may have a simple understanding of network related issues. Nevertheless, to guarantee certain QoS levels, they need to be able to map such requirement to the network (whether this network is internal to a single data centre or spanning across multiple data centres) on top of which services and applications run.

In XIFI we aim to support developers in the dynamic control of "virtualized" networks that interconnect the different services. Internal DC connectivity features rely on standard layer 2 functionalities controlled by Neutron [17], the OpenStack Networking framework. One of the options considered in XIFI to provide such network overlay in the DC is to use DOVE [18], [19], which is currently one of the FI-WARE generic enablers, as a plugin of Neutron. On the other hand, the connectivity across DCs is based on network services provided by the NRENs part of GÉANT. Management and control of E2E service connectivity, spanning several DCs, is one of the key innovations provided by XIFI. The mentioned control and management will be orchestrated by a XIFI Network Controller based on OpenNaaS [20] and some components of the Application Based Network Operations (ABNO) [21] framework, recently proposed in the IETF to deal with services that require on-demand and application-specific reservation of network connectivity. In the next paragraphs we discuss shortly the intra data centre and inter data centre connectivity solutions adopted in XIFI.

A. Intra-DC Network connectivity and management

The advent of cloud computing has led to important challenges within DCs networking such as large scale workloads, multi-tenancy and management. XIFI nodes, where Generic Enablers are deployed, deal with these challenges using network virtualization based on Neutron and DOVE. Neutron (formerly known as Quantum) is an OpenStack stand-alone service that allows users to design and manage virtual networks inside data centres. Using a programmatic and extensible API, users and administrators can create multi-tenancy networks without knowing how the underlying physical network is deployed. Moreover, Neutron can be used with a set of backends, called plugins, which are technology specific. The set of plugins currently available to enable the network virtualization, include NVP (from VMware/Nicira), Cisco UCS/Nexus, Ryu OpenFlow Controller, Open vSwitch (OVS) and others. Basically, these technologies adopt an overlay-based approach to decouple the logical topology from the physical topology and build multi-tenant networks and traffic isolation. This approach overcomes the problems of traditional VLANs that lack, among other things, scalability and ease of configuration. In this direction IBM has developed a Distributed Overlay Virtual Ethernet network (DOVE) architecture. DOVE provides logically isolated multi-tenant networks with a virtual network with layer 2 or layer 3 connectivity using Virtual eXtensible LAN (VXLAN) that is being standardized under IETF. DOVE is provided as Quantum plugin in the FI-WARE Data centre Resource Management (DCRM) GE [22].

B. Inter-DC federation: Network connectivity and management

XIFI network connectivity service consists of a multi-domain E2E provisioning service where circuits are established in order to provide connectivity among XIFI DCs, and enable the access to the XIFI nodes and Generic Enablers hosted in such DCs. To this target, the GÉANT and NREN community represent an excellent scenario for the XIFI connectivity service deployment, since NRENs are specialized Internet service providers dedicated to supporting the needs of the research and education communities. Therefore, XIFI leverages such network infrastructures to accomplish the full E2E network connectivity service between DCs involving several network segments (see Figure 1).

These segments comprise the intra-DC connectivity, the Point-to-Point (P2P) connectivity provided by GÉANT and NRENs and the so-called XIFI Demarcation Points. Due to current technology limitations and NRENs heterogeneity, the specific XIFI network connectivity service is provided between the demarcation points, assuming that to the date, there are no automated mechanisms that enable to dynamically establish connections across different administrative domains in GÉANT and the NRENs. NREN provisioning services entail a set of specific particularities that must be taken into account, since they impact the overall XIFI Network Service:

- NRENs and GÉANT network domains are capable of providing with E2E multi-domain layer 2 connectivity services. Recently, managed L3 services are being developed by GÉANT;
- E2E QoS capabilities rely on the NRENs and GÉANT provisioning capabilities;
- Connection to the Internet must be done at NRENs facilities, since they own public IP ranges
- For the intra-domain NREN connectivity, wherever possible, automatic provisioning mechanisms will be used such as GÉANT BoD. Otherwise, connectivity will need to be pre-engineered and pre-provisioned in advance.

C. Reference Scenario

In XIFI we aim at making accessible multi-domain dynamic network traffic management to cloud applications developers. A reference scenario, for example, is the case of largely distributed media repository based on cloud big data solutions. In the case of XIFI we will consider for example the problem of synchronizing and distributing (through streaming techniques) such media across two nodes. Applying SDN, XIFI will demonstrate how it is possible to improve QoS perceived by users and/or service reliability.

The scenario will leverage on network control through SDN technology at level of single node, and NaaS facilities at

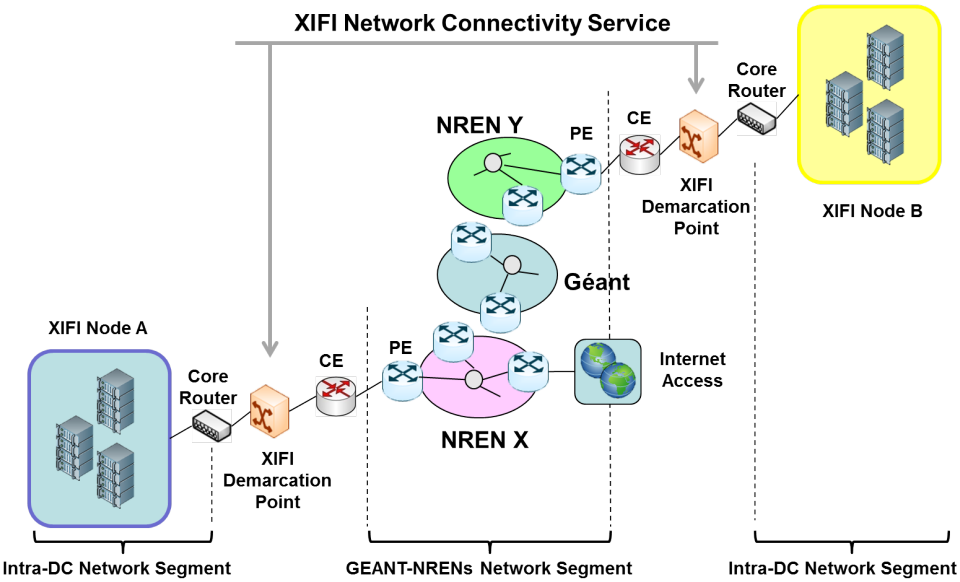


Fig. 1: Overall XIFI network composed of DCs, NRENs and GÉANT facilities.

cross nodes (spanning over multiple domains). For example, as regard media streaming, the application developer can improve the Quality of Experience of end-user accessing video streaming from different nodes of the XIFI cloud. The different nodes of XIFI cloud can be used as media repository and used as geo-localized access points for the end-users. The developer, then can decide whether synchronizing the media repository and hence, ensuring that on all the nodes there are copies of relevant media content (using SDN techniques to support this synchronization) or ensuring that streaming requests are served the access nodes with a given quality also from a central repository. In both cases, once the end-user access such application, following the configuration of node connectivity and prioritization of traffic defined by the developer, the underlying network is configured in order to ensure service continuity and an appropriate QoS, by means of a) fault prevention or fault recovery in case of network failure; and b) slicing of network resources to ensure QoS to users.

In the case of distribution of media across multiple nodes, SDN can be used for fault prevention and recovery, for example by replacing the streaming of the end user from node A, to node B in case of malfunctioning or of connectivity performance issues with node A.

IV. SDN-BASED NETWORK CONNECTIVITY SERVICE PROVISIONING IN XIFI

The XIFI architecture is composed of a federation of distributed DCs offering a shared infrastructure for cloud-based services. The inter-DC connectivity service provided by XIFI may deal with a number of requirements imposed to network resources [23], especially in terms of defining a fine-grained per-flow policy based control model and the dynamic adaptation to the traffic generated by cloud services. To this effect, network programmability is a powerful mean to achieve them.

End-users do not need to assume any specific negotiation with XIFI for the provision of network connectivity when deploying their applications and services across distributed DCs. The proposed architecture should enable a single operation point for requesting both computing and network resources. Also an integrated provisioning system is required to automate the connectivity setup locally in each of the DCs. Thus, a XIFI network controller is designed to orchestrate the complete service provisioning process. The XIFI network controller will smartly interact with the SDN controllers locally deployed in each DC, as part of the federation of SDN-based domains. Furthermore, the XIFI network controller will complement the local SDN controllers by providing the needed multi-domain coordination among multiple independent cloud systems allocated in different domains and connected through different networks. XIFI network controller will provide a north-bound Restful API. In particular, it is aimed to interact with Neutron to extend the connectivity among DCs. The next sections describe in detail the XIFI network controller.

A. *Applying the SDN paradigm to the XIFI architecture*

A single monolithic SDN controller is not able to implement the network management capabilities required for supporting the distributed connectivity of the XIFI federated framework, neither to deal with the integration of the envisaged heterogeneous network environments by its own. Therefore, the network controller in XIFI requires maintaining a logically centralized and abstracted view of the XIFI network infrastructure, composed of the XIFI nodes at DCs, and the inter-DC logical links connecting them. Thus, the XIFI Network Controller triggers the provisioning of the E2E service delivery according to the needs and requirements of the end-user, by configuring a number of SDN-based network nodes. A full automation is the key to enable the rapid deployment, consumption and management of cloud services. Moreover, in SDN-based scenarios the network intelligence and state are logically centralized, and the underlying network infrastructure

is abstracted from the applications. As a result, end-users are capable to define and abstract the service requirements independently of the underlying network technologies. For instance, the communication needed between application instances running in different locations may have stringent delay requirements that will be mapped to a specific connectivity design that can discriminate paths connecting the involved DCs. Each of these demands will have specific needs in terms of number of DCs to be connected, QoS to be guaranteed, addressing ranges, etc.

Service Providers (e.g., NRENs) offering network connectivity between the XIFI DCs need to protect their operational networks from external applications. As a consequence, the XIFI network controller is not allowed to directly control the NRENs network resources. Therefore, specific switching resources allocated at DCs will be deployed as part of the XIFI network, acting as demarcation points for XIFI connectivity services. They will be fully managed by the XIFI network controller by means of OpenFlow. These OpenFlow-based XIFI switches will connect to the NRENs infrastructure enabling the possibility of providing with E2E inter-DC connectivity services. In order to separate and isolate the different traffic flows of end-users, the connectivity service provides with overlay networks on top of the physical infrastructure among the XIFI demarcation points (Figure 1). Since the connectivity between the NRENs is IP-based, a tunneling mechanism is needed to build such overlay networks. XIFI proposes two potential solutions, either based on layer 2 (e.g., VLAN, Q-in-Q) or layer 3 (e.g., IP-in-IP, GRE) tunnels. The final choice will depend on the capabilities offered by the NREN that is providing the connectivity to the DC, and the functionalities supported by the OpenFlow version used. For instance, the XIFI switch could deliver VLAN-tagged frames to the NREN access router, which encapsulates the traffic in a GRE tunnel with destination a remote DC. Thus, the overall interconnection topology among DCs can be built based on a "mesh of tunnels" among the involved sites, so that different paths can be selected from the centralized XIFI network controller to accommodate the end-user flows according to their needs.

B. *Network Controller components*

The XIFI network connectivity service management is carried out by means of two key software components. On one hand, the XIFI service management is performed implementing the required extensions to the OpenNaaS platform [20], [24]. On the other hand, the network facilities management is carried through the ABNO [21] framework, which includes key functional blocks such as a Path Computation Element (PCE) [25]. The coordination between these two entities results crucial for a suitable implementation of the overall XIFI connectivity service.

1) OpenNaaS: XIFI Connectivity service is considered from the perspective of Network-as-a-Service (NaaS) model, since it aims to provide with an E2E network service among XIFI nodes. The NaaS model capabilities are particularly interesting for the benefit they can provide to GÉANT and the NREN community, enabling with full control, management and operations of the underlying network resources increasing flexibility and potentially reducing OPEX while implementing the XIFI connectivity service. The NaaS model has been brought forward with OpenNaaS and will promote an easy prototyping and proof-casing of XIFI connectivity service. OpenNaaS is an open-source framework, developed under the FP7 MANTYCHORE project [26] which provides tools for managing the different resources present in any network infrastructure. The software platform was created in order to offer a neutral tool to the different stakeholders. It allows them to contribute and benefit from a common NaaS software oriented platform for both applications and services.

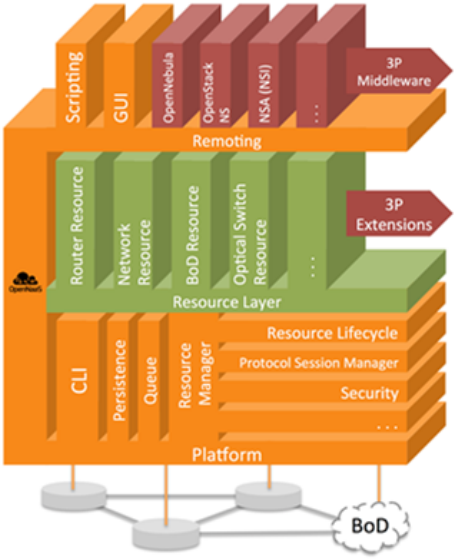


Fig. 2: Layered OpenNaaS architecture.

OpenNaaS is based on a lightweight abstracted operational model, which is decoupled from actual vendors specific details, is flexible enough to accommodate different and tailored modules and can provides to developers with the necessary tools to control the XIFI network behind distributed systems. Additionally, the OpenNaaS framework allows implementing the logic of an SDN-like control and management plane on top of the lightweight abstracted model, which can be applied to the control of the OpenFlow-based demarcation points in the XIFI nodes. Figure 2 depicts the Open-NaaS architecture. It is structured in three differentiated horizontal layers: the platform layer contains the reusable building blocks that form the core of the software and controls the access to the different infrastructure resources; the resource layer contains the different resource abstractions and defines manageable units with a set of capabilities that are mapped to the actions that can be performed over the managed resource. Finally, the upper layer, where the network intelligence resides, provides integration with external applications and implements specific interfaces towards external elements. Thus, the OpenNaaS framework provides a suitable framework for the management and operation of on-top developed services, such us the XIFI network connectivity service.

2) Application Based Network Operations (ABNO): The SDN-based network approach in XIFI follows the ABNO architecture, proposed in IETF as an SDN framework based on standard functional blocks to deal with application-specific reservation of network connectivity, reliability, and resources.

These functional blocks interact through standard interfaces. The set of standard functional blocks include elements such as the PCE, Virtual Network Topology Manger, topology databases, etc. The components of the ABNO framework permit a number of functions to act in an integrated way, including policy control, topology information, and network resource provisioning. Figure 3 represents the specific components (red color) of the ABNO framework considered in XIFI.

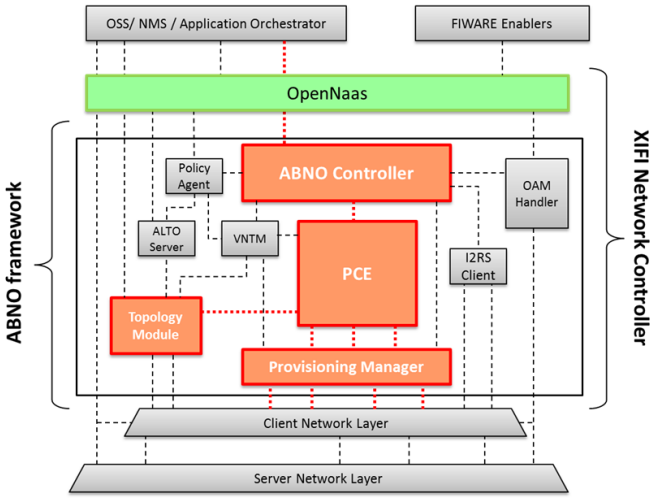


Fig. 3: XIFI Network Controller with OpenNaaS (green colored) and ABNO (red colored) components.

The ABNO Controller is responsible of orchestrating the connectivity service requests by invoking the necessary components in the right order according to specific workflows. It is connected to the OpenNaaS component from where the connectivity request is progressed. The PCE is in charge of the necessary calculations to find connectivity between network elements based on the topological and traffic engineering information, applying a wide set of constraints, including for instance bandwidth, QoS and exclusions. The Path Computation Element architecture supports coordination among several PCEs with different responsibilities, which is especially useful for multi-domain environments, in which each domain is independently administrated, or multi-technology networks, in which switching can be performed using different technologies, such as Ethernet or optical. As mentioned, the PCE relies on topological and traffic engineering information. In that sense, the Topology Module is in charge of discovering the network resources and their availability. The topology module is able to interface with third party tools (inventory, NMS), SNMP, NETCONF [27], IGP (OSPF-TE, ISIS-TE), BGP, OpenFlow, REST/JSON, etc. The Provisioning Manager is an element in charge of controlling the devices of the network to create the desired connectivity. There are two main approaches that can be followed. On the one hand, there can be a trigger to the control plane (e.g., GMPLS) and rely on it for the necessary provisioning. On the other hand, there can be a programming of each individual network element involved in the connection. Finally, the provisioning manager receives the requests in a standard format that is able to cover the different network technologies. In order to control the devices (either relying on a control plane or configuring the devices themselves), standard interfaces are used as a preferred option. Among the standard interfaces, NETCONF and PCEP [28], are candidate for option like OpenFlow, ForCES, GSMP and, again, NETCONF suit for the second option. In XIFI, the implementation is targeted at using OpenFlow as the configuration interface. The provisioning manager will instruct distributed SDN controllers local to each XIFI DC for performing the E2E connectivity setup, for example pushing the necessary OpenFlow rules.

V. Conclusion and Future works

In this paper, we presented the solution we propose for provide SDN based interconnectivity across multiple domains in the case of XIFI, the community cloud of the FI-PPP programme. The paper depicts the overall technical solution and provides hints of how we plan to solve common issues in data centre connectivity faced moving from single domain to multiple domains. XIFI is just started and there are still many challenges we have to face in the project. In this paper we present the architecture we plan to adopt in order to manage the SDN based interconnectivity among data centre sites across multiple domains. Surely, when the architecture will be deployed we will discover new constraints and issues not foreseen at this stage that will require us to adapt our solution. In the case of the inter-DC connectivity service, network control based on the SDN approach accounts a number of known challenges, like performance, scalability, security or interoperability [29]. While that challenges can apply mainly to the local SDN controllers, the XIFI environment presents additional challenges to be addressed in the future. The following is a list of the identified challenges with their respective foreseen actions.

- Full operation as adaptive network, according to internal network events (e.g., alarms). The XIFI project is developing a monitoring middleware able to report relevant network events. The ABNO framework considers the interaction of the ABNO Controller with an OAM module, and further work is needed to integrate both modules. Furthermore, the deployment of local policy agents could enhanced the architecture by triggering specific actions according to pre-defined policies (e.g., link load thresholds)
- Extension to vendor-specific control mechanisms (e.g., CLI), for network investment protection allowing integration with non-OpenFlow-based devices. The Provisioning Manager can extend its capabilities to manage non-OpenFlow devices by integrating specific adapters. The development of specific modules could provide a way for extending the XIFI architecture to conventional data centres.
- Security mechanisms for the connection to distributed SDN controllers. The XIFI network controller assumes the connection to local SDN controllers. In order to prevent security issues some procedures should be established to create a trusted relationship between the distributed SDN controllers and the centralized XIFI network controller.

Moreover, the exploration of the architecture in the development of cloud distributed applications will bring to the table new unforeseen requirements and challenges we will

need to tackle to ensure the success of the project. For example, the cloud manager or the application itself may use the APIs exposed by the network controller at run-time to modify the network behaviour (e.g. "scaling" the network bandwidth as nowadays VMs CPUs or memory scale) in according application specific dynamic requirements. This will open new scenarios that are of practical interest for the developers and that will move our challenges from now quite consolidated research on SDN, to new developments in the area of Application-defined Networks.

ACKNOWLEDGMENT

The work presented in this paper is supported by the EU funded XIFI project (FP7-ICT-604590).

REFERENCES

[1] FI-PPP. (2013) The Future Internet Public-Private Partnership. [Online]. Available: http://www.fi-ppp.eu

[2] XIFI. (2013) eXperimental Infrastructures for the Future Internet. [Online]. Available: https://www.fi-xifi.eu

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: experience with a globally-deployed software defined wan," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486019

[4] Open Networking Foundation. (2013) OpenFlow Switch Specification. [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow

[5] IEEE Standards. (2005) 802.1Q - Virtual Bridged Local Area Networks. [Online]. Available: http://www.ieee802.org/1/pages/802.1Q.html

[6] ——. (2006) IEEE 802.1ad - Provider Bridges. [Online]. Available: http://www.ieee802.org/1/pages/802.1ad.html

[7] ——. (2008) IEEE 802.1ah - Provider Backbone Bridges. [Online]. Available: http://www.ieee802.org/1/pages/802.1ah.html

[8] Narten, et al. (2013) Problem Statement: Overlays for Network Virtualization. [Online]. Available: http://tools.ietf.org/html/draft-ietf-nvo3-overlay-problem-statement-04

[9] Mahalingam, Dutt et al. (2013) VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. [Online]. Available: http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-04

[10] M. Sridharan et al. (2013) NVGRE: Network Virtualization using Generic Routing Encapsulation. [Online]. Available: http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-03

[11] Open Network Foundation. (2012) Software-Defined Networking: The New Norm for Networks. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[12] S. Azodolmolky, P. Wieder, and R. Yahyapour, "Cloud computing networking: challenges and opportunities for innovations," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. –, 2013.

[13] Open Data Center Alliance. (2013) Open Data Center Alliance Usage Model: Software-Defined Networking rev. 1.0. [Online]. Available: http://www.opendatacenteralliance.org/docs/Software_Defined_Networking_Master_Usage_Model_Rev1.0.pdf

[14] H. Yin, H. Xie, T. Tsou, D. Lopez, P. A. Aranda, and R. Sidi. (2013) SDNi: A Message Exchange Protocol for Software Defined Networks (SDNS) across Multiple Domains. [Online]. Available: http://tools.ietf.org/html/draft-yin-sdn-sdni-00

[15] The OpenStack Foundation. (2013) OpenStack: Open source software for building private and public clouds. [Online]. Available: http://www.openstack.org

[16] XIFI. (2013) XIFI Architecture. [Online]. Available: http://wiki.fi-xifi.eu/Public:Architecture

[17] The OpenStack Foundation. (2013) OpenStack Networking Neutron. [Online]. Available: https://wiki.openstack.org/wiki/Neutron

[18] L. Lewin-Eytan, K. Barabash, R. Cohen, V. Jain and A. Levin. (2012) Designing Modular Overlay Solutions for Network Virtualization. [Online]. Available: http://domino.research.ibm.com/library/cyberdig.nsf/papers/F59140F39B4A09E285257A750043F4A6/$File/h-0316.pdf

[19] K. Barabash, R. Cohen, D. Hadas, V. Jain, R. Recio, and B. Rochwerger, "A case for overlays in DCN virtualization," in *Proceedings of the 3rd Workshop on Data Center - Converged and Virtual Ethernet Switching*, ser. DC-CaVES '11. ITCP, 2011, pp. 30–37. [Online]. Available: http://dl.acm.org/citation.cfm?id=2043535.2043540

[20] The OpenNaaS Community. (2013) OpenNaaS. [Online]. Available: http://www.opennaas.org/

[21] D. King and A. Farrel. (2013) A PCE-based Architecture for Application-based Network Operations. [Online]. Available: http://tools.ietf.org/html/draft-farrkingel-pce-abno-architecture-00

[22] FI-WARE. (2013) IaaS Data Center Resource Management - Installation and Administration Guide. [Online]. Available: https://forge.fi-ware.eu/plugins/mediawiki/wiki/fiware/index.php/IaaS_Data_Center_Resource_Management_-_Installation_and_Administration_Guide#IaaS_Data_Center_Resource_Management_Installation

[23] International Telecommunication Union. (2013) Y.3510 - Cloud computing infrastructure requirements. [Online]. Available: http://www.itu.int/rec/T-REC-Y.3510-201305-P

[24] D. Wilson, "OpenNaaS and Virtual CPE: a new way to connect to the Internet," in *TNC*, D. Foster, Ed. TERENA, August 2013. [Online]. Available: http://www.terena.org/publications/tnc2013-proceedings/

[25] Farrel, et al. (2006) RFC 4655 - A Path Computation Element (PCE)-Based Architecture. [Online]. Available: http://tools.ietf.org/html/rfc4655

[26] The Mantychore Project. (2013) IP Networks as a Service, Deliverable D1.1 Project Presentation. [Online]. Available: http://www.mantychore.eu/wp-content/files/MFP7%20D1%20r4.pdf

[27] Enns, et al. (2006) RFC 6241 - Network Configuration Protocol (NETCONF). [Online]. Available: http://tools.ietf.org/html/rfc6241

[28] JP. Vasseur and JL. Le Roux. (2009) RFC 5440 - Path Computation Element (PCE) Communication Protocol (PCEP). [Online]. Available: http://tools.ietf.org/html/rfc6241

[29] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 7, pp. –, 2013.

Boosting Cloud Communications through a Crosslayer Multipath Protocol Architecture

Matthieu Coudron[†], Stefano Secci[†], Guido Maier[‡], Guy Pujolle[†], Achille Pattavina[‡]
[†]LIP6, UPMC, 4 place Jussieu 75005, Paris, France.
Email: firstname.lastname@lip6.fr
[‡]DEIB, Politecnico di Milano, Piazza Leonardo da Vinci 32 – 20133 Milano, Italy.
Email: firstname.lastname@polimi.it

***Abstract*— External reliability in data-center networking is today commonly reached via forms of provider multihoming, so as to guarantee higher service availability rates. In parallel, Cloud users also resort to multihoming via different device access interfaces (Wi-fi, 3G, Wired). Both practices add path diversity between Cloud users and servers, unusable with legacy communication protocols. To overcome this void, we present a holistic multipath communication architecture for Cloud access and inter-Cloud communications, and defend its possible implementation using three promising recent protocols functionally acting at three different communication layers: MPTCP, LISP and TRILL.**

***Index Terms*—Software defined networks, multipath routing, cloud computing, multihoming.**

I. Introduction

Multipath communications represent both a chance and a hassle for the current Internet. On the one hand, legacy Internet protocols have mainly been designed with a single active path paradigm in mind. On the other hand, multihoming practices at both endpoint and network levels can offer path diversity to data connections, potentially allowing effective end-to-end load-balancing and multipath communications [1]. Cloud networking is at the forefront of this trend. Indeed, it increases Cloud availability guarantee via different techniques : the external interconnection of data-centers (DCs) with multiple independent provider links, the deployment of intra-DC multipath layer-2 protocols and IP endpoint multihoming. In this paper, we tackle the challenge of establishing coordinated multipath communications in a Cloud environment composed of multihomed data-center networks and users.

We adopt a protocol interoperability perspective, aiming at increasing throughput and resiliency of Cloud communications, within and across administration domains. We present both stateless and stateful solutions to end-to-end path diversity management, involving novel protocols standardized in the last months: the Multipath Transport Control Protocol (MPTCP) [2], the Locator/Identifier Separation Protocol (LISP) [3] and the Transparent Interconnection of a Lot of Links (TRILL) protocol [4].

II. General Architecture

Our goal is to resort to multipathing to improve cloud access and inter-Cloud performance, more precisely to increase Cloud connections' throughput. Decreasing transfer times improves the user Quality of Experience, and boosts storage consolidation and virtual machine migration as well. As depicted in Figure 1, the envisioned network environment involves Cloud service users, potentially mobile, and data-center networks, with user-to-Cloud, intra-DC and inter-DC communications. Under this perspective, there are major challenges to address:

- How to send and receive data packets along different paths without disturbing applications?
- How to select disjoints paths in order to provide higher resiliency and to avoid bottlenecks?
- How to ensure packets really follow disjoint paths?

It is worth noting that using different paths in a uncoordinated way, TCP performance can be decreased rather than increased, namely because packet arrival disorder may trigger retransmission that may disturb the application workflow. Moreover, selecting Internet-wide end-to-end disjoint paths is commonly considered as an unrealistic dream, given the high heterogeneity and versatility of the Internet ecosystem. However, in the following we present rather simple functional blocks and protocol coordination mechanisms that are one step toward this goal, under realistic assumption and partially already available protocol functionalities.

In the following, MPTCP is considered as the transport layer protocol, as it is undoubtedly more scalable than other proposed alternatives and already deployable, even if it is deployed mostly at an experimental extent for the moment. MPTCP [2] is a TCP extension making use of several TCP subflows when possible to improve throughput and resiliency. It first asserts if the destination is MPTCP compliant (middleboxes such as firewalls might prevent the use of unknown TCP extensions), otherwise it falls back to legacy TCP. Once an MPTCP connection is established, endhosts can advertise their IPs, add or remove MPTCP subflows at anytime.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

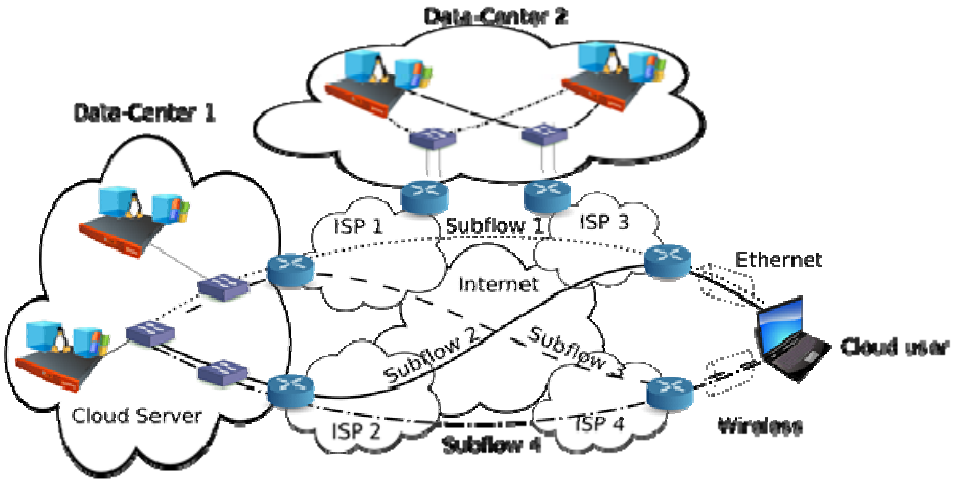


Figure 1 : Representation of the Cloud Networking Context

Basically a subflow could be defined as a TCP connection embedded in a more comprehensive TCP connection. MPTCP can mark a subflow as « backup only » to use it only if the other subflows stop transmitting. More interestingly, joint congestion control techniques using many subflows are documented in standardization documents. Performance gains are achievable if the round-trip-time (RTT) gap among subflows is not too high, so that packet disordering can be absorbed by TCP end-point buffering.

A. Cloud Network Elements

The Cloud fabric we envision should be sufficiently flexible to be extended and adapted to different technologies. The key role is taken by four node types:

- Virtualization Cloud servers, with MPTCP enabled at the hypervisor level.
- Cloud users, able to establish MPTCP connections even when single-homed.
- Border nodes, i.e., routers at the DC and user borders with its Internet Service Providers.
- Cloud Controllers, managing a DC network, enabling path discovery and establishment.

The implementation of MPTCP at the hypervisor level allows its scalable deployment: it somehow represents a TCP optimizer and it allows deploying MPTCP agnostic virtual machines. MPTCP can open multiple TCP subflows for a single TCP connection. These subflows differ with respect to their source or destination IP (if multihomed), or via the subflow TCP port (in case one server is singlehomed). Our proposition is an advanced yet simple protocol architecture that basically stitches MPTCP subflows to Ethernet- and IP-level paths, which are as much disjoint and RTT-similar as possible. The delivered network service improves resiliency and throughput, but comes at a cost: finding diverse and RTT-similar paths adds an overhead in processing time as well as in latency that may impede performance and scalability. Therefore, we foresee a special treatment only for flows for which the pros outweighs the cons, as described hereafter.

B. Functional blocks

A generic crosslayer multipath protocol architecture should implement the following blocks .

Flow Qualification Service: at the Controller or hypervisor level, it identifies which connection benefits from a multipath communication. The online classification ranks different metrics of a flow, e.g., jitter, latency, throughput, security, so that if needed we can sort flows according to policies. For instance, it appears appropriate to distinguish « elephant » flows (i.e., long-lived flows) from « mice » (i.e., short-lived) flows. The hypervisor triggers signaling for multipath communication for elephant flows only.

Flow Monitoring Service: at the Controller or hypervisor level, the state of either the network or the application might evolve during a communication, thus becoming obsolete. If a physical path becomes unreachable or underperforms, one may want to update the multipath configuration. With respect to RTT variations across subflows, the result of this service can allow virtually compensating the variations with artificial delays at the hypervisor software level.

Path Discovery Service: this service can be considered as a Traffic Engineering Database (TED) service; at the Controller or hypervisor level, it collects various information about the available path diversity, such as DC Link States, MTUs, load-balancers, DC and Internet topology, multipath capabilities, between the Cloud servers and users through border nodes . While DC-level discovery can be performed using existing operational tools, retrieving Internet path information can be difficult due to various factors (high versatility, unreliable knowledge of the topology, loose paths, etc.). In practice, the

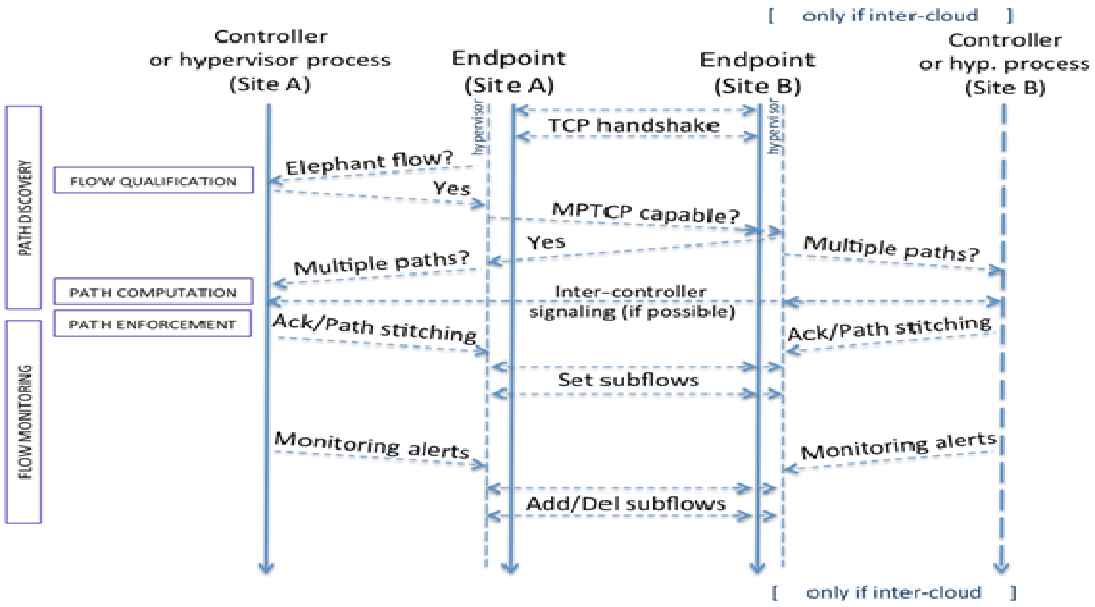


Figure 2. A signaling example in the case of extra-DC multipath communication

service is to be decomposed into an Intradomain Discovery Service (i.e., a controlled network scope, such as a DC network), and an Interdomain Discovery Service. One might also want to replicate the TED as a distributed database or cache, splitting it per layer or per subdomain, each using a pull or push discovery (these considerations are out of the scope of this document).

Path Computation Service: at the Controller or hypervisor level, this service is in charge of selecting paths to assign to flows when requested. The decision process takes into account constraints specified in the request (e.g., a jitter-sensitive path), and uses the TED to resolve matching paths and node capabilities.

Path Enforcement Service: once paths are computed, packets have to use these paths. This is possible through stateful and stateless modes:

- Stateful signaling: network path setup on per flow basis, e.g., using Software Defined Networking (SDN) or MPLS-based architectures.
- Stateful explicit forwarding: the source (i.e., the Cloud server's hypervisor, actively or passively via the Controller) lists in each packet the nodes the packet must pass through.
- Stateless forwarding: the source exploits network load-balancing algorithms and crafts packets' headers so that they follow foreseen paths [6].

C. *Multipath Communication Signaling*

Figure 2 highlights the signaling process and the associated functional blocks. For example, a TCP connection is established between two VMs in two different DCs in site A and site B. The hypervisor hosting the VM at site A detects the flow and queries the Flow Qualification process (could be a process running at the hypervisor, or an external controller). If the flow qualifies for a multipathed connection, the hypervisor acting as an MPTCP proxy verifies if endpoint B is MPTCP capable (or its hypervisor is); if yes, it queries the Controller for multipath capabilities to its destination. Controllers at the two sites may be able to collaborate in the computation of the paths. The Controller(s) compute(s) and enforce(s) the paths based on the information recovered by the Path Discovery Service (running ex-ante in a push mode or ex-post in a pull mode). In the stateful mode, intra-DC paths are virtually stitched with MPTCP subflows by the hypervisor, and at border nodes inter-DC (loose) paths are stitched with intra-DC paths. In the stateless mode, the hypervisor could be informed about the presence of load-balancers and possibly of the available path diversity at intra-DC and/or inter-DC segments. Data is eventually transferred using the multiple end-to-end paths. Guaranteeing a level of disjointness at the intra-DC and/or inter-DC level can allow boosting throughput, thus reducing transfer times. Appearance of network events such as node/link congestions/failures/additions could trigger respective hypervisors, causing subflow deletion or addition, possibly new path computation and enforcement.

In case the endpoint B is a user terminal, the right side of Figure 2 does not apply: there is no need for path stitching at site B. On top of that, if the user terminal is multihomed, the more access interfaces there are, the higher number of subflows can be opened. It is worth noting that neither the user terminal nor the DC nor the Cloud virtualization server need to be all multihomed: to open more than one subflow path, just one segment needs to have path diversity, where the segments are the intra-DC segment between Cloud servers and DC border nodes, the extra-DC segment between border nodes, and the end-to-end transport segment between terminal interfaces. ,

At each Cloud network segment, different protocols can act, hence interoperability between them is needed to propagate path diversity across segments. One solution could be having SDN protocols such as OpenFlow as ubiquitously as possible between source and destination, with however important scalability and reliability concerns. A reasonable alternative is to rely on three new distributed protocols recently defined to independently handling path diversity at each segment, with an adequate coordination as proposed hereafter.

.

III. Specific Implementation Using MPTCP, LISP and TRILL

In this section, we present one possible peculiar implementation of the previous architecture making use of three novel protocols: apart the already mentioned MPTCP [2] at the end-to-end transport segment, LISP [3] handles path diversity at the extra-DC segment between border nodes, and TRILL [4] can enable multipath Ethernet-layer communications within the DC. It is important to highlight that, since their standardization, all these protocols have been designed to be incrementally deployable in the existing Internet infrastructure.

Our intention by using these protocols is to address all the previously described use-cases as well as distributing path diversity across layers in a scalable and practical way. A key factor for this purpose is the multipathing ability they all offer (though optional in TRILL). These protocols do have also in common the functionality of mapping one upper layer logical point to many lower-layer logical points: one application port to many IP interfaces for MPTCP, one IP address to many IP routing locators for LISP, one MAC address to many Ethernet routing locators for TRILL. None of them are fully standardized, yet they have all been implemented to some degree: many vendors have started commercializing TRILL since a few months (e.g., Cisco, Huawei, Fujitsu); a TRILL OpenSolaris opensource version exists, and a Linux version is expected to be released closely. FreeBSD (OpenLISP[1]) and Linux (LISPmob[2]) versions of LISP are available; LISP is also already available in Cisco routers. Finally, a stable Linux version of MPTCP exists , adaptable to Android smartphones, so having it implemented at customer end-points and Cloud virtualization servers is not unrealistic. In the following, we first synthetically describe LISP and TRILL, and then discuss cross-layer coordination in the specific architecture. Finally, we present the results of partial experimentations.

A. Locator/Identifier Separation Protocol (LISP)

IP addresses assume today 2 functions : localization and identification of its owner. In LISP [3], each endpoint IP, named Endpoint Identifier (EID), is associated to one or many IP addresses of intermediate IP interfaces, named Routing LOCators (RLOC), typically supposed to be at border routers of the endpoint network. Upon reception of a packet from the local network to an outer EID, the border router acts as an Ingress Tunnel Router (ITR): it retrieves the EID-to-RLOC from a mapping system, and then it prepends to the packet a LISP header and an outer IP header with the RLOC as destination IP address. The receiving RLOC is an Egress Tunnel Router (ETR) that decapsulates the packet and forwards it to the destination EID. As the outer packet is a traditional IP packet, it can be routed on the legacy internet (though there might be Maximum Transfer Unit, MTU problems). If a site is not yet LISP compliant, the traffic might get encapsulated (or decapsulated) by a Proxy ITR (or a Proxy ETR). The usage of RLOC priorities and weights in the mapping system allows inbound traffic engineering, suggesting a best RLOC or an explicit load-balancing. An extension interesting for our architecture is the LISP Canonical Address Format (LCAF) [6] that, among other features, can allow enforcing explicit chains of RLOCs on the way toward the destination, hence boosting the traffic engineering capabilities in the extra-DC Internet segment.

B. Transparent Interconnection of Lot of Links (TRILL)

TRILL implements a logic close to LISP's at the Ethernet layer in order to solve switching tables scalability problems and improve network efficiency for DC environments. As LISP, TRILL uses data encapsulation with an outer standard header and a shim specific header, and it tunnels data units from an ingress node to an egress node, with a mapping system to update association of endpoint (MAC) addresses to network locator (MAC) addresses (this is a key feature as VM are migrated across DC racks, hence their location can be updated). On the other hand it differs from LISP since it integrates a multi-hop routing logic between ingress and egress nodes based on IS-IS (though LCAF somehow fills this gap). This is the reason why TRILL bridges are called RBridges (Routing Bridges). It is incrementally deployable: standard Ethernet segments can sit between RBridges, with each RBridge terminating a spanning tree instance. Multipath communications are therefore possible between RBridges at the Ethernet layer. It is worth mentioning that alternative Ethernet routing protocols exist, namely the IEEE 802.1aq Shortest Path Bridging (SPB), Layer-2 Label Switched Path (L2LSP), Provider Backbone Bridge with Traffic Engineering (PBB-TE) and (at some extent) OpenFlow; all could allow Ethernet multipathing too, yet both require a complete deployment at all bridges and especially for this reason they are, for the moment, often considered as less scalable and versatile.

IV. Specific Architecture

In an heterogeneous Cloud environment with MPTCP, LISP and TRILL, the Cloud network has MPTCP enabled at both endpoints, at the DC side either directly in the VM or (more reasonably) as a virtual middle-box in the hypervisor. TRILL

[1] http://www.openlisp.org (data-plane); http://www.lisp.ipv6.lip6.fr (control-plane)

[2] http://www.lispmob.org

is deployed between the Cloud virtualization servers and the DC border nodes, noting that an increasing interest in standardization activities is given to the implementation of RBridges also as virtual bridges at the hypervisor level. LISP is implemented at DC border IP routers, and it can be implemented by user endpoints (see [7]) handling each access interface as an RLOC, hence complementing MPTCP with an inbound traffic engineering control-plane when both are enabled in the user endpoint.

Therefore, from the one hand (DC side) TRILL and MPTCP coexist at the hypervisor level, and from the other hand (access side) LISP and MPTCP coexist at the user mobile node. It is also worth noting that, since a few weeks, the LISP data-plane is implemented in the well-known virtual bridge called OpenVSwitch[3], hence pushing down to the hypervisor LISP encapsulation/decapsulation and letting the three protocols coexist in the same DC node. While MPTCP capability at the endpoints is mandatory in order to enable multipath communications to the application layer, our architecture does not require LISP or TRILL to be implemented concurrently; a partial deployment with at least one of them is needed to associate loosely diverse path to subflows, yet both together would allow reaching higher performance. Finally, about the Cloud Controller, it can be based on classical Network Management Systems and the like, or on an SDN open controller accessed for instance via OpenFlow, or on a Path Computation Element (PCE[4]) [7] generalized for non-MPLS environments, or a mix of these technologies.

In the following, for each component described in the Functional Blocks section of the general architecture, we associate a specific solution based on MPTCP, LISP, TRILL and peculiar functionalities to close the gap between theory and practice.

Flow Qualification: the process can be implemented either as co-located with the hypervisor or as external server. The advantage of the latter case is to exploit network-level information and offload the hypervisor by excessive signaling, and also to implement advanced qualification criteria, for example based on destination's information and white/black lists. However, if simple criteria are used, e.g., a binary classification as elephant or mice flow, the implementation in the hypervisor does not need external information and appears as more appropriate. In such a case, a possible solution could be Mahout [8] implemented at the hypervisor level; basically, whether the socket fills up quicker than its emission rate, it marks the flow as elephant. In this case, we pursue the process, and proceed as explained in Figure 2.

Flow Monitoring: it is left to the MPTCP congestion control mechanism using per-subflow windowing. Upon failure of a TCP subflow (i.e. window size inferior to a threshold during a certain period of time), the Cloud server may request a new path to the Controller in order to replace this subflow with a new subflow following a new path. This request may notify the deficient path to the discovery service which may update the states about that specific path. As the MPTCP congestion control mechanism is per subflow, it is important that a subflow keeps using paths of equivalent quality; indeed, a change to a path of lesser quality may decrease the window, yet it takes time to recover the original window value.

Path Discovery: it can be decomposed into two subservices with different constraints.

- *intra-DC discovery service*: in DC environments, the assignment of VMs to servers should be orchestrated by a Cloud Management System that knows where each VM is placed. TRILL support this through an oracle-like system (non mandatory) called « directory system » [9], which returns the destination RBridge associated with a MAC, instead of resorting to ARP flooding, thus reducing L2 broadcast traffic. Associated to this information, the IS-IS substratum readily allows TRILL campus topology synchronization at RBridges, which can be enriched with TE metrics adopting the ISIS-TE extensions defined for IP networks [10] to TRILL-based Ethernet networks.
- *Interdomain discovery service*: MPTCP discovery is quite straightforward; if the distant host does not answer with the MPTCP capable option, then the connection falls back to legacy TCP. If both hosts are MPTCP compliant, they can either advertise their different IP interfaces to the other host, or directly try to open new subflows with these EIDs. As far as LISP is concerned, the hypervisor or the Controller can be easily enabled to query mapping servers to retrieve the RLOCs of the local site and the destination, along with LISP load-sharing information. Moreover, we may also try to rank Internet paths between xTRs, for example adding a TED companion to the LISP mapping information, using providers' services such as [11] or PCE-based [7], knowing however that Internet path information accuracy can be low.

The Controller could manage a single TED built merging TE metrics related to TRILL Ethernet routing tables, LISP RLOC metrics and inter-domain path metrics, as a separate database regularly and on-demand pulling data from TRILL, LISP and external databases.

Path Computation: this service will use the information gathered by the Path Discovery Service to compute a number of paths according to different constraints, which might be passed in the request (e.g., latency, jitter, throughput, bottleneck bandwidth). In the case of inter-cloud communications, the PCE communication Protocol (PCEP) [7] could be used to allow distributed computation between DCs, including also the possibility to involve the PCEs of external ISPs. The extra-DC path computation between LISP endpoints can include the possibility of involving

[3] http://www.openvswitch.org
[4] http://pce.ida-cns-group.net (opensource PCE implementation)

Reencapsulating Tunner Routers (RTR) between the ITR and the ETR [12] if LCAF is enabled.

Path Enforcement: both stateful and stateless methods are conceivable.

- *Stateful mode*: the computed path is enforced in a source-routing fashion. The way this can be implemented in TRILL and LISP does not rely on end-to-end reservation as in MPLS networks, but it implies the hypervisor is able to craft TRILL and LISP packets. This is proposed by [13] for TRILL: the hypervisor, called « TRILL smart endnode » directly queries the TRILL directory and encapsulates the packet with a TRILL header to lighten the load on the next RBridge. Even if not proposed elsewhere, with the above mentioned implementation of the LISP data-plane in virtual bridges such as OpenVSwitch, the hypervisor could become a « LISP smart endnode » as well. Conversely, explicit path prepending is described for LISP with RTRs and LCAF [6][12], allowing the enforcement of multihop xTR chains, and is currently not offered by TRILL. On the other hand, TRILL supports extensions, so we can imagine a similar feature implemented at RBridges.
- *Stateless mode*: with TRILL, we cannot use VLANs anymore to force a physical path, since any VLAN can get encapsulated in shared transport VLANs by the TRILL campus. However, with multipath load-balancing enabled, one can carefully compute the TRILL header's Time To Live (TTL) field so that packets with the same TTL follow the same path as of the result of hashing functions used in load-balancing, similarly to how described in [4]. This technique allows controllable per-flow load-balancing and prevents packet disorder. Similarly, the TTL in the IP header can also be tuned to enforce extra-DC egress load-balancing at ITRs.

To sum up, the stateful method adds LISP and TRILL headers overhead to packets, but in a controlled environment such as a DC, the MTU should not be a problem. As for the stateless method, enough LISP and TRILL nodes need to share the same hashing algorithm to make it interesting, which is a reasonable assumption. However middleboxes may also change the TTL value, thus nullifying the effect. Finally, in this specific architecture, hypervisors can implement all the services, but the heaviest ones such as the Path Discovery and the Path Computation Services that shall be taken on by the Controller.

To sum up, the stateful method adds LISP and TRILL headers overhead to packets, but in a controlled environment such as a DC, the MTU should not be a problem. As for the stateless method, enough LISP and TRILL nodes need to share the same hashing algorithm to make it interesting, which is a reasonable assumption. However middleboxes may also change the TTL value, thus nullifying the effect. Finally, in this specific architecture, hypervisors can implement all the services, but the heaviest ones such as the Path Discovery and the Path Computation Services that shall be taken on by the Controller.

V. CONCLUSION

Multipathed communications still remain a complex networking research field due to the necessary coordination between network layers and protocols. The architecture we propose in this paper unifies the different control planes thanks to the controller knowledge and coordinated routing mechanisms. Multipathed communications are becoming essential to augment Cloud communications; for instance, very recent experiments have shown that significant throughput could be achieved either locally - a local MPTCP throughput of 51.8Gbit/s between 2 servers containing each 3 dual-port 10 Gig NICs has recently been proven[5] – or on long distances - an intercontinental testbed [14] achieved a throughput of 15Gbits/s from Geneva to Salt Lake City. Our cross-layer multipath architecture can potentially overcome these values so as to further push network innovation at Internet edges. Indeed, coordination between the MPTCP, TRILL and LISP protocols, splitting flows at the transport, network and Ethernet layers, can allow carefully selecting communication paths, while controlling the computational load, and guaranteeing a semi-distributed reliable nature to the communication environment.

ACKNOWLEDGMENT

This article was partially supported by the NU@GE project (http://www.nuage-france.fr), funded by the French «Investissement d'avenir» research programme, and the FUI 15 RAVIR (Réseaux d'Accès Virtualisés au Cloud) project.

REFERENCES

[1] A. Akella et al., On the Performance Benefits of Multihoming Route Control. IEEE/ACM Transactions on Networking, Vol. 16, No. 1, pp : 91-104, 2008.
[2] A. Ford, C. Raiciu, M. Handley, O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses, RFC 6824, January 2013.
[3] D. Farinacci, V. Fuller, D. Meyer, D. Lewis. The Locator/ID Separation Protocol (LISP), RFC 6830, January 2013.
[4] R. Perlman, D. Eastlake 3rd, D. Dutt, S. Gai, A. Ghanwani. Routing Bridges (RBridges): Base Protocol Specification, RFC6325, January 2011
[5] G. Detal et al., Revisiting Flow-Based Load Balancing: Stateless Path Selection in Data Center Networks, Computer Networks (in press).
[6] D. Farinacci, D. Meyer, J. Snijders. LISP Canonical Address Format (LCAF), draft-ietf-lisp-lcaf-02, March 2013.
[7] A. Farrel, J-P. Vasseur, J. Ash. A Path Computation Element (PCE)-Based Architecture. RFC 4655. August 2006.
[8] A. R. Curtis, W. Kim, P. Yalagandula. Mahout: Low-Overhead Datacenter Traffic Management using End-Host-Based Elephant Detection, in Proc. of IEEE INFOCOM 2011
[9] L. Dunbar, D. Eastlake, R. Perlman, I. Gashinsky. TRILL Edge Directory Assistance Framework. draft-ietf-trill-directory-framework-04, Feb. 2013.
[10] H. Smit, T. Li, IS-IS Extensions for Traffic Engineering (TE), RCF 3784, June 2004.

[5] http://multipath-tcp.org/pmwiki.php?n=Main.50Gbps

[11] D. Saucez, B. Donnet, O. Bonaventure, Implementation and preliminary evaluation of an ISP-driven informed path selection, in Proc. of ACM CoNEXT 2007.
[12] D. Farinacci, P. Lahiri, M. Kowal, LISP Traffic Engineering Use-Cases, draft-farinacci-lisp-te-02, Jan. 2013.
[13] Radia Perlman et al., TRILL Smart Endnodes, draft-perlman-trill-smart-endnodes-01, Jan. 2013.
[14] R. van der Pol, M. Bredel, A. Barczyk , Experiences with MPTCP in an intercontinental multipathed OpenFlow network in Proc. Of SC201

Enabling Information Centric Networking in IP Networks Using SDN

Markus Vahlenkamp, Fabian Schneider, Dirk Kutscher, Jan Seedorf
NEC Laboratories Europe, Heidelberg, Germany
<first.last>@neclab.eu

***Abstract*—In this paper, we show how to enable Information-Centric Networking (ICN) on existing IP networks, such as ISP or data center networks, using Software-Defined Networking (SDN) functions and control. We describe a mechanism that *(i)* enables addressing and transfer through non-SDN controlled networks (i. e., the Internet), *(ii)* allows to identify ICN requests and responses, *(iii)* decouples forwarding from the object name, *(iv)* requires neither new or extended network/L3 and transport/L4 protocols nor changes of client and server OS, and *(v)* supports aggregation of routes inside the SDN controlled network. In addition, the proposed solution is agnostic of the specific ICN protocol in use, and does not require all network elements to be SDN-enabled. It supports advanced ICN routing features like request aggregation and forking, as well as load-balancing, traffic engineering, and explicit path steering (e. g., through ICN caches). We present the design as well as our first implementation of the proposed scheme—based on the Trema OpenFlow controller-framework and CCNx—along with initial performance measurements showing the feasibility of our approach.**

I. Introduction

Software-Defined Networking (SDN) is based on the concept of providing an API for packet forwarding devices such as switches and routers, which allows programmability of network elements and entire networks. In this paper, we apply the general SDN concept for a new type of networking: Information-Centric-Networking (ICN) [1]. In ICN, communication is not based on packets that are "sent from" and "destined to" hosts or host interface addresses. Instead, requesters send requests to the network, asking for Named Data Objects (NDOs) that have been published before and that are available in one or many copies in the network. The network elements that receive a request—unless they have a local copy in their cache—typically have to decide where to forward the request to, for example which interface to use for forwarding the request. Usually, network nodes participate in a routing protocol that helps to distribute this information. Alternatively, a network node can employ a Name Resolution Service (NRS) that can map NDOs to locators in underlying networks, for example IP. Once a request has been forwarded and reached the destination (e.g., a cache holding the copy of the NDO), a corresponding response message (i. e., the NDO itself or a locator) has to be relayed back to the requester, possibly passing one or more on-path caches that can cache the objects in order to satisfy future requests from their cache. This return path can be determined in different ways: For example network elements can maintain state or they can obtain some information from data structure inside messages, such as a label stack.

This paper addresses the problem of decoupling the ICN data plane (ICN request and NDO forwarding) from the ICN control plane, so that name-based routing and name resolution can be performed independently. In many networks where ICN becomes interesting, such as data centers, a centralized ICN control plane is viable and can take on additional functions. Examples include dynamically replicating NDOs across multiple nodes and load-balancing both ICN nodes and network resources.

The mechanisms for software-defined ICN that are described in this paper enable control plane entities ("controllers") to implement arbitrary control protocols for deciding about request/response forwarding. Software-Defined-Network elements (e.g. OpenFlow switches) can be kept relatively simple, focusing on efficient request forwarding. In particular, these data plane elements do not need to be ICN-enabled with the approach we present. A controller, for example, could be programmed to decide about the next hop for every message that a network element has no information for. It could also participate in large scale name resolution or name-based routing and provide corresponding hints to network elements. As a result of this approach, the ICN network can be dynamically adapted. Network elements can be kept simple and do not need to know about ICN routing or name resolution.

In the following subsections we give an overview on ICN deployment approaches, remaining challenges and the benefits of our solution. In Section II we explain our solution, explain its components, and detail on how to process ICN messages. In Section III we explain our first implementation of the approach using the Trema SDN framework and CCNx, and present preliminary evaluation of our implementation. We conclude and outline our next steps in Section IV.

A. Existing Approaches to Realize ICN

An ideal or native deployment of ICN requires all network elements, user devices, content sources as well as all intermediary network elements (routers, switches, ...) to be ICN aware. We do not believe that in the near future such an ICN deployment is viable given the need to exchange or at least update all networking equipment installed today. Therefore in this paper we aim for a step-by-step migration towards more and more part of the network being ICN enabled.

Running ICN approaches on the existing Internet—which is tailored to support host-to-host communication using IP and TCP/UDP—requires additional mechanisms. The most common approach is to create an overlay network on top of the existing Internet and implement the additional routing and

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

forwarding mechanisms in the overlay software. In particular overlays cause overhead for overlay management and encapsulation inside Internet protocols. The overlay approach and its overhead has been well explored in the context of P2P systems.

A slightly different solution has been presented by Blefari-Melazzi et al. [2], which aims in a similar direction as our approach. Their approach is based on OpenFlow switches and dedicated border nodes which perform name-to-location resolution (with the help of an external system) for the requested NDO. The proposal defines a new IP option which is supposed to include the requested NDOs name and an ICN specific transport protocol. Because the new IP option cannot be matched upon by OpenFlow, the border nodes encapsulate the original request in a new packet. A flow identifier (which represents the object's name with a 1:1 mapping) is embedded within the transport protocol's port numbers. Then the OpenFlow controlled network can use these port numbers to forward the packet to the appropriate location(s). The border node will keep a pending interest table to allow reversing the encapsulation and sending the response to the original requester.

B. Remaining Challenges and Problems

While aiming in a promising direction (i. e., moving away from overlay networks for deploying ICN), the approach of Blefari-Melazzi et al. leaves a set of problems and challenges:

- How does a client get to know the identity (and address) of a border node from outside the network, or where would it send its requests initially?
- The border node needs to be ICN-aware, run ICN software, and perform packet en/decapsulation. While this is an easy task for custom software running on a computer, this is difficult on today's network elements and at high network speeds. Moreover operators are reluctant to put "middle boxes" in there network, rendering this ICN-aware border node option unlikely to be widely adapted.
- By using IP options, packets will typically pass through the slow-path in network elements (i. e., they will be processed by the network element's GPU rather than the optimized network processing pipeline).
- The approach requires a new transport protocol which might be blocked at various network elements (esp. those outside administrative control).
- ICN nodes (both requester and caches/servers) cannot operate on default UDP (or TCP) sockets, thus requiring special kernel modules.
- ICN nodes need to know how to communicate with the name resolution service.

The approach we present in this paper solves all the problems above, while at the same time preserving the benefits of existing solutions, e. g., close(r) to native—all network elements are ICN aware—ICN deployment and support for aggregation of destinations/routes/rules so that forwarding tables remain manageable.

C. Benefits of our Solution

Our SDN backed ICN deployment approach seeks to provide the following benefits: *(i)* Facilitate ICN deployment over existing networks. We particularly focus on initial and partial ICN deployments that can emerge with time. We achieve this by allowing for the use of off-the-shelf network protocols and network stacks in host operating systems (i. e., unmodified IP and UDP) and enabling ICN hosts to send requests to that ICN network without any knowledge about cache locations and ICN node addresses. *(ii)* Routing or content location learning, which is a major problem in ICN solutions, is simplified through separating routing and forwarding through an SDN approach. The centralized view of the controller enables easier path selection/finding and more sophisticated forwarding decisions. Moreover, state maintenance and complexity requirements on network elements can be kept low and actually moved between SDN switches and controllers. *(iii)* Routing can be further improved through knowledge about both interests/requests and publications/content locations. Multiple requests can be aggregated into one or split across multiple paths. The controller can also manage cache content by triggering the pre-population/synchronization of caches or by routing contents through caches. The controller can also consider network utilization in the forwarding decisions thus achieving traffic engineering.

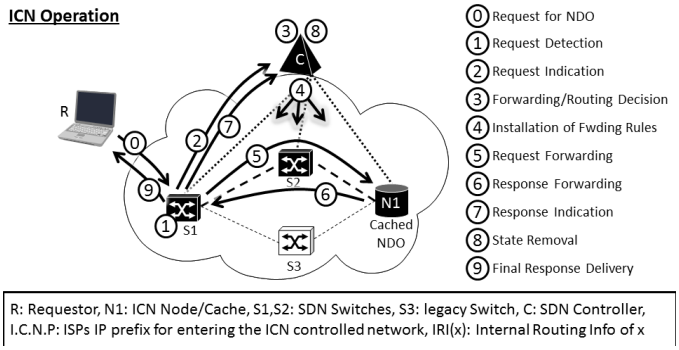


Fig. 1. Overall Operation

II. Proposed Approach

The generic method of our approach is to use SDN mechanisms to *(i)* detect ICN requests in a network, and to *(ii)* set up forwarding and return paths for ICN request/response message pairs. The general idea of our approach is to include a (message) identifier inside the default packet headers of network (IP) and transport (UDP or TCP) protocols while traversing the SDN controlled part of a network. This SDN controlled part should include most of the ICN nodes in a network. Then the identifier can be matched upon to determine forwarding paths through the SDN network[1], thus enabling close to native ICN forwarding. In order to direct packets to this ICN, we propose an IP prefix to which user devices should address their request packets. The presence of this prefix is also used to differentiate rewritten (inside SDN control) from original (outside SDN control) packets. Because the path setup inside the SDN network is determined by an ICN-aware SDN controller all of the typical ICN benefits can be realized without the need for ICN-aware network elements.

A. Overall SDN controlled ICN operation

Figure 1 depicts the overall operation: A network consists of ICN requesters (R), network elements (S1 to S3) and one or more object caches that hold copies of NDOs. The network is SDN-enabled and controlled by at least one SDN controller (C). An ICN-enabled requester R sends an ICN request for an NDO to an SDN network (step 0). A network element (S1) in that SDN network identifies (based on earlier configuration provided by an SDN controller) the request as an ICN request (step 1). The network element sends a request indication (containing the request packet) to the SDN controller (step 2). The SDN controller is expected to have ICN routing information and can identify one or more destinations for the request (forwarding decision step 3). The controller uses SDN control mechanisms to install forwarding rules in the network (step 4). These forwarding rules enable network elements to forward the request to the selected cache (step 5) and to forward an eventual response message back to R. Once the request has reached the Object Cache, it generates a response message with the requested object. This response message is forwarded according to the previously installed forwarding state back to S1 (step 6). S1 indicates the response message reception to the controller (step 7) which can then remove any state about the request (step 8). Finally the message is delivered to the original requester (step 9).

ICN protocols generally enable access to NDOs in network. Such protocols typically provide different message types such as a GET request (called ICN request above) and a corresponding response message. Parameters of a GET request include the name of the requested NDO, and often, as for the specific NetInf ICN protocol [3], a message identifier (Message ID) to distinguish messages in the network. However, our approach does not enforce a specific name structure.

We assume an ICN protocol on top of the Internet Protocol, specifically we assume that ICN messages are transferred with UDP and TCP. Yet, a dedicated ICN transport protocol would also work with our approach. In addition, we consider the ICN protocol messages to be read-only. However, the SDN controller needs to be able to understand the ICN application protocol. In particular, it needs to recognize different message types, the NDO's name, and the Message ID. The generic method is to enable an SDN controller to install appropriate forwarding state for an ICN request in a way so that network elements only have to support IP forwarding and do not require ICN protocol knowledge. This method is leveraging ICN protocol Message IDs and features of SDN instantiations such as OpenFlow [4, 5] to rewrite packet header information. The details of the method are explained in the following subsections.

B. Targeted Scenario & Conventions

Our solution focuses on the operation within a controlled IP network domain. However, the solution is as well applicable for a multi-domain scenario. In general we assume that the requester is outside the controlled network domain. The solution supports partial SDN deployment within the network domain. For example, assume an Internet Service Provider (ISP) that wants to deploy ICN caches (within domain) for its customers (outside domain). We begin by assuming UDP as transport protocol for ICN and explain the basic procedures. Later we discuss how TCP can be used for that purpose. For our solution we require:

1) An *ICN Protocol Identifier* that is carried by all packets belonging to ICN messages. A key requirement for this is that SDN network elements can match on this identifier. We propose to use a dedicated transport protocol port number, which is used as destination or source port depending on the message type (request or response). Another option would be a dedicated transport protocol which can be matched in the respective IP header field.
2) A *publicly routable network address* per domain. The purpose of this address is to allow for an easy mechanism to determine a target IP address from outside the controlled network domain. This can also be used to identify packets that have not been changed for SDN forwarding. We propose to select a specific IP prefix (can be a /32 for v4 or a /128 for v6) from the network domain's IP address pool. This could be announced on every SDN enabled network element and made public via e. g., DNS. Note, that the network domains IP routing protocol(s) should be used to distribute this prefix towards the edge of the administrative domain. This explicitly allows to announce this one prefix at multiple locations in the network, thereby accounting for geographically distributed entry points to the ICN/SDN controlled part of the network. We further assume that each ICN user device only needs the ICN IP prefix of the network operator it is currently connected to.
3) The *object's name*, which is used to determine the path of the packets (routing).
4) A *Message ID*, which will be used for forwarding decisions in the network elements on the determined path. The Message IDs will replace destination IP addresses in requests and source IP addresses in responses.
5) The SDN needs a *controller*, which has *knowledge about the location of NDOs* inside the network domain. And it needs to be able to *setup forwarding rules on all SDN network elements* in the network domain.
6) The SDN network elements need to support NAT-like IP address and port rewriting.

While 1, 3, and 4 (having a Message ID) are common to most ICN solutions and 5 (routing logic) is outside the scope of this paper, 2, 6, and 4 (encoding the Message ID in the address fields of the IP header) is a new contribution of our work. Additional contributions are the request/response processing procedures described in the following.

C. Request forwarding

Figure 2 shows the processing of ICN requests using the six requirements from the last section. When a Requester R wants to query an object from the ICN service it will first perform a lookup for the publicly routable network address specifying the ICN prefix (I.C.N.P) of the respective domain. Once the prefix has been determined an IP packet is sent (Step

[1] Strictly speaking it is rather a "path identifier" than a message ID.

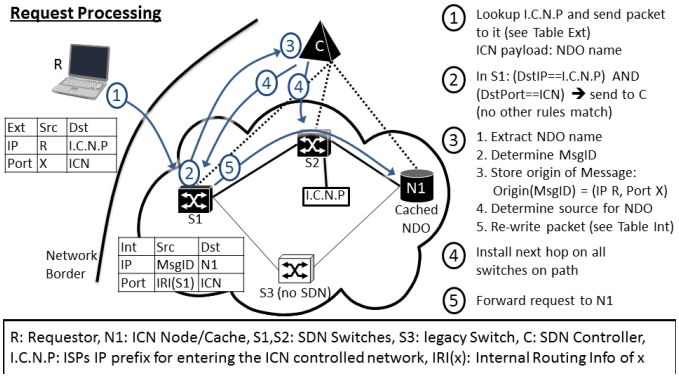


Fig. 2. Request forwarding

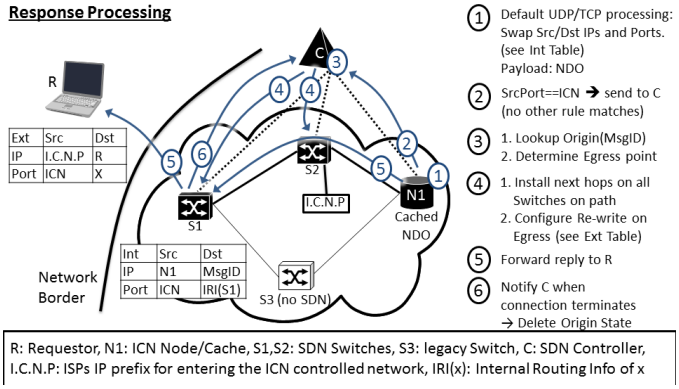


Fig. 3. Response processing

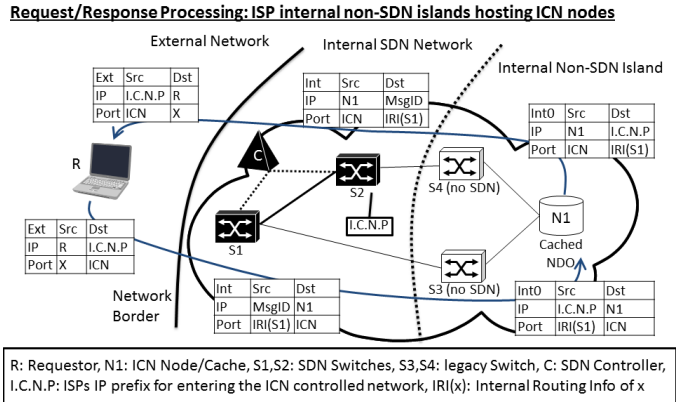


Fig. 4. Processing with non-SDN islands

1) with destination IP address of I.C.N.P. The destination port number will be set to the ICN Protocol Identifier. As usual, the source IP address and port number will be determined by the Requester's IP stack. For this example we chose R to represent the requester's IP address and X to represent the local port number at R (see also Table 'Ext' in Figure 2). As application payload R will follow the specification of the ICN protocol which includes at least the name of the requested object.

I.C.N.P must only be announced by SDN-enabled network elements. This will ensure that IP routing sooner or later delivers the packet to S1, the "ingress" SDN network element. S1 will try to match (Step 2) the packet to its rules and find that only the "default ICN" rule matches. This rule matches for the combination of destination IP and destination port and sends packets to the Controller C if they are I.C.N.P and ICN. Upon reception of the packet by the Controller C (Step 3), C will parse the ICN protocol payload and extract the requested object name. If the ICN protocol does not use 32bit Message IDs a new random (and currently not used) Message ID is generated. The origin of the request (that is IP address R and port number X) are stored. Next the location of the NDO, that is the address of a cache that can serve the requested object, is determined. Then comparable to NAT the IPs and Port numbers of the packet are rewritten (see also Table 'Int' in Figure 2): The new destination IP is the IP of the cache from which the object should be served. The new source IP is the MsgID. The destination port number is kept to identify the packet as ICN. In addition the source port number is changed, in order to identify the "ingress" network element. This internal routing info (IRI) can be used to aggregate rules (read routes) for response packets. Next (Step 4) C installs forwarding rules (read routes) on all the SDN network elements on the path to N1 and sends the re-written packet back to S1. All these rules must include a check if the port number matches ICN. The rules can either use the MsgID as match criteria, or can be aggregated using the destination IP addresses. Eventually (Step 5), the packet is forwarded to N1, the cache serving the object. Using the IP address of N1 as destination IP one could even use non-SDN paths (e.g. S1→ S3→ N1) towards the cache.

D. Response forwarding

Figure 3 shows the processing of ICN responses. The ICN node N1 does not need a special network stack. It can simple listen to the ICN port. When requests arrive they will be processed and a corresponding response (positive or negative) will be generated. As with usual UDP and TCP processing for the response packet(s), the IPs and port numbers are swapped (Step 1, see Table 'Int' in Figure 3). Now two options exist: 1.) N1 has a build-in SDN stack or 2.) N1 is configured to send all ICN traffic out on a port connected to an SDN controlled network element. Either way (Step 2), the first SDN enabled element will forward the first response packet to the controller. This can be achieved by matching for ICN in the source port. However the controller might already have installed a specific (based on the MsgID) or aggregated (based on the IRI) rule to forward responses; in that case the packet(s) are not sent to the controller. When the controller receives a response packet (Step 3) it will lookup the origin of the MsgID of that packet and determine a forwarding path back to the requester. This path and the reversal of the address and port re-writing rules will be installed (Step 4). Then (Step 5) the network forwards the packet(s) to the origin, performing the address and port re-write on the egress SDN node. When the response has been completely delivered to R the egress SDN node will notify C that all state for MsgID can be removed (Step 6). Note that while request forwarding is possible over non-SDN controlled network elements, the response forwarding does not work as it is using the MsgID as destination IP addresses.

E. Other Aspects and Benefits

Non-SDN-controlled network parts: When ICN nodes are deployed in non-SDN controlled islands, communication is still possible. Figure 4 shows an example. Since we cannot use the MsgIDs as destination IPs for the response packet(s), we use addresses from the already deployed I.C.N.P prefix. However, as we want to support existing network stacks, which only swap IPs and port numbers, we also need to change the request. Thus a solution is a second re-write operation before forwarding the requests to the non-SDN controlled part of the network.

Dealing with failed requests / Cache does not have the NDO: If the cache that is supposed to deliver the NDO does not have the NDO it cannot fulfill the request. This can happen either because the controller did not have complete information, or because the cache evicted the content to cache other objects. There are two options to deal with this situation. First, the cache can reply with a ICN message indicating the content is not available. Second, the cache can "forward" the request. The handling for the second case is shown in Figure 4.

ICN Request Aggregation and Request Forking: Request Aggregation in ICN is the concept of avoiding to send two requests for the same NDO through the whole network. If some part of the path towards the cache is shared, the ICN approach is to transmit the request and NDO only once over the shared path. Realizing this with an SDN controlled network is easy: When a second request for the same object arrives before the response of the first one was initiated, the response path for the NDO can be setup to include copying the response to both ingress network elements. Request Forking is the opposite of request aggregation. The ingress node will duplicate the request and send it out to multiple caches. Once the first cache generates a response, the other request can be ignored. With our approach, the SDN controller will generate multiple copies of the request and target them to different ICN nodes/caches.

F. State Management Trade-offs

The proposed solution does allow to trade-off state complexity in the network elements for complexity in the controller. That way the approach can be customized towards the capabilities of the deployed network and the degree of ICN utilization. To reduce the state in SDN network elements, SDN rules can be aggregated by destination. It is not necessary to install rules per MsgID. For the request direction default IP route aggregation is possible because the destination IP is a routable address. For the response direction the internal routing information (IRI) that is embedded in one of the port numbers can be used. An alternative possibility is to send all ICN packets to the controller and let it make all decisions. Rules do not need to be installed.

Likewise it is also possible to reduce the state in the controller by doing one of the following: 1) The response path can already be configured when the request is processed; 2) The address re-writing at egress can be pre-configured. No request origin information on the controller; 3) In combination with SDN rule inactivity timers the whole processing can be performed in a "fire-and-forget" manner from the Controller's perspective.

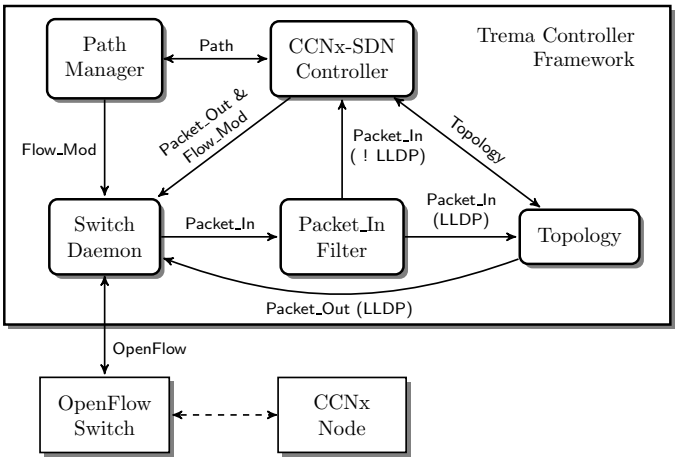


Fig. 5. ICN-SDN implementation architecture

III. Initial Implementation & Evaluation

In this Section we first provide an overview of our prototype implementation, called *CCNx-SDN-Controller*, which is based on CCNx [6] and Trema [7]. Then we briefly report on our initial evaluation of the approach.

CCNx is PARC's open source implementation of their "Content Centric Networking (CCN)" [8] ICN approach. Trema is an open-source OpenFlow controller framework developed by NEC, which allow to easily implement arbitrary network control applications.

A. CCNx-SDN-Controller Implementation

Our Controller implementation consists of different parts, as depicted in Figure 5. For each controlled switch (called Datapath in the OpenFlow terminology), the Trema *Switch Manager* (not depicted) forks a *Switch Daemon*, that is responsible for the communication with its associated *OpenFlow Switch*. Each packet yielded by the *OpenFlow Switches* is, via the *Switch Daemon*, first delivered to the *Packet_In Filter*, whose task is to filter out Link Layer Discovery Protocol (LLDP) packets and deliver them to the *Topology* component, which is likewise generating and receiving those LLDP packets for the purpose of detecting the network topology. According to our *Packet_In Filter* configuration, all other packets are handed over to the *CCNx-SDN-Controller* process.

The *CCNx-SDN-Controller* maintains a *FIB* to manage the object name to ICN node address mappings. It learns this information from observing name prefix announcement traffic of the controlled *CCNx nodes*. Moreover, it processes the incoming packets and further sends information back to the *OpenFlow Switch*, via Packet_Out messages, which are handed over to the actual *Switch Daemon*. The *CCNx-SDN-Controller* further utilizes the *Topology* component to calculate paths through the network. The *Topology* component, when queried with ingress and egress Datapath and there associated ports, provides a list of Datapath_IDs and there associated ingress and egress ports, that form the requested path. The actual path provisioning is subsequently partially delegated to the *Path Manager*, that consistently handles the path creation. It is just asked to provision the basic forwarding path, whereas the switch, that has to perform the packet rewriting actions is instructed by the *CCNx-SDN-Controller* itself. The flow entries

provisioned this way are created without a hard lifetime, but an idle timeout of 10 seconds. A side effect of this mechanism of deploying the path, is, that the *CCNx-SDN-Controller* will explicitly be notified by the *Switch Daemon* whenever its flow entries are removed.

ICN Request Path Provisioning Based on the source IP and port information the controller generates a MsgID. This MsgID along with FIB information is used to program the packet header rewriting according to our approach. We pre-provision the path to the content source on all intermediate switches using the *Path Manager* and send the Packet_Out message to the ingress switch.

Response Path Provisioning The backward path for content delivery can also be pre-provisioned upon request processing. However to allow finer grain control over the ICN operation, the response path can be explicitly provisioned as well. For this we reverse the tasks from the request provisioning, i. e., , first setup the path to the egress, and then programm the header re-writing.

B. Evaluation

Our prototype implementation shows that the benefits discussed in Section I-C can indeed be achieved and that our approach is viable and solves the challenges outlined in Section I-B. We do not use IP options or new/modified transport protocols, and thus we can implement our solution on existing operating systems. Also our solution does not require complex communication between ICN nodes and a name resolution service, as our ingress nodes can be simple SDN switches. Thus we avoid the placement of yet another set of middleboxes.

To further analyze the applicability of our concept, as well as to get a first impression of the effects on network performance, we set up an emulated network environment to run experiments with our ICN-SDN approach in comparison to a simple direct communication and overlay configuration. We find that our *CCNx-SDN-Controller* produced transfer times similar to those of a best case non-SDN scenario (in which the content consumer is already aware of the actual node that serves the requested content). Hence no additional hops are to be traversed.

We do not assume that this will be the common case. Very likely additional hops that perform CCNx routing decisions or name-to-location resolution will be required to reach a content serving node. So typically an ICN deployed as an overlay will correspond to a multi-hop ICN communication. We also ran experiments for such a scenario with 2 hops, which is representative of the ICN edge node design of Blefari-Melazzi et al. [2]. In such a scenario the transfer time increases by one third. Thus, our SDN approach provides even better performance in addition to the deployability benefits outlined above.

Our results are very initial and we plan to follow up with a broader evaluation and refinement of the approach.

IV. Conclusion & Future Work

In this paper we have introduced a mechanism to deploy ICN protocols in IP networks via the assistance of the SDN paradigm. Our approach utilizes smart packet header rewriting in combination with a single IP prefix to initially contact the ICN network. This approach is backed by the centralized SDN controller, that is used to generate paths through the SDN controlled network domain. The benefits include ease of deployment in existing IP networks, not requiring additional transport protocols or extensions to the IP protocol, and separation of the ICN data and control plane. The latter enables simpler ICN routing mechanism through centralized knowledge as well as improved traffic engineering capabilities. Furthermore our approach allows trading the need for keeping state in network elements for reduced interactions between the controller and network elements.

Through an implementation and subsequent evaluation, we have demonstrated the practicability and benefits of our approach. Our preliminary results show that compared to a typical overlay deployment we can significantly shorten the transfer times, and achieve almost optimal performance. As future work we intend to advance our general approach further, extend the evaluation, and apply it to different ICN instantiations.

Acknowledgment

This work has been supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, 2012.

[2] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An openflow-based testbed for information centric networking," in *Future Network Mobile Summit (FutureNetw), 2012*, 2012, pp. 1–9.

[3] D. Kutscher, S. Farrell, and E. Davies, "The NetInf Protocol," draft-kutscher-icnrg-netinf-proto-01, Internet Research Task Force, Aug. 2013.

[4] Open Networking Foundation, "OpenFlow Switch Errata, Version 1.0.1," 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.1.pdf.

[5] OpenFlow project at Stanford, "OpenFlow Switch Specification, Version 1.0.0," 2009, http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[6] "CCNx OpenSource project," 2013, http://www.ccnx.org/.

[7] "Trema project," 2013, http://trema.github.io/trema/.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09, 2009, pp. 1–12.

End to End Solution Testing over Software Defined Networking – Future Challenges for Service Providers

Saurav K Chandra
Senior Technology Architect
Infosys Limited
Bangalore, India
kantichandra@infosys.com

Roshni K. K.
Technology Architect
Infosys Limited
Bangalore, India
Roshni_kk@infosys.com

Abstract— **Adoption of Software Defined Networking has a disruptive impact on communication network deployment. Hence the adoption must start with a meticulously formulated deployment strategy. Deployment strategies are formulated on a few key decision points. These decision points are usually a combination of right business cases, interoperability with the existing networking standards, cost of deployment, integration effort requirement etc. The formulated strategy must be implemented to complete the successful adoption. Service providers may face a number of challenges while implementing the deployment strategy. One of those major challenge areas is the testing of the services on top of Software Defined Networking.**

Software Defined Networking changes the fundamental way a routing and switching system works. It decouples the data plane from the control plane and places the same in a general purpose computing system. This enables the possibility of software-wise programming and defining the network behavior. The real usage of Software Defined Networking in the service provider environment lies in bringing out network services which are not possible or very difficult by using traditional methods. The above unique business case of Software Defined Networking foretells that most of the future solutions riding on Software Defined Networking will come up with so far never seen deployment scenarios. This situation demands a rethinking of the end to end solution testing on Software Defined Networking.

This paper studies the existing end to end testing methods and attempts to identify the gaps in them for Software Defined Networking testing. This paper also attempts to bring out some additional unique challenges posed by Software Defined Networking compared to the traditional network testing. Finally, this paper summarizes the challenges and provides an approach of test strategy and mitigation of the challenges. We believe this paper will add value to various users of the Software Defined Networking technology, especially the testing stake holders like test equipment manufacturers, networking product verification team, service provider integration testing teams and every testing enthusiasts of this technology.

I. INTRODUCTION

End to end solution testing is a very important step for service providers to ensure the network and services are available, secured and dependable. The end-to-end solution test strategy involves an elaborate mapping among business functions, network functions, service functions, and user experience. The focus of a service provider's testing is to verify and validate the end-to-end functionality of the network.

Software Defined Networking is in its early stage of adoption and service providers are investing in research and small scale pilot deployments to strategically weigh the business benefits against the disruption. This is the time when a well-defined strategy needs to be thought out for adoption, deployment and testing. A holistic testing strategy will include devices, networks, platforms and most importantly the solutions. It is not that easy task to arrive at this strategy because there are numerous challenges in testing Software Defined Networking enabled network.

II. CURRENT STATE MAP AND GAP ANALYSIS OF TESTING

The testing strategy for the network of wireline and wireless service providers has evolved to maturity in past years keeping pace with the evolution of switches and routers. Basic packet forwarding at line rates is not the only activity that a switch performs today. Switches handle various other functionalities like network isolation, traffic engineering, packet processing, payload analysis etc. A switch also takes care of changing network topologies and processes the link failures and network error conditions. They have become scalable to meet the demands of next generation networks. The overall complexity of switches continued to increase. As a result, network testing became complex and it needs sophisticated tools and skilled resources.

The current testing strategy focuses more on vendor equipments and feature sets coming along with the equipment. It includes a set of standard protocol compliance testing. At a network level the testing concentrates on manageability of the equipments like provisioning, monitoring and maintaining. It includes service testing but the majority of that is concentrated on the infrastructure which hosts different components in Operation Support Systems and Business Support System components. There is

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

a very limited requirement of testing from the network side to ensure service quality because the legacy network supports a limited flexibility in creating a network configuration.

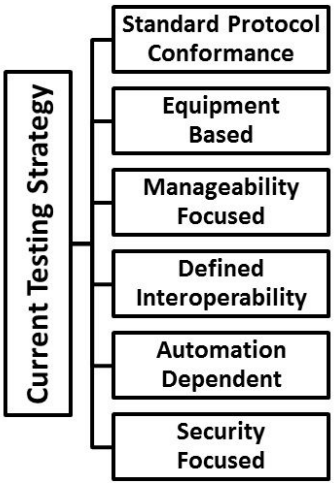


Figure 1: Current Testing Strategy

Software Defined Networking has introduced a fundamental change for the network. The control plane is decoupled from the data plane. Control plane, residing on a controller, has capability to run on a general purpose server controlling the switches. Switches have become mere packet forwarding engines. Multivendor switches can be controlled by single controller as there are a few open and interoperable standards like OpenFlow. The hardware centric implementation has now become software centric. It relies much on the software running on the controller.

The current Software Defined Networking is coming with an ecosystem behind it. Controllers, besides controlling the network, are opening up APIs for the ecosystem to build applications. The applications are capable of performing network automation, predictive maintenance etc. and essentially adding value to the services. This is definitely bringing in higher degree of vendor independence, but also introducing a bunch of testing challenges.

A. Gaps in the Current Strategy Posing Challenges In Testing over Software Defined Networking

1. Testing the cohesion in hybrid network:

The major challenge faced by service providers is ensuring the cohesion between legacy network and Software Defined Networking in delivering network services. Service providers would prefer a hybrid network with islands of legacy and Software Defined Networking capable switches. An end-to-end network service solution would traverse through the islands. Testing the end-to-end solution would ensure that the services are not impacted due to conflicts originating between the different control planes of islands.

2. Testing the programmability of network:

Providing programmability is a key value addition of Software Defined Networking. The control layer being abstracted, it provides a way of programming the network for specific services. The programmability comes in the form of network scripting language, flow modeling language, APIs to be used in application development or user interfaces. The objective of the programmability is to provide a level of easiness to service providers in creating and controlling services over the network while making them fool-proof from mishandling. The testing strategy would need to include testing the easiness and robustness.

3. Testing the manageability & operability of network:

The primary objective of manageability and operability testing is to ensure error-free service creation and service monitoring to meet customer service level agreements. Traditionally the manageability & operability of the network is carried out by its network management system. Integration with operation support system and business support systems also play a bigger role in this. These systems include functions like network configuration, deep packet inspection, policy management, security configuration etc. With Software Defined Networking, the scope of the manageability extends further. Service creation is no more a set of configurations across the network equipments, policy servers and load balancers. Service monitoring is not limited by equipments like deep packet inspectors, load balancers, delivery controllers and network management systems built on top of traditional network management protocols. A service over Software Defined Networking is created at a highly abstracted layer which has power to manipulate the control plane. The ecosystem provided applications will do a lot of service assurance functions. There will be additional integration points like cloud management systems who will dictate the network configuration from an infrastructure service perspective. A holistic approach including all additional integration points and the highly abstracted layer is a challenge for testing.

4. Testing the open-ended innovation platform:

The testing of solutions over Software Defined Networking becomes challenging because it is created in an open platform for vendors to introduce innovative solutions. Wide variety of end host applications, devices and protocols are involved in providing various end user functionalities which are never deployed and tested before. With the advent of newer ways of slicing network, virtualizing network, managing bandwidth in Software Defined Networking the testing strategy need to be redefined to accommodate functionalities. Software applications are developed by the ecosystem of vendors and sometime crowd-sourced. Absence of a strong regimen in this ecosystem may lead to potentially threatening application allowed to change network.

5. Testing the behaviorally changed network:

Reliance on software in defining the network changes the behavior of the network from a closed system to a multi-layered value-added platform. The utility of the created services is of primary importance in the changed network. The orchestration of the services is very different from what traditional testing strategy would include. The changed behavior supports convergence of network and infrastructure components by blurring the line between software-led-infrastructure and Software Defined Networking. Adopting this behavioral change in testing strategy is a significant challenge.

6. **Scalability & Performance Testing:**

Traditional testing includes performance testing by simulating the network load and user traffic patterns. The same will need a different dimension in Software Defined

Networking when the traffic is taken forward by flows. Newer bottlenecks arise due to newer interconnects across network islands of legacy elements and software defined networking capable elements.

The scalability testing approach traditionally is about introducing additional capacity in the network. It involves a manufacturer tested hardware installation. In Software Defined Networking, scaling out is more of a software change to create another boundary of virtualized network. The increased speed and indefinite possibility of scaling out present unique challenges in testing.

7. Security Testing:

Software Defined Networking brings forth vulnerable points in network. It opens up the control plane in secured channel for programmability but the vulnerability would arise in on top of this where applications become potentially powerful. The new set of vulnerability would present a challenge in testing.

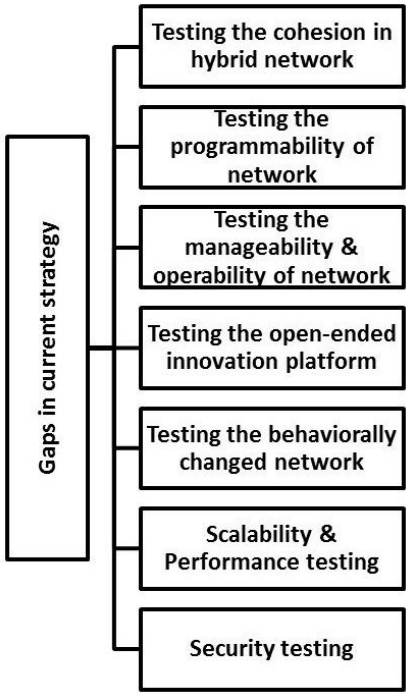


Figure 2: Gaps of Current Testing Strategy

B. Test Tools available for Software Defined Networking testing

Today's Software Defined Networking market has a handful of tools available to test the Software Defined Networking solutions. Here is snapshot of the different tools – commercial and open source.

Testing Controllers

- ✓ CBench focuses on testing OpenFlow Controllers. It generates packet-in events for new flows. It emulates a bunch of switches connecting to controller and creates test scenarios that sends packet-in messages and verifies the flow-mod messages.
- ✓ NICE is a python based tool for testing the NOX OpenFlow controller application.

Testing Switches

- ✓ OFTest is a python based test framework maintained by OpenFlow Foundation. This Test Framework is meant for testing OpenFlow switch implementations.

End-to-End Testing

- ✓ Spirent TestCenter provides a test solution to test the performance of OpenFlow networks. It provides an end-to-end test solution for Service providers, NEMs and enterprises. Customers can use the solution to verify and validate their networks and deploy various services.
- ✓ IXIA IxNAVL provides controller emulation and data-plane testing. It provides a framework for various benchmark tests and conformance tests.

Testbed Simulation

- ✓ Mininet emulates a testbed for developing OpenFlow applications. It creates scalable software-defined networks on a PC by using Linux processes. It supports up to hundreds of nodes, based on the configuration.

Performance Testing

- ✓ perfSONAR is an end-to-end testing tool for monitoring and troubleshooting various aspects of multi-domain network performance. It provides network engineers a way to test and measure network performance.

Test Automation Framework

- ✓ TestON an open-source extensible automation infrastructure.

III. TESTING STRATEGY FOR SOFTWARE DEFINED NETWORKING

The testing strategy for service providers for software Defined Networking needs to be created with a vision of exploiting the tremendous potential Software Defined Networking brings forth. It has to have the objective of reducing operating cost and capital cost together. Invariably, the strategy has to tackle the challenges and create a roadmap for wading through them. The strategy can be built on top of existing testing processes and frameworks but the critical success factors will be the approach for covering the existing gaps.

Service-centric testing approach is one such way which can mitigate the challenges, cover the existing gaps and assure benefits from Software Defined Networking deployments. Service-centric approach is fundamentally different from an equipment focused testing. It is inspired by the Service Oriented Architecture testing in software engineering. In Service-centric approach the use cases of testing is created by the services packaged in a solution offering of service provider. The test cases are driven by the atomic services. An atomic service could be part of the same service but scoped by its presence in different platforms in its lifecycle such as – abstraction layer, OpenFlow enabled switches, across network interconnects, across multiple vendor equipments etc. The perspectives in service-centric testing approach are:

(1) Creator of the Service
(2) Carrier of the Service
(3) Consumer of the Service
(4) Orchestrator of the Service

Creator of service drives the mitigating aspects of programmability challenges and challenges arising due to open ended innovation platform. Carrier of the service takes care of the network cohesion, scalability and performance challenges. Consumer of the service takes control of the requirements like Quality of Experience (QoE). Orchestrator of the service would mitigate the challenges regarding manageability, operability, security.
The testing levels are functional testing, non-functional testing, integration testing, regression testing, deployment testing. These are similar to the testing levels in a traditional network deployment testing. However in service-centric testing approach, there are different scenarios which are to be covered while applying it over Software Defined Networking. The same principle applies when the deployment happens in access (including radio) or core network.

The important aspects in each testing level in service-centric approach are:

(1) The regression and automation test suit will change as it has to incorporate the dynamicity and dependency of the applications on north bound APIs released by vendors. A change in an API will impact a lot of applications providing critical services.

(2) API testing will become a part of each testing level.

(3) Manageability testing will need to include vendor's approach to get the flow-statistics out of the network equipments.

(4) The development and test cycle from vendor and ecosystem providers will be very agile. The service centric approach will have to incorporate very dynamic and agile methods.

(5) In service centric approach, additional attention has to be given to the point that components will come from ecosystem providers working in silos and coming with a very less integration testing already done.

(6) A Horse-shoe model can be used for testing an end-to-end solution where left test leg tests the applications over abstraction layer and the right test leg tests the implication of the application on the network.

Critical Success Factors
(1) Test strategy with a service-centric testing approach and identifying atomic services for test user cases
(2) Clearly identifying the service flow from creator of service to consumer of service
(3) Keeping in mind the challenges which impacts testing over Software Defined Networking

Table 1: Critical Success Factors of Testing Strategy

IV. CONCLUSION

This paper attempts to bring out the key challenges for testing the Software Defined Networking based networks from an end-to-end perspective. The study shows that there will be a paradigm shift in the testing with the widespread adoption of Software Defined Networking. Hence, to successfully test this technology, the change has to happen from the test strategy itself. One has to bring in the service and application view into the test strategy. Again, all aspects of testing like the impact analysis, performance, regression etc will have a new meaning with Software Defined Networking. We believe there is lot of space and scope for innovation in these both from process and tools perspective. This paper provides one step towards the gap identification and possible avenues.

V. ACKNOWLEDGMENT

The authors acknowledge with thanks Mr. Sreekanth Sasidharan, Principal Technology Architect, Infosys Ltd, Bangalore for the guidance and anonymous reviewers for the constructive comments.

REFERENCES

[1] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.

[2] Gude, Natasha, et al. "NOX: towards an operating system for networks." ACM SIGCOMM Computer Communication Review 38.3 (2008): 105-110.

[3] Foster, Nate, et al. "Frenetic: A network programming language." ACM SIGPLAN Notices 46.9 (2011): 279-291.

[4] Canini, Marco, et al. "Automating the testing of OpenFlow applications." Presented at: The 1st International Workshop on Rigorous Protocol Engineering (WRiPE). 2011.

[5] Farias, Fernando NN, et al. "A proposal management of the legacy network environment using openflow control plane." Network Operations and Management Symposium (NOMS), 2012 IEEE. IEEE, 2012.

[6] De Lucia, Andrea, et al., eds. Emerging Methods, Technologies and Process Management in Software Engineering. Wiley. com, 2008.

[7] Canini, Marco, et al. "A NICE way to test OpenFlow applications." NSDI, Apr (2012).

[8] Pfaff, Ben, et al. "Extending Networking into the Virtualization Layer." Hotnets. 2009.

[9] OpenFlow Switch Consortium. "OpenFlow Switch Specification Version 1.0. 0." (2009).

[10] Rob Sherwood, Glen Gibby, Kok-Kiong Yapy, Guido Appenzeller, Martin Casado, Nick McKeowny, Guru Parulkary "Can the Production Network Be the Testbed? "

[11] Sherwood, Rob, et al. "Flowvisor: A network virtualization layer." OpenFlow Switch Consortium, Tech. Rep (2009)

[12] Staessens, Dimitri, et al. "Software defined networking: Meeting carrier grade requirements." Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on. IEEE, 2011

[13] Rothenberg, C. E., Nascimento, M. R., Salvador, M. R., Corrêa, C. N. A., Cunha de Lucena, S., & Raszuk, R. (2012, August). Revisiting routing control platforms with the eyes and muscles of software-defined networking. In Proceedings of the first workshop on Hot topics in software defined networks (pp. 13-18). ACM.

[14] Shin, Seugwon, et al. "FRESCO: Modular composable security services for software-defined networks." Proceedings of Network and Distributed Security Symposium. 2013.

[15] Koldehofe, B., Dürr, F., Tariq, M. A., & Rothermel, K. (2012, December). The power of software-defined networking: line-rate content-based routing using OpenFlow. In Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing (p. 3). ACM.

[16] Matias, J., Jacob, E., Sanchez, D., & Demchenko, Y. (2011, November). An OpenFlow based network virtualization framework for the cloud. In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on (pp. 672-678). IEEE.

[17] Spirent Network Centre. “The Definitive Guide to Switch Testing” (July 2007)

[18] ONF Market Education Committee. "Software-Defined Networking: The New Norm for Networks." ONF White Paper. Palo Alto, US: Open Networking Foundation (2012).

[19] Sherwood, Rob, et al. "Carving research slices out of your production networks with OpenFlow." ACM SIGCOMM Computer Communication Review 40.1 (2010): 129-130.

EmPOWER: A Testbed for Network Function Virtualization Research and Experimentation

Roberto Riggio[†], Tinku Rasheed[†], and Fabrizio Granelli[‡]
[†]CREATE-NET, Trento, Italy; *Email: name.surname@create-net.org*
[‡]University of Trento, Trento, Italy; *Email: granelli@unitn.it*

***Abstract*—Software Defined Networking (SDN) and Network Function Virtualization (NFV) are making their way into the research agenda of all the major players in the networking domain. Parallely, testbeds and experimental facilities are widely regarded as the fundamental step–stone to future "clean slate" networking. However, designing and building experimental facilities can hardly be considered a trivial step for either researchers and practitioners. Scale, flexibility, and ease of use are just some of the challenges faced by a testbed designer. These considerations are at the base of efforts such as GENI in USA, AKARI in Japan, FEDERICA, NOVI and OFELIA in Europe which provide federated and open facilities for the Future Internet research agenda. Albeit the importance of such facilities is unquestioned, today there is still a dearth of testbed exploiting SDN and NFV concepts in the wireless networking domain. In this paper we present *EmPOWER* an experimental testbed which aims at filling this gap by offering an open platform on top of which novel concepts can be tested at scale. The *EmPOWER* testbed is composed by 30 nodes and is currently used by both undergraduate and graduate students at the University of Trento and by the research staff at CREATE-NET.**

***Index Terms*—Software Defined Networking, Network Function Virtualization, WiFi, Testbeds, Open–source**

I. Introduction

SDN and NFV are two of the most promising concepts that are set to bring innovation in the ossified networking landscape. Current SDN efforts start from the consideration that, by providing full visibility of the network from a logically–centralized controller, it is possible to simplify network control and management tasks [1]. Nevertheless, the so called "north–bound" API exposed by today's controllers is still very primitive hindering the development of modular and flexible network applications: As a matter of fact, if with OpenFlow a practical and concrete forwarding abstraction has been found, considerable efforts are still required toward the definition of new programming models. In this regard, several SDN proponents argue in favor of high level declarative languages in order to specify the desired behavior of the network leaving to the underlying Network Operating System, or NOS, its actual implementation. Such vision is summarized by the seminal speech by Scott Shenker: "The Future of Networking, and the Past of Protocols". High–level languages [2], such as: Frenetic [3], Pyretic [4], Procera [1], The Flow Management Language [5], and Nettle [6], aim exactly at providing such level of abstraction.

SDN and NFV are in the research agenda of all the major projects and initiative in the broad Future Internet domain. Examples are GENI in USA, AKARI in Japan, FEDERICA, NOVI and OFELIA in Europe. Several open facilities such as Norbit (NICTA), w-iLab.t (iMinds), NITOS (UTH), Netmode (NTUA), SmartSantander (UC), and FuSeCo (FOKUS) already focus on wireless technologies. Some of these testbeds (Norbit, NITOS, Netmode) focus on research and experimentation on the WiFi domain allowing experimenters to gain full access of a variable number of open WiFi Access Points (APs) where custom software can be installed. Other facilities, such as w-iLab.t and SmartSantander, aim at providing support for experimentation in the IoT domain. Finally, FuSeCo delivers a research facility, integrating OpenIMS and a 3GPP Evolved Packet Core prototype platform. However, albeit the relevance of such facilities is beyond doubt, at the moment there is a dearth of fully virtualized experimental facilities exploiting NFV concepts in the Wireless Networking domain in general and for WiFi-based networks in particular. Moreover, the current OpenFlow ecosystem in term of controller, slicing platforms, and software/hardware switches, provides little support for the WiFi domain.

In this paper we present *EmPOWER* a novel open experimental testbed which aims at filling the gap in the experimental facilities offering for SDN&NFV research and experimentation. The *EmPOWER* testbed is composed by 30 nodes and is currently used by both undergraduate and graduate students at the University of Trento and by the research staff at CREATE-NET. Experiments can take full control of a slice of the network which is kept isolated (at a logical level) from the other slices. Traffic can come from either users that decide to opt–in a certain experiment or by mirroring the traffic of a production. Moreover, the experimenter can monitor in real–time and with the desired resolution the actual energy consumption at either device or slice level using the Energino open energy consumption monitoring and management toolkit [7], [8].

The remainder of this paper is structured as follows. The basic requirements that drove *EmPOWER*'s design are discussed in Sec. II. Section III presents the *EmPOWER* architecture. A particular use case focusing on energy efficiency is presented in Sec.IV. Finally, we draw the conclusions highlighting limitations and future work in Sec. V.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

II. Requirements

Implementing an effective SDN platform supporting advanced virtualization concepts and running on top of commodity WiFi devices raises several challenges. In this section we survey three of such challenges dealing with the conceptual slicing and programming model to be supported, with the collection of the actual state of the network, and finally with actual sharing of the facility resources among experimenters.

A. Slicing and programming model

The platform shall support high–level programming primitives with regard to the control of the network. Such primitive shall be powerful enough to relieve the experimenter from the implementation details specific to the WiFi standard, i.e. association/deassociation mechanisms, control frame exchange and status management. The feature is deemed of capital importance if the policies devised by the experimenters are to be ported to other wireless environments such as LTE–A. Moreover, the platform shall not impose additional requirements on the WiFi clients. This means that, unless the experimenter is planning to deploy custom WiFi clients, the platform shall not mandate for either software and/or hardware modification to the WiFi clients. Finally, the slicing mechanism shall put the experimenter in control of a portion of the network (AP and switches). An opt–in mechanism shall be available for users that want to use a certain slice for their traffic.

B. Querying the status of the network

The platform shall provide the experimenter with a rich set of primitive to query the status of the network. Such primitives shall be as much general as possible in order to support a broad spectrum of use cases. OpenFlow switches already allow collecting statistics related to ports and flows in the network in terms of number and size of the packets. A wireless deployment shall support statistics related to the wireless medium including, for example, RSSI, frame loss ratio, per–MCS (Modulation Coding Scheme) statistics, etc. Given the momentum generated by current research activities on energy efficient networking, the platform shall provide the experimenter with real–time energy consumption information with high temporal precision and small granularity. Such information shall be provided both at the device and at the sliver level, i.e. the platform shall report both the energy consumption of an AP as well as the energy consumption of a specific slice.

C. Federation Architecture

The instantiation of a virtual networks on top of the platform should be performed through either a web–based control framework or an equivalent command line interface which should allow an easy reservation of network resources, either nodes or links. Ideally, a user should be allowed to select the APs and the switches he/she intends to use during the experiment and then the system should instantiate the network in a transparent way to the user. Suitable traffic generation and collection tools shall also be available.

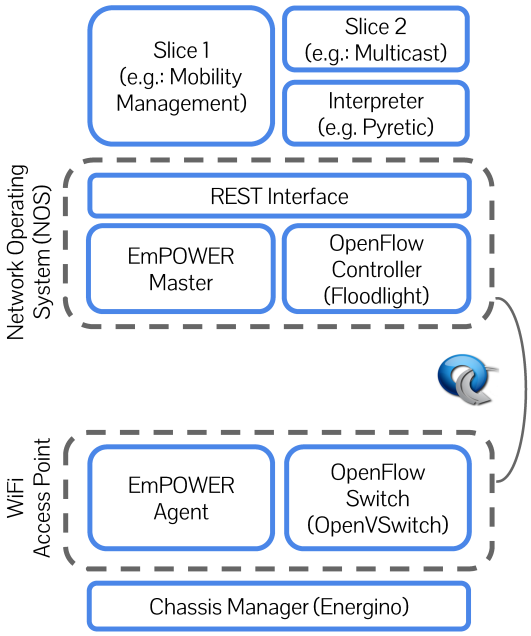


Fig. 1: The *EmPOWER* system architecture.

III. The *EmPOWER* platform

A. System Architecture

The system architecture, sketched in Fig. 1, consists of a single Master and multiple Agents running on each AP. The Master, implemented on top of an OpenFlow controller, has a global view of the network in terms of clients, flows, and infrastructure. The Agents allow multiple clients to be treated as a set of logically isolated clients connected to different ports of a switch. Network application run on top of the controller and can exploit either the embedded Floodlight REST interface or an intermediate interpreter, e.g. Pyretic [4]. Each network application effectively runs in an isolated slice controlling all or just a subset of the available APs.

The *EmPOWER* testbed is built around open and freely available toolkits: OpenVSwitch and the Click Modular Router for the datapath; Floodlight as the controller and Arduino as the power manager. Network applications, i.e. slices can either exploit the Floodlight REST interface or can be built on top of other SDN frameworks. The Pyretic interpreter is also being integrated in the framework in order to allow experimenters to take advantage of this composable programming language.

The *EmPOWER* framework builds on a light virtual AP (LVAP) abstraction [9] which decouples association/authentication from the physical connection between clients and AP. With LVAPs every client that tries to associate to the WLAN receives a unique BSSID, i.e. every client is given the illusion of having a dedicated AP. Similarly, each physical AP hosts an LVAP for each connected client.

Therefore, migrating an LVAP between two physical APs, effectively results in client handover without requiring any re-association and re-authentication. The agent running within each APs is implemented using the Click Modular Router [10].

Finally, Energino[1] is an Arduino add–on, which allows measuring the energy consumption of a device. The measurement circuit is composed of a voltage sensor (based on a voltage divider), and a current sensor (based on the Hall effect). The powering off is done using a mechanical relay. The maximum sampling rate for measurements is about 10.000 samples/s. Voltage and current measurements are periodically sent to the Energy Manager for statistical purposes. Fluctuations in the values read from the analog inputs are filtered out by continuously polling the voltage and the current sensors between update periods and by dispatching the average values. For example, if the sampling period is set to 1s, both the voltage and the current readings will be the average of ≈ 5000 samples. Finally, Energino acts also as chassis manager allowing the testbed administrator to power on/off any node in the network using an HTTP RESTful interface.

While, Energino allows the experimenter to measure the overall power consumption of the AP, it does not provide any info about the actual impact in terms of energy consumption of a slice on the testbed. In order to address this challenge we extended to acts as virtual power meter allowing the experimenter to gain insight into the energy efficiency of his/her network application. In order to do so, a set of power consumption models already developed by the proposers [11], [7] have been embedded into the control framework allowing it to isolate the actual contribution of each slice to the overall consumption of each AP. Energy consumption model take as input measurable network statistics, such as packet transmitted/received over a certain interface, CPU usage, etc. Such statistics are fed to a centralized entity which is in charge of estimating the energy consumption on a per–slice basis. Results are then made available to the experimenter over the controller north–bound interface.

The system exploits a "logically centralized" architecture in order to provide experimenters and network applications developers with a set of powerful programming abstractions to control the behavior of the network. Applications can, for example, register events associated with the actual network conditions and receive updates when such conditions change, e.g. a client moving away from an AP and closer to another. Such primitives can be used to devise and implement novel resource allocation and/or mobility management schemes without having to deal with all the WiFi–dependent implementation details, such as directly handling the IEEE 802.11 state machine or devising workarounds to the limitations of the IEEE 802.11 standard that do not allow the infrastructure to control clients' handovers. Moreover, the availability of a real–time energy monitoring platform will provide experimenters with empirical evidence about the energy consumption performances of their solutions.

[1]Online resources available at: http://www.energino-project.org/

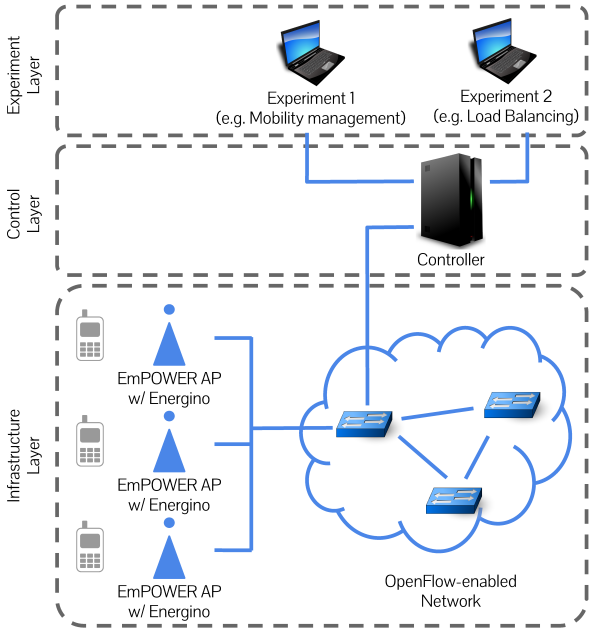


Fig. 2: The *EmPOWER* network architecture..

B. Network Architecture

The *EmPOWER* testbed architecture is sketched in Fig. 2. Each programmable Access Point is equipped with two Ethernet ports. One of them is connected to the control and management network. This allows experimenters collect network statistics and to perform administrative tasks without affecting the actual user traffic that flows trough the second Ethernet interfaces. VLANs are used at the switch in order to keep control and data traffic separated. The *EmPOWER* testbed is currently equipped with the following devices:

- 30 programmable APs based on PCEngines ALIX 2D2 (500MHz x86 CPU, 256MB of RAM) platform and equipped with two Mikrotik R52Hn IEEE 802.11 interfaces (a/b/g/n). The AP exploits OpenWRT 12.09 as operating system. Each AP runs and instance of OpenVSwitch version 1.9 together with an instance of the Click Modular Router.
- 30 Energino power meters. Each Energino is monitoring the power consumption of the AP it is attached to with a sampling period as low as 100 usec and with a resolution of 10mW. Statistics are exported in a format compatible with IoT platforms such as Xively. A REST interface for integration with additional monitoring and management systems is available. Energino acts also as "chassis manager" allowing the testbed manager to power on/off APs remotely.
- 2 Pronto 3295 switches supporting the OpenFlow version 1.0 protocol with 48 Fast Ethernet interfaces.

Each node is equipped with two Ethernet ports. One of them is connected to the control network allowing the controller to collect statistics without affecting the experiment. The second interface is connected to the OpenFlow switch and is used for running the actual experiment's traffic. Finally, another network collects the energy consumption statistics generated by the Energino devices. It is worth noticing that, unlike other WiFi testbeds, *EmPOWER* does not allow the experimenter to upload a custom OS on each AP but rather provides a set of APIs trough which the experimenter can control the behavior of the AP from a centralized controller.

The server runs the latest available software for Floodlight and FlowVisor. It is worth stressing that in the *EmPOWER* architecture new services and algorithms are deployed in the form of Network Applications on top of the Floodlight controller and exploiting its native REST interface. Additional interpreters, such as Pyrethic, are being ported to the platform. Each application is logically isolated from the others and has complete control over its slice, however physical level parameters such as the operating frequency for the hotspot are not. Nevertheless the application can control parameters such as Modulation and Coding Scheme and Transmission Power on a per–frame basis (if required by the experiment).

IV. Energy Programmable WiFi Networks

In this section we shall describe in details a particular use case that make full use of the features made available from the *EmPOWER* testbed starting from the SDN framework for WiFi to the Energino energy monitoring and management toolkit.

A recent report from CEET (Center for Energy Efficient Telecommunications, University of Sydney) stated that by 2015 the wireless access infrastructure will account for 90% of the entire energy footprint of the Wireless Cloud domain, which includes also datacenters and distribution networks [12]. WiFi hotspots are increasingly deployed to relieve cellular networks from the burden generated by data-hungry mobile applications. Such deployments generally cater for the worst case scenario, which leads to a sub-optimal usage of resources when little or no traffic is present. Real improvements in this context can only be delivered with true programmability of network functionalities which in time will allow better resource management, seamless handover between different technologies, and always best connected services. Recently, energy efficiency has also emerged as one of the evaluation metric for new networking solutions.

Hence, it is necessary to gracefully adjust the network to the current demand, improving both energy consumption and traffic pollution. Using the *EmPOWER* testbed, a researcher can test novel energy aware mobility management schemes over a realistic infrastructure. In this regard the *EmPOWER* testbed provides support both in the sensing part allowing the experimenter to subscribe a series of event such as RSSI of a client at one or more APs, energy consumption at device and network level (per–slice). The facility provides the experimenter with set of APIs to both query the actual state of the network and to implements handover policies.

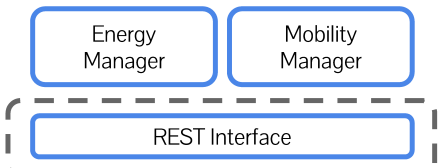


Fig. 3: The *EmPOWER* system architecture particularized for energy programmable WiFi networks use case.

This use case, which has already been demonstrated by the authors in [8], aims at demonstrating that *EmPOWER* can exploited to implement real–time energy consumption monitoring and management solutions aiming at reducing the actual energy consumption of WiFi infrastructures. The argument here is that, in WiFi networks,the extent of energy savings is limited by the actual client distribution (i.e., even if a single client device is attached to an AP, then the AP needs to stay on). This limitation can be traced back to the IEEE 802.11 standard that places all the (re)association initiation to the clients. However, the *EmPOWER* testbed allows the controller to dynamically handover WiFi clients between APs and to selectively shutdown the part of the network that is not deemed necessary.

Figure 3 sketches this use case implementation as two separated network applications running on top of *EmPOWER*. The system exploits a joint mobility and energy management solution. In particular the Energy Manager is responsible of energy management in the network. The decisions that lead to client handovers are handled by the Mobility Manager.

The reference network model for this use case is sketched in Fig. 4. APs are partitioned into clusters with a single Master (represented in blue) and multiple Slaves (represented in either gray or red). Masters are manually chosen at deployment time to provide full coverage and must remain always active. Slaves are deployed for providing additional capacity, and can be selectively turned on/off by the Energy Manager.

In this experiment, APs can support multiple operating modes. Possible events and corresponding transitions between modes are implemented as a finite state machine (FSM) by the Energy Manager. For this use case, we focus on two main operating modes. In the *Online* mode, an AP and all its wireless interfaces are on. In the *Offline* mode, the entire AP is turned off and only the Energino is powered. It is worth noticing that, due to the shared nature of the infrastructure, experimenter are not allowed to actually turn APs on/off. Nevertheless APs can be put in a *virtual* power down mode where no traffic is sent to the slice controller and where the power meter reports a null energy consumption.

We define $W_n \in \mathbb{N}^+$ as the number of clients that must be present in the AP n's cluster so that the AP must remain active. Based on the FSM, a *Slave* AP n belonging to a cluster

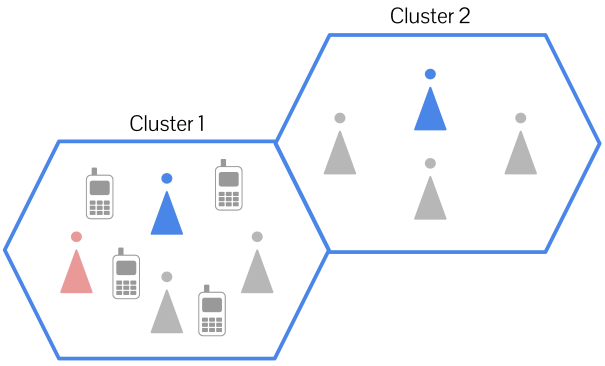


Fig. 4: Reference network model for the Energy Programmable WiFi Network use case. A minimum set of APs (Masters, in blue) providing full coverage must remain always on, while the remaining APs (Slaves, in red or gray) are at disposal of the Energy Manager.

with less than W_n clients and that has been inactive for at least T_{idle} seconds is transitioned to the *Offline* state. Here, inactive means that no LVAPs is hosted by the AP, i.e. no client is connected to the AP. If there are more than W_n clients in the cluster and if the AP has been offline for at least $T_{offline}$ seconds then the AP is brought back to *Online* mode. Notice that, W_n is statically defined for each AP at deployment time and that $W_n = 0$ only for *Master* APs.

This FSM provides a simple example, but it can be extended to support other operating modes according to the APs' capabilities, e.g., single or dual band, support for HT–rates. For example, if an AP has two interfaces, one can be tuned on the 2.4 GHz band and the other tuned on the 5 GHz band. Different operating modes can be created by turning on and off different the interfaces depending on, for instance, the existence of clients supporting the 5GHz band in the cluster.

Clients joining the network are handed over by the Mobility Manager to the AP that provides the best performance in terms of Signal–to–noise ratio (SNR). However, in order to trade–off performance with energy consumption, the Mobility Manager is allowed to handover clients to APs with lower SNR if their W_n is smaller. The rationale is that, by consolidating clients around APs with a small W_n, the Energy Manager will be allowed to turn off APs with bigger W_n. More precisely, if $S(n)$ is the SNR between the client and the AP n, N is the number of clients in the cluster, and $0 \leq \delta \leq 1$ is a tuning parameter specifying how much performance degradation are we willing to accept w.r.t. the best SNR $\hat{S}$, we define the optimal AP $\hat{n}$ as follows:

$$\hat{n} = \underset{n \in \psi}{argmin}(W_n), \quad \psi = \{n \in V | W_n \leq N,\ S(n) \geq \delta \cdot \hat{S}\}$$

where we assumed that V is the set of $|V|$ APs in a cluster (including the *Master* AP. Notice that, since $W_n = 0$ only for *Master* APs, the Mobility Manager will always try to handover clients to a cluster's *Master* AP if its SNR is acceptable.

Notice that albeit simplistic in nature, this use case shows the potential of the *EmPOWER* framework as practical platform for research and experimentation in the Wireless SDN domain. The actual implementation of the described energy management solution consists in $\approx$ 120 lines of java code and could be simplified even further by introducing more sophisticate domain–specific programming languages.

V. Conclusions and Future Work

The paper articulated the design of a novel open testbed aimed at offering an experimental facility for furthering SDN&NFV research and experimentation. Implementing an effective SDN platform supporting advanced virtualization concepts and running on top of commodity WiFi devices is a significant challenge which we address in this paper with the *EmPOWER* experimental testbed. The *EmPOWER* framework builds on top an SDN framework for WiFi networks combining OpenFlow with an open energy consumption monitoring and management toolkit. The facility is currently being extended to include programmable wireless base stations (LTE eNodeBs) to deploy and test heterogeneous scenarios and to experiment with programmable cellular networks.

References

[1] A. Voellmy, H. Kim, and N. Feamster, "Procera: a language for high-level reactive network control," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 43–48.

[2] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger *et al.*, "Languages for software-defined networks," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 128–134, 2013.

[3] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," *ACM SIGPLAN Notices*, vol. 46, no. 9, pp. 279–291, 2011.

[4] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," in *Proc. of USENIX NSDI*, 2013.

[5] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 1–10.

[6] A. Voellmy and P. Hudak, "Nettle: Taking the sting out of programming network routers," in *Practical Aspects of Declarative Languages*. Springer, 2011, pp. 235–249.

[7] K. M. Gomez and Roberto Riggio and Tinku Rasheed and Daniele Miorandi and Fabrizio Granelli, "Energino: An hardware and software solution for energy consumption monitoring," in *WinMee*, 2012.

[8] Roberto Riggio and Cigdem Sengul and Lalith Suresh and Julius Schulz–Zander and Anja Feldmann, "Thor: Energy programmable wifi networks," in *Proc. of IEEE INFOCOM*, 2013.

[9] L. Suresh et al., "Towards programmable enterprise wlans with odin," in *Proc. of ACM HotSDN*, 2012.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[11] Roberto Riggio and Douglas J.Leith, "A measurement-based model of energy consumption in femtocells," in *IEEE Wireless Days*, 2012.

[12] CEET, "The Power of Wireless Cloud: An analysis of the energy consumption of wireless cloud," Centre for Energy-Efficient Telecommunications, Tech. Rep., April 2013.

Edge-to-Edge Virtualization and Orchestration in Heterogeneous Transport Networks

Domenico Siracusa, Elio Salvadori, Tinku Rasheed
CREATE-NET
Via alla cascata 56/D, 38123 Trento, Italy
Email: {domenico.siracusa, elio.salvadori, tinku.rasheed}@create-net.org

***Abstract* — Nowadays, Internet services demand for the mobilization of computing and storage resources located in proximity of the final users, and for the support of an agile and programmable network infrastructure. This calls for a holistic vision in which control, management, and optimization of both computing and networking resources are combined, resulting in a progressive blur of the capabilities of the latter towards the edges of the transport segment. However, due to its multi-provider nature, the Internet itself is the main obstacle to this change. In such a context, Network Virtualization (NV) and Software Defined Networking (SDN) solutions can enable business opportunities and provide technical benefits for the operators managing the different domains of a transport infrastructure. In this paper we discuss in detail the technical challenges and outline potential architectural approaches exploiting the above mentioned solutions to provide edge-to-edge virtualization, abstraction, control, and optimization of heterogeneous transport networks composed of both packet- and circuit-switched technologies.**

***Keywords*— Network Virtualization; Software Defined Networking; business opportunities; heterogeneous transport networks.**

I. INTRODUCTION

Network control is rapidly blurring from the core to the edge of the transport networks, where it is expected to provide a tighter integration with the IT resources. In fact, modern services require for the support of a reliable and effective network infrastructure and needs for an access to processing and storage capabilities close to the end-users. From the networking perspective, the significant role that it is expected to play in satisfying service requirements between the edges of the transport network calls for a vision in which the infrastructure is opened and exposed to upper-layer applications, enabling for a combined management, control, and optimization of computing and networking resources. Unfortunately, such a vision requires fundamental changes in the current Internet transport infrastructure in order to allow *heterogeneous technologies* (i.e., L1 optical circuit-switching devices, L2/L3 packet-switching devices) and *architectures* to co-exist and cooperate to support the applications on top.

In such a context, an emerging trend in telecommunications called *Network Virtualization* [1] is paving the way to a complete reshape of the network infrastructure, which may indeed start from the edges, in order to bring the intelligence closer to the end customer. NV can greatly facilitate the Telco-IT convergence by enabling the commoditization of the infrastructure and overcoming its ossification. In its simplest form, a Network Virtualization Mechanism (NVM) allows a Service Provider (SP) to use multiple heterogeneous architectures called Virtual Networks (VNs) composed by an isolated subset, or a slice, of the nodes and links that are composing the real infrastructure, which is owned and managed by one or more Infrastructure Providers (InPs). The management of the VNs is up to the SPs, which should tackle the issues related to the control of the underlying virtual network resources. The basic building blocks to carry out this task are: path selection algorithms and control plane mechanisms to provision paths and to configure devices and interfaces. In this field the viable solutions are different, but, as the NVMs continue to proliferate, there is a recent push towards *Software Defined Networking* [2] as the way to provide cost-effective, programmable and centralized control of the devices of the VNs.

The introduction of these capabilities can enable the dynamic composition of independent VNs across the edges of the transport networks. Within a VN, programmability and orchestration of the available resources can be ensured by advanced control solutions. This will allow the InPs to provide a widespread offering to their customers by differentiating the quality of service offered by each VN, and the SPs to be able to focus on their core business: service delivery, product marketing, and customer orientation. However, investigation on virtualization and orchestration solutions is still on an initial stage and, due to some practical and technical reasons, they are not mature enough to produce the expected benefits on a transport network scale.

The aim of this paper is to discuss about the technical challenges and the potential solutions that can enable edge-to-edge virtualization and orchestration in transport networks composed by heterogeneous technologies. The rest of the

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

paper is organized as follows: Section II presents the state of the art on virtualization and control solutions; Section III discuss about the business opportunities provided by the introduction of NV in transport networks, while Section IV details some of the technical challenges related to such introduction; Section V presents two approaches to realize edge-to-edge NV and orchestration in heterogeneous transport networks and provides some use-cases and examples; finally, Section VI draws the conclusions.

II. State of the art and technological enablers

First of all, an elucidation on the connection between NV and SDN: for many networking actors overlay network virtualization solution is actually SDN, for some others it is not. This is clearly just a matter of nomenclature, but, to have a defined and consistent reference cataloguing in the rest of the paper we will assume that NV and SDN are different. More specifically, NV is a solution, while SDN is just a mechanism. SDN is like the programming language and NV is a program build with some programming language (that we have previously named NVM). The definition of SDN that we follow is the "original" one, that is, SDN is a network architecture where the control plane is decoupled from the data plane and the hardware from the data plane is generalized to allow more functions.

A. Network Virtualization

Network Virtualization has been often described as the long-term solution to the ossification problem that is affecting the existing Internet [1]. Surveys of a wide range of recent works and projects on NV can be found in [1] and [3]; they are not reported here due to space reasons. NV can be realized by means of Virtual Local Area Networks (VLANs), Virtual Private Networks (VPNs, VPN over MPLS, L1VPN, L3VPN), but also by means of SDN. In fact one of the currently most popular SDN based implementation to instantiate VNs is FlowVisor (FV) [4], which is based on the OpenFlow (OF) protocol [5].

A NV solution for L2 packet switches based on FV is presented in [6]. FV has been mainly introduced to deal with L2 switching technology, but since its potential benefits have also become evident in the context of optical networks, it has been recently extended to deal with the optical lightpaths [7]. The work of [7] tries to address some of the issues related to the extension of NV to the optical domain [8], which are mainly related to physical domain. However, the proposed approaches are not actually able to operate with packet devices. The convergence between network and IT, with the embedding of Virtual Infrastructures over physical infrastructures comprising both optical and IT resources is studied in [9]. The study also focuses on applications and services, while only considering the optical substrate.

Slicing is the main purpose of NV carried out by means of FV. Another key parameter of NV is the abstraction of network resources, which is not natively provided by FV. Authors of [10] address this issue by proposing a virtualization platform able to instantiate two different representations of a VN by enhancing FV. The first one is a VN composed of virtual links and virtual nodes (which may differ from the physical substrate) that can be fully managed by the customer. The second representation is a single node made of different Virtual Links (VLinks), which are composed by a module of the software called virtual topology planner. However, these two proposed extreme options do not cover the plethora of configurations that can be required by a customer; some further options can be investigated. Furthermore, such an approach has been proposed only for L2 switched networks and not for optical networks.

B. Network control and orchestration

NVMs slice and abstract the physical resources, which must be controlled then, by adopting either distributed or centralized mechanisms. In distributed networks, there is no specific node with a prominent role; the nodes should exploit a common protocol to exchange information about the network resources and configuration. For example, in MPLS standard transport networks [11], Provider Edge (PE) nodes exchange reachability information through the BGP protocol and then establish MPLS tunnels to deliver the traffic encapsulated into Virtual Private Networks (VPNs). In optical networks, the classical distributed control layer architecture is based on the suite of GMPLS protocols [12], namely OSPF-TE, RSVP-TE and LMP. These protocols take care of a set of network control functionalities like lightpath establishment, lightpath restoration, etc.

On the other side, in a centralized approach the components of the network are under the control of a single element that possesses all the needed information and may be able to orchestrate the resources in the most appropriate way. To this account, the role of the central intelligence can be demanded to an entity called Path Computation Element (PCE) [13], which has been initially proposed for multi-domain packet transport networks, but has been largely used also in optical networks. The most used centralized implementation based on the SDN paradigm, OF, has been originally designed for packet switching (i.e., Ethernet switches); it has been introduced into the optical networking domain only recently [14], and the attention has mainly been focused on its exploitation in transport networks [15], rather than in access networks. It can be employed in cooperation with a PCE. Some papers compare GMPLS distributed control plane solutions with the SDN implementations in optical and circuit switched networks (e.g., [14][16]); they show that using OF can significantly improve the performance of the distributed control plane and to reduce the lighpath setup time, at the cost of an increased complexity and a decreased responsiveness and reliability. In addition, a centralized control can provide to the applications and to flows coming from the access segment

a specific control on different network layers (e.g., at the MPLS or at the OTN layer) [17].

III. BUSINESS OPPORTUNITIES

Service providers and companies often adopt network virtualization techniques to achieve network isolation, security and traffic anonymity. In this section we briefly present some of the business opportunities that network operators can exploit by adopting NV between the edges of their network, according to different scenarios [18].

A telecom operator may be interested in *focusing on the delivery of services* and on the related marketing and customer orientation operations, promoting those activities as its core business. In this case, the management and control of the devices between the edges of the transport network is left to a third party InP that steps in and benefits from NV to capitalize its investments on the infrastructure and in new network deployments. Both the two parties can benefit from the application of the abstraction, since with a single node representation the whole underlying physical substrate assigned to a VN is exposed to the controller of the customer (SP) as a single router or switch, which is easier to be controlled. On the other side, the InP offer can be differentiated according to the service level required by the customers (e.g., maximum latency or packet loss between the ports of the VN edges).

On the other hand, an operator acting as an InP can *resell the infrastructure* to a customer, offering an enhanced VPN with the possibility of managing the capacity of the deployed routers. In this case, the physical substrate is a real asset for the infrastructure provider, which can be sold according to different pricing strategies from different stakeholders. Thus, in the previous case the customer of the InP benefits from an abstracted vision of the transport network hiding all the complexities related to the physical layer, since it is interested in having a sort of black box transporting the traffic injected at the edges with a certain quality of service. Conversely, in the latter case the customer is interested in having a control of the devices of the assigned slice.

Another business opportunity is provided to multiple operators (two or more) that are interested in deploying *common physical substrate* of a transport network to share the capital expenditure (CAPEX). In such a scenario, NV operating between the edges of such a network is necessary to provide the proper isolation of the different portions of the network.

IV. CURRENT PRACTICAL AND TECHNICAL CHALLENGES

Unfortunately, current virtualization solutions and orchestration mechanisms have not reached yet the flexibility and the maturity envisioned in the previous section: they are still undertaking an evolution aimed at overcoming some practical and technical issues.

The first issue we report is mainly practical (it has been already anticipated in Section II.A): the most used SDN implementation to instantiate VNs, FlowVisor (FV), is not able to provide flexibility in the level of network abstraction that an InP would need when exposing the VN to its customers. As highlighted in [19], most of the studies on network virtualization have focused on the "*network of virtual routers*" model, in which a portion of the available routers is leased together with associated partitioned links. However, this poses the problem that a customer renting such a kind of VN still has to manage and operate a physical network infrastructure and to deal with critical scenarios like network congestion or failure conditions that may eventually not depend on its own behavior. This complexity decreases the appeal of the NV for those SPs that want to focus on the services. In addition, the InP has to face additional complexity since (1) it is very difficult to plan and coordinate maintenance when a third party (i.e., the customer) is managing the allocation of the resources; (2) due to partitioning, it is very difficult to accurately control the resources: the InP has to overprovision the allocated resources, so that the customer will benefit from the agreed service level; (3) the business model behind this service is based on connectivity: the InP is simply offering a commodity service and there is almost no opportunity to differentiate itself and to provide a peculiar offering to its customers. For all these reasons, it is worthwhile to take a wider vision of the NV allowing the infrastructure provider to *expose different levels of network abstraction* for the proposed VNs. This requires an additional mechanism that we call hereafter Network Abstraction Mechanism (NAM) that can eventually run on top of a NVM or it can be an extension of it. The currently available NAM based on FV [10] has been designed and experimented for packet networks; it has not been extended to optical networks, which usually represent the core of the transport networks.

The remaining issues we would like to discuss about are technical and are mainly related to the heterogeneous nature of the transport networks in terms of switching capabilities. Transport networks are usually divided into different domains in which either packet or optical devices are deployed. Thus, in order to instantiate virtual networks between the edges of a transport network it is necessary to investigate how to *compose virtual networks across technological heterogeneous domains*. This actually fits with the presented business scenarios: the equipment of an operator composing a virtual network or sharing the physical substrate with other operators may be heterogeneous. To achieve this result, it is necessary to deal with technical issues that are specific for each technology at the same time: packet and optical domains are different in terms of switching operations, control procedures, protocols, and values associated to key parameters (e.g. power consumption, latency). To the best of our knowledge a NVM able to provide seamless virtualization between the different domains of a transport network has never been presented and

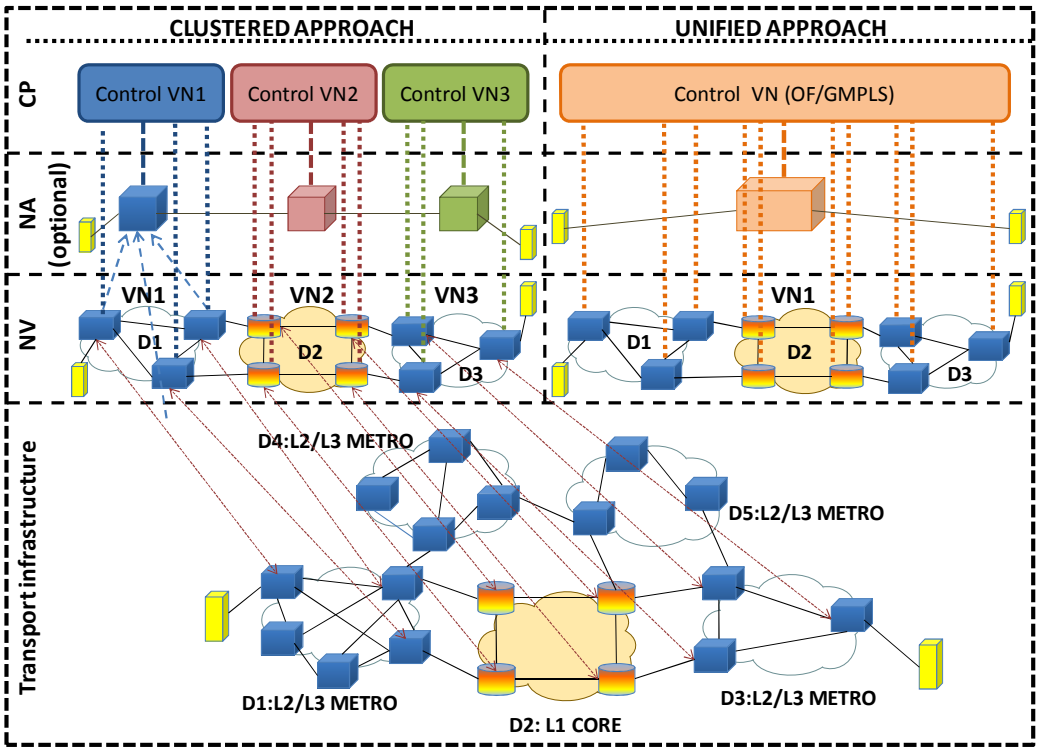


Figure 1. Edge-to-edge network virtualization approaches: clustered and unified.

validated in a real environment. In addition, aside of this latter issue, which is mainly related to the design and implementation efforts, there is a lot of room for research on the VN composition algorithms or on the orchestration mechanisms in heterogeneous domains, due to the complexity of this problem. In fact, while it would be reasonable to think that when mapping the physical substrate to the logical one or when computing a path on a slice, it would be better to avoid optical resources to save energy, it is equally evident that this can increase the latency of the traffic. At the same time, picking the best path on a domain may reveal not to be an effective choice on an end-to-end basis, since it influences the selection of some resources on a domain that may have strict technological constraints. There are many other possible scenarios, but the aspect that evidently matters here is to make effective choices, to perform computations on an edge-to-edge basis, thus to propose *smart dynamic algorithms able to jointly optimize the networking parameters of a heterogeneous network*.

V. Virtualization and Orchestration Approaches

The evolution we are envisioning in this paper aims at *enabling the full potential of Network Virtualization and Orchestration in heterogeneous transport networks on an edge-to-edge basis*. This requires an advanced networking solution supported by an effective architecture to control the devices, which is currently not available. To this account, some relevant progress beyond the state of the art should be pursued on the following topics:

1. Heterogeneous VN composition: extend the current virtualization approaches to deal with heterogeneous networks composed by nodes with different switching capabilities;
2. From Virtualization to Abstraction: enable different levels of virtualization/abstraction on top of edge-to-edge virtual networks;
3. E2E optimization: propose and demonstrate advanced VN embedding solutions or path selection algorithms able to effectively use the physical resources and increase the network performance, while saving energy.

We have identified two main approaches to provide virtualization of the transport network starting from the edge. They are illustrated in Figure 1, in which we consider three layers: the transport infrastructure layer, the NV layer and the Control Plane (CP) layer. The control plane, either OpenFlow-based (OF) or GMPLS can be supported by the PCE to make effective choices in the path selection. In addition, we consider the possibility to provide network abstraction functionalities on top of the sliced resources (in order to clearly distinguish the abstraction from pure virtualization), thus we have added an (optional) Network Abstraction (NA) layer between the NV and the CP layers. If a Network Abstraction Mechanism (NAM) is enabled, the CP controls abstracted representation (dashed line in Figure 1); otherwise, it directly controls the sliced nodes (dotted line in Figure 1). In the first case, the CP computes and establishes edge-to-edge paths on the sliced VNs. In the second case the NAM makes this computation by

performing the mapping between the physical and the logical resources, i.e. by composing the virtual links (VLinks), while the role of the CP is to manage the interfaces of the abstracted node provided by the abstraction mechanism. The physical substrate is composed by different metro network domains with layer-2/layer-3 (L2/L3) electronic switching nodes and a core domain with layer-1 (L1) optical switching nodes; two servers are attached to two peripheral domains. The NV approaches are described in the following.

The first approach is the *clustered approach*: the NVM (e.g., FV and its extensions for the optical technology [7]) of each domain slices the network by selecting a subset of the available resources, thus composing as many VNs as the number of virtualized domains (three in Figure 1). Each slice or VN is assigned to a different CP solution, which can be an OpenFlow controller, or a GMPLS CP. Each controller makes the decisions on its virtual network without any coordination with the other controllers. Consider that there could be a peering relationship between the controllers, but this would imply the definition of the exchanged messages and the definition of a common path computing solution. However, the customers still need to control each network; hence, network abstraction can be enabled. In this case, a NAM can provide a single node representation of each VN to different controllers managed by the SPs by creating some VLinks between the edges of each virtual network, thus hiding the network complexity to the controller.

The second approach is the *unified approach*: the NVM of each domain (or a single NVM managing the whole network) slices the resources and composes a single edge-to-edge VN (VN1 in Figure 1). Such a virtual network is assigned to a single CP mechanism (OF or GMPLS) that manages the sliced resources by making a *global decision*. If network abstraction is enabled, the NAM exposes to OF/GMPLS controller a single node representation of the managed slice. As pointed out in Section II, these operations are not currently feasible with the available NVMs and NAMs; there is the need to design new virtualization and abstraction software able to manage packet and circuit devices at the same time. Eventually, the composition of the edge-to-edge VLinks (i.e., the mapping between physical resources and logical links) can be optimized with respect to some parameters, like power consumption or latency. Hence, the unified approach can blur the virtualization to the edge and enable the application of optimized dynamic embedding algorithms.

Please note that, in addition to the presented solutions, another possible approach is the *layered* one, in which there are different virtualization mechanisms cooperating with a client/server relationship. For instance, there could be an edge-to-edge L2/L3 client NVM that request to the server NVM/NAM of each intermediate L1 domain the composition of a single-node abstracted VN representing the L1 domain, which is then controlled transparently. However, also in this case, it is necessary to define the messages and the interactions between the two NVMs. If this communication is properly defined and implemented, the layered approach should provide similar capabilities to the unified approach with respect to the edge-to-edge computations.

Each approach has strengths and limitations. The clustered one should be able to scale more than the unified, but the edge-to-edge computation for the VLinks/paths is only based on local choices. Thus, the clustered approach should fit better the scenario in which the owners of the physical domains are different InPs; in fact, they may not be inclined to disclose information about their domains and to have a third party NVM that works on their physical substrate, by limiting their possibility to differentiate the offering. Conversely, the unified approach is the most suitable when the infrastructure owner is a single InP. Such approach is very complex, both from an implementation (a single NVM should address at the same time the technological issues of different switching solutions) and algorithmic (a lot of parameters to be managed at the same time) perspective. The increased complexity should decrease the scalability of the unified approach. On the other hand, it allows the operator to perform globally optimized choices when composing the VN or computing end-to-end paths.

A. *Use Cases*

Here we present an example and some use-cases to demonstrate the differences between the two approaches. We assume the network topology shown in Figure 2, with one optical domain (D2) and three packet domains (D1, D3, D4). The VN request is between servers A, B and C, which actually corresponds to three bidirectional VLinks/paths to be embedded/computed (i.e., A-B, A-C, C-B).

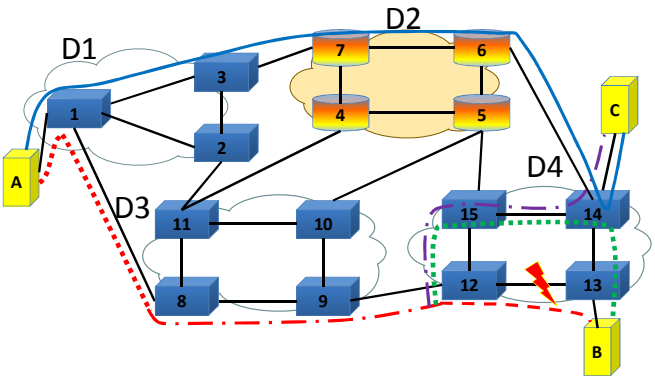


Figure 2. Clustered and unified approaches in a network example.

We initially assume a shortest path metric. First VLink/path is A-B. According to the clustered approach, the different domains are assigned to different NVMs (or NAMs if abstraction is exploited, we refer hereafter only to NVMs for simplicity) or computation elements. The NVM/PCE of D1 computes the shortest path from D1 to the following domain (let us assume it is D3, the path to reach it is shorter than the one to D2), so it chooses the link that goes directly to D3 (red

dotted line). Then, the NVM/PCE of D3 computes the shortest path from D3 to D4, i.e., 8-9 (red dotted-dashed line). Finally the NVM/PCE of D4 computes the shortest path to server B, i.e., 12-13 (red dashed line). The resulting path is composed by 4 hops (links from/to the source/destination are not included). In the unified approach, the unique NVM/PCE calculates in once the edge-to-edge path 1-8-9-12-13, actually the same of the clustered approach. For the second VLink/path, A-C, in the clustered approach the same path from node 1 to node 12 is calculated, then, the NVM/PCE of D4 computes the shortest path from node 12 to 14, which is 12-15-14 (purple dotted-dashed line); so, the resulting path is 1-8-9-12-15-14, composed by 5 hops. The unified approach here performs better, since it computes the edge-to-edge shortest path 1-3-7-6-14 (blue solid line), composed by 4 hops. The third VLink/path is B-C, which clearly provides the same result with both approaches.

This is the observed behavior if the metric relies on the shortest path. Considering a low-latency solution, the unified approach could choose path 1-3-7-6-14-13 for A-B connection, since it traverses more optical devices and links, which should introduce less latency. Conversely, if an energy-efficient metric is applied the unified approach could choose the 1-8-9-12-15-14 path for A-C connection, since it excludes optical devices, which are expected to consume more power. The objective of our future works is to provide optimizations in such heterogeneous networks, e.g. by finding the optimal tradeoff between latency and power consumption. Performance indicators to assess the quality of proposed solutions can be the responsiveness to activate connections and to react to network events, as well as the optimality of the virtual network composition or path computation with respect to different parameters like the amount of available resources, the energy-consumption or the edge-to-edge latency.

Finally, we can also consider the reaction to network events, such as failures or traffic congestion. For instance, let us assume a failure on link 12-13, which affects connection A-B. In the clustered approach, the NVM/PCE of the D4 has to re-compute the path for the domain it is controlling, thus choosing 12-15-14-13 (green dotted line); the resulting final path (1-8-9-12-15-14-13) is composed by 6 hops. The unified approach re-computes the edge-to-edge path, by choosing the path 1-3-7-6-14-13, which is globally shorter since it is composed by 5 hops.

VI. Conclusions

Network Virtualization coupled with the programmability provided by SDN is paving the way for a complete renovation of the Internet infrastructure that, applied between the edges of a transport network, can foster new business opportunities for telecom operators. However, the current technology is not mature enough to support such a drastic change. In this paper we presented some open challenges that are actually preventing the exploitation of effective solutions and mechanisms for the virtualization and orchestration of heterogeneous transport networks composed by packet- and circuit-switching devices. The proposed virtualization approaches basically differs for the capability in controlling the network embedding or the orchestration on a *per-domain* or on an *edge-to-edge* fashion, depending on the network and the market scenarios. We have shown some use-cases and sketched the agenda for the future activity on this topic, which should be basically focused on the implementation and validation of virtualization solutions for heterogeneous networks, supported by advanced embedding or path selection algorithms able to increase the overall network performance.

References

[1] N. Chowdhury, R. Boutaba, "A survey of network virtualization", Computer Networks, vol. 54, no. 5, pp. 862-876, 2010.

[2] N. McKeown, "Software Defined Networking", INFOCOM 2009.

[3] A. Wang, M. Iyer, R. Dutta, I. Baldine, "Network virtualization: technologies, perspectives, and frontiers", IEEE Journal of Lightwave Technology, vol. 31, no. 4, pp. 523-537, 2013.

[4] R. Sherwood et al., "Can the production network be the testbed?", Proc. Of USENIX OSDI, 2010.

[5] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks," ACM SIGCOMM Comp. Comm. Rev., vol. 38, no. 2, pp. 69-74, April 2008.

[6] E. Salvadori, R. Doriguzzi Corin, A. Broglio, M. Gerola, "Generalizing virtual network topologies in OpenFlow-based networks", IEEE Global Telecommunications Conference, GLOBECOM, 2011.

[7] S. Azodolmolky et al., "Optical FlowVisor: An OpenFlow-based Optical Network Virtualization Approach", OFC/NFOEC 2012, JTh2A.41.

[8] J. Elbers, A. Autenrieth, "Extending Network Virtualization into the Optical Domain", OFC/NFOEC 2013, OM3E.3.

[9] G. Landi et al., "A Network Control Plane architecture for on-demand co-provisioning of optical network and IT services", FUNEMS 2012.

[10] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. De Pellegrini, E. Salvadori, "VeRTIGO: network virtualization and beyond", EWSDN 2012.

[11] M. Bocci et al., "A Framework for MPLS in Transport Networks", IETF RFC 5921, 2010.

[12] E. Salvadori et al., "Distributed Optical Control Plane Architectures for Handling Transmission Impairments in Transparent Optical Networks," *IEEE/OSA JLT*, vol. 27, no. 13, 2009.

[13] A. Farrel, J.-P. Vasseur, J. Ash, "A Path Computation Element (PCE)-Based Architecture", IETF RFC 4655, 2006.

[14] M. Channegowda et al., "Experimental Evaluation of Extended OpenFlow Deployment for High-Performance Optical Networks", ECOC 2012.

[15] S. Gringeri, N. Bitar, T.J. Xia, "Extending Software Defined Network Principles to include Optical Transport", IEEE Communication Magazine, vol. 51, n. 3, pp. 32-40, 2013.

[16] S. Das, G. Parulkar, P. Singh, D. Getachew, L. Ong, N. McKeown, "Packet and Circuit Network Convergence with OpenFlow", OFC 2010.

[17] S. Das, G. Parulkar, N. McKeown, "Why OpenFlow/SDN Can Succeed Where GMPLS Failed", ECOC 2012.

[18] J. Carapinha, J. Jiménez, "Network Virtualization – a view from the bottom", 1st ACM workshop on Virtualized Infrastructure Systems and Architectures (VISA), pp. 73-80, 2009.

[19] E. Keller, J. Rexford, "The "Platform as a Service" model for networking", USENIX INM/WREN, 2010.

A Node Plug-in Architecture for Evolving Network Virtualization Nodes

Yasusi Kanada
Central Research Laboratory, Hitachi, Ltd.
Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan
Yasusi.Kanada.yq@hitachi.com

***Abstract* – Virtualization nodes, i.e., physical nodes with network virtualization functions, contain computational and networking components. Virtualization nodes called "VNodes" enabled mutually independent evolution of computational component called programmer and networking component called redirector. However, no methodology for this evolution has been available. Accordingly, a method for evolving programmer and redirector and developing new types of virtualized networking and/or computational functions in two steps is proposed. The first step is to develop a new function without updating the original VNode, which continues services to existing slices, using a proposed plug-in architecture. This architecture defines predefined interfaces called open VNode plug-in interfaces (OVPIs), which connect a data and a control plug-ins to a VNode. The second step is to merge the completed plug-ins into the original programmer or redirector. A prototype implementation of the above plug-in architecture was developed, tested, and evaluated. The prototype extends the redirector by adding new types of virtual links and new types of network accommodation. Estimated throughputs of a VLAN-based network accommodation and a VLAN-based virtual link using network processors are close to a wire rate of 10 Gbps.**

***Keywords* – Network-node plug-in architecture, Data plug-in, Control plug-in, Network virtualization, Virtualization node, VNode, Virtual link, Network processors.**

I. INTRODUCTION

In Japan, several projects targeting new-generation networks (NwGNs) have been conducted [Aoy 09] [AKA 10]. These projects aim to develop new network protocols and architectures (i.e., the "clean slate" approach [Fel 07]) as well as various applications that are difficult to run on IPs but work well on NwGNs. In the Virtualization Node Project (VNP), a virtualization-platform architecture consisting of virtualization nodes (VNodes), namely, a "VNode architecture," was developed by Nakao et al. [Nak 10]. They have also developed a high-performance, fully functional, virtualization testbed in JGN-X, which is a testbed widely used by network researchers. The goal of this VNP is to develop an environment in which multiple slices (virtual networks) with independently and arbitrarily designed and programmed NwGN functions run concurrently, but are logically isolated, on a physical network.

The VNode architecture enabled mutually independent development and evolution of *programmers*, i.e., programmable computational node-components in VNodes, and *redirectors*, i.e., networking node-components in VNodes [Nak 12] [Kan 12a]. In future, a VNode may contain various types of programmers and redirectors. If they are modular, and the interface between them is clearly defined and works efficiently, each vendor can develop software and/or hardware components independently from other components. No method for this VNode evolution, however, has been available.

In the present study, such a method for evolving VNodes, especially for developing advanced redirectors and new types of virtual links, is proposed. By means of this method, a VNode is evolved in two steps. The first step is to develop a new redirector or programmer component as software and/or hardware plug-ins and to install and to connect them to the redirector or the programmer of an existing VNode, without updating the original VNode, through predefined open interfaces. The VNode can continue services to existing slices because they are isolated from slices that use the plug-ins. Combinations of data and control plug-ins are used. The second step is to merge the plug-ins into the redirector or the programmer and, thereby, to create an evolved VNode. Plug-in interfaces and prototype plug-ins were implemented, and the evolved VNode can be used to create new types of virtual links and new methods of network accommodation.

The rest of this paper is organized as follows. Section II describes the method for virtualizing a network on the evolvable architecture platform and the independently evolvable VNode architecture. Section III describes the two proposed evolution steps, and Section IV describes the plug-in architecture for the first step of the evolution including the open VNode plug-in interface (OVPI) and control and data plug-ins. Section V first describes a prototype plug-ins that implements this architecture and then presents the results of an evaluation of this architecture. Sections VI describes related work, and Section VII gives some concluding remarks.

II. METHOD FOR NETWORK-VIRTUALIZATION

Network virtualization, the structure of a virtualization platform (i.e., a physical network), and the structure of the virtual network are described as follows.

A. Network virtualization

When many users and systems share a limited amount of resources on computers or networks, virtualization technology creates the illusion that each user or system owns resources of their own. Concerning networks, wide-area networks (WANs) are virtualized by using virtual private networks (VPNs). When VPNs are used, a physical network can be shared by multiple organizations, and these organizations can securely and conveniently use VPNs in the same way as virtual leased lines. Nowadays,

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

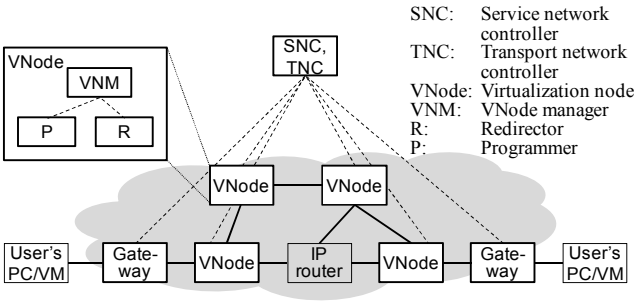


Figure 1. Physical structure of virtualization platform

networks in data centers are virtualized as VLANs, while servers are virtualized as virtual machines (VMs).

Many programmable virtualization-network research projects have been carried out, and many models, including PlanetLab [Tur 07], VINI [Bav 06], GENI [Due 12], and Genesis [Kou 01], have been proposed. Slices are created by network virtualization using a *virtualization platform* (substrate) that operates the slices.

In the VNP, network-virtualization technology was developed by Nakao et al. [Nak 10][Nak 12]. This technology makes it possible to build programmable virtual-network environments in which slices are isolated logically, securely, and in terms of performance (QoS) [Kan 13a] In these environments, new-generation network protocols can be developed on a slice without disrupting other slices.

B. Structure of virtualization platform

In the VNP, a physical network is assumed to consist of one or more domains, which are managed by a service network controller (SNC) and a transport network controller (TNC). SNC was formerly called a domain controller (DC) [Nak 12][Kan 12a]. Each domain has two types of nodes: VNode and gateway (**Figure 1**).

An overlay technology is used in the current version of VNode platform; that is, a VNode forwards packets on the platform, and each packet on the platform contains the contents of a virtual packet in a slice as the payload. VNodes are connected by tunnels using a protocol such as Generic Routing Encapsulation (GRE) [Far 00], and the Internet Protocol (IP) is used in the current version of the virtualization platform. A domain may contain conventional routers or switches that do not have virtualization functions. A slice is therefore neither constrained by the topology of the physical network nor by the specific functions of these nodes. A VNode can operate as a router or a switch for platform packets, so it can be deployed in conventional networks. VNodes can thus be distributed to any place connected by the IP. An arbitrary packet format and protocol can be used in a slice, so they can be used in a VNode anywhere.

A VNode consists of three components: a programmer, a redirector, and a VNode manager. A programmer processes packets on slices. Slice developers can inject programs into programmers. A redirector forwards (redirects) packets from another VNode to a programmer or from a programmer to another VNode. A *VNode manager* (*VNM*) (a software component) manages the VNode according to instructions from the SNC.

C. Structure of slices

In the virtual-network model developed by the VNP, a virtual network is called a *slice*, which consists of the following two components (**Figure 2**) [Nak 10][Nak 12].

- *Node sliver* (virtual-node resource) represents computational resources that exist in a VNode (in a programmer). It is used for node control or protocol processing of arbitrary-format packets. A node sliver is generated by slicing physical computational resources.
- *Link sliver* (virtual-link resource) represents networking resources such as a virtual link that connects two node slivers and that any IP and non-IP protocols can be used on. A link sliver is mapped on a physical link between two VNodes or a VNode and a gateway. A link sliver is generated by slicing physical-network resources such as bandwidth.

Both node slivers and both link slivers are isolated and work concurrently, so two slices that consist of these slivers are also isolated and work concurrently.

The SNC of a domain receives an abstract slice design by using an XML-based *slice definition*. The SNC distributes the slice definition to each VNM, which sends the necessary definitions to the programmer and the redirector: the programmer receives information required for configuring a node-sliver, and the redirector receives the information required for configuring link slivers. For example, a slice definition may contain an abstract link-sliver specification such as the following link sliver with two virtual ports, i.e., end points: vport0 and vport1.

```
<linkSliver type="link" name="virtual-link-1">
 <vports>
  <vport name="vport0" />
  <vport name="vport1" />
 </vports>
</linkSliver>
```

D. Independent evolution of programmers and redirectors

An aim of the VNP is to enable mutually independent development and evolution of programmers and redirectors. Programmers consist of programmable hardware and software and implements node slivers. Redirectors consist of flexible (and maybe programmable) hardware and software and implement link slivers. It is necessary to establish modularity of these components to enable their independence; in other words, the interfaces between the components (both data-plane and control-plane interfaces) must be clearly defined.

In future, virtualization platforms will probably consist of computational and networking hardware and software developed by various vendors. A VNode may contain various types of computational components, such as Linux VMs, Microsoft Windows VMs, network processors, and GPGPUs. A network composed of VNodes may consist of various types of networking components, such as VLAN, WDM, and light paths.

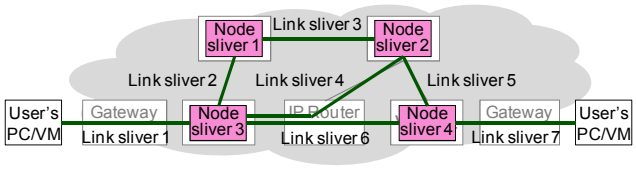
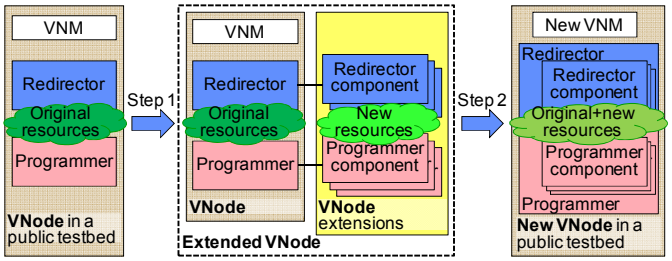


Figure 2. Example of slice design



(a) Original VNode (b) Evolving VNode (c) Evolved VNode

Figure 3. Proposed VNode evolution steps

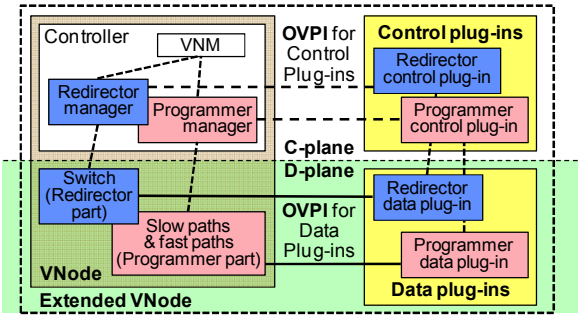


Figure 4. Open VNode plug-in (OVPI) architecture

If the interface between the software/hardware components and subcomponents is clearly defined and works efficiently, a component can be evolved independently from other components, and the components will be modular. That means they can be freely chosen and used in combination and can be freely enhanced or replaced by other components in accordance with emergence of new technology. No method for this evolution, however, has been available.

III. Proposed Evolution Steps

The proposed method for evolving VNodes is explained as follows. This method is applicable to not only VNodes but also other types of nodes for software-defined networks (SDNs). However, the architecture described in the previous section is assumed for simplicity.

As for the proposed method, redirector/programmer plug-ins are used for developing new functions, such as creating or deleting new types of virtual node or link, in two steps (see **Figure 3**). The first step is to develop new subcomponents of redirector or programmer as plug-ins and to install and to connect them to the redirector or the programmer of an existing VNode. The second step is to merge the plug-ins into the redirector or the programmer and to create an evolved VNode.

An "evolvable" VNode is created in the first step; that is, in this step, the plug-ins can be updated at any time without affecting the operation of the original VNode. Not only the redirector and the programmer in the original VNode but also the SNC and TNC (i.e., network managers) and the VNM (i.e., the management part of the VNode) remain unchanged. They manage the resources and the configuration of the original virtualization platform, but they do not manage the resources and the configuration of plug-ins.

The plug-ins can be tested by using newly created slices that specify the VNode and the plug-in. The VNode can continue services to existing slices while the plug-ins are developed because the existing slices do not use the plug-ins and are isolated from the testing slices. If the VNode has isolation function that separates packets generated by the plug-ins from packets for existing slices, the plug-ins can be tested without significant interferences with existing slices.

The resources and the configuration of the plug-ins must be managed by the plug-ins themselves. Because the resource managers are separated and the original managers do not know the new resources, the original resources and the new resources must be completely separated. The new resources may be new types of virtual node with a new type of network processor, new types of virtual link, or new types of physical subnode or link. If information on the plug-ins must be exchanged between two or more VNodes through the management components, the information should be passed through the components without interpreting or testing it. This tunneling mechanism can be called *control information tunneling* (*CIT*).

If redirectors, programmers, and the management components, i.e., SNC, TNC, and VNM, are designed to exclude interference between them and newly introduced plug-ins, a publicly available platform can be used for the development of new functions. It was intended to apply this method to JGN-X, i.e., a testbed that contains VNodes. The original VNode is probably placed in a place, such as a carrier's building, that is not easily accessible for temporary experimental purposes. However, the plug-ins can be placed in private environments for experiments, such as university laboratories or offices of vendors, and are connected by a layer-2 network such as a VLAN or a layer-2 tunnel over IP networks.

In the second step, an evolved VNode is created; that is, the plug-in functions developed in the first step are introduced into the core part of the platform. The programmer data-plug-in functions are merged into the programmer, the redirector data-plug-in functions are merged into the redirector, and the functions of control plug-ins developed in the fist step are introduced into the management components including programmer manager, redirector manager, SNC, TNC, and VNM. Because the resource managers are merged into the core part, the original and new resources are also merged. As a result, they can select the best method and resource from various methods and resources that were originally implemented in the VNode and added to it for fulfilling slice developers' requests.

IV. Open VNode Plug-in Architecture

A. Outline

The plug-in architecture described in this section is used in the first step of the VNode evolution. Plug-ins are installed and connected to a VNode using a predefined interface called an *open VNode plug-in interface* (*OVPI*), which should be built into both the programmer and the redirector of the VNode (see **Figure 4**).

There are two types of OVPI: a data-plane (D-plane) interface and a control-plane (C-plane) interface. The D-plane interface connects *data plug-ins* that handle data packets to slow-path or fast-path components (software

and/or hardware components) in the case of programmer extension or to a switch (which is a part of the redirector) in the case of redirector extension. The C-plane interface connects *control plug-ins* that manage the data plug-ins and the programmer manager or the redirector manager. The data and control plug-ins are therefore used in combination. The management interface between a control plug-in and a data plug-in is a private interface, which has no predefined specification.

Plug-ins may be placed at a distant place from the VNode. A VNode may exist in a publicly available network, and the plug-ins may exist in a private environment such as a university laboratory.

Many implementation methods can be used for the OVPI. For control plug-ins, command-line interfaces (CLIs) and APIs (such as remote procedure calls or XMLs) can be used. Because an OVPI is an interface connected through networks, the host name or the (IP) address is required to identify the host node of the plug-in in the case of a control plug-in. In the case of data plug-ins, packet headers such as VLAN header or GRE can be used. Interface parameters can be passed through procedure arguments, XML tags, VLAN identifiers, GRE keys, and so on.

B. C-plane plug-in interface

The following identifiers and parameters must be specified in a control message of an OVPI for a control plug-in.

1. *Host name or address* specifies the host that contains the plug-in. In usual cases, a domain name or an IP address is used, but a non-IP address or another type of name may also be used.
2. *Plug-in identifier* specifies a plug-in in the host. This identifier may be structured; namely, plug-ins may be hierarchical.
3. *Parameters* specify control information including information that identifies the slice that the information represents. Plug-in parameters may be named or positional; that is, each parameter may have an identifier and a value or parameter values may be specified in a specific order without identifiers.

Two examples of C-plane interfaces are described here. A CLI is used in the first example. In this example, the host name or address is specified as the **`ssh`**/**`telnet`** server's domain name or address, a command name (or a file name) can be used for the plug-in identifier, and command arguments can be used for specifying parameters. For example, the following command with named parameters may be used for creating a virtual-link plug-in (see section V-C):

add_link vlan=*id* esmac=*p1* edmac=*p2* ismac=*p3*.

This command specifies a set of control information for a data plug-in; namely, it specifies an addition of a virtual link (i.e., link sliver) between external virtual ports specified as *p1* and *p2* by the specified VLAN identifier (*id*). Virtual port *p3* specifies the internal port of the node. In this example, all the parameters are named and can be specified in an arbitrary order.

The second example is as follows. The same contents are specified by an XML-based interface, such as XML-RPC [XML] or SOAP [Mit 03]. In this case, the plug-in identifier and the parameters are passed to the host as XML elements and attributes.

In general, identifiers and parameters in an OVPI must be supplied by the slice definition or the VNode (i.e., redirector, programmer, or VNM). An example of a link-sliver specification, which is similar to the link sliver shown in section II-D, is shown below. This definition contains the domain names or addresses of the control plug-ins and the physical data ports of data plug-ins. The VLAN identifier and MAC addresses are not included in this definition because they are generated by the VNodes and the control plug-ins.

```
<linkSliver type="link" name="virtual-link-1">
 <vports>
  <vport name="vport0"
   <params>
    <param key="controller" value="plug-in-0-addr" />
    <param key="port" value="data-plug-in-0-port"/>
    <!-- Additional parameters -->
   </params>
  </vport>
  <vport name="vport1"
   <params>
    <param key="controller" value="plug-in-1-addr" />
    <param key="port" value="data-plug-in-1-port"/>
    <!-- Additional parameters -->
   </params>
  </vport>
 </vports>
 <params>
  <param key="ExtensionName" value="vlan_link" />
  <!-- Additional parameters -->
 </params>
</linkSliver>
```

C. D-plane plug-in interface

The following parameters must be specified in a data packet as an OVPI for a data plug-in.

1. *Plug-in channel tag*: In contrast to the C-plane interface, a host and a plug-in are not specified separately. A tag, which may be a protocol parameter such as a VLAN identifier, specifies a channel or a collection of plug-ins. Multiple plug-ins specified by a tag may be in one host or distributed to multiple hosts connected by a network channel (such as a VLAN).
2. *Parameters*: Plug-in parameters are specified as protocol parameters. Some parameters identify the slice of the data path that the plug-in implements. Some parameters may be used for identifying a plug-in among the plug-ins specified by the plug-in channel tag.

Two examples of D-plane interfaces are described here. In the first example, a VLAN is used for the D-plane protocol. In this case, the plug-in channel tag may be specified as a VLAN identifier. The parameters, which represent the end-point addresses of the virtual link, are expressed as source and destination MAC addresses. If only one (or a few) VLAN identifier can be used or if no tagged VLANs can be used, plug-ins may be distinguished by a set of MAC addresses; in other words, they can be used for specifying both a plug-in tag and parameters.

In the second example, GRE/IP is used for the D-plane protocol. In this case, the plug-in tag is represented by a key in the GRE header, and the parameters are represented by addresses in the IP header.

V. PROTOTYPING AND EVALUATION

A version of the OVPIs was implemented, and two sets of plug-ins were installed and connected by using the OVPIs and partially evaluated. The hardware and software for the OVPIs and the plug-ins are described below first; the design, implementation, and preliminary results of an evaluation of the plug-ins are described after that.

A. Hardware and software environment for plug-ins

The prototype system and the environment used for prototyping and evaluation are described as follows. A preliminary version of the OVPIs is implemented in the redirectors of the VNodes. A CLI is used for the C-plane interface, and a VLAN-based interface is used for the D-plane. Data plug-ins are implemented in two sets of PCs with CentOS (Linux). Each PC has a PCIe board with a network processor, Cavium Octeon [Cav 10]. This board is called WANic-56512 (developed by General Electric Company). An open and high-level language called CSP (Continuous Stream Programming) and its development environment, "+Net," for Octeon [Kan 13b] is used for developing the plug-in programs. Control plug-ins are implemented in the PCs.

Two sets of plug-ins were developed. The first set of plug-ins, a control plug-in and a data plug-in, implements a network-accommodation function that connects a slice to an external network through a VLAN, and the second set of plug-ins implements VLAN-based virtual links (link slivers) between VNodes.

B. Re-implementing network-accommodation function

The first set of plug-ins implements a network accommodation function that connects a slice of the VNode platform to an external network through a VLAN (see **Figure 5**). This function is similar to that of "network accommodation equipment" (NACE or NC) [Kan 12b], which is built into the VNode platform. However, this function is re-implemented to test the plug-in architecture and the network-accommodation-function implementation.

The data plug-in converts the packet format; that is, the packet format for the external network is X/Ethernet, where X is usually IP but other protocols can also be used, and the internal format for a VNode is X/Ethernet/Ethernet. The outer MAC header contains the platform parameters, and X/Ethernet (including the inner MAC header) is the packet format for the slice.

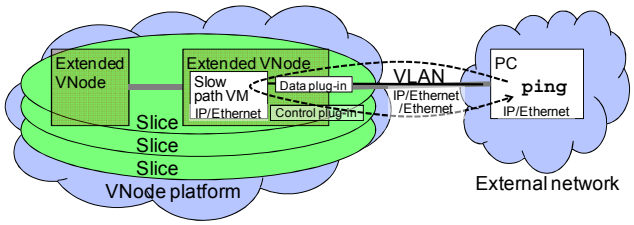


Figure 5. Re-implementation of network-accommodation function

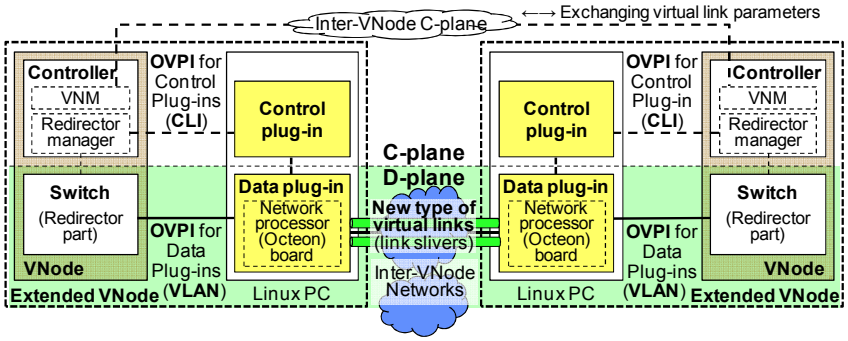


Figure 6. VNode plug-in and interaction architecture for extension of a virtual link

As shown in Figure 5, successful IP communication between a PC in the external network and a VM in the virtual node was confirmed by a **`ping`** command. The performance of the whole prototype system, which contains two VNodes and plug-ins, has not yet been measured. However, the performance of the data plug-in implemented on the Octeon board was measured. When packet size was sufficiently large, i.e., 600 bytes or larger, the throughput was measured to be 8 Gbps or more, namely, close to the wire rate, i.e., 10 Gbps.

C. Implementing a new type of virtual link

The second set of plug-ins implements a new type of virtual link. GRE-based virtual links are the only type available in the current version of VNodes. VLAN-based virtual links are thus implemented by using the plug-ins. The architecture for the VLAN-based virtual link is shown in **Figure 6**, and the packet formats and example contents are shown in **Figure 7**. To separate a programmer from the network and other programmers, internal MAC addresses of the programmer, which are part of the data plug-in interface, must be hidden outside of the programmer [Kan 12a]. The redirector data plug-in therefore swaps the MAC addresses in data packets as shown in Figure 7.

To operate a virtual link correctly, control plug-ins in two VNodes, which are the end-points of the virtual link, must exchange control parameters through the inter-VNode C-plane (see the top of Figure 6). The end-point addresses in the control parameters identify the slice to which the virtual link belongs. This negotiation should be performed by the VNode managers (VNMs) of the VNodes when a virtual link is created or deleted. However, currently they only have negotiation function of

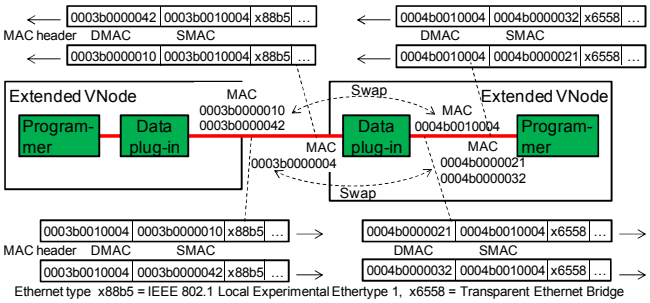


Figure 7. Packet formats for the VLAN-based virtual link

GRE-based virtual links. Therefore, in this preliminary and temporary implementation, the GRE-based link parameters, i.e., IP addresses (and a GRE key), are passed to the control plug-ins and they are converted to the VLAN-based link parameters, i.e., MAC addresses (and a VLAN ID). In a future version of redirector plug-in architecture, VNMs should implement a tunneling mechanism, i.e., CIT. The VNMs can exchange VLAN-based link parameters or any other type of control information that control plug-ins manage using CIT.

Successful IP communication between the virtual nodes connected by the VLAN virtual-link was confirmed by a **`ping`** command; although virtual links in VNodes can transmit arbitrary format packets such as IPEC packets [Kan 12c], IP was used because it requires only two commands (i.e., **`ifconfig`** and **`ping`**) built into the virtual node. The performance of the whole prototype system was not measured, but the throughput of the data plug-in was measured to be 9 Gbps or more when the packet size was 900 bytes or larger.

VI. Related Work

Click [Koh 00] is a software architecture that uses two-level description for describing routers modularly. The lower-level components, which are described in C, can be regarded as plug-ins. The higher level is described in a domain-specific language, which connects modules in several ways. Both data and control plug-ins may be described by using Click; however, Click is suited to (data) packet processing but not well suited to control processing and hardware plug-ins.

OpenFlow [McK 08] enables easy implementation and extension of network control. It is easy to use OpenFlow to design a plug-in architecture for management and control. It cannot, however, be used to implement data plug-ins.

Active networks enabled ad hoc extension of data paths. Capsules, or active packets [Wet 98], which are packets containing programs, may be regarded as a temporary plug-ins. There are, however, two issues concerning capsules. First, capsules are not suited to repeatedly used functions because of redundancy; that is, multiple packets contain the same program. Second, they cannot be used for hardware plug-ins. Other types of active networks, such as SwitchWare [Ale 98], solve the first issue but not the second one.

In contrast to OpenFlow and active networks described above, the plug-in architecture proposed in this paper can be used for both control and data plug-ins, and for both software and hardware plug-ins.

VII. Conclusion

A method for evolving programmer and redirector, i.e., computational and networking components of a VNode, independently was proposed and tested. This method is composed of two steps. In the first step, plug-in interfaces called "open VNode plug-in interfaces" (OVPIs) for both data and control plug-ins are used. These OVPIs are built in both the programmer and the redirector of VNodes.

A prototype of OVPIs and plug-ins were developed and evaluated. The evolved VNode can implement new types of network accommodation functions and can create new types of virtual links. The throughput of the network accommodation and the VLAN-based virtual links is close to a wire rate of 10 Gbps. This result means that the first step of VNode evolution was succeeded for these new functions.

Future work includes implementing CIT to the VNM and implementing new types of virtual links and network accommodation methods, including non-IP-protocol based ones, using advanced technologies and methods. It also includes applying this method, including the second step, to VNodes in JGN-X.

Acknowledgments

Part of the research results is an outcome of "Advanced Network Virtualization Platform Project A" funded by the National Institute of Information and Communications Technology (NICT). The author thanks Kazuhisa Yamada from NTT, Akihiro Nakao from the University of Tokyo, Toshiaki Tarui from Hitachi, and other members of the above project for their valuable discussions on the VNode evolution process. The author also thanks Yasushi Kasugai, Kei Shiraishi, Takanori Ariyoshi, and Takeshi Ishikura from Hitachi for implementing the plug-in interfaces in the redirector.

References

[AKA 10] AKARI Architecture Design Project, "New Generation Network Architecture — AKARI Conceptual Design (ver 2.0)", May 2010.

[Ale 98] Alexander, D. S., Arbaugh, W. A., Hicks, M. W., Kakkar, P., Keromytis, A. D., Moore, J. T., Gunter, C. A., Nettles, S. M., and Smith, J. M., "The SwitchWare Active Network Architecture", *IEEE Network*, Vol. 12, No. 3, pp. 29–36.

[Aoy 09] Aoyama, T., "A New Generation Network: Beyond the Internet and NGN", *IEEE Communication Magazine*, Vol. 47, Vol. 5, pp. 82–87, May 2009.

[Bav 06] Bavier, A., Feamster, N., Huang, M., Peterson, L., and Rexford, J., "In VINI Veritas: Realistic and Controlled Network Experimentation", *SIGCOMM 2006*, pp. 3–14, September 2006.

[Cav 10] "OCTEON Programmer's Guide, The Fundamentals", Cavium Networks, 2010, http://university.caviumnetworks.com/downloads/-Mini_version_of_Prog_Guide_EDU_July_2010.pdf

[Due 12] Duerig, J., Ricci, R., Stoller, L., Strum, M., Wong, G., Carpenter, C., Fei, Z., Griffioen, J., Nasir, H., Reed, J., and Wu, X., "Getting Started with GENI: A User Tutorial", *ACM SIGCOMM Computer Communication Review*, Vol. 42, No. 1., pp. 72–77, January 2012.

[Far 00] Farinacci, D., Li, T., Hanks, S., Meyer, D., and Traina, P., "Generic Routing Encapsulation (GRE)", RFC 2784, IETF, March 2000.

[Fel 07] Feldmann, A., "Internet Clean-Slate Design: What and Why?", *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 3, pp. 59–74, July 2007.

[Kan 12a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *IEEE International Conference on Communication Systems* (*ICCS 2012*), November 2012.

[Kan 12b] Kanada, Y., Shiraishi, K., and Nakao, A., "High-performance Network Accommodation into Slices and In-slice Switching Using A Type of Virtualization Node", *2nd*

International Conference on Advanced Communications and Computation (*Infocomp 2012*), IARIA, October 2012.

[Kan 12c] Kanada, Y. and Nakao, A., "Development of A Scalable Non-IP/Non-Ethernet Protocol With Learning-based Forwarding Method", *World Telecommunication Congress 2012* (*WTC 2012*), March 2012.

[Kan 13a] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-resource Isolation for Virtualization Nodes", *IEICE Trans. Commun.*, Vol. E96-B, No. 1, pp. 20-30, 2013.

[Kan 13b] Kanada, Y., "Open, High-level, and Portable Programming Environment for Network Processors", *IEICE 7th Meeting of Network Virtualization SIG*, July 2013 (in Japanese).

[Koh 00] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Frans Kaashoek, M., "The Click Modular Router", *ACM Transactions on Computer Systems* (*TOCS*), Vol. 18, No. 3, pp. 263–297, 2000.

[Kou 01] Kounavis, M., Campbell, A., Chou, S., Modoux, F., Vicente, J., and Zhuang, H., "The Genesis Kernel: A Programming System for Spawning Network Architectures", *IEEE J. on Selected Areas in Commun.*, vol. 19, no. 3, pp. 511–526, 2001.

[McK 08] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J., "OpenFlow: Enabling Innovation in Campus Networks", *ACM SIGCOMM Computer Communication Review*, pp. 69–74, Vol. 38, No. 2, April 2008.

[Mit 03] Mitra, N., and Lafon, Y., "SOAP version 1.2 part 0: Primer", W3C Recommendation 24 (2003): 12.

[Nak 10] Nakao, A., "Virtual Node Project ― Virtualization Technology for Building New-Generation Networks", *NICT News*, No. 393, pp. 1–6, Jun 2010.

[Nak 12] Nakao, A., "VNode: A Deeply Programmable Network Testbed Through Network Virtualization", *3rd IEICE Technical Committee on Network Virtualization*, March 2012, http://www.ieice.org/~nv/05-nv20120302-nakao.pdf

[Tur 07] Turner, J., Crowley, P., Dehart, J., Freestone, A., Heller, B., Kuhms, F., Kumar, S., Lockwood, J., Lu, J.,Wilson, M., Wiseman, C., and Zar, D., "Supercharging PlanetLab ― High Performance, Multi-Application, Overlay Network Platform", *ACM SIGCOMM Computer Communication Review*, Vol. 37, No. 4, pp. 85–96, October 2007.

[Wet 98] Wetherall, D., et al. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *1st IEEE Conference on Open Architectures and Network Programming* (*OPENARCH'98*), pp. 117–129, April 1998.

[XML] XML-RPC Home Page, http://www.xmlrpc.com/

Network Virtualisation Trends: Virtually Anything is Possible by Connecting the Unconnected

Eleni Patouni
Department of Informatics & Telecommunications
University of Athens,
Athens, Greece
elenip@di.uoa.gr

Andreas Merentitis
AGT International
Darmstadt, Germany
amerentitis@agtinternational.com

Panagiotis Panagiotopoulos, Aristotelis Glentis, Nancy Alonistioti
Department of Informatics & Telecommunications
University of Athens,
Athens, Greece
{ppanagiotis, arisg, nancy}@di.uoa.gr

Abstract— The large and increasing variety of proprietary hardware appliances in existing networks poses great difficulties in both their management and the launch of new network services. This is related to a plethora of requirements at the core network part such as energy costs, capital investment challenges and skills necessary to design, integrate and operate increasingly complex hardware-based appliances. At the end user side, the evolution of the Internet of Things envisions to increase the number of connections by yet another order of magnitude (from ~10 billion currently connected "Things"), bringing unprecedented challenges in network scalability, resource efficiency, and privacy considerations. This paper tackles the previous challenges under the prism of network function virtualization focusing on the following use cases: Software Defined Networking controlled wireless integration service, virtual cell management, and sensor networks virtualization. We discuss how the different facets of Virtualization promise to alleviate or resolve some of these challenges, acting as a catalyst for the realization of disruptive networking paradigms, both in the core network and in the end-user side.

Keywords—Future Networks, sensor virtualization, virtual cells, SDN, NFV

I. INTRODUCTION

Future networks are becoming highly dynamic and distributed, typically consisting of various multi-operator heterogeneous networks and a large number of network elements and users' devices. At the network side, the high level of interconnections through middle boxes, e.g. Network Address Translation (NAT), performance-enhancing-proxies, and other application-specific gateways exacerbates the Internet network ossification problem. At the end-user side, the notions of Machine to Machine Communications (M2M) and the Internet of Things, are expected to increase the number of connections by yet another order of magnitude (from ~10 billion currently connected "Things"). In this hyper-connected world that is full of data [1], it is apparent that in order to 'connect the unconnected' a paradigm shift is required. This paradigm shift, possibly waiting just around the corner, can bring disruptive change to both the core network and the end-user part. Nevertheless, a number of important technical challenges need to be resolved before such visions are realized.

Specifically, at the core network part the dynamicity and adaptability of the environment makes it difficult for network operators to achieve revenue generation and rapid take-off through new services, while also making a reduction of network CAPEX and OPEX challenging. To this end, automated network management solutions are called to address efficient resource utilization, spanning from the energy levels, to the physical and virtual resources up to the spectrum efficiency [2]. One key question that needs to be resolved is where such mechanisms should be introduced. One trend is to move functionality at the network edge, thereby reducing complexity at the network core, leveraging on the increasing computing and processing power of the end-user devices [3].

Network virtualization and Software Defined Networking (SDN) form a promising solution for orchestrating the sharing of physical and virtual resources that can be instrumental in this direction. Network Functions Virtualization (NVF) distinguishes logical services from physical resources, also allowing for moving resources to other network locations [4]. This concept is different from the notion of SDN networks which proposes the full decoupling of the network control and data planes, moving the control of the network behavior to third party software running in external dedicated or distributed servers [3]. However several technical challenges are still under investigation at the network side, such as: (1) Isolation, that ensures high performance for certain users groups and applications; (2) Quickness: that automates the allocation of transport resources to service managers; (3) Open control, that allows dynamic specification of the VNs topology based on traffic demands; and (4) Dynamic reallocation, that deals with dynamic reallocation of physical and virtual resources according to traffic and user requirements, e.g. load balancing between virtual networks [6].

At the end-user part, the myriad of interconnected devices envisioned by the Internet of Things is finally starting to become a reality. M2M communications are gaining momentum and seem to be mature enough for real-world applications, including smart grid monitoring, smart office and smart building installations, etc. Nevertheless, before the vision of reaching 50 or 100 billion interconnected devices can be fully realized, unprecedented challenges need to be addressed with respect to the scalability, resource efficiency, and overall management of the smart "things". Last but not least, in an environment where "everything" can be measured and everything is interconnected, the protection of private information becomes more relevant than ever.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

This paper elaborates on how different aspects of virtualization can alleviate some of the identified challenges. At the core network part we elaborate on the role of emerging trends such as Network virtualization & SDN in case studies relevant for the current and emerging network ecosystem. Our work focuses on maximizing the use of physical and virtual resources and at the same time commoditize processing, storage and networking. At the level of the interconnected smart things, the notion of Virtual Sensor Network is discussed as a means for copying with scalability challenges between heterogeneous sensors and sensor networks, while enforcing least privilege properties. Finally, Virtual Sensors bring the ideas of virtualization to the extreme by supporting perhaps the key concept in the Internet of Everything [21]: going from data exchange to the exchange of relevant information, produced on the spot as and when required.

The rest of the paper is organized as follows. First, related work in the area of the different virtualization facets is presented. Next the network and end-user virtualization levels are analysed in section III. Moreover, three case studies, namely SDN controlled wireless integration service, virtual cell management and sensor virtualization are introduced in sections IV, V and VI respectively. Finally conclusion remarks are drawn in section VII.

II. Related Work

Several research efforts have been investigating the application of SDN and NFV as solutions for the abovementioned problems. For example, the notion of Cloud Radio Access Network (C-RAN) has been launched by China Mobile as an emerging umbrella vision comprising cooperation radio with high higher spectrum efficiency, open platform real-time cloud infrastructure and BS virtualization technologies [7]. The latter enables processing aggregation and dynamic allocation, reduced power consumption and increased infrastructure utilization. The proposed Intel Architecture introduces call processing in the cloud, utilizing centralized BBU pool shared among large number of virtual BS, cooperative radio as well as dynamic resource allocation to combat tide effect in utilization [7]. Intel targets a variety of systems including LTE-Advanced; evaluation studies from China Mobile have shown improvements of 50% in OPEX and 15% in CAPEX for a seven-year projection [8]. Several related activities also take place in the standardization domain. NGMN has been working on suggestions on how traditional RAN could evolve to C-RAN step by step and has already produced a stable report on such outcomes [16]. ETSI also launched in 2012 a new network operator-led Industry Specification Group (ISG) to work through the technical challenges for Network Functions Virtualisation [4].

Significant research efforts have also been focused on the end user side and especially the M2M communications and the Internet of Things part. An early definition for the IoT envisioned a world where computers would relieve humans of the Sisyphean burden of data entry, by automatically recording, storing and processing all the information relevant to the things involved in human activities, while also providing “anytime, anyplace [...] connectivity for anything” [1]. While significant progress has occurred in this direction, another (r)evolution is already on the horizon, driven by industry championed initiatives such as the General Electric concept of “Industrial Internet” [20], and the CISCO initiated “Internet of Everything” [21]. The goal is to support scalable, predictable and self-adaptive network behavior (“more relevant connectivity” in CISCO IoE terminology, “pushing the boundaries” in the GE Industrial Internet notion).

III. Virtualisation Levels

Different facets of virtualization promise to provide solutions for the performance, network/service/resource management technical challenges posed at the network operator level, as well as for the scalability, data isolation and generation of relevant information the end-user side. To this end, two main virtualization levels are introduced: the network level and the end-user level.

The concept of network virtualization proposes the abstraction of network elements and transport resources and their combination into a common distributed pool, which may involve different network locations. Depending on the network location placement the following categories may be considered: a) same network location, where the pool comprises the physical resources of a single network element which are partitioned to form virtual resources; b) different network locations, where two distinct options have been identified: 1) moving the functions of a specific network element to a standard hardware server that is located anywhere in the network or 2) separating between physical resources and logical services of network elements. In this case the network element incorporates the logical service management and the supporting hardware appliances of the physical resources may be placed anywhere in the network [16]. The connection between the logical service points and the physical hardware is realized using high connectivity means, such as fibers. In both cases, one critical issue is how to realize effective and efficient resources utilization together with flexible and robust management. Different solutions are applicable depending on the virtualization technique. At the network infrastructure level we focus on two relevant use cases, namely SDN controlled wireless integration service, and virtual cell management.

Another level of virtualization concerns the end-user, in terms of interconnecting the different user hardware appliances/things (e.g. sensor or embedded devices) and is closely related to the evolution of the Internet of Things. However, arguably the biggest innovation is that it targets to include processes and people in the loop, enabling communications that are more relevant in order to offer new capabilities, richer experiences and unprecedented economic opportunities. To pave the way for this vision, sensor virtualization will play an important role towards: (1) addressing scalability challenges in the interconnection, control and management of a plethora of heterogeneous smart things, (2) promoting cooperation between the different elements in an energy efficient way, and (3) providing a basis over which the data analytics and sensor network trends can evolve and converge, independent of manufacturer-specific hardware or software perks [22]. In this work, we investigate sensor virtualization from the perspective of extracting relevant

information from a large network of heterogeneous sensors, in a secure and efficient and device-agnostic way.

Based on the defined virtualization levels, we introduce the following functionality within the network elements and user equipment to support virtualization:

- The Human-to-Network interface, which allows for the remote management of the network element by the network management system or the network operator [17]. The Human-to-Network tool comprises a user-friendly control panel for the network operator facilitating the following functions: a) to dynamically set resource allocation strategies, e.g. considering users groups/classes and b) to receive feedback on the resources utilizations and schedule repartitioning and load balancing actions.
- The SDN Control Layer which implements the programmable control plane towards the data plane, thereby centralising the network state within this layer [18]. It allows for the dynamic shaping of traffic flows through the network devices, acting as virtual software switch or router for the physical resources. Thus, the application of the SDN Control Layer overcomes the limitations of typical static flow management schemes, which implement network traffic routing in the physical layers. More details on the pros and cons of using SDN for virtualization are described in the first case study.
- The Thin Software Virtualization layer, which is incorporated in the end-user devices to support dynamic formulation, merging and splitting of sensor network subsets that serve different applications and are possibly administered by different entities. This software caters for (1) interoperability of heterogeneous sensors from different vendors, (2) exposure of the sensor basic functionality to the data consumer and sensor assignment to tasks, (3) data isolation and enforcement of the least privilege properties, and (4) collaboration with other sensors and/or consideration of analytic models that connect the underlying phenomena so that the sensed data can be transformed to relevant information, produced and transmitted on demand.
- The Network Infrastructure Virtualization layer, which is incorporated in the core network elements in order to support resource reusability and flexible resource pooling at the PHY and MAC layers. While often collocated with SDN functionality, its purpose is to facilitate efficient usage of the network resources and not to abstract and aggregate their management from a central point. Nevertheless, it can cooperate with the SDN Control Layer so that together they can support rapid and efficient resource (re)allocation as a response to traffic peaks or other emergency situations in the network.

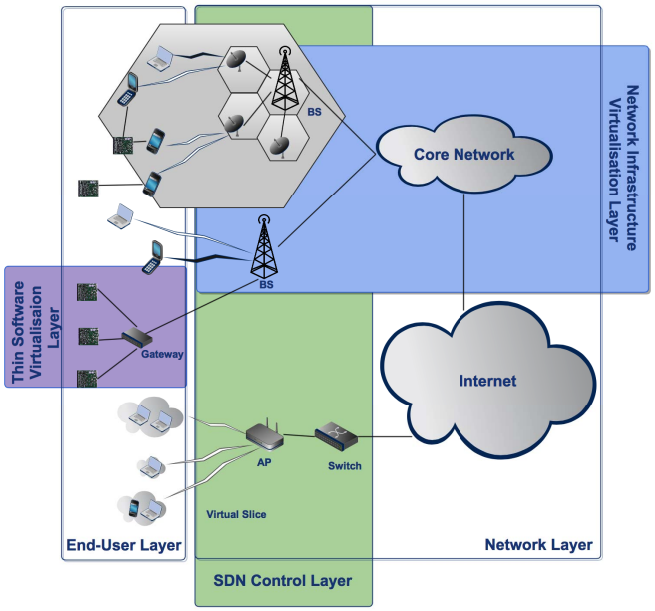


Figure 1:Virtualisation levels and layers

IV. SDN CONTROLLED WIRELESS INTEGRATION SERVICE

The key idea behind Software Defined Networking (SDN) is to provide an abstraction of lower level functionality in order to decouple the system that makes decisions regarding where traffic is sent (the control plane) from the traffic forwarding part (the data plane) [10]. The fine-grained control over traffic flows achieved with the use of a central aggregation point in a general purpose server (common API) allows for supporting multitenant configurations with significantly reduced management cost and minimized human intervention. Nevertheless, despite the significant advantages of SDN, a number of important challenges need to be addressed before it is ready for adoption in mainstream markets. Such challenges include: a) the ability to preserve the network operation (i.e., avoid having a single point of failure) in case of failures in the SDN host server/connecting switch to the network; b) the need for a realistic migration plan and a roadmap that protects investments in existing networking infrastructure [10]; c) the alleviation of security issues, e.g. with respect to the SDN protocol operation [10].

Up to now, the SDN paradigm has been actively developed and focused to the core network; its extension to wireless networks is expected to play an important role in future networks. However, the adoption of SDN in wireless scenarios poses additional requirements, namely [10]: a) introducing a wireless data plane with a single data aggregation point from the wireless infrastructure, b) supporting node mobility, c) coping with wireless links reliability issues, d) triggering a need for changes in the underlying hardware in order to offer the required functionality for efficient implementation (e.g., newer antennas, software controlled MAC layer).

This paper investigates how to deploy SDN in wireless networks focusing on two axes: the first is the extensions of wireless networks in order to support SDN functionality efficiently, and the second is the implementation of SDN functionality using current network infrastructures.

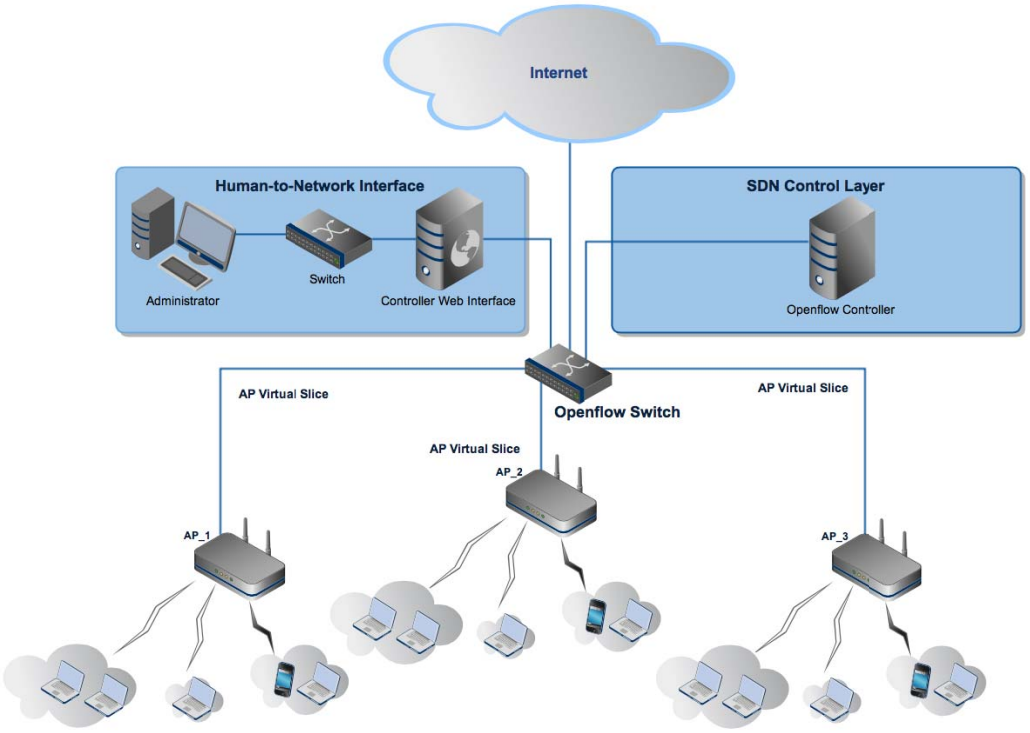


Figure 2: Overview of the WLAN prototype system for SDN controlled wireless integration service

In the considered scenario, we assume deployment of the core SDN management functionality in a single network element within the same network location. We also assume that the physical resources are partitioned using the resource slide as the main resource allocation unit. As regards the resources sharing, existing approaches can be considered and enhanced; for example, [12] proposed reconstruction algorithms to implement the goal of on-demand allocation through resource combination, resource split and resource random adjustment.

Through the Human-to-Network Interface the network operator can dynamically change the data rules on the fly, either automatically or even manually if needed. The SDN Control Layer realizes the traffic flow shaping within a resource slide or between the resource slices using offloading schemes, based on the rules defined in the Human-to-network interface. Thus, the manipulation of the logical map of a network resource slice can introduce a new service or customize the network behavior. Moreover, the traffic of the network can be controlled through a centralized network console, constraining the amount of traffic per application, per host, etc.

Below we consider the realization of this scenario assuming WLAN Radio Access Technology; however, the presented concepts are applicable to other RATs or even to multi-standard network elements. The main novelty of our work is the combination of traditional access points and SDN enabled switches to introduce the SDN capabilities in WLAN environments. It should be noted that we have deployed the presented functionalities in a proof-of-concept prototype system for WLAN resources portioning, using slices enabled by SDN technology. The Human-to-Network interface has been implemented as a control panel embedded in a web-based application. The administrator is able to configure the whole network through a web server (Controller Web Interface) running Django web framework that dispatches the requests to the OpenFlow Controller and forwards the commands to the OpenFlow switch. The SDN control layer has been implemented through the OpenFlow Controller which realizes the splitting of the physical resources into virtual slices. In addition, multiple Basic Service Set Identifications (BSSIDs) have been created with just one antenna. Each BSSID has been assigned to a different VLAN (AP virtual slice), which ends at an open flow switch. Thus the wireless interfaces have been converted to VLAN and the latter have been administered by the SDN control layer using OpenFlow. Finally, all the access points terminate at an OpenFlow switch that is handled by a controller. Figure 2 presents a high level view of our prototype system.

V. VIRTUAL CELL MANAGEMENT

The notion of “virtual cells” relies on cell grouping in neighboring geographic regions, with the formation of a large virtual cell, i.e., a cluster of cooperating small cells that appears to the user as a single distributed network element [13]. This concept was initially introduced in legacy WLAN systems, for handling the limitations of co-channel interference or APs operating in the same channel in WLAN deployment. Herein we consider the concept of “virtual cells” for eliminating the physical boundaries between the wireless resources in the heterogeneous environment, as well as within

different radio access technologies [9]. In this setting, handovers would occur only at virtual cell boundaries. To support the notion of virtual cells, the following mechanisms need to be introduced:

- Novel techniques for resource pooling are required to enable the unification of the physical access points layer towards the application connected to the virtual cell. Such mechanisms will allow the network capacity to grow linearly with the dynamic incorporation of each radio. To this end, existing algorithms can be considered and extended; e.g. [14] presents a solution for efficiently embedding requests for interconnected virtual resources into a pool of heterogeneous substrate resources with finite capacities in real-time. Greedy heuristic approaches with link mapping have been also proposed [14].
- Resource partitioning mechanisms will enable the dynamic partitioning of the unified virtual resources into separate, protected resources for each application or user; such procedure is driven by policies representing the network operators' objectives/technological constraints.
- Decision-making functionality for controlling the virtual resources pooling and partitioning needs to be specified and introduced within existing network nodes/virtual server. Such functionality will enable the dynamic transformation of network operator's goals and business objectives, expressed via human-to-network tools, into low-level actions to be enforced at the resource level.
- Supporting functionalities for virtual resources management need to be introduced, including: a) cross-system algorithms to improve the common usage of the unified resources by different operators, b) scheduling algorithms to transport the traffic associated with different users over the unified resource [15] and c) reservation procedures to support resource portioning.

For example, we examine the application of this concept for creating small virtual cells, considering an LTE BS and the distribution of the BS antennas across the entire cell. To this end, we achieve improved coverage for the end-users, since the placement of the distributed antennas can cover a larger geographical area than before. The distributed antennas are connected to a common processing unit via fiber adopting the C-RAN solution for BS functions distribution (Figure 3) [16]. If we assume a multi-operator environment, another research challenge is the optimal distribution of L1, L2, and L3 functions between Radio Unit (RU) and Digital Unit (DU) nodes.

Both the case where the operators have to be isolated and the case where agreements between them allow the virtualization unit to exchange certain parameters can benefit from the virtual cell management approach. Different architectural configurations can be evaluated with respect to tradeoffs in flexibility/responsiveness of resource sharing, energy consumption and the burden posed in the optical network [16].

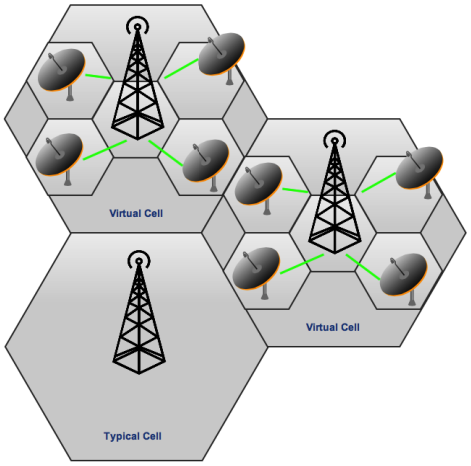


Figure 3:Virtual Cell Management

VI. SENSOR VIRTUALIZATION

Going from the infrastructure networks to the Internet of Things domain uncovers a few common but also several different challenges compared to the problems that network operators typically face in their systems. As discussed in previous sections, virtualization at the level of sensors or smart things is called to address requirements related to heterogeneity, scalability, and resource efficiency, as well as to provide the common layer over which truly disruptive technologies (e.g., from data exchange to sharing of relevant information) can be applied in a device-agnostic way.

Virtual Sensor Networks (VSNs) are emerging as a novel form of collaborative wireless sensor networks [26] that can establish the basis over which the evolution from connecting "Things" to the efficient interaction of the "Things" with processes and people can be realized [22]. A VSN can be formed by supporting logical connectivity among collaborative sensors [27], [26], [28]. Nodes are grouped into different VSNs based on the phenomenon they track (e.g., number of cars vs. NO_2 concentration) or the task they perform (e.g., environmental monitoring vs. traffic control). VSNs are expected to provide the protocol support for formation, usage, adaptation, and maintenance of the subset of sensors collaborating on a specific task(s).

Even nodes that do not sense the particular event/phenomenon (directly or indirectly by the notion of Virtual Sensor - VS) could be part of a VSN if they permit sensing nodes to communicate through them. Thus, VSNs can utilize intermediate nodes, networks, or other VSNs to deliver messages across VSN members. The same physical infrastructure can be reused for multiple applications, promoting scalability and resource efficiency.

The VSNs may also evolve into a dynamically varying subset of sensor nodes (e.g., when a phenomenon develops in the spatial domain, the sensors that can detect it change over time). Similarly, the subset of the users or processes having access rights to different subsets of the VSN can vary (e.g., the

people that have access to the network change with time or specific operations on a sensor network subset are only available to specific groups of people based on their role, etc.). This node grouping, merging and splitting property makes it easier to define, apply, and update policies (e.g., least privilege access) based on conceptual models rather than by configuring each of the myriad nodes independently.

Having alleviated part of the scalability and information protection/privacy requirements through the VSN concept is a good starting point for progressing on an even more ambitious front: going from data exchange between sensors to the sharing of relevant information, produced on the spot as and when required, so that it can be consumed on demand by processes and people. This paradigm is also promising to address the transmission and processing challenges that traditional large scale sensor installations face. The latter include various Big Data scalability issues with respect to the centralized gathering, logging and processing of the sensor data. The Virtual Sensor notion is instrumental in this effort.

In this paper, we use the term Virtual Sensor (VS) to refer to a software entity that can serve as an aggregation point for multiple sensors, using physical sensor entities and a computational model to combine their measurements [22]. The Virtual Sensor can be a thin layer of virtualization software that is executed on physical sensors (often referred as embedded hypervisor) or it can be a mathematical model for aggregating information residing in a sensor management platform similar to [23].

These different realizations of the VS notion face different types of challenges. For example, centralized or hybrid semi-centralized solutions based on analytic engines have to address the challenges of data fusion from heterogeneous sources, both functional (different credibility levels of the sensors, co-dependent sensor observations, difficulty to link human information needs to sensor control, etc.) and non-functional (scalability and performance problems, security and privacy requirements, etc.). On the other hand, the embedded hypervisors have to cope with the integrated nature of embedded systems, and the related need for isolated functional blocks within the system to communicate rapidly, to achieve real-time/deterministic performance, and to meet a wide range of security and reliability requirements [24], [25]. Nevertheless, bringing more processing capacity and intelligence in the end devices is both inevitable and necessary in order to cope with scalability challenges [29]. Related efforts, often referred as “analytics at the edge” are focusing on realizing this transition.

However, despite the different types of challenges, both embedded hypervisors and analytically/computationally realized virtual sensors share a number of key properties. First and foremost, the VS is doing more than interpolating values of physical sensors measuring the same phenomenon, as translation between different types of physical sensors is a far more interesting topic when models for the relations between the underlying phenomena are available.

An interesting use case for this translation process in an urban setting is the estimation of car pollution based on a model that combines car counting (e.g., by induction loops or cameras) and weather conditions, while possibly utilizing also the information from the few available pollution sensors [22]. In this case the VS can be configured to report periodically the estimated pollution value and give a warning if the pollution is above certain regulations (relevant information), instead of continuously reporting all the data.

Another example that is applicable in smart grid scenarios is the calculation of electric grid parameters (e.g., the load on given points in the transmission network, or the sag of transmission lines). Such information can be deduced by the Virtual Sensor indirectly from correlated values and a model of the related phenomena, even with a sparse network of different sensors (e.g., voltage and temperature sensors for the sag case, coupled with measurements of wind speed from a nearby weather station). Again the Virtual Sensor can issue warnings or alerts when some dynamic threshold values are exceeded instead of producing and transmitting all the information continuously. It is important to note that both the embedded hypervisor and the platform-based realizations of Virtual Sensors can employ state of the art signal processing techniques such as compressive sensing (for efficiently reconstructing a signal from relatively few measurements taking advantage of sparseness properties) or robust statistics (for copying with outliers, impulsive interference, etc.).

At their core, VSNs and VS are building on and/or extending existing collaborative networking paradigms, therefore classifying them with respect to the ways that cooperation is realized in more conventional cooperative communication schemes is of great value. Taking into consideration the properties of Virtual Sensors and VSNs discussed previously, an updated model of the 3D cooperative methods taxonomy introduced in [29] that also captures the different sensor-level virtualization aspects described above is provided below. Figure 4 depicts the scope of the cooperation as planes in a 3D space (information exchange, decision and configuration control, and layer mechanisms), with each dimension being associated to a set of enablers and technical areas [22].

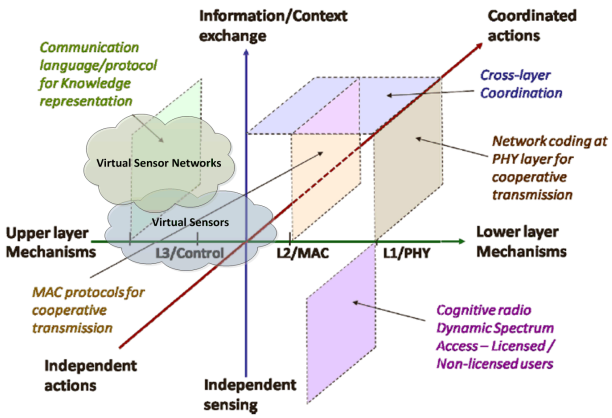


Figure 4: Sensor Infrastructure Virtulization depicted over the various dimensions of cooperative decision making and control.

VII. CONCLUSIONS

This work looks into the dynamics of the "hyperconnected full of data" telecommunications environment. To address the technical requirements at both the network and end-user side, an emerging solution lies in moving the functionality to the network edges. Network virtualization and SDN have been recognized as solutions for optimising physical and virtual resources management and allocation. The different facets of virtualization have been approached through three case studies: Software Defined Networking controlled wireless integration service, virtual cell management, and sensor networks virtualization. We also discussed evolution scenarios based on network and infrastructure virtualisation towards a fully interconnected Internet of Everything environment. The latter offers both enhanced QoS/QoE to the end-user and gains for the network operator in terms of automation, energy consumption and CAPEX/OPEX.

ACKNOWLEDGMENT

Part of this research was funded by EU LiveCity project [5], grant agreement No. 297291. Part of this work has been performed under the research project FutureNET; the research project is implemented within the framework of the Action «Supporting Postdoctoral Researchers» of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

REFERENCES

[1] A. Manzalini, "Building the Edge ICT Fabrics", Future Internet Assembly, May 2013, Dublin, Ireland.

[2] P. Chatzimisios, Ch.Verikoukis, I.Santamaria, M. Laddomada and O. Hoffmann, "Mobile Lightweight Wireless Systems", ISBN: 978-3-642-16643-3, 2010.

[3] A. Manzalini et al., "Clouds of virtual machines in edge networks," IEEE *Communications Magazine, IEEE* , vol.51, no.7, July 2013.

[4] ETSI NFV white paper "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action, "SDN and OpenFlow World Congress", Darmstadt,Germany, October 2012.

[5] R. Kokku, R.Mahindra, H. Zhang, S. Rangarajan, "NVS: A Substrate for Virtualizing Wire less Resources in Cellular Networks". IEEE/ACM Transactions on Networking, vol. 20, no.5, pp.1333-1346, 2012.

[6] A. Masuda, et al.,'Experiments on Multi-layer Network Virtualization towards the Software Defined Transport Network'. SNPD 2012: 661-665.

[7] China Mobile Research Institute, 'C-RAN: the road towards Green RAN', version 2.5, October 2011.

[8] R. Gadiyar, 'Cloud RAN with Intel® Architecture', Wireless World Research Forum October 18, 2011.

[9] R. Q. Hu, S. Talwar, & P.Zong, "Cooperative and Green Heterogeneous Wireless Networks", 23rd International Teletraffic Congress (ITC 23), San Francisco, USA September 2011.

[10] S. Ortiz, "Software-Defined Networking: On the Verge of a Breakthrough?", IEEE Computer, vol. 46, no. 7, July 2013, pp. 10-12.

[11] S.Costanzo, L.Galluccio, G.Morabito,and S.Palazzo. Software Defined Wireless Networks: Unbridling SDNs. Proc. of EWSDN 2012. Darmstadt,Germany, October 2012.

[12] X-J. Chen, "Resource reconstruction algorithms for on-demand allocation in virtual computing resource pool", International Journal of Automation and Computing vol. 9, no. 2, April 2012. p. 142 –154.

[13] E. Kudoh and F. Adachi, "Power and frequency efficient multi-hop virtual cellular concept", IEICE Trans. Commun., vol. E88-B, no.4, pp.1613-1621, April 2005.

[14] C. Papagianni, A. Leivadeas and S. Papavassiliou, "A Cloud-Oriented Content Delivery Network Paradigm: Modeling and Assessment", IEEE Transactions on Dependable and Secure Computing, vol.10, no.3, pp. 287 – 300, Sept.-Oct. 2013.

[15] E. Patouni, N.Alonistioti and L.Merakos, "Cognitive Decision Making for Reconfiguration in Heterogeneous Radio Network Environments", in IEEE Transactions on Vehicular Technology (TVT), special issue on "Achievements and the Road Ahead: The First Decade of Cognitive Radio, vol. 59, issue 4, May 2010, pp. 1887-1900.

[16] NGMN Technical Document, "Suggestions on Potentil Solutions to C-RAN by NGMN Alliance", January 2013, version 4.0.

[17] E. Patouni, B. Fuentes and N. Alonistioti, 'A Network and Service Governance Framework: Case Study for Efficient Load Balancing', in the Proceedings of the IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks 2012 (CAMAD 2012), 17-19 September, Barcelona, Spain.

[18] Software-Defined Networking:The New Norm for Networks, ONF White Paper April 13, 2012.

[19] International Telecommunication Union "The Internet of Things" 2005.

[20] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines", GE White Paper, Nov. 26, 2012, [Online]. http://www.ge.com/docs/chapters/Industrial_Internet.pdf, last access: July 2013.

[21] D. Evans, "The Internet of Everything: How More Relevant and Valuable Connections Will Change the World", CISCO White Paper [Online]. http://www.cisco.com/web/about/ac79/docs/innov/IoE.pdf, last access: July 2013.

[22] A. Merentitis, et al., "WSN Trends: Sensor Infrastructure Virtualization as a Driver Towards the Evolution of the Internet of Things", International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), September 29 - October 3, 2013, Porto, Portugal.

[23] V. Gazis, K. Sasloglou, N. Frangiadakis, P. Kikiras, A. Merentitis, K. Mathioudakis, and G. Mazarakis, "Architectural Blueprints of a Unified Sensing Platform for the Internet of Things" – International Conference on Computer Communications and Networks (ICCCN), 30 July – 2 August, 2013, Nassau, Bahamas.

[24] A. Merentitis, N. Kranitis, A. Paschalis, and D. Gizopoulos, "Low Energy On-Line Self-Test of Embedded Processors in Dependable WSN Nodes", IEEE Transactions on Dependable and Secure Computing, 2011, pp-86,100.

[25] A. Merentitis, G. Theodorou, M. Georgaras, N. Kranitis, "Directed Random SBST Generation for On-Line Testing of Pipelined Processors", International On-Line Testing Symposium (IOLTS), 6-9 July 2008, Rhodes, Greece.

[26] L. Sarakis, T. Zahariadis, H. Leligou, and M. Dohler, "A framework for service provisioning in virtual sensor networks" EURASIP Journal on Wireless Communications and Networking 2012, pp. 1-19.

[27] H. M. N. D. Bandara, A. P. Jayasumana, and T. H. Illangasekare, "Cluster tree based self organization of virtual sensor networks," GLOBECOM Workshops Nov. 2008, pp.c 1-6, New Orleans USA.

[28] N. M. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," Computer Networks, vol. 54, no 5, Apr. 2010, pp. 862–876.

[29] G. Koudouridis, et al., "Enablers for Energy-Aware Cooperative Decision and Control in Wireless Networks", Vehicular Technology Conference C2POWER Workshop, May 2011, pp.1-5, Budapest, Hungary.

[30] National Instrumnets, Data Acquisition Technology Outlook 2013, [Online]. http://www.ni.com/daq-trends/, last access: July 2013.

On the implementation of NFV over an OpenFlow infrastructure: Routing Function Virtualization

Josep Batallé, Jordi Ferrer Riera, Eduard Escalona and Joan A. García-Espín
Fundació i2CAT
Distributed Applications and Networks Area (DANA)
C./ Gran Capità 2-4, 08034, Barcelona
{josep.batalle,jordi.ferrer,eduard.escalona,joan.antoni.garcia}@i2cat.net

***Abstract*—Network Function Virtualization (NFV) has emerged as an operator proposal for offering network services with network functions implemented in software, which may be located in Data Centers, network nodes or even in a virtual machine. NFV is theoretically applicable to any network function and aims at simplifying the management of heterogeneous hardware platforms. Using as a basis one of the network functions offered by network routers, this article presents the analysis, design, and the first implementation, in a virtualized manner, of the routing function. Considering the current co-existence of IPv4 and IPv6 and the possibilities brought into the arena by OpenFlow-enabled infrastructures, the article describes the design of the virtualized routing protocol, enabling a simple management and avoiding signaling messages overhead in the control plane level, and the different scenarios considered to validate the virtualized function. In essence, the manuscript describes one of the first implementations of the functional NFV concept, through the virtualization of the routing function over an OpenFlow network. The different scenarios validated in the article are used to demonstrate the applicability of the NFV-powered implementation proposed into actual production environments.**

I. Introduction

Economical constraints and restrictions are nowadays present in all the business areas, and telecom markets are not an exception. Network operators have to deal with a number of factors when designing and deploying their new network infrastructures. Several parameters must be considered apart from the desired technical capacities (e.g. man power, number of network nodes, or overall power consumption). The deployment of future networks infrastructure requires new dynamic, simple and economic management mechanisms. One of the most promising approaches that has emerged in the recent years is the Software Defined Networks (SDN)[1] concept. SDN advocates for the separation between the data forwarding plane and the corresponding control plane, which is implemented out of the box in a separate SDN controller. With this approach, the network can be dynamically programmed and administrators are able to build customized solutions to manage the infrastructure services.

OpenFlow[2] is a configuration protocol that enables SDN architectures. It was originated from research work conducted at Stanford University and the University of California. Currently, OpenFlow is the most used solution within SDN, although it is not the only feasible approach. OpenFlow allows the separation of the control plane from the data plane and defines the communication between two types of devices, a controller and a switch. The switch is responsible for the packet forwarding to other network devices and the controller manages the network forwarding elements. The communication between the data plane inside the switches and the control plane in an external controller is done with the OpenFlow protocol using a secure channel. Commonly, an OpenFlow controller is connected to multiple switches and provides the management of all of them because it has a centralized view of the network.

Along with SDN, NFV [3] has arisen as an operator-promoted initiative with the objective of increasing the flexibility of network services deployment and integration within the operators network. This is achieved by means of implementing network functions through software, avoiding specific hardware-based devices and increasing the return on investment. This concept is applicable to a number network elements and offer several benefits thanks to the enhanced network programmability. NFV in fact aims to transform the way that network operators architect their networks. Although related and highly complementary, SDN and NFV do not depend on each other. There are several challenges to be faced in order to apply NFV, contributing to a dramatic change in the telecom industry landscape (e.g. reduce equipment costs and power consumption, enable a wide variety of eco-systems through openness, or supporting multi-tenancy).

Network operators and service providers are interested in NFV because it enables the establishment of new mechanisms to deploy and operate network and infrastructure services [3]. However, currently, NFV is a new functional idea defined that it is still not yet implemented. Considering the definition, each network element has a set of different functions that can be potentially extracted and allocated into external elements that can later be moved, instantiated, duplicated, and managed individually. This approach opens a wide range of possibilities to be adopted in different scenarios.

This article focuses on the proof-of-concept virtualization of the routing network function over an OpenFlow-enabled network by externalizing routing decisions from the actual equipment. Depending on the scenarios, it becomes reasonable to virtualize the routing function and logically centralize the routing knowledge, being capable of moving its location along the network and duplicating it if required, and also being capable of maintaining dynamic and simplified routing tables between the different network elements. Once the NFV proposal is demonstrated with a functional prototype, we provide solutions for specific use cases: (i) the separation between IPv4

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

and IPv6 routing and (ii) offer an inter-domain routing under OpenFlow network. The first use case is proposed due to the emergent migration to IPv6 required by network operators. And the other use case tries to offer support for inter-domain routing because we trust that a centralized function can helps with the management of the routing through different domains managed by different OpenFlow controllers.

One of the benefits of our implementation is the reduction of OPEX and CAPEX. Cost reduction is achieved through an NFV approach since the management of the routing is completely centralized and externalized, reducing configuration and deployment time as well as the manufacturing costs, since the physical device is completely devoted to forward packets, while the intelligence is located in the corresponding controller or in the routing element. Additionally, the host responsible of the routing can be managed easily through dedicated APIs, or even shared between different operators. The application of a Network as a Service (NaaS) paradigm can also add value to the management of a virtualized routing function, since the network function can be seen as a component of the network that is managed and provided through service delivery frameworks.

To the best of our knowledge, we provide for the first time results of experimenting with different NFV scenarios for the virtualized routing function. The paper represents a novel proof-of-concept on how the NFV concept can be implemented and how the separation of control and data planes fostered by SDN provide benefits for network management. Although focusing on routing, we believe the elements and methodology used to implement this experiment can be exploited in the procedures of virtualizing other network functions. The rest of the manuscript is structured as follows. Section II presents the related work performed with routing over OpenFlow networks as well as some background components. Then, Section III contains the description of the design and implementation performed to achieve the virtualization of the routing function over an OpenFlow-enabled scenario. Section IV describes the evaluation of the system implemented as well as the description and analysis of the different metrics utilized to evaluate it. Finally, Section V gives the conclusion and some future research actions that can be taken.

II. Related Work

OpenFlow is the most common protocol for enabling SDN. It is already functional in different research, educational and even production infrastructures. Additionally, there are several open source projects developed by the community that provide new services and tools for such operational networks. OpenFlow controllers (e.g. Floodlight[1]) provide basic features such as routing, topology discovery, security, or even firewall rules. The first implementations of the controllers are centralized, so these components hold a complete view of the network. We believe that the integration of these functions inside the controller is useful in small-sized networks; however, when applied to bigger scenarios, moving such features into an external element enables (i) the management of the network function from a centralized point of view, and (ii) the management of the host where the network function is deployed.

The basis of OpenFlow operation is the usage of L2, since communication in this layer requires less intelligence to process the packet (i.e. less encapsulation of datagrams). It operates as follows. A host sends a packet towards the switch. The switch receives it and attempts to match the header of the packet in its own Flow Table. If there is no matching, the switch asks the controller about the route of the specific packet through a packet-in message. Then the OpenFlow controller floods the network with ARP requests in order to obtain the path. Once the controller is aware of the route, it sends a FlowMod and a packet-out messages to update the flow tables of the corresponding switches, which are now capable of forwarding the packet through the specified ports.

Unlike a simple OpenFlow network, the communication between different entities takes place at Layer 3 through the IP protocol. Different proposals have studied IP routing features. The design and implementation of a routing control system is demonstrated in [4]. Furthermore, other proposals contain relevant information about routing platforms, such as RCP [5]. In the case of OpenFlow networks, the routing function is directly included into one research project; RouteFlow [6] implements an IP virtual routing service in an OpenFlow network, and provides this function using a single OpenFlow controller and a single RouteFlow server. In contrast to this approach, our proposal includes one IP routing service towards one or multiple controllers, considering the NFV concept. Indeed, one of the feasible scenarios for our approach is the inter-domain routing, where several controllers are involved.

Other projects like HyperFlow [7] or Flowvisor [8] present different solutions for the design and management of networks with multiple controllers. The first is a distributed control plane for OpenFlow networks, while the second is a special-purpose controller used to create rich slices of network resources and delegate control to other controllers. These approaches do not externalize any network function, but they concentrate them inside the controllers. Furthermore, since all the functions are concentrated, there is no need to develop any communication protocol between the controller and any other external entity.

Authors in [9] present the *controller placement problem*, where they describe the importance of taking into account the location of the controllers in SDN networks. In essence, this problem is duplicated when virtualizing any network function and deploying it into an external host. Besides, network size grows with the corresponding increase of number of elements and different type of resources [10]. Our proposal also includes resource management using the OpenNaaS[2] framework, one of the most used NaaS platforms, which allows to manage in a centralized manner different physical or virtual resources (or even network functions) that are contained within a given network.

III. Routing Function Virtualization

The implementation done in this article is separated in three different components. The first one involves the development of a new module for a given OpenFlow controller in order to detect the packet received by the switch, analyze the corresponding information (e.g. type of protocol, whether it is IPv4 or IPv6, or source or target address) and send it to

[1] http://www.projectfloodlight.org/floodlight/

[2] http://www.opennaas.org

the virtualized routing function if required. Then, the second component is the externalization of the routing function itself for receiving the different routing requests and provides a feasible path for a given packet. Finally, the third component is the communication protocol between the controller and the routing function, which needs to cover the different scenarios. Figure 1 depicts an overall view of the global architecture.

The idea behind the virtualized function is to enable the network routing, both IPv4 and IPv6, when a switch does not have the knowledge to forward traffic packets. In this case, the communication protocol designed is responsible for enabling the interaction between the switch controller and the virtualized routing host, deployed in a different machine. To implement the prototype, we have modified the Floodlight[11] controller, an OpenFlow controller developed in Java and commissioned to make decisions on how to handle traffic packets; on the other hand, a new bundle in the OpenNaaS [12] framework has been created in order to offer the routing function.

A. NFV Module

Floodlight is an open Java-based OpenFlow controller available to the community of developers. It is the selected controller for the experiments in this article. The functionalities of Floodlight are defined in different modules, where each one of these modules implements a specific task. Most current OpenFlow networks are deployed in education and research environments where the networks are small and with few numbers of hosts and switches. Usually, in these types of networks all switches are connected to the same controller. The deployed scenario implements a new Floodlight module that is deployed in the controller that is managing the domain. This module is responsible for detecting the packet-in received from the switch and for processing the header of the packet to detect if it is an IPv4 or an IPv6 packet. Then, it sends a request to the corresponding virtualized routing element in order to obtain the corresponding path. Once the module receives a response from the routing function, it sends a packet-out and pushes the corresponding entry to the Flow Table of the switch.

B. Routing Resource

The second part is the implementation of the routing function as a resource in OpenNaaS. OpenNaaS is an open-source framework that provides tools to manage the different resources present in any network infrastructure. OpenNaaS provides a neutral tool to different stakeholders to enable mutual contribution and benefit from a common NaaS software-oriented stack for both network applications and services management. It is developed by an open community under LGPLv3 license. The virtualized routing function developed in this project is implemented as an OpenNaaS resource with a set of capabilities and an associated model used to maintain the overall view of the routing state.

The model is defined following a route table-like structure and each routing resource contains two types of tables: one for IPv4 routing and one for IPv6 routing. The routes are defined by the Source IP, Destination IP, Datapath Identifier (DPID)[3], input port and output port. In the case of OpenFlow switches,

[3]Identifier of the switch, that corresponds to the MAC address

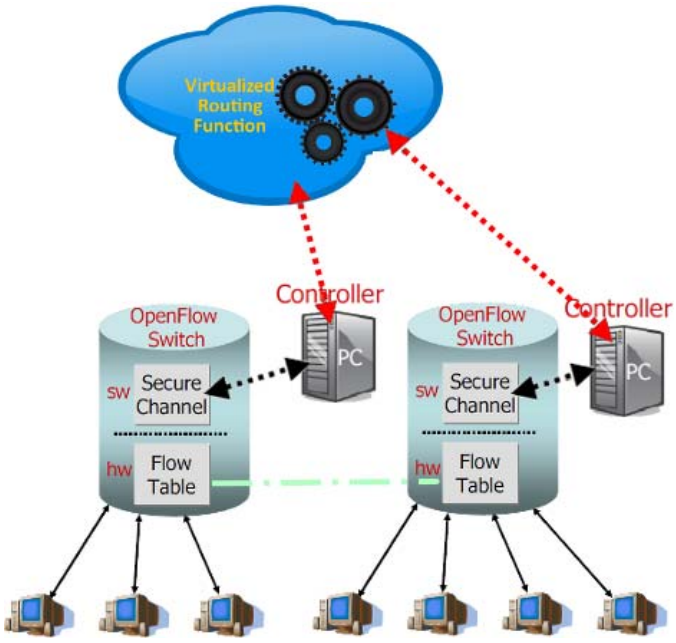


Fig. 1. Architecture of the RFV

the ports are identified by numbers. Therefore, the path of one route is identified by the input and output port of the switch. Furthermore, each route is associated with an identifier and a lifetime stamp. The basic functionalities defined in the virtualized routing resource provide the path finding capabilities for specific input parameters and for management features such as insert or delete a given route, and retrieve information corresponding to the whole routing table.

OpenNaaS provides a Command Line Interface (CLI) and a REpresentational State Transfer (REST) interface that give access to the resource using command line or through Web Services respectively. Through this interface, the above-mentioned capabilities and resource model can be accessed using a URL address and the corresponding parameters. This REST interface can be called from our Floodlight module in order to obtain the output port from a route. In addition, this feature allows us to create a Web GUI for managing the centralized routing function, retrieve the route table and insert new routes.

C. Communication Protocol

Figure 2 shows a flow diagram that describes in detail the procedure followed by each packet received by the switch. Basically, the part performed by the switch is the same as the normal operation of OpenFlow. The newly implementation functionalities, highlighted in red, are enabled when the controller receives the packet-in message from the switch. First, the controller parse the message, and then it triggers a request to the Routing Function Virtualization (RFV) machine using the REST interface. Then, the RFV finds a path in its route table and sends a response back to the controller. This response contains the output port of the switch and is included in the FlowMod message.

One of the drawbacks of NFV is the latency and the externalization of the routing function further increases the

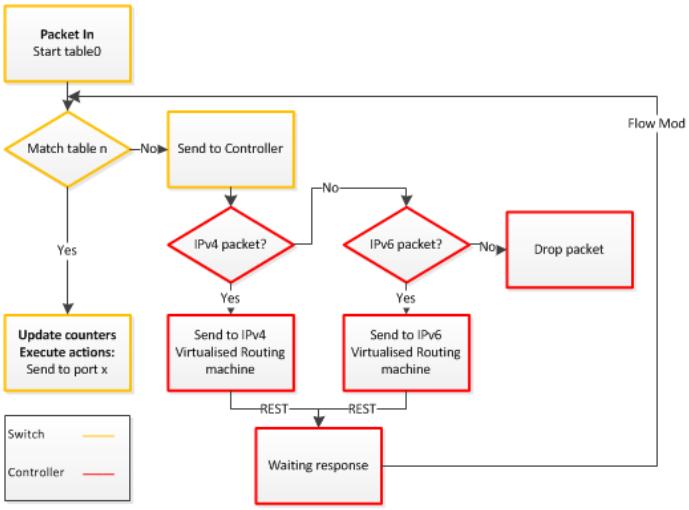


Fig. 2. Flow diagram of the implementation

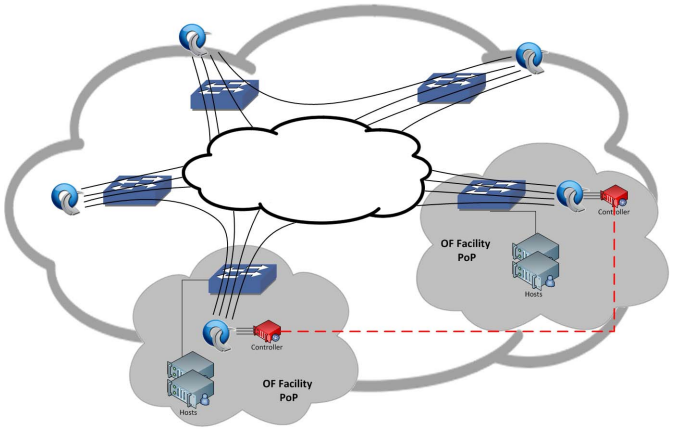


Fig. 3. Network topology

delay of the system caused by the communication between the controller and our routing machine. We have implemented a proactive mechanism that informs the switch when a packet is going to be received within few milliseconds. With this mechanism, the communication is started by the RFV bundle, which is able to call the Floodlight REST interface asynchronously and proactively.

Figure 1 shows the architecture of the RFV. The data plane is connected with other switches in order to establish L2 communication. Then, the control plane of the switch is connected to a controller using a secure channel that provides the OpenFlow protocol. Finally, through a Virtual Private Network (VPN) link each controller is connected to the RFV, which can be located anywhere in the network or even outside. Therefore, there is a complete separation between the data and the control plane.

The exchange messages in the communication between the OpenFlow Controller (OFC) and the RFV can be done in two directions:

- **OFC to RFV**. The message contains the source and destination IP of the packet, the DPID of the switch and the input port where the packet enters the switch. The routing machine responds with a string that contains the output port of the switch and the subnets that the switch can handle in order to create new rules related to subnetworks.
- **RFV to OFC** This is a response to the previous message received from the OFC. Another message can be used in order to offer a proactive strategy.

These two messages are defined using a REST interface; therefore, the responses are received following the set of HTTP rules. For example, the response for the first type of messages contains the HTTP code status, output port and the subnetwork information about the source and destination hosts. Once the OFC receives the response, it informs the switch what is the destination of the packet (the output port) using the packet-out event defined in the OpenFlow specification. This process allows to forward the packet instantly, but the flow is not inserted in the Flow Table. Then, a FlowMod message is required for adding the rule in the Flow Table of the switch.

IV. Evaluation

We have defined different scenarios to be validated over the test-bed and demonstrate that our proposal can be deployed in a real environment. The test-bed is based on L2 switching devices with multiple OpenFlow islands (group of interconnected OpenFlow switches). Specifically, our scenario is composed of five islands, each controlled by a controller (i.e. Floodlight). These controllers include the new developed modules proposed in the previous sections. The test-bed also contains an instance of the OpenNaaS framework deployed in an independent host with the virtualized routing bundle instantiated. Communication between the controllers and OpenNaaS is performed through a dedicated Signaling Control Network (SCN) deployed over a VPN. Among the different evaluated use cases, the most relevant are the inter-domain routing management between different OpenFlow islands, and the separation of IPv4 and IPv6 routing, since they can be deployed independently in different OpenNaaS resources.

The main purpose of each experiment has been to enable the communication between hosts located at different islands (e.g. video streaming) and analyze how the OpenFlow controller and the virtualized routing function perform using the designed protocol and the proof-of-concept prototype. Different approaches for the routing (i.e. reactive, and proactive) have been analyzed depending on the traffic load, as well as different performance indicators such as the number of flow entries in the switch or the overhead caused by the signaling packet exchange.

Figure 3 depicts the reference network topology used for this experiment, and based on the OpenFlow infrastructure of GÉANT [13], a real scenario deployed by GÉANT that offer an infrastructure to try out and evolve OpenFlow-based SDN solutions. But due to the lack of availability of the infrastructure, the network is simulated using Mininet [14], which allows to create virtual switches running OpenVSwitch [15]. Each switch is connected to one controller, and manages a Point of Presence (PoP) providing the communication with other switches. The controllers run in servers that can be accessed through Internet and are interconnected between them

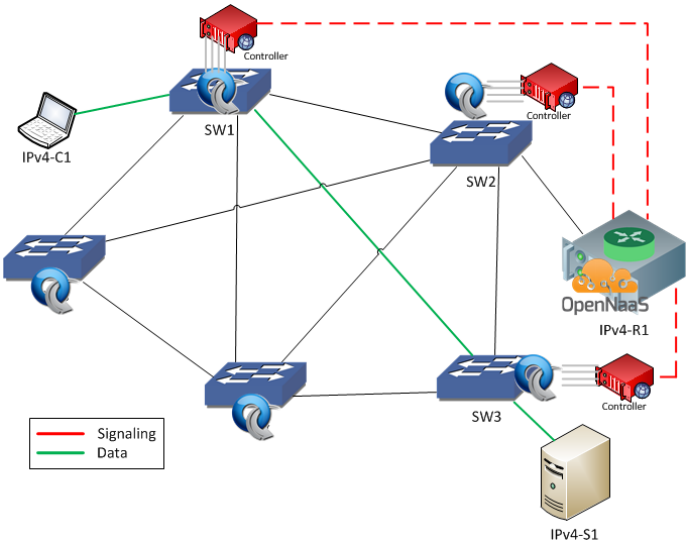


Fig. 4. Scenario with IPv4 routing

using a VPN. This network tries to simulate an inter-domain network, where each switch is the entry point to another network. We include two different scenarios in this article. One with point-to-point communication between two single hosts, and another scenario with fifty nodes populating each PoP.

A. *Round-Trip delay Time (RTT)*

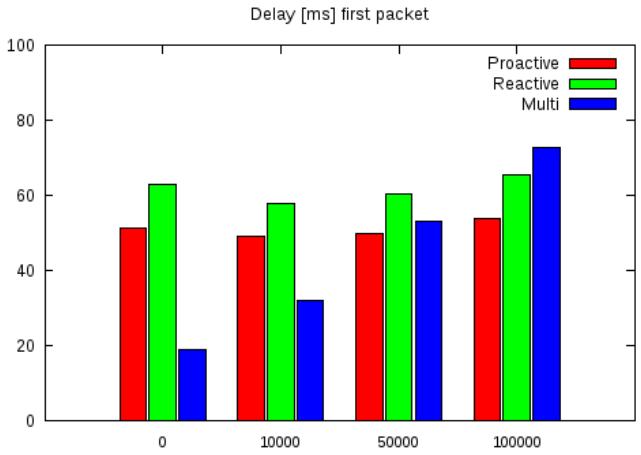
According to the authors in [3], the performance of the network can be degraded when NFV solutions are deployed, being the latency one of the affected metrics. Adding a new element in the routing decision mechanisms implies that the RTT will increase.

In this evaluation, the RTT is compared with the normal operation of the Floodlight controller and with our implementation using both the reactive and the proactive design. In addition, our scenarios try to emulate an inter-domain network, where the number of controllers increases depending on the number of switches to be controlled. The tested scenarios are two: the first implies a communication between two hosts where each host is located in a different PoP and is represented in the Figure 4. The related test tries to communicate the source host located inside the SW1 with other host located inside SW3. The second scenario is equivalent, but the communication is done between 50 source hosts and 50 destination hosts located in other PoP.

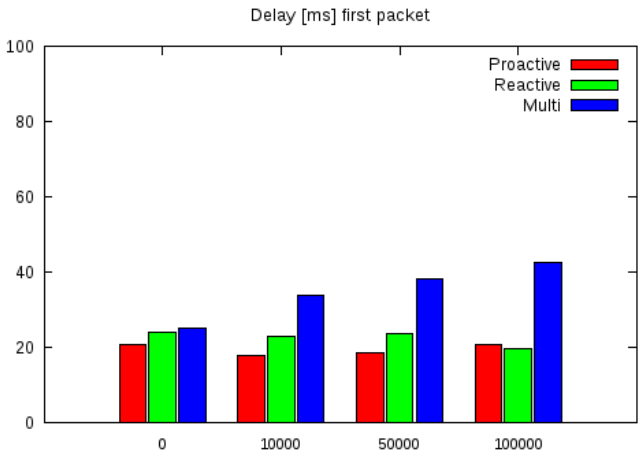
The results of the first scenario are depicted in Figure 5(a). As expected, the addition of a new element in the chain results in increased delay compared to the normal operation of Floodlight. However, when the number of flow entries increase, the RTT in our proposal maintains a stable behavior. Moreover, the proactive mechanism improves the results of the reactive strategy. The major difference takes place when the number of flow entries is zero. Another conclusion of this graph is that using the proactive mechanism it is possible to reduce the metrics related to the average delay when the number of flow entries is high.

Figure 5(b) depicts the RTT in the same scenario, but with 50 hosts in the origin and destination PoPs. The RTT measured contains the average value of the communication between all hosts. In this case, the average is lower than before because the number of requests to the controllers is lower. Then, our proposed implementation outperforms the results of the previous scenario for different reasons. First, the number of flow entries is significantly lower using a proactive mechanism compared to the reactive mechanism and the normal Floodlight operation. Then, the number of destination hosts makes the ARP flooding more inefficient. Furthermore, our proposal remains stable when the number of flow entries increase. And finally, using a proactive strategy, the number of request to an external machine is also lower because the first communication can establish a flow for the source host.

By means of using the proactive routing, we can avoid sending too many messages to the RFV, thus reducing the overall RTT. However, the fact that these scenarios are small-sized and are simulated using virtual machines imply very low transmission delays, therefore, packets can sometimes be transmitted faster (from SW1 to SW3) than the FlowMod messages that are pushed to SW3. This means that in real networks the positive effect of the proactive mechanism would increase.



(a) RTT in scenario 1



(b) RTT in scenario 2

Fig. 5. Simulation results first communication (a) and (b)

		RFV	
Scenario	**Default**	**SW1**	**SW2**
1	2	4	4
2	2500	102	102

TABLE I. NUMBER OF FLOW ENTRIES REQUIRED IN EACH SWITCH FOR EACH SCENARIO

B. Number of Flow entries

Currently, when a controller receives a packet-in and finds the path to the destination host, it creates one flow per communication between the origin host and the destination host. For a single communication each switch needs to contain four flow entries, two in order to send the packet through one direction, and the other two for the other direction. These two packets for each direction are ARP packets and IPv4 packets. In our proposal, the intelligence of the system is moved to an external machine prepared to do the programmed work. This machine can do in fact some calculations, and can manage the routes with different data bases. This provides a way to reduce the entries in the flow tables of each switch.

This evaluation compares the number of flows saved in each switch using an unmodified Floodlight, and then using our proposal. Theoretically, in the first case the number of flow entries needed by the switch for establishing a communication between hosts is defined by the following operation: $n * 4$, where n is the number of communications. However, usually Floodlight uses the MAC address to create flows. Using our proposal, the number of flow entries needed is defined by the following expression: $2 * n + 2$ where n is the number of communications. Table I shows the comparison with the same scenario of the Figure 4. The first column shows the scenario (where the first scenario the value of n is 1, and for the second scenario is 2500 due to the communication between the 50 hosts), and the second and third column show the number of flow entries for each case, using a default Floodlight configuration and with our NFV proposal. When the number of communication increases, our proposal reduces the number of flow entries in a half plus two. The reduction is caused because using the routing machine the management of subnets is enabled and the RFV is in charge of calculating routes and inserting the best flow entry. In addition, this means that our proposal makes a network more scalable than the normal Floodlight.

V. CONCLUSIONS

We have presented a proof-of-concept implementation of the routing function virtualization, compliant with the NFV approach. We have shown what are the different components required to deploy the prototype, as well as the designed protocol to implement the communication between the controllers and the virtualized function. In order to validate the implementation, we have proposed, executed, and analyzed a set of experiments.

Our approach aims at reducing the number of specific routing devices, and consequently the required equipment, space and energy consumption associated to these devices, thus reducing capital expenditures. The results show that the RTT remains stable in the proposed scenarios when the number of requests increases. Furthermore, the number of flow entries in the switches can be reduced due to the transfer of the intelligence to an external machine.

To the best of our knowledge, these are the first validation results of a specific NFV-like implementation. Results show that performance and scalability are assured in the presented scenarios, but in order to demonstrate the robustness of the virtualized function, more evaluation are needed. We have also presented two sound use cases where our proposal would cover all the requirements (i.e. inter-domain routing, IPv4 - IPv6 migration). Last but not least, it is worth to mention that a lot of future research directions are opened from here, going from implementing dynamic routing protocols in the virtualized host towards different optimization policies for the routing.

REFERENCES

[1] O. M. E. Committee., "Software-defined networking: The new norm for networks.," *Open Networking Foundation*, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[3] "White paper on "Network Functions Virtualisation"." http://portal.etsi.org/NFV/NFV_White_Paper.pdf, 2012. [Online; accessed 18-Apr-2013].

[4] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The case for separating routing from routers," in *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA '04, (New York, NY, USA), pp. 5–12, ACM, 2004.

[5] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, (Berkeley, CA, USA), pp. 15–28, USENIX Association, 2005.

[6] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, "Virtual routers as a service: the routeflow approach leveraging software-defined networks," in *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, (New York, NY, USA), pp. 34–37, ACM, 2011.

[7] A. Tootoonchian and Y. Ganjali, "Hyperflow: a distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, (Berkeley, CA, USA), pp. 3–3, USENIX Association, 2010.

[8] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[9] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *Computer Communication Review*, pp. 473–478, 2012.

[10] H. Kim and N. Feamster, "Improving network management with software defined networking.," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[11] "Floodlight, Java-based OpenFlow Controller. ." http://www.projectfloodlight.org/. [Online; accessed 2-Jul-2013].

[12] "OpenNaaS platform." http://www.opennaas.org. [Online; accessed 20-May-2013].

[13] "Technical Information GÉANT OpenFlow Facility." http://www.geant.net/opencalls/Overview/Documents/Open%20Call%20Technical%20Annex%20B%20GEANT%20Openflow%20Testbed%20Facility%20FINAL.pdf, 2013. [Online; accessed 19-Sept-2013].

[14] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, (New York, NY, USA), pp. 19:1–19:6, ACM, 2010.

[15] "Open vSwitch Features." http://www.openvswitch.org/, 2013. [Online; accessed 19-Feb-2013].

CONTENT Project: Considerations towards a Cloud-based Internetworking Paradigm

Kostas Katsalis, Thanasis Korakis
University of Thessaly, Greece
kkatsalis,korakis@uth.gr

Giada Landi, Giacomo Bernini
Nextworks,Pisa, Italy
g.landi,g.bernini@nextworks.it

Bijan R. Rofoee, Shuping Peng
University of Bristol, UK
bijan.rofoee,shuping.peng@bristol.ac.uk

Markos Anastasopoulos, Anna Tzanakaki
Athens Information Technology, Greece
manast,atza@ait.gr

Dora Christofi, Michael Georgiades
PrimeTel PLC, Cyprus
dorac,michaelg@prime-tel.com

Renaud Larsen
Juniper Networks Inc.,Belgium
renaudlarsen@juniper.net

Jordi Ferrer Riera, Eduard Escalona, Joan A. Garcia-Espin
Fundació i2CAT, Spain
jordi.ferrer,eduard.escalona,joan.antoni.garcia@i2cat.net

***Abstract*—Although cloud computing and the Software Defined Network (SDN) framework are fundamentally changing the way we think about network services, multi-domain and multi-technology problems are not sufficiently investigated. These multi-domain, end-to-end problems concern communication paths that span from the wireless access and the wireless backhaul networks to the IT resources through optical networks. In this paper we present the CONTENT project approach to network and infrastructure virtualization over heterogeneous, wireless and metro optical networks, that can be used to provide end-to-end cloud services. The project goal is to drive innovation across multi-technology infrastructures and allow ICT to be delivered and consumed as a service by Virtual Network Operators. The communication mechanics between wireless and optical domains and the physical layer abstractions of a CONTENT Virtual Network are presented and the relation of the proposed approach with the SDN framework is investigated.**

Keywords*—Network virtualization, inter-domain networking, wireless-optical networks, cloud computing

I. Introduction

The mass adoption of the Internet is continually driving the technology in finding ways to face age-old obstacles, like user mobility, and physical infrastructure and network limitations. Network and infrastructure virtualization technologies give the opportunity to physical infrastructure providers to add value to their existing business propositions and generate an interesting new turnover, mainly because of the way they are able to handle these limitations [1]. To address the requirements of future virtual network operators, a combination of knowledge obtained by the recent advances in virtualization and Software Defined Networking (SDN) technology must be investigated, while in order to build and deploy efficient cloud based services, we must depart from models that consider intangible multi-domain end-to-end settings.

In this work we present the CONTENT project approach [2], an EU funded effort, for interconnecting geographically distributed computational resources through ubiquitous converged virtual network infrastructures. CONTENT project aims to deliver an SDN-enabled inter-domain cloud networking platform, engineered with a bottom up approach to meet the requirements of an end-to-end cloud computing environment. Thanks to the approach devised, the successful Mobile Virtual Network Operator (MVNO) model that is used to provide wireless services over the physical network provider, can be extended to a Mobile-Optical Virtual Network Operator (MOVNO) model. In this model a virtual operator, besides the wireless provider "sliced" resources, will be able to use virtualized resources on the optical metro network to seamlessly interface with virtualized resources in the Data Center.

The key motivation behind the CONTENT initiative is that multi-domain and multi-technology problems are not sufficiently investigated. Active research is striving to consolidate frameworks that, with respect to SDN philosophy, will be able to provide performance guarantees, not per segment but end-to-end. New protocols and frameworks, like OpenFlow [3] and OpenContrail [4], are emerging solutions, which seem promising to address these issues. However, global standards for Control Plane to Control Plane communications over diverse technologies are still missing, while the technology to support QoS and performance guarantees in SDN networks is still in an immature state. A main component of the CONTENT architecture is the communication framework that bridges the Control Planes residing in different technology domains. More specifically, it investigates how TSON (Time Shared Optical Network) virtualized optical networks [5], that utilize GMPLS Control Plane mechanics, will be able to communicate with virtualized converged WiFi/LTE networks, in such a way that traffic isolation between the virtual wireless networks is preserved and QoS guarantees are provided end-to-end.

In a nutshell, the CONTENT project is about building an end-to-end virtualization framework that allows for seamless orchestrated on-demand service provisioning across heterogeneous technology domains. The proposed architecture is based on the integration of metro optical-wireless access domains used to interconnect the end users with virtualized computational resources. We propose to leverage a solution

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

that uses standards-based protocols and provides components for network virtualization, SDN-based network control, virtual routing functionalities and will be open to accept analytics engine and Northbound APIs, for enhanced network applications and cloud service orchestration. Two testbeds, the wireless NITOS testbed [6] and the optical TSON [5][7] will be the building blocks of a reference CONTENT testbed used for policies and framework evaluation and for proof of principle demonstrations.

This paper is organized as follows: in section II a description of the the envisioned architecture, the motivation behind CONTENT and related work is made; in section III the end-to-end virtualization strategy is presented; in section IV issues on multi-domain networking and the CONTENT testbed are described; and section VI concludes this work and presents the project's future plans.

II. Vision, Motivation and Related Work

An end-to-end infrastructure facilitates the interconnection of Data Centers (DCs) with fixed and mobile end users through a heterogeneous network, integrating optical metro and wireless access network technologies. The CONTENT project proposes an architecture design that integrates an advanced optical network solution, offering fine (sub-wavelength) switching granularity, with wireless WiFi and LTE access network technologies. To support the IaaS paradigm as well as the diverse and deterministic QoS needs of future Cloud and mobile Cloud services, the proposed architecture adopts the concept of virtualization across all technology domains (see Figure 1) and relies on a common DC infrastructure fully converged with the broadband wireless access and the metro optical network.

CONTENT defines a new stakeholder, the MOVNO, who will offer to the operators the ability to exploit new business opportunities. The MOVNO comprehends converged virtualization of WiFi, LTE, optical metro and IT resources for providing, voice, data and IT services to its customers. The MOVNOs can use the infrastructure of another provider, minimizing their technology systems to billing and customer care, content delivery management and business support systems. The end-user can experience a better quality of service in terms of improved reliability, availability and serviceability, whereas physical infrastructure providers can gain significant benefits through improved resource utilization and energy efficiency, faster service provisioning times, greater elasticity, scalability and reliability of the overall solution. In this context, the adoption of cross-domain and cross-technology virtualization facilitates migration towards a fully converged infrastructure.

The proposed architecture identifies three main actors, the Physical Infrastructure Provider (PIP), the Virtual Operator (VO) and the Service Provider (SP). The PIP owns the physical infrastructure and provides virtual infrastructure resources on top of its physical resources to the VO. The VO in turn has the ability to provide to the SP options for providing new services to its customers. The MOVNO will be able to reduce their CAPEX costs and allow a large percentage of population to be covered by the deployed infrastructure. This will also allow MOVNOs to spend more money on enhancing service provisioning. PIPs will have the opportunity to enhance their

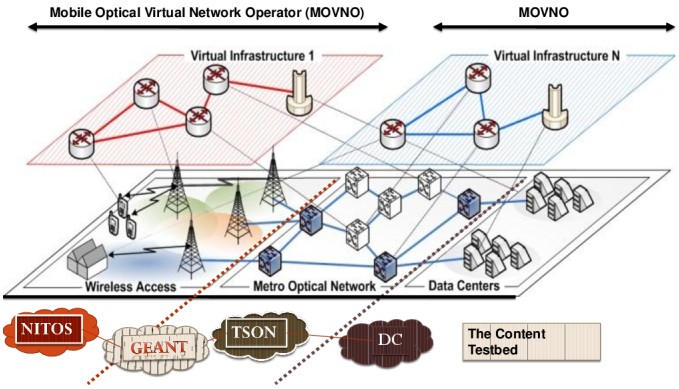


Figure 1. MOVNO and the Content testbed

infrastructure and generate revenues from MOVNOs including custom made solutions; the VO will be in position to provide end-to-end cloud services to its customers with guaranteed wireless broadband connectivity whilst being able to monitor and manage its own virtual network and cloud resources; and the SP will be able to offer to its customers cloud-based application services as part of the more general broadband network access services.

A. Related Work

Virtualization across multi-technology domains has gained significant attention over the last few years and several solutions already exist [8]. Typical examples include the pipe and the hose Virtual Private Network models [9]. Their main difference is that in the pipe model the bandwidth requirement between any two endpoints must be accurately known in advance, whereas the hose-model requires only the knowledge of the ingress and egress traffic at each endpoint having the advantages of ease of specification, flexibility and multiplexing gain [10].

Other similar research efforts have been focused on embedding Virtual Infrastructures (VI) over converged optical data center networks [11], [13] or across multiple substrate networks [8],[14]. A key assumption is that the details of each domain are not communicated to the other domains. Therefore, each infrastructure provider has to embed a particular segment of the VI without any knowledge of how the remaining VIs have already been mapped or will be mapped [8] (referred to as the PolyViNE) with the objective to unilaterally maximize its payoff. A typical example includes the case where an infrastructure provider wishes to optimally allocate its resources without considering the impact of its decision on the other domains.

Although existing approaches (e.g. PolyViNE) target relatively straightforward solutions from an implementation perspective, they may lead to inability in meeting some cloud and mobile cloud service requirements (e.g. end-to-end latency) and inefficiencies in terms of resource requirements, energy consumption etc. [15]. Various aspects of SDN related to transport networking, and issues to be addressed, are presented in [16]. Existing mobile cloud computing solutions allow mobile devices to access the required resources by accessing a nearby resource-rich cloudlet, rather than relying on a distant cloud [17]. In order to satisfy the low-latency requirements of several content-rich mobile cloud computing services such as

high definition video streaming, online gaming and real time language translation [18], one-hop, high-bandwidth wireless access to the cloudlet is required. In the case where a cloudlet is not nearby available, traffic is offloaded to a distant cloud such as Amazons Private Cloud, GoGrid [19] or Flexigrid [20]. However, the lack of service differentiation mechanisms for mobile and fixed cloud traffic across the various network segments involved, the varying degrees of latency at each technology domain and the lack of global optimization tools in the infrastructure management and service provisioning make the current solutions inefficient.

III. Content End to End Virtualization approach

In this section we present the challenges in virtualizing the optical and the wireless segments and a short description of the virtualization technologies available. Then we classify the virtualization approaches as a function of the layer where the virtualization happens followed by a discussion on the proposed scheme.

A. Challenges in wireless/optical segment virtualization

Optical Domain Virtualization: Optical network virtualization is defined as the composition of isolated virtual optical networks (VONs) simultaneously coexisting over shared optical physical network infrastructure. Within a VON, the granularity of virtual links is inherited from the switching capabilities supported by the physical devices (e.g. sub-wavelength, wavelength, waveband, or fiber). The two most important characteristics of VONs are isolation and coexistence.

Unlike Layer 2 and Layer 3, optical network resources and transport formats are characterized by their analogue nature. Optical layer constraints such as wavelength/spectrum continuity and physical layer impairments (PLIs) are examples of optical network differentiators. Hence, optical virtualization paradigms should take into account the physical characteristics of optical networks [21]. Linear and non-linear impairments in optical communication systems (noise, dispersion, or cross-talk) can affect the reach and quality of the optical signals, and hence limit the scale and flexibility of virtualization in optical networks. Apart of the wavelength continuity, time slice continuity should be also considered for the time-shared sub lambda networks. This means that if the burst of data is transmitted on a specific period of time, that period (considering the delays) should remain untouched by other signals otherwise contention occurs.

Virtualization techniques for different types of optical networks of different technologies and granularities may vary, according to the node and link characteristics in a particular network. For example, in a wavelength switched network with optical cross connects (OXCs) and Reconfigurable Optical Add-Drop Multiplexors (ROADMs) virtualization approaches are different from networks with sub-wavelength granularity of switching and control. Initial work on optical routers and switches was performed by Qiao [22] in the context of the Optical Burst Switching (OBS) proposal. This effort has lead to some very interesting research and prototype developments, such as an OBS ring testbed in [23], the OPST network solution by [24] and TSON sub wavelength switching [5]. Sub-wavelength switching also is enabled by using Optical Orthogonal Frequency Division Multiplex (OOFDMA) solutions.

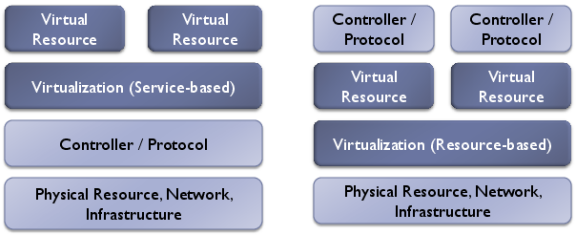


Figure 2. Service Virtualization (left) and Resource Virtualization (right)

The OOFDM sub-wavelength solution offers great flexibility in data rates and modulation levels, however the technology is still in its infancy to be properly deployed in a network environment.

Wireless Domain Virtualization: A number of challenges exist in the wireless domain that do not allow for a straightforward adoption of a specific virtualization strategy, if QoS and service differentiation must be provided, besides traffic isolation between the virtual networks. Like optical network that suffer from impairments and nonlinearity, wireless networks suffer from time varying channel characteristics. There is no well established theory for computing the channel capacity, or providing simple bounds to the maximum information rate based only on the channel impulse response. In addition, unlike wired networks, co-existence of slices with bandwidth-based and resource-based reservations is not straightforward, since the bandwidth achieved by a slice from a given amount of resources varies with the channel quality of the users, the number of users that are sharing the medium and the flow scheduling policies. Other concerning issues include non-location specific addressing, Association, Authentication (AuthN) and synchronization complexity and channel configuration [25].

In the wireless medium, radio resources can be shared and thus virtualized in different ways such as in time, space, and frequency. Splitting and slicing techniques of the wireless medium may use specific time slots (Time Division Multiplexing), space (Space Division Multiplexing), frequencies (Frequency Division Multiplexing) or combinations[26]. Ideas and techniques of virtualizing 802.11 access points are examined in [27] and [28], where we note that in [28] the proposed solution utilizes programmable data plane technologies to guarantee flow differentiation. Work on Base Station (LTE/WiMax) virtualization is presented in [29] and [30]. We also note that OpenFlow [3] and flow-based control offers the potential for end-to-end virtualization in the flow level. For example, OpenFlow technology can be used for slicing wireless backhaul networks and offer communication between controllers that reside in different domains.

1) Virtualization models: In [31], the authors provide a classification of the virtualization approaches as a function of the layer where the virtualization happens. Additionally, in [32], the authors focus on a theoretical approach. They provide a technical classification of the virtualization algorithms presents in the literature as a function of the different parameters optimized by the allocation algorithm, without considering specific technologies of the scenario where the algorithm is applied. From the CONTENT project perspective, and in order to align the virtualization to the software-defined networking requirements, we provide a generic definition of the different types of network virtualization, using as a basis the initial

virtualization classification that has been done within the IT realm, with the virtual machines. Therefore, we distinguish between two different types of virtualization, resource-based virtualization (bare-metal), and service-based virtualization. Figure 2 depicts both approaches and their relationship with the SDN control planes.

Resource-based virtualization (or resource virtualization) becomes the most suitable option in order to have the finest feasible granularity in any case, as well as the maximum level of flexibility to control the different virtual resources. However, in terms of implementation, the complexity of this approach is higher, since it needs to completely control the physical resource itself. On the other hand, service-based virtualization loses part of the flexibility, since the virtualization system is located on top of the network service. As an example, the L3VPN, represents a case of service-based virtualization, since the virtualization happens of top of the IP protocol. Granularity is also limited for this approach, since it depends on the northbound interface of the service itself.

B. *Virtualization in CONTENT*

Applying different virtualization schemes on the different technological segments has different effects. In CONTENT a cross-domain virtualization approach is adopted that aims to overcome these difficulties in order to provision virtual infrastructures composed of resources coming from multiple heterogeneous domains. Based on the functional requirements, for each technology segment there is a specific virtualization system that follows the resource-based approach. On top of each domain-specific system, there exists the cross-domain component, which holds a holistic view of the infrastructure and which is responsible to perform the virtualization over the different domains.

In CONTENT, we adopt TSON as the underlying optical network technology. Its unique features bring more challenges on virtualization. The bursty nature of the signals is more vulnerable to the noise of optical components, such as Erbium Doped Fibre Amplifier (EDFA) and transceivers, and also other impairments, which limits the reach and OSNR. Besides the requirements on the wavelength/spectrum continuity, TSON also requires the time slice continuity. During the process of TSON virtualization, available time slots should also be abstracted and exposed as virtual resources. In TSON, global synchronization among the network nodes is needed due to lack of buffers. Therefore, in each virtual slice the synchronization should also be taken into account and guaranteed. Moreover, in order to flexibly accommodate with user's diverse demands, the virtualization of transceivers in the sub-wavelength network is essential and also a big challenge considering the limitations on the hardware.

In the wireless domain, programmable data planes, software routers and software switching technologies have already been proposed in the literature to perform network slicing or provide other network functionalities. Such as technology is the Click Modular router [33]. With respect to the SDN framework where data, control, management and service planes must be clearly separated, in order to provide a slicing mechanism over a converged WiFi/LTE network, a combination of technologies like OpenFlow in combination with programmable

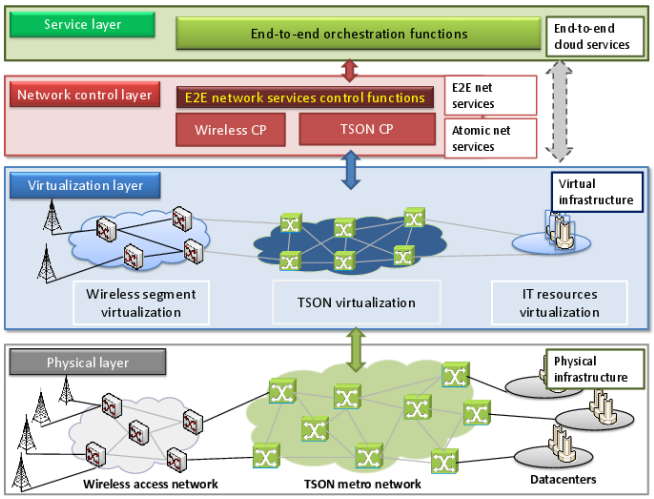


Figure 3. CONTENT Layering Structure

data plane technologies, is adopted. These technologies can be used to perform traffic shaping and slice isolation from the edge network to the wireless backhaul/access network. We note that in works like [28], programmable data plane technologies, besides providing potential technologies for traffic shaping and policy enforcement in the edge network, they are also used to virtualize a single Access Point itself. The combination of OpenFlow (or Openflow like) technologies with programmable data planes gives us the ability to perform flow isolation, traffic shaping and QoS policies, for both the ingress traffic that comes from the optical network and the egress traffic flows that span from the wireless access networks through the wireless backhaul to the wireless edge.

C. *CONTENT Control Plane*

The CONTENT Control Plane (CP) operates on top of the hybrid virtual infrastructures composed of wireless and TSON domains and virtual data centers. Following the SDN paradigm and the four planes approach (data, virtualization, control & management, service layers) it is fully decoupled from the physical layer and interacts with the underlying virtual infrastructure through open and vendor-independent interfaces.

The CONTENT control plane relies on an SDN-like approach with functional entities operating as controllers of the virtualized networks. These entities provide management, control and analytics procedures through three main internal functions: *Configuration functions* responsible for translating the high-level data model into a lower level form suitable for interacting with network elements; *Control functions* responsible for propagating this low level state to and from network elements; *Measurement functions* responsible for capturing real-time data from network elements, abstracting it and presenting it in a form suitable for administration purposes or to consume by applications.

The CONTENT CP is structured in two hierarchical layers. The lower layer is related to the control of technology-specific domains through dedicated CPs responsible for providing connectivity within the wireless and the TSON networks respectively (see also Figure 3). The upper layer uses the functionalities exposed by wireless and TSON CPs in order to provide end-to-end and multi-layer network services to

interconnect the data centres and the mobile users, as required by the cloud orchestration functions deployed on top of the architecture.

In this context the wireless CP handles the connectivity in the wireless domain and it manages the intra- and inter-technology handovers. Also it is responsible for the configuration of the Ethernet backhaul network, which is composed of programmable switches, through an SDN controller. On the metro side, the TSON CP implements all the basic functionalities to provide intra-domain network services with flexible capacity (through the support of the sub-wavelength bandwidth granularity), dynamicity and resiliency guarantees. Both the wireless and the TSON CPs expose at the north-bound side an interface to access the basic service primitives implemented in each domain, so that enhanced network control applications with end-to-end scope can be easily developed on top of them.

These applications will operate on the entire virtual infrastructure through the abstracted services exposed by the wireless and TSON CP north-bound interface, without dealing with the specific network-oriented semantics, internal technologies and protocols implemented in each domain. These end-to-end network applications manage the cross-domain connectivity taking into account the requirements of the cloud services in mobile environments. As an example, they can dynamically modify the resource allocation in the two network segments depending on different parameters or network conditions, such as the real-time traffic load in the metro between the data centers, the network utilization, the scheduled cloud services, the characteristics of the traffic generated by the mobile users.

IV. The CONTENT Testbed

The CONTENT testbed aims to evaluate the integrated solution and the proposed CONTENT framework, though performance metrics that quantify the efficiency of ubiquitous fast broadband access, the flow isolation, the QoS and QoE that the end users enjoy and the signaling overhead. As shown in Figure 4 the wireless segment of the CONTENT testbed is the NITOS testbed [6] and the Optical segment is the TSON testbed [5]. Custom virtualized infrastructure or cloud based resources will be used to support the necessary back-end DC infrastructure.

A. Description of the CONTENT Testbed

TSON, is designed and implemented as a novel time multiplexing metro network solution, which uses multiple wavelength for transportation of time multiplexed data, while offering dynamic connectivity with fine granularity of bandwidth in a contention-less manner. TSON data plane is augmented with an extended GMPLS control plane for signaling and reservation of network resources for establishing lightpaths. At the heart of the control plane, two centralized routing and resource allocation drivers of Path Computation Element (PCE) and Sub Lambda Assignment Engine (SLAE) manage the network resources for the incoming lightpath requests and the existing ones. TSON uses high performance FPGA platforms of XILINX V6HT for high speed 10GE frame processing and EOC conversion of the frames into bursts of traffic at the edge nodes, whilst TSON core nodes use the high speed FPGA platforms to transparently bypass the traffic,

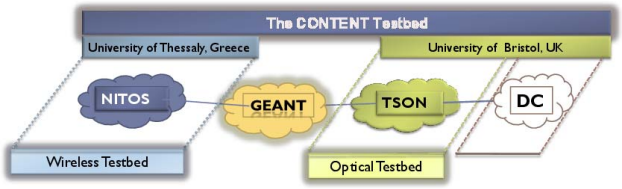


Figure 4. The CONTENT Testbed

via controlled advanced PLZT based fast switches (10ns). Therefore the TSON data plane is capable of offering up to 10 Gb/s data rate over each wavelength with the granularity of 100 Mb/s. The interconnection and communication between the TSON data plane nodes and control plane elements take place through web service interfaces; this approach decouples the functionalities of the two layers from each other, therefore enhances the manageability and extensibility of the network.

The wireless segment of the CONTENT testbed architecture is provided by an heterogeneous wireless testbed called NITOS, offered by University of Thessaly (UTH) in Greece. NITOS testbed is a FIRE facility and the current deployment includes more than 50 WiFi enabled nodes, a wide-range WiMAX/LTE Base Station, a WiMAX/LTE experimental network and an extended wireless sensor network. Furthermore, wireless nodes feature USRP boards that are software defined radio systems. Through these, a user can build from scratch PHY layer interfaces as well as the needed MAC layer. NITOS is based on open-source software and drivers allowing experimenters to modify any layer of the network protocol stack, while the control and management of the testbed is based on the OMF/OML framework. In addition, NITOS offers experimentation capabilities using fully programmable OpenFlow switches and allows for the design, build and evaluation of SDN scenarios that span both the wireless access and the wireless backhaul network. This is very important for the aims of the CONTENT project where a convergent WiFi/LTE network is examined.

B. Physical interconnection between the different testbeds

The physical interconnection between the wireless testbeds (NITOS in Volos, Greece) and the optical testbed (TSON in Bristol, UK) is made through the GEANT network [34]. VLAN-based connections are currently used, while the GEANT network robustness guarantees the delivery of speed and services to enable greater collaboration. CONTENT project will exploit all the advances and updates on the GEANT network and the services offered, like for example the Bandwidth on Demand service (BoD). The involved NRENs (GRNET in Greece and JANET in UK), are already carrying out pilots for the introduction of the BoD service. With the BoD service, multi-domain provisioning is offered to users that can make reservations in advance, or provision in real-time point-to-point circuits with the capacity needed.

The interconnection of the two technology domains includes interfacing of the testbeds both at the physical layer and also at the logical and software layer. For the physical interconnections, the two technology domains need to support common protocols to enable seamless traffic interchange between them. Ethernet 802.1Q VLAN tagging and techniques

like QnQ tagging and EVPN have been considered to interface the two networks, which enable inter domain traffic transport, are compatible with other Ethernet based technologies (thus enhancing the scalability of the network), and facilitate the establishment of end-to-end virtual networks as services.

For connecting the two testbeds (TSON and NITOS), data rate characteristics of both testbeds are considered. TSON as the metro backhaul solution provides from 100 Mbps up to 20 Gbps connectivity for a variable number of LTE/WiFi cells, depending on the rates required at each access point and the level of aggregation to be performed. The rate matching between the two domains can cause inefficiency in using the available bandwidth. However TSON as the sub-wavelength switching technology with fine data rate granularities and dynamic setup and tear-down of connections can adopt to the requirements in the access quite swiftly.

The control of the physical setup and interconnection of the two networks needs to be addressed in the control layer. This way the allocation/optimization mechanisms will have global understanding of the resources, can have access to these resources (for e.g. nodes, ports, spectral, time units) and so create optimum end-to-end slices of the network. In this regard, the TSON system exposes the information in its edge nodes out to the controller through a web service interface, while NITOS does it via both a web based interface and SFA/mySlice [35]. This enables the controller to take advantage of the programmability and flexibility of the TSON and NITOS technologies to establish optimum paths stretching from users in wireless access areas to the distributed data centers through the metro/core network.

V. Concluding remark: CONTENT project and the SDN framework

The CONTENT layered architecture adopts the SDN paradigm since the control functions are separated from the physical devices and operate on top of programmable entities through open interfaces. The SDN approach is reflected in several architectural choices, from the introduction of virtualization mechanisms between the physical infrastructure and the control plane, up to the internal design of the control plane itself.

The virtualization layer in CONTENT generates isolated virtual infrastructures and provides an abstracted view of the different domains, hiding the vendor-dependent details while exposing the full technology capabilities through a unified interface. This design is aligned with the SDN approach, where control plane functions operate on top of programmable devices, in this specific case represented by the virtual network resources (TSON nodes in the metro networks, base stations and backhaul L2 switches in the wireless segment). The interfaces enabling the control of the virtual nodes follow the same design principles of SDN protocols, like OpenFlow. In particular, they are conceived to support synchronization, configuration and monitoring actions through commands and asynchronous notifications expressed in an open language, independent from the specific protocols and interfaces implemented on the corresponding physical devices.

At the control plane level, on top of the virtual infrastructures, the wireless and TSON virtual domains are managed through dedicated lower-layer control frameworks: the wireless and the TSON CPs. These are customized to deal with the specific technology constraints (e.g. management of sub-wavelength switching capability in the TSON metro network) and implement a set of basic intra-domain mechanisms, like connectivity setup and monitoring. With reference to SDN architectures, the local CPs act as a sort of macro SDN controllers responsible of the basic network functions in each segment. On the other hand, all the end-to-end enhanced functionalities for on-demand provisioning, maintenance, resiliency, monitoring and dynamic re-configuration of network connectivity in support of mobile cloud services are delegated to upper layer network applications.

It is clear that this internal organization of the CONTENT CP is strongly based on the programmability concept, that represents one of the pillars of SDN architectures. In the CONTENT architecture, specialized network applications implement cloud-oriented functions over generic controllers offering simpler network service primitives. In other terms, the controllers provide a further level of service abstraction on top of the resource virtualization layer, while the dynamic programmability of the network is handled through the upper layer applications, tightly integrated with the orchestration functions at the cloud service layer.

VI. Conclusions & Future work

This paper presented the CONTENT project approach for building virtual networks over converged multi-domain network infrastructures. The proposed architecture is based on a sample environment with distributed data center sites interconnected through a TSON metro network, while a hybrid wireless network, based on LTE and Wi-Fi technologies, provides the access for mobile cloud users. The joint orchestration and cooperation between cloud and network services through an enhanced cloud-to-network interface, will allow for the strict requirements imposed by cloud services to be met, in terms of dynamicity, resiliency, end-to-end QoS and performance.

The CONTENT testbed has been designed to validate the proposed architectural solution using relevant cloud-based applications. Trials and experimental tests are planned during the next year of the project, to evaluate the performance of different types of cloud services on top of a CONTENT-enabled virtual infrastructure. A “Mobile broadband services by MOVNO” scenario will be the pilot use case for which the CONTENT testbed will provide the virtual infrastructure used for the provisioning of cloud services.

The project consortium closely follows the advancements in the cloud computing technology, in the SDN framework and the Network Functions Virtualizaton (NFV) that are related to the project’s goals and can potentially be exploited by the proposed framework.

Acknowledgements

This work has been funded by the EU Project 318514 “Convergence of wireless Optical Network and IT rEsourcesiN support of cloud services” (CONTENT). We thank all our project partners for their valuable contributions.

REFERENCES

[1] M. Armbrust, et al., "A view of cloud computing". Commun. ACM 53, 4 (April 2010)

[2] CONTENT Project: http://content-fp7.eu/

[3] OpenFlow Switch Specification, version 1.3.1, Open Networking Foundation, September 2012

[4] The OpenContrail Project: www.opencontrail.org/

[5] G. Zervas, J. Triay, N. Amaya, Y. Qin, C. Cervell-Pastor, and D.Simeonidou, Time Shared Optical Network (TSON): a novel metro architecture for flexible multi-granular services, Optics Express, 19 (26).

[6] NITOS test-bed, http://nitlab.inf.uth.gr/NITlab/

[7] TSON testbed http://www.bris.ac.uk/engineering/research/pho/hpn/

[8] M. Chowdhury, F. Samuel, R. Boutaba, PolyViNE: policy-based virtual network embedding across multiple domains, in Proc. of ACM SIGCOMM, workshop on Virtualized infrastructure systems and architectures (VISA '10), 2010.

[9] J. Chu, Chin-Tau Lea, New architecture and algorithms for fast construction of hose-model VPNs, IEEE/ACM Trans. Netw. 16, 3, 670-679, June 2008.

[10] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. V. der Merwe, A flexible model for resource management in virtual private networks, in proc. ACM Sigcomm , San Diego, California, USA, August 1999.

[11] D.A. Schupke, Multi-Layer and Multi-Domain Resilience in Optical Networks, Proceedings of the IEEE, vol. 100, no. 5, May 2012.

[12] A. Tzanakaki et. al., Planning of Dynamic Virtual Optical Cloud Infrastructures: The GEYSERS approach, IEEE Communications Magazine, Special Issue on Advances in Network Planning, Jan. 2014.

[13] K. Georgakilas, A.Tzanakaki, M.P. Anastasopoulos and Jens M. Pedersen, Converged Optical Network and Data Center Virtual Infrastructure Planning, IEEE/OSA Journal of Optical Communications and Networking, Vol. 4, No. 9, pp.681691, Sep.2012.

[14] D. Dietrich, A. Rizk, P. Papadimitriou, AutoEmbed: Automated Multi-Provider Virtual Network Embedding, Demo, ACM SIGCOMM 2013

[15] A. Tzanakaki, M.P. Anastasopoulos, G. Zervas, B. Rofoee, R. Nejabati and D. Simeonidou, Virtualization of Heterogeneous Wireless-Optical Network and IT infrastructures in support of Cloud and Mobile Cloud Services, IEEE Communications Magazine, Aug. 2013.

[16] Mcdysan, D., "Software defined networking opportunities for transport", Communications Magazine, IEEE , vol.51, no.3, pp.28,31, March 2013.

[17] M. Satyanarayanan, P. Bahl, R.Caceres and N. Davies, The Case for VM-Based Cloudlets in Mobile Computing. IEEE Pervasive Computing 8 (4), pp.14-23, Oct. 2009.

[18] K. Mun, Mobile Cloud Computing Challenges, TechZine Magazine, http://www2.alcatel-lucent.com/techzine/mobile-cloud-computing-challenges/

[19] http://www.gogrid.com/

[20] http://www.flexiscale.com/

[21] S. Peng, R. Nejabati, D. Simeonidou, Impairment-Aware Optical NetworkVirtualization in Single-Line-Rate andMixed-Line-Rate WDM Networks, IEEE/OSA Journal ofOptical Communications and Networking, 5(4), pp. 283-293, April 2013.

[22] C. Qiao, M.Yoo, Optical burst switching (OBS) - a new paradigm for an optical Internet, Journal of High Speed Networks - Special issue on optical networking

[23] Ning Deng; et al "A novel optical burst ring network with optical-layer aggregation and flexible bandwidth provisioning," Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference , vol., no., pp.1-3, 6-10 March 2011

[24] Verismaivx 8000, Optical Packet Switch and Transport Intune Networks, 2011

[25] Hong-jiang Lei, et al., "Survey of multi-channel MAC protocols for IEEE 802.11-based wireless Mesh networks", The Journal of China Universities of Posts and Telecommunications, v.18, Issue 2, April 2011,

[26] Aljabari G., Eren E., Virtualization of Wireless LAN Infrastructures, thThe 6 IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2011

[27] Aljabari G., Eren E., "Virtual WLAN: Extension of Wireless Networking into Virtualized Environments", International Journal of Computing Research Institute of Intelligent Computer Systems, 2010

[28] Bhanage, G.; Vete, D.; Seskar, I.; Raychaudhuri, D.; , SplitAP: Leveraging Wireless Network Virtualization for Flexible Sharing of WLANs, IEEE GLOBECOM, 2010

[29] R Kokku, R Mahindra, H Zhan et al,Remote Virtualization of a Cellular Basestation, US Patent , US 2012 0002620 A1

[30] Bhanage G., Seskar I., and Raychaudhuri D, "A virtualization architecture for mobile WiMAX networks." SIGMOBILE Mob. Comput. Commun. Rev. 15, 4 (March 2012)

[31] Chowdhury, N.M.M.K.; Boutaba, R., "Network virtualization: state of the art and research challenges," Communications Magazine, IEEE , vol.47, no.7, pp.20,26, July 2009

[32] Fischer, A.; Botero, J.; Beck, M.; De Meer, H.; Hesselbach, X., "Virtual Network Embedding A Survey," Communications Surveys & Tutorials, IEEE , vol.PP, no.99, pp.1,19 ,2013

[33] Kohler, Eddie, et al. "The Click modular router." ACM Transactions on Computer Systems (TOCS) 18.3 (2000): 263-297.

[34] GEANT network, http://www.geant.net

[35] http://sfawrap.info/

IEEE Software Defined Network Initiative

A Proposal

Mehmet Ulema, Manhattan College, USA
Nirant Amogh, Huawei, India
Raouf Boutaba, University of Waterloo, Canada
Cagatay Buyukkoc, AT&T Labs, USA
Alex Clemm, Cisco, USA
Jiang (Linda) Xie, University of North Carolina-Charlotte, USA
Mehmet C. Vuran, University of Nebraska-Lincoln, USA
Antonio Manzalini, Telecom Italia, Italy
Roberto Saracco, EIT ICTLABS, Italy

***Abstract* — This paper outlines a proposal for setting up an IEEE initiative on software defined networks (SDNs) to facilitate professional and academic exchange of SDN-related ideas, research, and development. The proposal is a result of an intensive effort of a team consisting of the authors. After a comprehensive gap analysis, gaps and key opportunities were identified. Finally, a specific set of components along with schedule and financial consideration were proposed in the areas of publications, conferences, standards, education, certification, and publicity.**

Keywords—SDN, Software Defined Networks, Virtualization, IEEE, Abstraction.

I. INTRODUCTION

Software Defined Networking (SDN) has become one of the most active research areas in networking and communications today. There is a flurry of activities in the industry and academia and yet, there is no major professional organization leading the way in a substantial and comprehensive way. This prompted IEEE Communications Society and IEEE Future Directions Committee to set up a task force to propose an IEEE initiative to make the IEEE as "the place to go" for any SDN related activities such as publications, conferences, and standardization.

The authors of this paper formed a task force to prepare a proposal, which is the subject of this paper. The team members shared the load, each taking on a specific area of the proposal. The team met on a weekly basis to discuss the findings and future directions. The methodology used by team was straightforward: perform a gap analysis by reviewing current activities, different interpretations of SDN, and related technologies such as cloud computing, and identify overlaps. Then, identify the gaps, open areas, and key opportunities in general. Finally, identify opportunities for IEEE in the areas of publications, conferences, standards, education, certification, etc.

It was quickly determined that there are different views on SDN and the well-known interpretation [1,2,3] is rather limited! Despite the existing activities, we emphasize that the field is just emerging and will grow quickly. The related technologies such as Cloud Computing, Cognitive Radio Networking, are not threat to SDN. On the contrary, SDN can be a strong enabler for other emerging technologies due to its significant flexibility and overarching scope.

An important finding of the team is that there is a need to broaden the vision of SDN to cover all components end-to-end, and to include all types of technologies (wireless, optical, etc.) involved. This broader vision is especially important for IEEE to fill the gap and provide a larger scope for its products in publications and conferences.

The primary objective of the proposal is to establish an IEEE-wide initiative with a comprehensive program based on a broad vision of SDN. The proposal emphasizes a prioritized launch of products and services to utilize resources efficiently. Accordingly, the mission of the proposal includes keeping up, leading, and co-branding as much as possible with other IEEE and non-IEEE activities. The highlighted individual areas are as follows:

- An IEEE Magazine on SDN and a Journal on SDN
- A major conference and regional and topical conferences
- A Standards Committee on SDN to drive standardization
- Tutorials, e-courses, training courses, and webinars
- Certification programs for people, devices, and testbeds
- Web Portal for links to IEEE SDN programs, repository, communications

The rest of the paper provides a detailed vision of the future of SDN. Section II provides a summary of the analyses and opportunities identified by the team. In Section III, the individual components of the proposal are presented. The management issues related to the initiative are discussed in Section IV. Finally we conclude the paper in Section V.

II. ANALYSES AND OPPORTUNITIES

SDN has become widely accepted as a new paradigm in designing network(s) and their components. Technology and market trends have enabled an open framework to create modular networks that can be functionally separated, to achieve user, application and service awareness; manageability; and controllability. Ability to use abstractions allows programmability and open innovation and separation of basic hardware from various control and management functions.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

SDN started with a narrow scope of separating data and control planes, but it is evolving [1, 2, 3]. SDN can be used in generalized areas of networking (User Equipment (UE), Radio, Transport, Application) abstractions that allow management [17], cloud, cognitive, and smart grid [4, 5].

SDN's key characteristics include: Modularization of Hardware, Software, and Functions, Multi-Level Virtualization, Multi-Tenancy, Centralization of Control, Programmability, and Open Innovation.

There are numerous SDN related events under way, including standardization, conferences, and industry alliances. OpenFlow is the most widely known standardization activity in this area. It is an interface specification between a path controller and a data forwarding engine. In addition to Open Networking Foundation (ONF) [6, 7] (the defacto home of the OpenFlow specs.), Internet Engineering Task Force (IETF) [8, 19], Open Daylight [9], Distributed Management Task Force (DMTF) [10], Open Management Group (OMG) [11], Association of Computing Machinery (ACM), European Telecommunications Standards Institute (ETSI) (most active in Network Functions Virtualization (NFV) [12]), Optical Internetworking Forum (OIF): [13], as well as IEEE are some of the other organizations that have some SDN activities.

There are several technologies that seem to relate to SDN, but it is clear that SDN is not a replacement of those, but rather a facilitator to make the implementation of the those technologies easier. Briefly, Cloud Computing refers to the use of computing resources delivered as a service over a network. SDN framework is increasingly [14] adapted in cloud computing/networking [17]. Cognitive Networks refer to those networks with cognitive processes that can learn from the past and use this knowledge to improve the decisions in the future. For Cognitive Networks, SDN can provide mechanisms to support self-learning and self-adaptation and the autonomic network management needed by cognitive networks [15]. Similar arguments can be extended to Service Oriented Architectures (SOA), Network Functions Virtualization (NfV), Application Programming Interfaces (API), etc.

Our analyses reveal that there are indeed a large number of gaps; that SDN provides opportunities not only for IEEE, but the whole academic and industrial world to explore. These include creating and using abstractions, Information Centric networking (ICN), new programming capabilities/models, SDN for Network Management [17], SDN for Self Organization Networks (SON) [20], SDN for IEEE created technologies (e.g., Ethernet, WiFi). SDN Support for Open Application Innovation and Business Models, as well migration between different SDNs and migration to SDN from legacy networks are also important opportunities [21].

Furthermore, SDN will impact networks evolution both in the core and in the edge (i.e., access, distribution) segments. The edge is arguably the network segment where SDN innovation will be exploited more disruptively. On the other hand, the edge will become smart and much differentiated. Software will become a powerful enabler to create dynamic interoperability of multiple pools of processing, storage and link resources (here and now). There will be a change of paradigm in the business: from centrally managed networks to application-driven networks, from Quality of Service (QoS) to Quality of Applications.

Telecommunications sector is going through a very significant change, for several reasons (regulations, costs reductions, larger competition, etc.). It will be important to point out that the SDN will be a major driving force in the current and future regulation scenarios of the sector. It will be also necessary to analyze these scenarios and simulate the cooperation-competition among diverse players like N.Os, OTTs, but also Municipalities, Providers of Consumers Electronics, Enterprise Networks, etc.

Our gap analysis also reveals that a broader vision, a broader framework can be used to create an ecosystem, which may include: User Equipment (UE), Radio [23], Transport, Application, network management, cloud, cognitive, and smart grid.

The broader vision will apply to many different segments of a network such as wireless networks [22], optical networks as well as different architectures such as service oriented architectures, and the management plane (E2E automated control of business by BSS and OSS). Most importantly, the broader vision for "Software Defined-concept" will be very important for IEEE to base its activities in publications, conferences and standards

III. Components of the Initiative

The components of the initiative are identified along the lines of IEEE's strength areas: conferences, standards, training/education, and certification. The initiative will also have an additional component to supplement these strength areas: Web Portal and Communications.

A. Publications

A quick analysis indicate that there is no dedicated venue for publications in SDN today; neither IEEE nor otherwise. There has been some feature issues in the past in IEEE Communications Magazine [16] and in Springer Journal on Network and System Management. Therefore, SDN needs a publication outlet to bring together a sizeable SDN research community from various outlets (conferences, workshops, journal special issues and series). We need to make sure a dedicated journal will be sustainable since much of the interest seems to be driven from industry; but the interest from academia has been growing exponentially.

The initiative includes a magazine style publication, called IEEE SDN Magazine. High interest is expected from researchers and even more so from industry. It will create a forum for SDN researchers, developers, Open Networking Summit – type audiences. It will balance new topics and survey-style articles. The plan is to begin with a quarterly magazine in the first year, and increase the frequency in later years.

The initiative also includes journal publications. The plan is to have special issues on SDN on existing venues such IEEE Communications Magazine and IEEE Transactions on Network and Service Management. A new SDN Series in IEEE Journal on Selected Areas in Communications (JSAC) will be

launched immediately. The plan is to start with two dedicated issues per year and ramp up to 4 issues in the second year. This will serve as incubator for a new IEEE Journal on SDN. There is a need to establish at least one journal as the "place to go" for archival quality SDN papers. Based on response / submission numbers, early experiences with SDN JSAC series, SDN magazine, special issues, conference/workshop publications/workshops publications on SDN, an IEEE journal will be launched during Year 3 of SDN Initiative. The plan is to publish four issues a year initially and ramp up to more based on expected increase of paper submissions due to visibility offered by a journal dedicated to SDN research.

The initiative plans to leverage IEEE Press/Wiley book series to publish a set of books on SDN. Since the topic is in flux now, books only on certain point topics (e.g. Openflow) are expected to be published. These can be edited books with chapters contributed by field experts.

B. Conferences

The current landscape shows us the following: There are several existing events dedicated to SDN including a number of industry summits such as Open Networking Summit (Oct. 2011, Apr. 2012, Apr. 2013), China SDN and Open Networking Summit (Dec. 2012), SDN Summit (Mar. 2013), SDN World Congress (Jun. 2013), SDN & NFV (Oct. 2013), and SDN Asia (Dec. 2013). There were a number workshops on SDN as well: HotSDN: ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (Finland – Aug. 2012, Hong Kong – Aug. 2013), SDN: IEEE ICC Workshop on Software Defined Networks (Canada – Jun. 2012), and EWSDN: European Workshop on Software Defined Networking (Germany – Oct. 2012, Germany - Oct. 2013). It is important to note that three have been numerous panels, tutorials, papers and plenary talks at general systems conferences: mainly ACM and USENIX sponsored events (e.g., SIGCOMM, NSDI, OSDI, NDSS, IMC, CoNext, HotNets, HotICE, PAM, etc.), and a few are IEEE sponsored (e.g., IM, NOMS, CNSM, INFOCOM, ICC/GLOBECOM).

One can easily conclude that IEEE needs to play a more active role in SDN-related events worldwide. There is little IEEE presence so far including a few dedicated workshops: SDN Workshop co-located with ICC (2012) and European Workshop on SDN (EWSDN'12) and a few general purpose conferences: only a few compared to other learned societies/organizations, namely ACM and USENIX.

There is a great potential for IEEE led conferences, workshops, and forums: ACM SIGCOMM HotSDN 2013 attracted 86 paper submissions (will NOT continue beyond next year per SIGCOMM policy); SDN tutorial at IEEE IM 2013 attracted over 75 registered attendees; worldwide industry SDN adoption; increasing interest in academia judging by the number of grant proposals; and no major SDN conference venue today, neither IEEE nor otherwise.

Based on the above assessment, the initiative proposes the following 4 category of conference activities:

- Sponsor SDN related activities in IEEE Conferences: This should include keynotes, technical sessions, panels, tutorials in IM, NOMS, ICC, GLOBECOM, INFOCOM, etc. Also it includes workshops in conjunction with IEEE Conferences such as IM, NOMS, ICC, GLOBECOM, INFOCOM, etc. Industry Forums and Industry Exhibits on SDN could also be part of ICC, Globecom, IM, NOMS, etc.
- Launch "Annual IEEE World Conference on SDN": This would be positioned as the flagship IEEE conference on SDN. Hhigh interest and attendance from academics and the industry are expected.
- Sponsor IEEE Regional conferences and workshops in Asia Pacific, Europe, and Latin America.
- Sponsor IEEE Topical SDN Conferences/Workshops on Software Defined Radio, Software Defined Optical, Software defined Data Centers, etc.

C. Standards

Many efforts are underway by major Standards Developing Organizations (SDOs) principally in requirements analysis (SDN-SP, POMI, ETSI/NfV, IRTF SDN, IETF SDNP, and OSU) and few standards specification (OpenStack, NIST, IEEE NGDON, OpenDaylight, ONF –F, IWWW 802, SDR Forum, OMA DM2) for enabling the SDN.

There exists a large gap in realizing the SDN, primarily because the standards landscape needs more focus. There is high potential to apply Software Defined Standards in enabling business cases/applications for Smart Grids, Entertainment, Health, Finance, etc. The following is a snapshot of the hot areas in the next 3 years for standards in Software Defined Ecosystem:

- User: Virtualization of Customer Equipment Functions; Centrally controlled automated P2P relations for Wireless Adhoc Networks. Dynamic detection and attachment of User sessions in a Het Net; Emerging research in programmable machines (movable parts) (Creating custom antenna function, user tracing antennas, vehicle networks, etc.).
- Transport and Access: Global Flow control through flow control negotiations among *providers*.
- Service and Control Infrastructure: Virtualization of service functions for enabling programmable service delivery and operations.
- Application & Management: E2E automated control of business by BSS and OSS. Business orchestration to dynamically create Network orchestration.

To take advantage of these opportunities, the initiative will set up a Software Defined Ecosystem (SDE) Standards Committee to promote, co-ordinate, collaborate and accelerate focus on SDE Standards. Also support the standards development through IEEE-SA. The responsibilities will include providing IEEE SDN Standards Representation in Global Standards Collaboration (GSC) - proposing a Question on SDN and Collaborate on Global Standards, identify a subset of these organizations and their thought leaders and establish a collaboration board around SDE.

The IEEE SDN Standards Committee will be created with major industry and academic players. This committee will be instrumental in creating the PARs for IEEE SDN Test Bed, in extending the scope of existing IEEE Standards Initiatives – 802.x, P1903.x, etc., in supporting the Standards Boards in various IEEE societies to create and support new PARs in SDE domain (CPE, App interface, Management, INTEROP) and provide relevant technical support, as well as in enhancing participation in Standards by collaborating and publicizing with IEEE Regional Chapters.

IEEE SDN Initiative Standards Committee will be responsible to setup SDE Technical Advisory Group (TAG) to build the E2E SDN blue print for various Operators (Carriers, MVNO, ISP, App Provider, etc) for implementing end to end SDN based on world wide standards. TAG will provide a test bed to support validating or certifying the products based on these standards to help the certification of the E2E SDN implementation both for operators and vendors. Based on business demand support for additional TAG spin-offs (Eg. SmartGrid–SDN TAG, Data Center–SDN TAG, Carrier-SDN TAG, etc) will be provided as well. And eventually, transfer the activities to relevant IEEE Technical Societies like ComSoc, SA, Computer and others will be coordinated by TAG as well.

Setting up and maintaining a SDE knowledge repository (website) for next 3 years of world wide efforts in SDN specifically from Research and Standards aspects will be done in coordination with the IEEE SDN Web Portal. This will support in publishing SDN business case study white papers and publications from IEEE SDE Standards Committee.

Promotion of IEEE SDE Standards by supporting commercial demonstrations by operators in key events (IEEE and non-IEEE) by providing logistical support will be among the other activities of this committee.

D. Training/Education

Existing educational events dedicated to SDN include various tutorials given at conferences such as ONS (Open Networking Summit), ACM/USENIX/IEEE sponsored conferences and some online tutorials such as SearchSDN.com, Bay Area Network Virtualization. There are also several education and training courses, most notably by "SDN University", SDNCentral. The format varies and includes free webinars, paid webinars, on-site training. SDN Academy also offers education courses and on-site training. In addition, there is Georgia Tech free online course on SDN offered via Coursera.

IEEE needs to play a more active role in SDN-related education worldwide by providing a set of comprehensive SDN-related educational activities online and in local regions including developing regions and by providing systematic training for supporting certification, standards, and testbed related activities.

Based on the above assessment, the initiative proposes the following four category of educational and training activities for IEEE:

- Provide a set of comprehensive face-to-face and online short (2-3 hours) tutorials and webinars on SDN. This may include introductory, intermediate as well as advanced tutorials on SDN and SDN related topics. Online tutorials could be once a month. The face-to-face tutorials could be part of the major conferences. Some examples of topics include: Introduction to OpenFlow: standards, current development, and open research issues; Overview of SDN: SD-related research and projects; SDN Framework and Functional Components; Current activities in SDN standardization
- Provide online and face-to-face day-long (6-8 hours) learning courses. These include training courses and education courses that could be targeted for certification, for standards, and for test-beds as well.
- Integrate/collaborate with other components to make IEEE SDN training and education activities more meaningful and practical for the industry. The collaboration with Portal activities will increase public awareness, broaden audience to SDN via Facebook, Twitter, magazine/newspaper. Tutorials, eLearning courses, and Webinars will be publicized through the portal, which will also be useful in collecting potential interested topics for tutorials, eLearning courses, webinars, and summer schools. Collaboration with Publications and Conferences activities will help publicize eLearning courses in IEEE publications. Also, obtaining videos of conference talks (keynote, distinguished talks) and tutorials for Webcasts, online tutorials, and education courses will be very helpful in creating a portal warehouse for SDN material. Finally collaboration with Certification, Standards, and test-beds activities is essential to develop training/education courses: e.g., certification curriculum, certification classes.
- Provide education and training activities for IEEE local sections/chapters. These could be handled by interacting with the Distinguished Lecturer Tours and sponsoring humanitarian talks in developing regions and 1~2 summer schools around the world.

E. Certification

It seems that there is very little certification related activities on SDN. There is Cisco Certified Network Professional (CCNP) Service Provider certification program, which is aimed at developing the skills and knowledge of IT professionals to deploy and next-generation networks; not specifically tailored for SDN. There is als Indiana University's InCNTRE Lab, which is a certification lab for Open Flow.

Certification for SDN is definitely a fertile area! This is a great opportunity for IEEE to get into. IEEE can provide certification in the following three areas:

- Professional Certification: This may consist of the establishment of Body of SDN Knowledge (BoSK); creation of BoSK Learning Tools which should include the curriculum, books, newsletters, classes, as well as the support. In addition, development of BoSK

Examination Tools including SDN-P Certification Exam and Renewal is needed.

- Device Certification: This will require that device requirements, SDN-D Specs, application areas, and benchmarks need to be developed by solicitations and interactions with the industry. Revisions need to be managed. Also, evaluation tools for SDN Certification test bed, SDN-D Benchmark tests, and SDN-D Certification need to be established. The process for certification rrenewals needs to be established as well.
- Internal SDN-test bed development: Certification procedures should be developed based on the initial test bed evaluations. Implementation of a test bed via Internet 2.0 and local data centers/supercomputer centers is necessary to utilize state-of-the-art resources. Education activities will be included in the SDN-test bed for education/curriculum activities. Third party requirements which require SDN Testbed specs and heavy Industry relations need to be established. Finally, an evaluation board for testbed certification need to formed for managing SDN-T Re-evaluation-Renewal-Expiration-Cancellation activities.

F. Web Portal and Communications

Publicity and presence of a well developed, user friendly web portal are two essential ingredients for the success of any initiative. Currently, there is only one independent SDN portal (SDN Central). Other portals are all either company owned or standards body specific: Open daylight (Cisco-IBM led), Company specific portals: SAP, Jupiter, Cisco, etc. Standards Specific: OpenFlow, Open Networking Foundation.

The SDN portal will be an essential infrastructure for the IEEE initiative, which is needed to provide support broad based interpretation of SDN, to become a "store front" for an easy access to all IEEE SDN Initiative programs, to publicize IEEE SDN related activities, to be repository of SDN specific materials (videos, presentations, documents, etc), to provide a platform for SDN specific job postings, news, events, etc. And furthermore the portal has a great potential for revenue generation through ads.

The SDN portal will have links to all IEEE SDN activities. It will be a user interface to access to IEEE SDN products and services. It will include SDN related news, SDN related job posting, and advertisement (banners, buttons, context sensitive ads, etc.). The portal will be the clearinghouse for all SDN related presentations, videos, documents, etc. The portal will be integrated with social media such as Facebook and Twitter.

The development of a portal is time consuming and expensive. But, the presence of a portal is essential as soon as possible to support the other components of the initiative. A phased development approach is the key to speed up the launch of the portal as early as possible.

Communications is an important aspect of this initiative. We need to place IEEE on the SDN map and for this we should have a strong communications component, which includes marketing, public relations, and public visibility. Web Portal is an important component of this component. Other communications related activities include: Social Media, Co-Branding, co-sponsoring activities in related events, and advertisement in other SDN portals and other publications.

IV. PROJECT MANAGEMENT

The project management part of the initiative provides an insight into its milestones and schedule, estimated expenses, potential revenues and how the initiative will be managed. The proposal includes a specific organizational structure which consists of an executive committee for day-to-day operations of the activities as well as a steering committee to oversee and make fundamental decision on the operations. The schedule proposed takes into account of the fact that prioritized launch of products and services is necessary to utilize resources efficiently and to keep up with and to lead the community, and utilize co-branding as much as possible.

The proposal also established certain rubrics to measure the success of the initiative. For publications, the number of submissions, the number of downloads, and advertising revenue will be the criteria. For conferences, the number of attendees and the surplus will be the keys. For standards, the number of PARs, number of members in each WG, and number of companies participating will the basic criteria. For education, the umber of attendees; for certification, the number of applications, and for the portal, the number of visits, number of downloads, and advertisement revenues will be used as the basic criteria to measure success.

We recognize that there are numerous challenges. Predicted interest in academia may not materialize. Then a shift from journals to more industry oriented publications may be necessary for publications. Moreover, a shift from academic conferences to more industry oriented conferences, meetings and events may be necessary as well. Industry may not be interested in IEEE standards and again, the predicted interest may not materialize which will impact the education and training components as well. The need for certificaction is closely tied to the acceptance of the SDN technology in general. The challenge for the web portal would be that we need to make it attractive enough (content and appearance) to catch attentions and draw advertisers; make it user friendly and useful.

V. CONCLUSIONS

This paper describes a proposal which is a result of an intensive effort by a team of experts supported by an advisory board. This is a comprehensive proposal for an IEEE-wide initiative on Software Defined Network (SDN), which is a new paradigm in designing networks and its components

The report includes a gap analysis on the field and related technologies and associated activities in publications, conferences, standards. The report also identifies opportunities and specific proposals for IEEE in the areas of publications, conferences, standards, education, certification, and web portal. The report proposes a 3-year program with a specific schedule, milestones, and estimated budget.

We conclude that there is a need to broaden the vision of SDN to cover all segments of a network (i.e., end-to-end), and

to get all types of technologies (wireless, optical, etc) involved. This broader vision is especially important for IEEE to fill the gap and provide a larger scope for its products in publications, conferences, etc. It is essential that we establish an IEEE wide initiative with a comprehensive program based on a broad vision of SDN in order to make the IEEE a major player in this field.

REFERENCES

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner , OpenFlow: Enabling Innovation in Campus Networks, CCR 2008

[2] The Future of Networking, and the Past of Protocols, Scott Shenker (video of talk at Ericsson, slides of talk at ONS'11)

[3] What OpenFlow is (and more importantly, what it's not) http://networkheresy.com/2011/06/05/what-openflow-is-and-more-importantly-what-its-not/ & What Should Networks Do For Applications? http://networkheresy.com/2013/04/13/what-should-networks-do-for-applications/#comment-922

[4] Soudeh Ghorbani, Matthew Caesar, HotSDN 2012 Abstractions for Network Update,

[5] Stephen Gutz, Alec Story, Cole Schlesinger, Nate Foster, HotSDN 2012 Splendid Isolation: A Slice Abstraction for Software-Defined Networks,

[6] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, Teresa Vazao, HotSDN 2012

[7] OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking, Jafar Haadi Jafarian, Ehab Al-Shaer, Qi Duan, HotSDN 2012

[8] IETF Software Driven Networks http://nerdtwilight.wordpress.com/2012/01/30/sdn-double-vision/

[9] Open day light web site

[10] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, Shriram Krishnamurthi, HotSDN 2012 Walk the Line: Consistent Network Updates with Bandwidth Guarantees,

[11] Rick McGreer, HotSDN 2012 Hierarchical Policies for Software Defined Networks,

[12] Li Erran Li, Z. Morley Mao, Jennifer Rexford, EWSDN 2012, Towards Software-Defined Cellular Networks

[13] ONF, Brocade, "Open Flow Products - Brocade MLX Series", https://www.opennetworking.org/sdn-openflow-products/662-brocade-mlx-series

[14] Mohammad Banikazemi, David Olshefski, Anees Shaikh, John Tracey, Guohui Wang, IEEE Communications Magazine Feb 2013 Meridian: An SDN Platform for Cloud Network Services

[15] Wireless Innovation Forum (SDR forum): http://www.wirelessinnovation.org/

[16] Improving Network Management with Software Defined Networking, *Hyojoon Kim, Nick Feamster,* IEEE Communications Magazine Feb 2013

[17] A Clean Slate 4D Approach to Network Control and Management, Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, Hui Zhang, CCR 2005

[18] Qiang Duan, Yuhong Yan, and Athanasios V. Vasilakos, A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 9, NO. 4, DECEMBER 2012

[19] IETF General Switch Management Protocol (GSMP) V3 RFC 3292 http://datatracker.ietf.org/doc/rfc3292/history/

[20] Self-Organizing Networks (SON): http://www.3gpp.org/SON

[21] Bram Naudts, Ghent University – iMinds, " Techno-economic analysis of SDN", 2012 IEEE EWSDN

[22] Firew Siyoum, Marc Geilen, Orlando Moreira, Rick Nas, Henk Corporaal, "Analyzing Synchronous Dataflow Scenarios for Dynamic Software-defined Radio Applications"

[23] Manu Bansal, Jeffrey Mehlman, Sachin Katti, Philip Levis, HotSDN 2012, OpenRadio: A Programmable Wireless Dataplane

Software Networks at the Edge: a shift of paradigm

Antonio Manzalini
Strategy, Future Centre, Innovative Architectures
Telecom Italia, Italy

Roberto Saracco
EIT ICT LABS

***Abstract* — Technology advances and costs reductions in processing, storage and communications will determine, in the next years, a growing amassing of IT and networking resources at the edge of current networks (i.e., not only in the distribution and access segments, but up to the end Users devices). Moreover, the edge is also where we are witnessing "intelligence" migrating, already since a few years. This paper argues that the adoption, in this area, of Software Defined Networks and Network Function Virtualization solutions will create a sort of distributed communication "fabric" offering an enormous processing and storage power, to execute functions and services, and to store data. This, together with the economic drive given by a myriad of new Players (entering the arena) along with the rapid expansion of ICT services, will lead to a shift from a network infrastructure, whose capabilities are statically "designed and provisioned", to an highly flexible edge fabric whose dynamics is an emergent property of the ever changing aggregation and usage of resources.**

Keywords— Software Defined Networks, Network Functions Virtualization, Edge Fabric, Self-Organization

I. INTRODUCTION

IT technology is progressing at an impressive rate: computing is still following the Moore's curve, doubling in capability roughly every 18 months; storage is doubling every 12 months, fueling the connectivity demands to access the network; optical and radio bandwidth are, in turn, continuously increasing. Hundreds of TB of storage at residential level will become possible by the end of the next decade, although the technological availability may not translate in adoption if the network will be able to provide seamless and unlimited connectivity.

These progresses are impacting the evolution of Telcos networks and services infrastructures: transport networks are going to become less and less hierarchical, with a limited number (and types) of optical nodes, inter-connecting metropolitan areas. In turn, these areas, at the edge, are going to be densely populated by thousands of small aggregation nodes (with processing and storage capabilities) and by an incredible number of devices and machines, accumulating around Users.

The number of devices connected to the network is growing at an exponential rate. For every cell phone today we have already a few sensors and by the end of this decade there will be more than hundred sensors per each cell phone, bringing the number of connected objects to the hundreds of billions.

In this evolutions, software will be the true challenge. Future networks will rely more and more on software, which will accelerate the pace of innovation (as it is doing continuously in the computing and storage domains) and will reduce costs (e.g., determined by the virtualization of middle-boxes [1]). This, together with the economic drive given by a myriad of new players entering the market, will lead to a shift of paradigm, capable to drive investment outside of the network infrastructure boundary and stimulate the advent of new communications paradigms. The Web 2.0 paradigm will evolve from having a network of services to applications made available by a plethora of (small) players.

Software Defined Networking (SDN) [2] and Network Function Virtualization (NFV) [3] can be seen as steps in this direction. In particular, in SDN, the network control (S/W) and data forwarding plane (H/W) are decoupled, so that in principle the network infrastructure could be abstracted from functions and business applications. SDN should not be confused with NFV, which is about virtualizing some network functions that could be executed on standard H/W, and that could be moved and instantiated in various locations. SDN and NFV could be seen as mutually beneficial but they are not dependent on each other: e.g., network functions can be virtualized and deployed without an SDN being required and vice-versa.

It's true that already today, there is a certain level of adoption of resource virtualization, allowing the deployment, on the same physical infrastructure, of diverse coexisting and isolated virtual networks. On the other hand, SDN and NFV transformation might impact to such depth the legacy operations processes (e.g. based on centralized OSS, BSS) of the current infrastructures that it is questionable whether these paradigms will be really adoptable in Telcos WAN or core networks.

Overall we are saying that the recent progress of telecommunications has been strongly affected by computers and IT in general. Computers helped in coding and decoding the information, they manage the network resources, they support all administrative and customer care interactions. The whole network has slowly become a massive distributed computer system. Lately, also its termination, cell phones and a variety of connected devices, are also computers of some sort.

Still, by far, we tend to keep a distinction between the "network" and what connects to it, the terminals. More and

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

more this distinction becomes an artificial one, as more and more functionalities can be performed in the network and/or in the terminals.

If it makes economic sense for a User to add, move or change functionality (and NFV will allow that soon) in his/her cell phone (e.g. to pay less) he/she could do it. The cost to develop an app for Android or iPhone is measured in tens of dollars (of course complex applications may require much bigger investment) and even teen-agers can do it, and they do. This trend is unstoppable, because of the continuous technology evolution that enables different economic paradigms.

Shifting the focus from the network to the terminals makes evolution possible at the micro scale. It cost less, it scales more gracefully, and leads to immediate revenues. Evolution in terminal is much faster than in network equipment, for their sheer number (a new cell phone appears on the market every week!) and it is viral. This creates a volume and an economic market that will drive investment outside of the network infrastructure boundary and stimulate the advent of new communications paradigms.

The edges will be transformed into complex Data Centers consisting of a number of diverse and autonomous, but interrelated nodes, devices, machines.

As known, complex systems cannot be easily described by global rules and their characteristics are not reducible to one level of description: they exhibit properties (e.g. self-organization) that emerge from the local rules and interactions of their parts. These self-organization capabilities will be achieved by introducing autonomics and cognition, as transformative methods and systems.

The paper is organized as follows. Section II describes two main ways of potential adoption of SDN (in the core and at the edge). Section III elaborates about how approaching one of the main challenges behind the exploitation of SDN at the edge, which is dynamically instantiating, orchestrating and relocating multiple VMs executing network functions. Section IV presents the main components of the SDN fabric, defined as distributed edge infrastructure consisting of loosely coupled storage, networking and processing functions interconnected by high bandwidth links. Section V summarizes the backstage for next decades communications infrastructures and elaborates about the related economics.

II. SOFTWARE DEFINED NETWORKS: TWO WAYS

In this section we'll motivate why SDN and NFV will follow the shift of paradigms (from the network to the terminals), thus focusing at the edge of current networks.

We have to distinguish SDN from software (defined) networks: the former is what most people are talking about. It is a sort of derivation of past networking thoughts, where the potential separation of H/W from (control) S/W – feasible today thanks to technology advances - can be potentially used to improve and optimize certain functions (e.g. Traffic Engineering over a WAN interconnecting Data Centers), and to reduce investments by means of low cost pieces of equipment and by customizing open source software.

It is questionable whether this type of SDN will be really adoptable in Telcos core networks. It is more likely that Telcos will disappear, or be outcompeted, before such SDN will be deployed. The impact on their internal processes and procedures will be too disruptive, and they are not ready, from a "cultural" viewpoint to adapt to this change seamlessly. On the other hand, some OTTs are, as demonstrated by some recent examples of exploitation of SDN-like WAN interconnecting Data Centers. So Telcos have to play the SDN game on a different ground, i.e., at the edge.

In fact, the latter way of considering software (defined) networks is about creating very dynamic virtual networks out of a variety of aggregation nodes, devices, elements located at the edge of current networks, up to around Users. Some of these elements usually are not considered yet as network nodes: for example, machines, sensors, actuators, any Users' devices, smart things with embedded communications, etc. In other words this is about developing a "fabric" made of an enormous number of nodes and elements aggregated in an application driven way.

The term "fabric" has been used in the past to refer to a distributed computing system consisting of loosely coupled storage, networking and processing functions interconnected by high bandwidth links. It has also been used to describe a flat, simple intra data centers network optimized for horizontal traffic flows, mainly based on a concept of "server-to-server connectivity".

Leveraging on these meanings, the term fabrics is extended, in this paper, to indicate the edges of the metro networks, becoming as distributed Data Centers consisting of loosely coupled processing and storage resources interconnected by pervasive wired and wireless links.

As a consequence of this, several processes or activities normally carried out in Data Centers, such as allocation, migration and cloning of VMs (e.g. for server consolidation, load balancing, etc.) will be also performed in the aggregation nodes, even up to Users' devices.

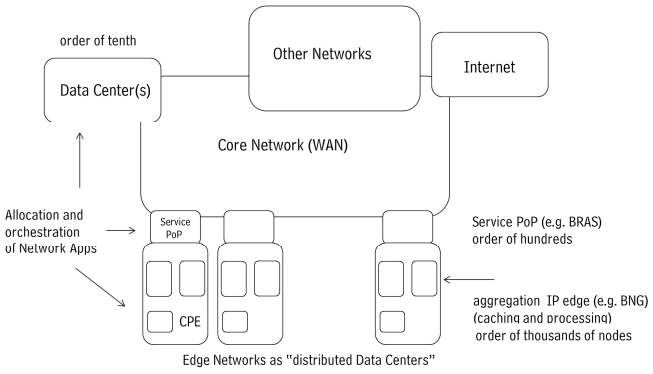


Figure 1 – Example of overall network architecture

So, it is claimed that the exploitation of SDN and NFV will follow this shift, focusing at the edge of current networks. One of the main challenges will be the capability of dynamically

instantiating, orchestrating and relocating multiple Virtual Machines (VMs) across various network locations (Fig. 1).

Ideally, at the edge fabric it will be possible to create, program, instantiate or migrate dynamically different types of virtual networks, functionality and services. In addition, we are going to see the emergence of local data exchange, edges-to-edges and terminal-to-terminal.

No more ossified architectures, but a sort of ephemeral (temporary) virtual networks of resources capable of self-adapting elastically and flexibly to application dynamics.

The edges will be like huge distributed Data Centers. A 1TB cell phone will be soon available, media centers in the home will host the entire life production of a family in their multi-TB storage. There are already Companies planning of using residential storage as an alternative distribution means, EB (a billion of billions of bytes) will become commonplace in the next decade.

III. Towards Self-Optimizaton with Local Rules

We've mentioned that one of the main challenges will be the capability of dynamically instantiating, orchestrating and relocating multiple VMs executing network functions.

Some network functions (e.g. firewall) could be even embedded as-a-service in the hypervisor (a layer of software between the hardware and the operating system) kernels. Packets sent from a VM could be inspected by the stateful function services in the hypervisor kernels before being sent to the network, thus limiting the need of traffic steering.

Figure 2 is showing the general architecture of the Universal Edge Node (UEN) concept. This node architecture should be very modular, capable of scaling from a CPE to an aggregation node (embedding processing and storage) or a Service Point of Presence (PoP).

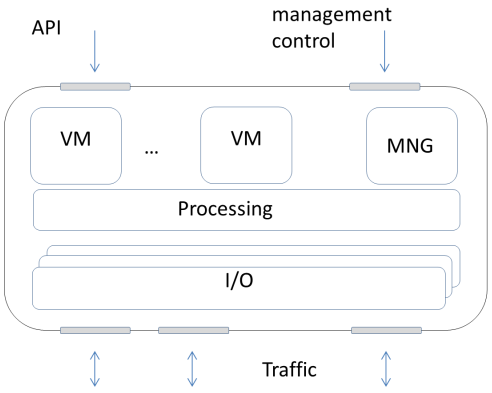


Figure 2 – General architecture of the Universal Edge Node

The edge fabric will be composed by a myriad of UENs interacting each other. Out of this complexity, it will not be possible to adapt centralized management approaches. Software networks at the edge should be able to self-adapt and self-configure themselves (with limited human intervention).

In this section how self-organization could be achieved as a property emerging from the interactions of nodes local behaviors, solving constrained optimization problems with statistical approaches and to data analysis. The communications fabric is the key enabler both at the level of raw data harvest and, at the level of processing into metadata, processing of metadata versus specific application and customer demand and distribution of results.

Let's consider an example coming from statistical mechanics: the behavior of a gas (e.g., modelled by the state equation) can be described at the microscopic level in terms of the position and velocity of each molecule, which appear as random processes. On the other hand, the local random behavior of the molecules causes the gas as a whole to solve a large scale constrained optimization problem.

Similarly the behaviour of electrons in an electrical network can be described in terms of random walks. This simple description at the microscopic level leads to rather sophisticated behaviour at the macroscopic level: for example, patterns of potentials in a network of resistors is minimizing heat dissipation for a given level of current. Again, this is like to say that the local random behaviour of the electrons causes the network as a whole to solve a large scale constrained optimization problem.

Moving to living entities, a termite colony can be seen as a self-organized ecosystem. We can imagine each termite having its own utility function (e.g., concerning its survivability), and we may argue that, in this sense, evolution selected those behaviours capable of maximizing said utility functions. Termites are interacting each other, and with the environment, whilst living in their colonies. Another time, this is like saying that the termites' simples behaviors, and cross-interactions, are solving constrained optimization problems: and this appears as an emergent self-organization.

In summary, one may say that any large scale ecosystem tends to move from a state with higher energy (i.e., higher cost) to a state with lower energy (i.e., lower cost); local minima are usually related to the stable stationary states (as a consequence of functional minimization). Indeed, maximize profits, minimize costs, minimize the loss are typical economics problems, which can be mathematically modelled as constrained optimization problems.

It is argued that the same principles could be taken for the edge: a guided self-organization will become a property emerging from the interactions of nodes local behaviours, solving constrained optimization problems.

In literature there are similar approaches for minimization (or maximization) of an objective function subject to constraints on the possible values of the independent variables. For example "Layering as Optimization Decomposition" [4], [5] integrates the various protocol layers into a single coherent theory, considering them as carrying out an asynchronous distributed computation over the network to implicitly solve a global Network Utility Maximization (NUM) problem.

Since then, many research studies have been carried out on distributed network resource allocation using the language of NUM. These efforts have found many applications in network resource allocation algorithms and Internet congestion control protocols, e.g., [6], [7], [8], [9]. For example, the TCP/IP

protocol can be seen as an example of an optimizer: its objective is to maximize the sum of source utilities (as functions of rates) with constraints on resources. In fact, each variant of a congestion control protocol can be seen as a distributed algorithm maximizing a particular utility function.

The exact shape of the utility function can be reverse-engineered from the given protocol. Similarly, other recent results also show how to reverse engineer Border Gateway Protocols (BGPs) as a solution to the Stable Path Problem, and contention-based Medium Access Control (MAC) protocols as a game-theoretic selfish utility maximization.

Also cross-layer interactions can be characterized by viewing the process of network layering as the decomposition of a given NUM problem into many sub-problems. These sub-problems are then "combined together" by certain functions of the primal and dual variables [5].

Normally these maximization/minimization of complex functions (or functionals) are achieved with metaheuristics. On the other hand, these approaches are not that scalable (and quickly converging) for highly distributed systems composed by huge amount of interacting nodes, which is the case for the edge fabrics. So centralizing such metaheuristics is not a direction to pursue.

This paper adopts another approach which is really achieving guided self-organization as a property emerging from the interactions of nodes local behaviors. The main problem becomes designing these local rules and their control parameters.

As a simple example, imagine introducing into the edge nodes some "controllable" local rules, making the nodes capable to learn computing input-output data mapping with desired actions or properties (e.g., local rules means rules capable of changing for example the "strength" of the interconnections of a node with the immediate neighborhood, in line with the Hebb's postulate of learning).

Edge software networks could be controlled by a few hierarchical entities but could also be actuated locally, as result from decision of autonomous entities. An example of a hierarchical entity is the ambient, a dynamic collection of devices that all together create a well-defined context (like a home, a mall, a classroom....). These ambient are connected through the big network (i.e., via big communication pipes) one another forming bigger, sometimes overlapping ambient, like a city, a hospital, a school, but also a project domain... The ambient could act as a gateway to access backbones (although the physical gateway can be distributed and embedded in the various devices).

IV. Components of the Edge Fabric

The advent of autonomic systems, the multiplication of networks, the presence of huge storage capacity at the edges of the old network (more specifically in the terminals, cell phones, media centre) and the growing intelligence outside the network, will change significantly the network paradigms.

The overall communications environment will consist of millions of data and service hubs connected by very effective (fast and cheap) optical and wireless links.

The edge fabric will be based on these two main components:

- personal virtual terminal that connects Users' information space to the fabric: the actual terminal will depend on what is needed and what is available in a given context. What really matters is the an User identity and through this identity any device will become the "personal device". The set of personalization is part of User identity in the connection fabric and the exact instantiation depends on the type of device.
- edge nodes (e.g. UEN) and devices (i.e. aggregation nodes, terminals, machines, sensors, actuator...): each device with sufficient processing, storage and communication capacity can in principle become a network node and share its capacity in the network. By aggregating with other devices can creates a local network and by selecting an appropriate gateway it can connect that network to the big network. Each device can act as an autonomous system and its behavior depends on its "goal" and on the ambient it is part of. Decisions are made based on local conditions, they change the ambient and these changes percolate to broader and broader contexts.

As mentioned, the edges will be transformed in distributed Data Centers: the information space will be formed by a federation of clouds each one having different ownership. A device might be part of a cloud and might contribute to create a cloud by sharing its data, the access rules and the processing capability. If we look at the processing and storage capacity in the sum of all smart phones in Italy they exceed the processing and storage capacity of the largest DC. If one can share these capacity at the edges one can really create a most powerful cloud.

From a purely functional perspective, an edge node (Fig. 2) could be also seen as consisting of a set of services, leveraging the concept of Self-Managed Cell architecture reported in [10]. For example, the discovery service discovers resources and components being part of the node and the other nodes entering in the communication range (each single node is clearly designed for interactions). A policy service could be in charge of managing the policies specifying the node behavior.

The overall fabric can be seen empowered with three different kinds of behaviors (Fig.3):

- automatic behavior: a capability achieved with fast pre-defined rules for self-configuration in predefined contexts; it could be designed, for example, by means of simple and fast automatic rules deployed into edge nodes;
- autonomic behavior: a capability responsible for local self-adaptation and it is achieved by exploiting unsupervised learning capabilities. The layer could be designed, for example, by introducing reinforcement learning and cognitive methods and algorithms into edge nodes;
- self-organization behavior: a global emergent capability achieved through the orchestration of local reactions (e.g.,

through activation-deactivation of rules) by means of control information diffused for reaching the local and global goals.

In other words, the information space will include a sort of global coordination field (i.e. a global context), injected by nodes (and potentially control points) in the network and autonomously propagating.

All nodes are interacting with each other and with the environment by simply generating, receiving and propagating distributed metadata structures, representing context information.

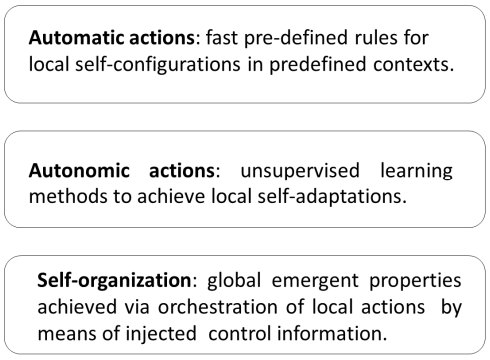


Figure 3 – Behaviors of the Edge Fabric

This global control field is providing the nodes with a global representation of the situation of the fabric (to which they belong). This representation is immediately usable: a node is moving in this field like an object is moving in a "gravitational" field. Environmental dynamics and nodes' local decisions will determine changes in the field, closing a feedback cycle. This process enables a distributed overall self-orchestration.

As an example this control field can be made of tuples of data which can be injected and diffused by each node. Local reading of these tuples of data (e.g. through pattern matching) can trigger local self-adaptation behaviors.

A simple event-based engine, monitoring configurations and the arrival of new tuples, reacts either by triggering propagation of other tuples or by generating events.

V. ECONOMICS OF FUTURE NETWORKS

In the next decade, we are going to see that a variety of edge devices will indeed absorb progressively the network, if not all of it a significant (economically speaking) part of it. From an economic point of view we are already there: consider the business of cell phones and look at the business of cellular equipment and you'll discover that 70% of the biz is in the terminals and just 30% in the network equipment.

Let's summarize the backstage for next decades communications infrastructures:

- increase intelligence at the edges (a variety of Users' devices, machines, etc);
- increase in the number of communicating objects (hundreds of billions of sensors, actuators, etc);
- economic drive and focus on the edges, rather than on the network;
- thousands of players (and consolidation of existing Network Operators);
- complete decoupling of services from the infrastructure;
- potentially each edge node or terminal becomes a network node and along with neighbor terminal can create a local network fabric.

In the next decade the technological substrate and the economic drive will create a shift from communications infrastructures to a communication fabric (that is not, as it is the case today, created to connect devices).

The communication fabric will be created by the devices themselves, a continuously changing fabric whose behavior shows awareness and where semantic is the engine.

These devices that eventually (and dynamically) will form the network fabric through aggregation will be deployed and owned by independent parties so that no centralized planning/coordination will be feasible. Hence the takeover of the autonomic paradigm and the creation of a "small world" fabric whose reliance and robustness will depend on its very basic rules of aggregation, as it is the case in our brain as well as in social insect colony.

Technology is going to become basically accessible to all enterprises in any part of the world on an equal basis, further reducing any competitive advantage due to location. Hence, the real differentiator rests on the capacity to innovate continuously. The challenge for Telecom Operators, that may want to position themselves at the enabling layer of a business ecosystem, is how to transform their management infrastructure into an ecosystem management fabric, moving from a centralized, hierarchical management to a distributed multi-domain management.

In this direction, semantic meaning is the value and more and more the economics will shift from the economics of resources (becoming commodities) to the economics of information (and its related context). Notice that the economics of information has a lower relative value than the economics of resources (bits cost approaches zero) and this results in a lower barrier to entrance and therefore leads to a larger number of players.

VI. CONCLUSIONS

This paper has proposed a vision stemming from IT technology advances and the cost reductions due to adoption (and consolidation) of standard hardware in network and service infrastructures.

These trends are unstoppable [11]: technology is going to become basically accessible to all enterprises in any part of the world on an equal basis. The economic drive given by a myriad of independent players will lead to a shift of paradigm impacting the Telco environment.

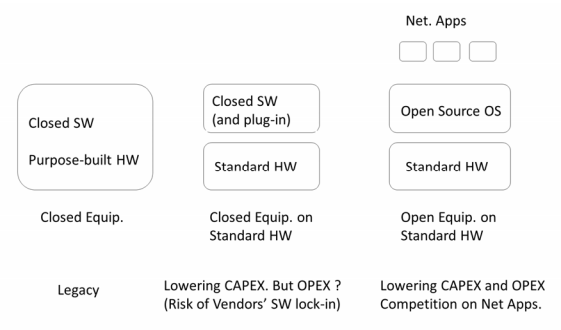


Figure 4 – SDN and NFV impacts on CAPEX and OPEX

SDN and NFV are claiming network cost reductions, due to the adoption (and consolidation) of standard hardware, capable of running network applications. Nevertheless this model (Fig. 4) will be successful only if SDN and NFV will be really based on open source software solutions (closed solutions, in fact, would move costs from hardware CAPEX to software OPEX, thus erasing the claimed advantages of SDN and NFV in terms of cost saving). Actually, a number of related open source software initiatives and communities are emerging in these areas. It is argued that this r-evolution will happen at the edge, first, as it cost less, it scales more gracefully, and it leads to immediate revenues. Evolution in Users' devices, CPE or small aggregation edge nodes is much faster than in core or WAN equipment.

As a matter of fact, the large number of nodes, devices and systems being deployed at the edge, up to Users' premises, are offering an enormous processing and storage power. It is argued that exploiting these resources, closer to the Users, to execute network functions and services will bring several advantages, both in term of improved performance and cost savings. Moreover, the exploitation of these principles will transform the edge into a fertile ground for the flouring of new ICT ecosystems and business models.

Technology is going to become basically accessible to all enterprises in any part of the world on an equal basis, further reducing any competitive advantage. The economic drive given by a myriad of independent players along with the rapid expansion of services will lead to a shift from a network infrastructure to a network fabric edge whose capabilities and behavior is an emergent property of the ever changing aggregation and usage. No more ossified architectures, but a sort of ephemeral (temporary) virtual networks of resources capable of self-adapting elastically and flexibly to application dynamics.

Eventually, because of this evolution, several economists, as well as technologists, have started to wonder if the usual representation of relationships among a myriad of players in a certain area can still be modelled on the bases of value chains. There is a growing consensus that value chains modelling shall be complemented by a broader view considering business ecosystems.

ACKNOWLEDGMENTS

Thanks are due to Roberto Minerva for helpful discussions.

REFERENCES

[1] IETF RFC3234 http://tools.ietf.org/html/rfc3234;

[2] White paper on "Software-Defined Networking: The New Norm for Networks" https://www.opennetworking.org/;

[3] White paper on "Network Functions Virtualisation"; http://portal.etsi.org/NFV/NFV_White_Paper.pdf NfV;

[4] Chiang M., Low S. H., Calderbank A. R. & Doyle J. C. (2007). Layering As Optimization Decomposition: A Mathematical Theory of Network Architectures. *Proceedings of the IEEE*, Vol. 95, No. 1., pp. 255-312;

[5] Palomar D.P. & Chiang M. (2006). A Tutorial on Decomposition Methods for Network Utility Maximisation. *IEEE Journal Selelected Areas Communications*, vol. 24, no. 8, pp. 1439–1451;

[6] Low S. H. (2003). A duality model of TCP and queue management algorithms. *IEEE/ACM Transactions Networking*, vol. 11, no. 4, pp. 525–536;

[7] Kunniyur S. & Srikant R. (2003). End-to-end congestion control: utility functions, random losses and ECN marks. *IEEE/ACM Transactions Networking*, vol. 10, no. 5, pp. 689–702;

[8] La R. J. & Anantharam V. (2002) Utility-based rate control in the internet for elastic traffic. *IEEE/ACM Transactions Networking*, vol. 9, no. 2, pp. 272–286, 2002;

[9] Kelly F.P., Maulloo A. &Tan D. (1998). Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*; 49:237–252;

[10] Sloman M., Lupu E.: Engineering Policy-Based Ubiquitous. Systems Published by Oxford University Press on behalf of The British Computer Society, 2005; doi:10.1093/comjnl/bxh000;

[11] A.Manzalini, R. Minerva, F. Callegati, W. Cerroni (2013). Clouds of Virtual Machines at the Edge, IEEE Com. Mag. "Future Carriers Networks", July 2013.

Research Directions in Network Service Chaining

Wolfgang John*, Konstantinos Pentikousis^, George Agapiou[%], Eduardo Jacob[¤], Mario Kind[$], Antonio Manzalini°, Fulvio Risso[§], Dimitri Staessens[&], Rebecca Steinert[†], and Catalin Meirosu*

* Ericsson ^ EICT [%] OTE Research [¤] University of the Basque Country [$] Deutsche Telekom AG
° Telecom Italia [§] Politecnico di Torino [&] iMinds [†] SICS
Corresponding author email: *wolfgang.john@ericsson.com*

***Abstract*—Network Service Chaining (NSC) is a service deployment concept that promises increased flexibility and cost efficiency for future carrier networks. NSC has received considerable attention in the standardization and research communities lately. However, NSC is largely undefined in the peer-reviewed literature. In fact, a literature review reveals that the role of NSC enabling technologies is up for discussion, and so are the key research challenges lying ahead. This paper addresses these topics by motivating our research interest towards advanced dynamic NSC and detailing the main aspects to be considered in the context of carrier-grade telecommunication networks. We present design considerations and system requirements alongside use cases that illustrate the advantages of adopting NSC. We detail prominent research challenges during the typical lifecycle of a network service chain in an operational telecommunications network, including service chain description, programming, deployment, and debugging, and summarize our security considerations. We conclude this paper with an outlook on future work in this area.**

Keywords—Network Service Chaining; SDN; NFV

I. INTRODUCTION

Infrastructure network operators are currently struggling to meet growing user and traffic demands on their traditional connectivity services, for example, in terms of providing sufficient capacity and mobility support. While subscribers enjoy a constantly (and often drastically) declining "cost per bit", operator investments (CAPEX) and operational costs (OPEX) for the increasingly complex network infrastructure are rising: new technologies have to be incorporated while older investments are still operational and will be so for the foreseeable future. From a technical point of view, "over-the-top" service providers (OTT) can innovate and introduce new technologies at a rapid pace, while vendors close to the physical network enhance access technologies by orders of magnitude within a decade. This is only possible because the middle part of the protocol stack remains largely unchanged.

The price to pay for this approach, which calls for flexibility and innovation concentrating at the edges of the protocol stack, is the so-called "network ossification". Architectural kludges implemented through the introduction of middleboxes exacerbate this further: service chains must be carefully crafted from statically-assembled components chosen at design time. Then, once a network service is defined, little can change: operators can mainly perform minor configuration changes (e.g. parameter tuning) and address scalability through further infrastructure investment to reach more subscribers. In short, a whole network is purpose-built and optimized for a few static services. This modus operandi is advantageous in terms of service quality guarantees and has served, up to now, the telecommunications industry well. But it is particularly inflexible in the current market and technological conditions. Earlier investments in specialized hardware are difficult to re-tool and re-deploy with new functionality: network service providers (NSPs) cannot weave together best-of-breed technologies to form novel full service chains at will. In the end, NSPs operate, manage, and maintain costly and monolithic service silos deployed for decades.

To some extent, the current network service chaining (NSC) model is reminiscent of how mainframes were built in the early years of high-performance computing. For example, deployment models for advanced services, such as intrusion detection and prevention systems (IDS/IPS), firewalls, content filters and optimization mechanisms, deep packet inspection (DPI), caching, etc., are typically centered on monolithic platforms installed at fixed locations in or at the edge of the carrier core network. Besides being rigid and static, deployment of advanced network services and connectivity between network and service platforms often lack automatic configuration and customization capabilities, leading to significantly stretched deployment times and large operational complexity. Operational complexity is further aggravated by NSP organizational "silos" - separate teams and software systems manage particular network domains, treating service fulfillment and assurance as separate processes. As a result, troubleshoot times may vary greatly (from hours to days or even weeks).

Efforts to overhaul NSC gained significant traction recently in both research and standardization fora as NSPs seek to offer advanced services beyond basic connectivity, while optimizing infrastructure use and operational efficiency. For example, the IETF contemplates the creation of a dedicated working group on NSC. In this case, Quinn et al. [1] define a service chain loosely as "*the required functions and associated order that must be applied to packets and/or frames.*" Conversely, Zhang et al. [2] motivate the need for "*steer[ing] traffic at the granularity of subscriber and traffic types*" and "*through the right inline service path*" but do not actually define NSC formally. In sum, earlier literature sketches the overall NSC context but falls short of providing a clear and concise definition of what NSC entails in detail.

We fill this gap in the following two sections and discuss our considerations regarding NSC in carrier-grade infrastruc-

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

ture networks. The remainder of this paper is dedicated to the current research directions in NSC, starting with service lifecycle aspects, including service chain description, programming, deployment, and debugging. We then introduce a holistic approach for tackling these issues in line with work planned in the EU-funded FP7 UNIFY project and conclude this paper with an outlook of the NSC area.

II. MOTIVATION

Several drivers are expected to impact network and service infrastructure evolution in the coming years: technological progress in commercial of the shelf (COTS) hardware, cost reductions in processing and storage systems, growing availability of open source software defined networking (SDN) solutions, and "intelligence" migration towards user devices. These drivers will open new business opportunities and increase the competition in the ICT arena. The ossification of IP over transport networks hinders the flexible deployment of new network layer functionality (e.g. routing algorithms) or in-network security services (e.g. firewalls).

Changes to existing network platforms need careful engineering and customizations to address inter-dependencies between functional components and to meet high expectations on quality. Consequently, introducing new functionality into a deployed network is complicated, time-consuming and thus expensive. This rigidity provides few, if any, opportunities for re-tooling the network, and inhibits, in practice, the emergence of new revenue sources. In this context, reducing the "time-to-market" by minimizing the duration of the current network operator innovation cycle is critical.

Complexity compounds as third-party operators, such as wholesale customers, cellular, and cable network operators, call for access to the network platform and demand services beyond what the operator offers in retail. By the same token, access/aggregation carriers are no longer the only customer-facing infrastructure substrate and seek access to other access/aggregation operator's platforms in return, e.g. metro or cable network providers.

Significantly higher degrees of automation within management and configuration tasks could reduce both OPEX and CAPEX. OPEX reduction can be achieved, for example, by reducing the network touch-points (and thus possible configuration mistakes) and by assisting human administrators in configuring and managing equipment. CAPEX reduction can be achieved, for instance, by delaying network resource investments (e.g. by re-factoring and optimizing the use of available resources), through the virtualization of certain network functions that can run on standard data center hardware, and that can be instantiated in various locations, as considered by the ETSI Network Function Virtualization (NFV) [3] effort.

In today's network architectures, increasingly more complex services, such as IPTV, security services, and delivery optimizations, have been introduced through the deployment of middleboxes, both in the operator-controlled network as well as beyond the reach of the carrier in the home network environment. One example of an increasingly complex network platform is the 3GPP Evolved Packet Core (EPC) and its optimizations for content delivery and security. In addition to the typical eNodeB, S/PGW, MME and other network functions [4], an EPC deployment requires the following functions typically installed in independent boxes: i) Network Address Translation (NAT) from private IPv4 addresses to public IPv4/v6 addresses; ii) service access policing, e.g., for VPN, video platforms and VoIP; iii) infrastructure firewall protection; iv) a content distribution network (CDN) solution for efficient popular content distribution; and v) transcoding engines for optimized picture and video delivery.

In this case, customer traffic crosses several middleboxes, which effectively requires operators to define statically-provisioned service chains. Each middlebox is a stateful system supporting a very narrow set of specialized network functions and is based on purpose-built and typically closed hardware. This agglomeration of all kinds of middleboxes contributes to network ossification, and is responsible for a significant part of the network CAPEX/OPEX. As highlighted above, today an operator cannot re-use any of these middleboxes as they are carefully crafted to provide a single service: take one box out and the whole chain breaks. In contrast, the introduction of SDN and NFV in operator networks enables flexible allocation, orchestration and management of L2-L7 network functions and services and provides the substrate for dynamic network service chains.

Early-stage related work in SDN-based service chains originates in OpenFlow demonstrations. For example, OpenPipes [5] explored the possibility of using a modular network component system design in which self-contained in-network functions, such as digital image filters for video content distribution, could be called upon to create a video processing system. However, this work is basically a feasibility study and does not consider a carrier-grade network. Bari et al. [6] survey how virtualization can improve flexibility, scalability, and resource efficiency for data center operators and point to future research directions in that area, but provide few clues on virtualization benefits for telecommunication operators. On the other hand, work such as MobileFlow [7] details a possible way forward for introducing carrier-grade virtualization in EPC, but does not delve into NSC to a significant extent.

III. NETWORK SERVICE CHAINING

In general terms, we define dynamic Network Service Chaining as *a carrier-grade process for continuous delivery of services based on network function associations.* In this context*, continuous delivery* means dynamic network function orchestration and automated (re-)deployment used to improve operational efficiency. *Carrier-grade* means that the entire process is designed for high availability and fast failure recovery, with reliable testing capabilities integrated in every step of the process.

Fig. 1 illustrates how data travels from source to destination with and without the introduction of a dynamic NSC architecture. Today, each and every data packet has to be processed by a predefined series of (often hardware-based) "services" such as a security gateway service, a Deep Packet Inspection service (DPI), a firewall (FW), a load balancer, and so on. Fig. 1 is representative of what NSC principles can accomplish, resulting in a dynamic, software-configurable and upgradable system. NSC provides the means so that data can flow naturally without the intervention imposed by different services residing at different nodes. Network services can be implemented as part of a dynamic chain where each flow is

processed by various service functions thus avoiding the need for deploying different physical network elements. Hence, NSC benefits from virtualization, allowing physical entities, whether manually configured or implemented with software, to be integrated seamlessly at a higher layers.

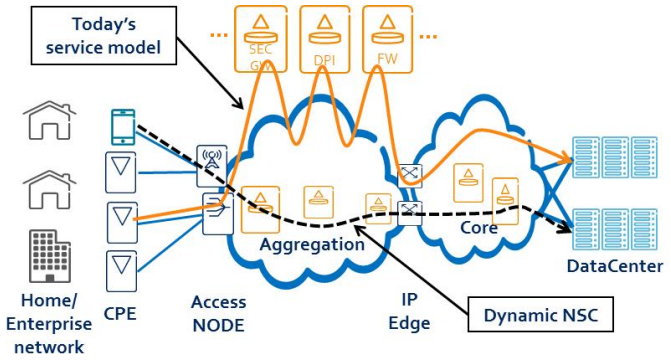


Figure 1: The bold orange line depicts traditional service creation models, following a predefined order of monolithic service elements. The dashed black line depicts dynamic NSC, passing physical and virtual service functions embedded into different network domains.

Another example of the benefits of adopting dynamic NSC is depicted in Fig. 2. When data travels in the same network (e.g. communication between two users who belong in the same network) it goes through different policy enforcement points, balancers, security gateways, traffic schedulers, etc., which may be in software or in hardware depending on the size and the complexity of the network. The goal of these elements is to provide better security and fairness to the end users. However, when traffic has to traverse different network domains, additional operator investment in terms of software and hardware are required, e.g. in the form of a provider edge (PE) gateway which consists of software (different policy elements) and hardware for routing and forwarding the traffic. In these cases, things become more complex and data may experience heavy deterioration in terms of delay depending on the cross-domain network load and the number of the policy elements through which this data is passing. With dynamic NSC implemented in each of the network domains and by exploiting the NSC functionality on the PE elements, it will lead to more intelligent traffic steering and thus provide traffic performance acceleration. Sensitive data and multimedia flows can cross different networks in a reliable and, more importantly, predictable manner when NSC is implemented in the edge of the different network domains.

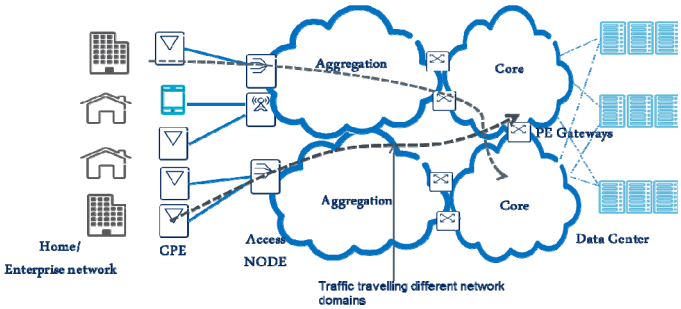


Figure 2 Service provider network interconnection

However, dynamic service chaining and the evolution of network virtualization from data centers into carrier networks do not come without their own challenges. Due to the dynamic nature of the service path, it will no longer be feasible to allow for lengthy discovery processes separating the fulfillment and assurance steps. In addition, each network service chain could evolve automatically to include new service components, if necessary, or shed components that are no longer needed at run time. The flexible bundling of service components customized to individual subscribers will lead NSC operators to manage large numbers of services and service instances. This is unlike the current operational environment, where carriers manage dozens of services, which apply to millions of subscribers. To be able to handle large numbers of flexibly created dynamic network services, we define *continuous network service delivery* as the operator ability to introduce customized services at a rapid pace while maintaining carrier-grade end-user quality of experience.

IV. NSC Research Directions

In order to enable dynamic NSC in future operator networks, several challenges need to be addressed. This section follows the lifecycle of a compound service realized via NSC, which includes description and programming, service instance deployment, continuous network service delivery, as well as security, and identifies the associated research directions.

A. NSC Description and Programming

Network service chains can be considered as particular cases of service composition. As a research topic, service composition has been studied extensively before [8][9]. Research questions which are still open include optimization strategies for decomposition and aggregation of services and service blocks, service modeling languages, design for personalization, mobility, context awareness and adaptation, modeling and enforcement of policies, risk and trust.

Although every link in the service chain could be treated as a service in its own right, it is yet unclear to what extent the *aggregation of service blocks* needs to be performed through interfaces that are highly descriptive or whether simple RESTful interfaces might be more appropriate for the task. There are similarities between characteristics that are desirable for chain links and the netlet concept from the NENA architecture [10].

Recent work in the domain of network programming languages (such as Pyretic [11] and Maple [12]) shows how to implement network functionality by controlling the flow space in an SDN/OpenFlow switch in a programmatic manner. Netcore [13] allows policies to be described in terms of arbitrary functions that cannot be directly realized on physical switches. To handle such policies, the compiler generates an underestimation of the overall policy using a simple static analysis, and then uses partial evaluation to refine this underestimation at run time using the actual packets seen in the network. Less generic solutions, such as SIMPLE [14], have been proposed to address challenges related to mapping towards physical resources and controller visibility into the functionality exposed by a middlebox.

The environment of a dynamic *service chain* calls for *programming languages* that address complex policies in accordance to the packet processing capabilities in the chain links.

For instance, complex functionality, such as caching or intrusion detection, needs programmability constructs that go beyond simple manipulation of flow tables and address handling. As service chains will be deployed on both physical and virtual infrastructures, it is reasonable to believe that virtual switches may in time develop characteristics that are beyond the reach of their physical counterparts in terms of flexibility, feature complexity and frequency in release cycles.

Ways to accurately describe the service characteristics are needed in order to enable automated NSC deployment and optimization. *Service description* needs to cover both the service level and resources involved, from hardware (or a virtual representation) and software point of view. The problem of accurately describing high-level services has been tackled from different angles. The Grid community has focused its efforts on Semantic Grid and OGSA [15]. The cloud computing community has been working on the topic too [16]. Regarding resource description languages with a network orientation, some work is available in the literature (e.g. VXDL[17]). From a NSC perspective, VXDL includes temporal constraints difficult to synchronize between orchestration engines, and not directly supported by resources.

Dynamic service chains are expected to be instantiated in large numbers, likely in the order of the number of subscribers to a particular network. Service and resource description languages need to address such *scalability aspects* natively in order to allow for efficient deployment. Beyond simple temporal constraints, other constraints related to QoS, resource sharing and mobility, security and energy efficiency need to be supported. Particular attention needs to be paid to describing network flows (in OpenFlow terms) that must be forwarded between elements. This task is not trivial because the OpenFlow flow-match-action definition is so rich (and starting with version 1.2, expandable) that the possibilities for aggregating, dividing and defining flows are almost endless.

Virtualization technologies enable resource sharing in a transparent manner between multiple service chain instances. In an OpenFlow context, the OFELIA Control Framework [18], Layer 2 Prefix-based Network Virtualization [19], FlowVisor [20] and FlowN [21] are examples in this respect. Dynamic flow reconfiguration at the architectural level of the infrastructure by different actors owning different service chain instances will have a large impact on scalability requirements (mainly in terms of signaling) which must be addressed. Virtual machine migration can, too, impact behavior at the flow level and lead to sub-optimal resource utilization when the virtual and physical infrastructure descriptions are not able to encompass all applicable constraints. Maintaining optimal resource usage under such dynamic conditions requires adaptive monitoring and optimization approaches, crucial for tracking and optimizing resources between multiple service instances relative to usage and policy constraints.

The definition of a dynamic service chain needs to facilitate *monitoring and problem troubleshooting* for chain instances under live operations. Certain steps were made in this direction, ranging from more theoretical proposals such as [22], to specifying a set of key performance indicators as part of the Service Measurement Index [23] and developing Application Programming Interfaces (API) such as the TMForum Simple Management API [24]. The exchange of service monitoring information is generally well understood and challenges relate more to the shear amount of information to be provided as well as privacy concerns. However, as illustrated by the discussion in the IETF ALTO working group regarding privacy requirements in the exchange of topology data [25], certain information that could be used to facilitate troubleshooting is considered sensitive by the providers. More than a simple modeling exercise, determining what troubleshooting-related information needs to be made available between links in a dynamic service chain requires establishing a balance between the utility of every bit exposed versus the potential business risks. However, in order to facilitate troubleshooting via automated tools, fairly detailed information as well as troubleshooting-related actuators need to become available through programmatic interfaces.

B. Service Instance Deployment

Datacenter networking is typically based on rather homogeneous platforms. In stark contrast, telecommunications gear has traditionally used a mixture of different components, such as network processors, ASICs, and a wide variety of processing elements. Heterogeneity limits platform openness to general programmers. In practice, it is difficult to allow anyone but the network equipment manufacturer to create, install and deploy software on physical devices.

Future network equipment, suitable for NSC, could follow the SDN datacenter model, i.e. migrate toward a *universal node* paradigm, in which the computation and storage resource architecture is mutated from the standard high-volume hardware deployed in datacenters. This would lead the entire path from network edge to datacenter to be seen as a homogeneous programmable platform, enabling software deployment at any place of this (long) programmable path. Similarly, currently monolithic (and complex) functions can be split into several components, each one running at the location that is the best suited for the overall service operation.

As an illustrative example, consider a complex function running on current networks such as a Broadband Remote Access Server (BRAS), which is typically implemented as a dedicated network element with deeply integrated monolithic software. NSC based on universal nodes, on the other hand, would allow function refactoring into modules, with some of them executed at the network edge (e.g., user session termination), others in the core (e.g., content caching), and the rest in the datacenter (e.g., user authentication). From the network operator point of view, this NSC-based BRAS is a unique function, without any reference to the exact physical location of different modules. In principle, automatic dispatchers would optimize the location of each component of this function by (dynamically) relocating each module based on different parameters such as its CPU/memory requirements, the network traffic generated by the communication between different modules, and so on.

While service chaining distributed across the operator network and datacenters is a likely way forward, future physical architectures of network devices are not clear yet, which opens up an entire area of research. For instance, one option is to have virtual network devices that result from the aggregation of several distinct components, such as a network switch combined with a set of traditional servers with processing and storage resources. A second option would be to integrate di-

verse resources (e.g., components specifically targeting network tasks such as network processors or ASICs, plus general purpose hardware such as mainstream CPUs, memory, etc.) in each device. A third option would avoid altogether network-specific components in deployed gear, assuming that the overall performance through the use of solely general purpose hardware is acceptable. In fact, it is worth mentioning that network functions may also include data plane components, i.e., modules that need to inspect, and potentially modify, large amounts of network traffic and therefore benefit from dedicated hardware accelerators in network devices.

A second important research question is independent from the physical architecture of future network devices and relates to the *modular design of network functions*. For instance, at one end of the spectrum we may have a single, monolithic function that can be installed at any network location. On the other end, we can imagine a highly granular partition of the same function into very small components such as regular expression matching, lookup table processing, etc., with each one operating at a different location of a programmable path.

Finally, *mapping service chain components to available resources* in the network is still an open research topic. A mapping function needs to determine where certain blocks may be installed. This component may require additional performance monitoring models that measure/predict resource state and provide input to the mapping functions. When multiple mappings are possible, an optimal solution imposing constraints on the amount of resources to be reserved for these blocks (CPU cycles, memory, network interfaces, physical location and so on) could be chosen, or some other policy may be selected. Such optimization problems have long been known as NP-hard ([26][27][28]) and various heuristics were developed to make them computationally tractable. Methods based on probabilistic approaches, capable of accounting for uncertainty and variations in the network, are promising in this respect [29][30][31].

C. Continuous Network Service Delivery

Service chains may be assembled manually through a user interface or dynamically via algorithmic development. Either way, operators can no longer afford extensive field trials before introducing new services and changes. Instead, they need to be empowered with a toolbox that facilitates daily operations and troubleshooting, in a fashion similar to the DevOps tools gaining popularity in the IT world [32]. DevOps includes tools common to both development and operations teams. In SDN, the connection between implementing network policies and easily determining the source of performance problems was highlighted by Kim and Feamster [33].

DevOps borrows from agile software development methodologies in order to facilitate cross-team communication and greatly increase infrastructure automation. In this context, *workflow definition* for testing, validation and troubleshooting may be considered a challenge. As discussed in [34], even software-defined networks require a fairly complex and time-consuming constructed workflow in order to troubleshoot network functionality using state of the art tools. Dynamic service chains need such workflows to be tailored to their needs which can evolve dynamically. Therefore we need mechanisms to track and incorporate such changes in troubleshooting processes and tools.

NSC also calls for modern *model checking methods* as testing must be performed before a service instance is activated and delivered to the customer. Due to the dynamic properties of the chain, currently defined static methods used today for checking, e.g. connectivity services (ITU T-Y.1564), are not suitable. Software development model checking techniques have recently been employed in an SDN context [35]. However, model checking for dynamic service chains ought to address a series of challenges in terms of the number of instances that could be tested simultaneously on a given infrastructure and duration of tests. The effectiveness of such approaches is currently limited by the state-space explosion due to the composition of service chain segments. Moreover, it would be natural that such checkers are generated on demand and automatically configured to account for policy restrictions for particular chains. This line of work appears very promising as we move forward in NSC research.

As service chains are deployed, we need an infrastructure that dynamically keeps track and can "zoom in" on particular components that could be problematic within a chain, i.e. we would like to have *programmable observation points* within each chain covering both network and service components. With programmable, we mean that the observation points are embedded within the service chain at service definition time alongside other policies. As a result, the infrastructure can perform situation analysis in an autonomic fashion, determine the exact context, decide and react to changing conditions, e.g. when to implement the service instance migration.

Furthermore, programmability also means the possibility to programmatically *assemble basic observation constructs* (such as counters, active measurement capabilities, etc.) into tailor-made tools that can detect specific problems at various service chain components. This type of functionality obviously requires tradeoffs between security, confidentiality, visibility, and resource consumption when it comes to infrastructure internal observation capabilities and the level of exposure toward the customer on the service chain interface. Furthermore, the required dynamicity should be implemented with low signaling overhead towards the infrastructure. Information will be mapped to the troubleshooting workflow, summarized and presented in a manner that makes it easier for human operators to take a decision on further actions.

With respect to the vantage point over the entire set of service chains deployed in the field, *scalable observability* is a major concern. Wide-scale deployment of programmable observation points may potentially lead to huge amounts of monitoring data. In practice, for highly dynamic large-scale systems, a reasonable tradeoff is to avoid reflecting the exact network state information continuously and at a very fine granularity. Network state can be approximated, reaching a balance between estimation accuracy and degree of tolerable uncertainty. In other words, capturing the network state in an accurate, efficient and scalable manner requires monitoring components that are adaptive to changing network conditions - a promising research area in NSC. Scalable and flexible tools for SDN fault management and performance monitoring also require efficient real-time data processing to handle massive amounts of network data, for example by combining probabilistic approaches [36] and big data analytics.

D. Security Considerations

NSC introduces interesting security research topics for different reasons. From a classic security domain point of view, a service chaining process (which takes place in the provider domain) might be triggered by an end user (which is in another domain). Note that today the user domain relies on various legacy access networks that connect to the core, where connectivity is granted after an authentication process such as PPP or serial number matching, with further services requiring additional authentication/authorization processes. These techniques are not very well suited to services composed through dynamic NSC as discussed in this paper.

Short-lived network services, for instance upgrading connectivity performance for a given period (e.g. to offer a "premium" service in terms of network delay for supporting online games while one is connecting to a game central), bringing the service to targeted users in a single subscriber network (e.g., to benefit only one of the home users), or supporting user-service mobility (watching on-demand movies using one's own subscription while visiting friends at their home), are difficult to support with the current schemes. Security demands increase if we take into account the deployment of these services not only on the subscriber's access network, but on visited access networks too. Additional services that will rely on a richer definition of security services include delegating rights to other users, e.g., granting access to a service bundle to another user while the subscriber is travelling. A security architecture for dynamic NSC will have to deal not only with legacy and lower layer protocols used in access networks today, but also shift from a (CPE) device-based authentication model to a user identity based one. To support this vision, research around mixed private/public key architectures that will be able to cope with the huge number of users and interactions expected in dynamic NSC will need to be undertaken.

In the provider domain, the use of static services with proprietary equipment has been beneficial in the past in terms of security. The dynamic aspect of NSC implies that new effort will be needed in deployment design. In fact, it is widely recognized that virtualization technologies can help to control the scope in which services are deployed. Finally, as Quinn et al [1] already detect, but do not elaborate on, there are clear security implications on data and control plane management when deploying NSC which future research ought to consider.

V. Future Work

After detailing the relevant research directions in NSC, we take the opportunity to introduce the EU-funded FP7 project UNIFY, which sets out to tackle many of the issues mentioned above through a holistic approach [37]. The core project aim is to provide the means for flexible service creation within the context of unified cloud and carrier networks, especially focusing on network functions. Specifically, UNIFY finds current carrier networks to be slow, rigid in terms of functions and resources, and inflexible with respect to service creation. Thus, UNIFY envisions an architecture where the entire network from home devices to data centers forms a unified production environment. As a consequence, an NSP can distribute functions and state anywhere in the network, aided by automated orchestration engines. In other words, UNIFY envisions an automated, dynamic service creation platform, leveraging fine-granular NSC.

To accomplish such a unified production environment, the project will focus on four key aspects. First, UNIFY will consider the network services for a converged fixed-mobile network and study the decomposition of these traditional network functions into more fine granular components. UNIFY will identify the minimal set of components which, once in place, can provide more flexibility for network service chaining. Second, UNIFY will define a service abstraction model and a proper service creation language suitable for dynamic NSCs. This includes aspects dealing with orchestration and network function placement optimization through novel algorithms, enabling the automatic placement of networking, computing and storage components across the infrastructure. Third, in the framework of Service Provider DevOps, new management technologies will be developed, based on the experience from data centers, and integrated into the orchestration architecture, addressing the challenges of dynamic service chaining. Finally, UNIFY will evaluate the applicability of a universal node based on commodity hardware in order to support both network functions and traditional data center workloads, with an investigation of the need of hardware acceleration.

VI. Conclusion

This paper discussed network service chaining in the context of future infrastructure networks. After illustrating how service chains are crafted today, we motivated the need for dynamic NSC and presented how service chains can be employed in future operator networks in order to provide cost reductions, increased flexibility, and time-to-market acceleration throughout the network. We then went through our design considerations for NSC, including the key role it can play in accelerating the design, implementation and deployment of novel service offerings in infrastructure networks as well as the potential for carrier CAPEX and OPEX reduction. The key contribution of this paper is a detailed array of research directions in the context of NSC, including service instance deployment, network service definition, programming, and operations, as well as the concept of continuous network service delivery. Finally, we summarized how the EU- FP7 UNIFY project aims to address some of the research challenges introduced in this paper.

Dynamic NSC is a very promising research area with several topics that will need to address challenges which hitherto were unknown in telecommunications networks. For example, service chain "debugging" in an NSC era and the corresponding network fault isolation processes today entail completely different aspects. Therefore, research results towards dynamic NSC will have significant impact in the way we design, operate, and maintain networks in the coming years. Furthermore, in contrast to major players in datacenter networking, network infrastructure carriers prefer solutions that are interoperable and have global reach and applicability. As such, we also expect that as research in NSC matures and is demonstrated to work well in practice, some of the NSC focus will be diverted towards interoperable solutions across operator networks and thus international standardization.

ACKNOWLEDGMENTS

This work was conducted within the framework of the FP7 UNIFY project, which is partially funded by the Commission of the European Union. Study sponsors had no role in writing this report. The views expressed do not necessarily represent the views of the authors' employers, the UNIFY project, or the Commission of the European Union. We thank Fritz-Joachim Westphal, Ioanna Papafili, Róbert Szabó and Pontus Sköldström for constructive comments as well as all partners who contributed in the UNIFY preparation phase.

REFERENCES

[1] Quinn, P., Kumar, S., Agarwal, P., Manur, R., Chauhan, A., Leymann, N., Boucadair, M., Jacquenet, C., Smith, M., Yadav, N., Nadeau, T., Gray, K., McConnell, B., Glavin, K., "Network Service Chaining Problem Statement," IETF Internet draft, Informational, Aug. 2013

[2] Zhang, Y., Beheshti, N., Beliveau, L., Lefebvre, G., Manghirmalani, R., Mishra, R., Patney, R., Shirazipour, M., Subrahamaniam, R., Truchan, C., Tatipamula, M., "StEERING: A Software-Defined Networking for Inline Service Chaining," Proceedings of. IEEE ICNP 2013, Goettingen, Germany, Oct. 2013.

[3] European Telecommunications Standards Institute (ETSI)," Network Function Virtualization", http://www.etsi.org/technologies-clusters/technologies/nfv , retrieved Okt. 21, 2013

[4] Balbas, J.-J.P.; Rommer, S.; Stenfelt, J., "Policy and charging control in the evolved packet system," Communications Magazine, IEEE, vol.47, no.2, pp.68-74, February 2009.

[5] Gibb, G., et al., "OpenPipes: Prototyping high-speed networking systems," Proc. SIGCOMM (Demo), Barcelona, Spain, 2009.

[6] Bari, M., et al., "Data Center Network Virtualization: A Survey," IEEE Commun. Surveys & Tutorials, vol. PP, no. 99 (early access article).

[7] Pentikousis, K., Wang, Y., and Hu, W., "MobileFlow: Toward Software-Defined Mobile Networks", Communications Magazine, IEEE, vol. 51, no. 7, pp. 44-53, July 2013.

[8] ter Beek, M.H., Bucchiarone, A., and Gnesi, S., "Formal Methods for Service Composition". Annals of Mathematics, Computing & Teleinformatics 1, 5 (2007), 1-10.

[9] Strunk, A., "QoS-Aware Service Composition: A Survey," Web Services (ECOWS), 2010 IEEE 8th European Conference on , vol., no., pp.67,74, 1-3 Dec. 2010

[10] Martin, D., Völker, L., Zitterbart M., "A Flexible Framework for Future Internet Design, Assessment, and Operation". Computer Networks, Vol. 55, No. 4, ISSN 1389-1286, pp. 910-918, Mar 2011.

[11] Monsanto, C., Reich, J., Foster, N., Rexford, J., Walker, D., "Composing software-defined networks," Proceedings of NSDI'13, Berkeley, CA, 2013

[12] Voellmy, A., Wang, J., Yang, R., Ford, B., Hudak, P., "Maple: simplifying SDN programming using algorithmic policies", SIGCOMM 2013

[13] Monsanto, C., Foster, N., Harrison, R., Walker, D.,"A compiler and run-time system for network programming languages,".POPL '12, 2012

[14] Qazi, Z., Tu, C., Chiang, L., Miao, R., Yu, M., "SIMPLE-fying Middlebox Policy Enforcement Using SDN". SIGCOMM, 2013.

[15] Foster, I., Kishimoto, H., and Savva A., editors (24 July 2006). "The Open Grid Services Architecture, Version 1.5". Open Grid Forum. Retrieved 15 September 2013.

[16] Sun, L., Don, H., Ashraf, J., "Survey of Service Description Languages and Their Issues in Cloud Computing," Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on , pp.128,135, 22-24 Oct. 2012

[17] Koslovski, G.P., Primet, P.V.B., and Charao, A.S.,. VXDL: Virtual Resources and Interconnection Networks Description Language, in 2nd International Conference on Networks for Grid Applications, 2008

[18] "OpenFlow in Europe" OFELIA FP7 Project, OFELIA Control Framework, http://fp7-ofelia.github.io/ocf/

[19] Matias, J., Tornero, B., Mendiola, A., Toledo, N., Jacob, E., "Implementing Layer 2 Network Virtualization using OpenFlow", EWSDN, 2012

[20] Sherwood, R., Gibb, G., Yap, K., Appenzeller, G., Casado, M., McKeown, N., Parulkar, G., "Can the Production Network Be the Test-bed?," OSDI 2010

[21] Drutskoy, D., Keller, E., Rexford, J., "Scalable Network Virtualization in Software-Defined Networks," IEEE Internet Computing, vol. 17, no. 2, pp. 20-27, March-April, 2013

[22] Prieto, A.G.; Dudkowski, D.; Meirosu, C.; Mingardi, C.; Nunzi, G.; Brunner, M.; Stadler, R., "Decentralized In-Network Management for the Future Internet," Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on , 14-18 June 2009

[23] CloudCommons, Introducing the Service Measurement Index, Cloud Service Measurement Initiative Consortium, 2012.

[24] TMForum TR-196. Multi-Cloud Service Management Pack - Technical Guide.
[Online] http://www.tmforum.org/TechnicalReports/865/home.html.

[25] Stimerling, M., Kiesel, S., and S. Previdi, "ALTO Deployment Considerations", draft-ietf-alto-deployments-08 (work in progress), October 2013.

[26] Fernandez-Baca, D., "Allocating modules to processors in a distributed system", IEEE Trans. Software Energy. 15(11), 1989, 1427-1436.

[27] Wang, Z., and Crowcroft, J., "Quality of Service Routing for Supporting Multimedia Applications," in JSAC, vol. 14. Institute of Electrical and Electronics Engineers, Sept. 1996.

[28] Gu, X., Nahrstedt, K.; Chang, R.N.; Ward, C., "QoS-assured service composition in managed service overlay networks," Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on , pp.194,201, 19-22 May 2003

[29] Elmroth, E., Tordsson, J., Hernández, F., Ali-Eldin, A., Svärd, P., Sedaghat, M., and Li, W., "Self-management challenges for multi-cloud architectures", Towards a Service-Based Internet, pages 38–49, 2011.

[30] Gonçalves, P., Shubhabrata, R., Begin, T., and Loiseau, P., "Dynamic Resource Management in Clouds: A Probabilistic Approach".IEICE Transactions on Communications, 95(8):2522–2529, 2012.

[31] Konstanteli, K., Cucinotta, T., Psychas, K., Varvarigou, T., "Admission Control for Elastic Cloud Services". In Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pages 41–48. IEEE, 2012.

[32] Rockwood, B., "The DevOps Transformation", keynote at the USENIX LISA'11 conference, https://www.usenix.org/conference/lisa11/devops-transformation, retrieved Aug 1, 2013

[33] Kim., H., Feamster, N., "Improving network management with software defined networking," Communications Magazine, IEEE , vol.51, no.2, pp.114,119, February 2013

[34] Heller, B., Scott, C., McKeown, N., Shenker, S., Wundsam, A., Zeng, H., Whitlock, S., Jeyakumar, V., Handigol, N., McCauley, J., Zarifis, K., Kazemian, P.,."Leveraging SDN layering to systematically troubleshoot networks,". Proceedings of HotSDN '13, Hong Kong, Aug. 2013

[35] Canini, M., Venzano, D., Peresini, P., Kostic, D., Rexford, J.,"A NICE way to test OpenFlow applications," Proceedings of NSDI, San Jose, CA, Apr. 2012

[36] Prieto, A. G., Gillblad, D., Steinert, R., Miron, A.,"Toward decentralized probabilistic management," Communications Magazine, IEEE, 49(7), 80-86, 2011

[37] Császár, A., John, W., Kind, M., Meirosu, C., Pongrácz, G., Staessens, D., Takács, A., Westphal, J., "Unifying Cloud and Carrier Network," Proceedings of. DCC, Dresden, Germany, to appear Dec. 2013

Some Controversial Opinions on Software-Defined Data Plane Services

Fulvio Risso
Dept. of Control and Computer Engineering
Politecnico di Torino
Torino, Italy
fulvio.risso@polito.it

Antonio Manzalini
Telecom Italia Strategy
Future Centre
Torino, Italy
antonio.manzalini@telecomitalia.it

Mario Nemirovsky
ICREA Research Professor
Barcelona Supercomputing Center
Barcelona, Spain
mario.nemirovsky@bsc.es

***Abstract*—Several recent proposals, namely Software Defined Networks (SDN), Network Functions Virtualization (NFV) and Network Service Chaining (NSC), aim to transform the network into a programmable platform, focusing respectively on the control plane (SDN) and on the data plane (NFV/NSC). This paper sits on the same line of the NFV/NSC proposals but with a more long-term horizon, and it presents its considerations on some controversial aspects that arise when considering the programmability of the data plane. Particularly, this paper discusses the relevance of data plane vs control plane services, the importance of the hardware platform, and the necessity to standardize northbound and southbound interfaces in future software-defined data plane services.**

I. Introduction

The idea of transforming the network into a programmable platform is probably one of the hottest topics in the current research domain, which originates from the impossibility to deeply change the behavior of current network devices. In fact, in most cases only the manufacturer of the network equipment has the privilege to create the software that controls the device itself, while the possibilities for any other actor (e.g., a network operator) are more limited. In fact, a network operator can only *configure* the software already provided by the network manufacturer (changing parameters, choosing a routing protocol instead of another, etc.) but it cannot directly install *its own* software on the network device, such as a routing protocol customized for its particular environment.

Among the approaches that have been proposed so far toward a greater flexibility, we can cite Software-Defined Networks, Network Functions Virtualization [1] and Network Service Chaining [2]. Software-Defined Networks (SDN) represent a new architectural model in which the control plane is transformed into a programmable entity and is decoupled from the data plane. Instead, Network Functions Virtualization (NFV) and the IETF Network Service Chaining (NSC) proposals can be considered fairly orthogonal to SDN and aimed at simplifying the complex data plane processing path present in network operator's networks. Although a more in-depth discussion of SDN and NFV/NSC is left for Section II, we can summarize how SDN is more oriented to *control plane* programmability, while NFV/NSC focuses on *data plane* functions. Briefly, control plane refers to the set of functions that influence how packets are forwarded to the destination; for instance, the control plane faces the problem of *managing network paths* between source and destination hosts. Vice versa, data plane refers to the set of functions that can inspect, and potentially modify, the content of the packets in transit, i.e. it faces the problem of *processing each single packet*.

This paper discusses the problem of customizing data plane services with a view limited to a *single network device*, and assumes that future NFV/NSC solutions will allow users to deeply change the behavior of the data path of the network, e.g., by installing and running custom applications that operate on an arbitrary portion of the network traffic. In case of such of this event, we speculate that some assumptions that may be valid for the SDN world may no longer be appropriate when deep data plane programmability comes into play.

This paper presents the personal (and potentially controversial) opinion of the Authors on some issues related to the future software-defined data plane services, namely the importance of a programmable data vs control plane (Section III), the necessity of the network hardware to evolve (Section IV) and the necessity to define standard interfaces for future data plane services (Section V). The paper includes also an introduction to the existing SDN and NVF/NSC concepts (Section II), and a final section (Section VI) that summarizes current findings and presents some conclusive remarks.

II. Background

A. Software-Defined Networks

Software-Defined Networks are based on the separation between the *control* and the *data* plane of the network. The former, which is supposedly where most of the intelligence is, is transformed into an open programmable platform that can potentially host any network control application, usually provided by the network operator. This may allow different actors (e.g., network operators) to finely control the forwarding decisions taken in any portion of their network and implement the best traffic forwarding strategy according to their necessities. As a consequence, SDN can enable the implementation of smart algorithms e.g., to balance traffic across different links based on several criteria such as the sender/receiver of the traffic, the application, or anything else that is considered useful for the network operator.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

SDN predicates that the control is a logically centralized function (although the implementations may be distributed) and it should be programmed through a set of well-defined interfaces, possibly standard, which could allow independent developers to execute their applications on different controllers.

While SDN represents definitely an important innovation in the networking area, it limits its scope to the control plane. In other words, it offers interesting possibilities to control the path traversed by a generic network flow, but it is not appropriate for data plane-oriented tasks that are very common in nowadays networks, such as many functions implemented by dedicated middleboxes (e.g., firewalls, network address translators, web caches, etc.) often placed at the edge of the network.

B. Network Functions Virtualization and Network Service Chaining

Network Functions Virtualization (NFV) focuses on the problem of consolidating and optimizing the processing of the network traffic that needs to traverse many middleboxes mentioned before, which is an increasingly important problem particularly in network operators' networks. In fact, new network services often require the traffic to traverse a large set of those boxes, each one implementing a specific function, with a huge impact in terms of costs (CAPEX, power, management, physical space), reliability and complexity of the network.

NFV is based on flow processing [3] and proposes to implement in software the network functions that today run on proprietary hardware, leveraging high-volume standard servers (e.g., Intel-based blades) and IT virtualization. This potentially enables greater flexibility and reduces costs, e.g., by consolidating several functions on a few physical servers. NFV can exploit control plane technologies such as OpenFlow [4] to dynamically reprogram the paths of network flows and allow them to traverse exactly the set of components (called *functions* or *applications*) that are needed for the selected service.

Network Service Chaining (NSC) is currently an unofficial IETF working group still in the embryonic Birds-of-a-Feather (BoF) discussion stage, whose vision looks similar to NFV. In fact, NSC goal is the standardization of the general architecture and the building blocks that are required to create network service chains, such as protocols (for setting up, configuring and managing the service chain), metadata (for passing additional information among different services) and more.

Although NFV/NSC are oriented to the data plane services, currently they do not enter into the detail of the implementation of each single function. For instance, a possible implementation consists in having applications running as virtual machine images, executed on hypervisors installed on standard servers. This allows to exploit the best of both worlds, i.e., servers execute applications, and network devices forward traffic, but that in this case data plane processing blocks are not *integrated* in the network device, but are located at its *border*, which may be a possible source of inefficiency. However, this possible implementation can be seen as a pragmatic move toward a (near) future with customized applications running on the data plane, enabled by an architectural model that leverages the highest possible number of existing components instead of starting the design from a blank new sheet. For instance, the reuse of existing components is one of the explicit goals of NSC. In essence, the architectural model proposed to implement the NFV/NSC in the near future is shown in the block in the middle of Figure 1: flexible data plane processing is not achieved by changing the architecture of the network device and integrating the new customized data plane applications in it; instead, service chains are created by establishing a closer collaboration between network equipment and mainstream computing platforms. Finally it is worth mentioning that the NFV/NSC architecture supports distributed processing chains, in which services are installed in different locations, even in a remote datacenter, and then chained in the proper order.

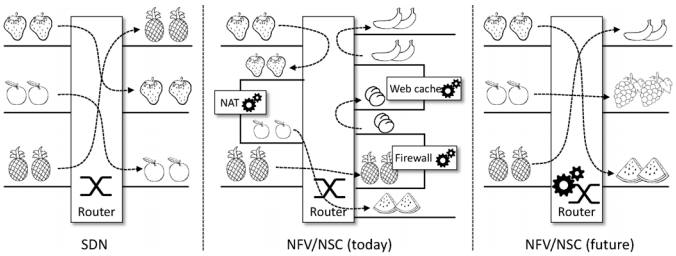


Fig. 1. An overview of the SDN and NFV/NSC (present and future) paradigms with respect to data plane functions, focusing on a single node.

C. A long term view on NFV/NSC

This section focuses on the implementation of the single functions defined in NFV/NSC on a long term horizon. Although predicting the future is always a difficult activity, we can foresee the presence of at least two fundamental differences from today. First, we foresee that future data plane architectures will spot a deeper integration between networking and general purpose processing components, possibly deeply integrated within the same network box[1]. Second, we foresee that new network devices will allow third parties (e.g., network operators) to install their own data plane applications directly on network device. This new software could either complement the standard data plane processing code provided by the device manufacturer with new functions or could replace it totally, and it will potentially operate on the all the packets flowing through the network device. In fact, several proposals in this direction have been made so far; among the others we can cite Click [5] and some recent papers such as [6], [7], [8], [9] and [10].

While everybody agrees that future network devices should become hybrid platforms that can execute both traditional networking tasks and new complex data plane processing applications, the hardware architecture needed for those new network

[1]Although many network manufacturers already offer network equipment that supports also general-purpose processing blades, currently those components are not properly integrated with the other data plane modules.

devices is still under debate. Options range from replacing the currently dedicated network hardware with standard high-volume components (e.g., clusters of servers equipped with mainstream CPUs) to the creation of hybrid devices that integrate both dedicated networking components (e.g., special purpose ASICs) and general purpose linecards.

Independently from the resulting hardware implementation of future network devices, the different approaches (control vs. data plane) taken by SDN/NFV/NSC may suggest the necessity of different hardware architectures, which are summarized in Figure 1. For instance, an SDN router may appear as a simple (dumb) switch (indeed, the intelligence is in the control plane), a NFV/NSC router may look like a (dumb) switch connecting several (smart) modules that operate on the data plane, while a future data plane router should resemble to a more integrated device, including switching and customized data plane processing functions in the same box. As a consequence, we foresee that future routers for data plane services will become smart devices with the capability to deeply modify the network packets, such as changing the value of some fields (e.g., in NAT applications) or even implementing complex data plane applications such as transparent web proxies, WAN accelerators, and more.

III. On the relevance of the data vs. control plane

SDN speculates that the network control plane is the place where the intelligence should be, while the data plane should look similar to a fast (but dumb) switch. As a consequence, SDN assumes that the control plane is much more relevant than the data plane because of the presence of more high-level functions.

While not questioning that the control plane is where most of the network intelligence would be, we would like to analyze the relevance of the data vs. control plane by looking at the point of view of two different entities, namely *network operators* and *end users*. With *network operators* we intend all the entities that are in charge of operating a network, such as companies whose business consists in selling network services (e.g., connectivity) to end users, companies that own a datacenter and sell added value services to third parties, and more. Vice versa, with *end users* we intend all the entities that buy network connectivity and, possibly, services from network operators, such as domestic ADSL users or corporate networks connected to the Internet.

This section makes use of three steps to present the personal view of the Authors on the relevance of the control vs. data plane. First, we analyze the relevance of the data vs. control plane from the point of view of the network operator. Second, we move our focus to end users and we discuss the possibility of giving them the privileges to to customize the behavior of the network (being it the control or the data plane). Finally, we discuss the relevance of the control vs. data plane focusing on the point of view of the end user.

A. The view from the network operator

The capability to customize the control path of the network represents a big value for a network operator, which can use this technology to optimize its infrastructure, create overlays or partitions in the network (e.g., Virtual Private Networks), avoid bottlenecks, and more. In essence, a clever control plane allows the network operator to improve its costs and to provide new services, although, from private conversations with some network operators, Authors infer that the first objective (i.e., reducing costs) looks more important than the second (i.e., providing new services).

Similar advantages can be achieved by a programmable data plane as well. The network provider can improve its operating cost by consolidating and optimizing many functions currently in use (firewalls, intrusion detection systems, network monitors, transparent web caches, etc.). Furthermore, it can introduce new services with shorter setup time and higher customization capabilities, such as parental controls, personal security software, network mobility solutions (e.g., LISP [11]), etc. For instance, a recent study [12] on the Italian market, which accounts 41.37M Internet users and 60.9M inhabitants at December 2012, evaluates the potential customers and the corresponding revenues coming from the five possible data plane services listed in Table I, all belonging to the security domain. The result is that those services alone have the potential to bring to the telecom operators more than 500M€, on a market that is estimated at about 60 billions euro/year. Although this preliminary number does not account the cost of delivering the service, nor the fact that a potential customer may decide not to buy the service, it shows that the introduction of a programmable data plane can offer to the telecom provider interesting opportunities for new added value services.

Although previous numbers should be considered as ball-park estimations[2], they seem to suggest that data plane customization may become an important source of revenues for the network operator, possibly originating a bigger economic impact than control plane customization.

B. User customization of the control/data plane

Historically, network manufactures looked suspiciously at the idea of allowing other actors (e.g., network operators) to create and install their own software on their network devices. In addition to the several reasons that come from the business side, we can cite the problem of guaranteeing the integrity of the "core" functions of the devices, which should run unaffected by the behavior (including bugs) of the external applications.

Nevertheless, due to market demand, some network equipment manufacturers recently opened (partially) their boxes to other actors, but, interesting, the previous problem did not

[2]For instance, in addition to the sources of uncertainty related to the economic impact of data plane programmability already mentioned in the text, other potential issues are the lack of estimates about the revenues that may come from a customized (programmable) control plane, and the cost reductions achievable through programmable control/data planes.

TABLE I
POSSIBLE YEARLY REVENUES FOR A TELECOM OPERATOR FROM NEW DATA PLANE SERVICES (ITALY ONLY)

Market description	Potential users	Pricing	Revenues
Customers (families) interested in a parental control software	1383180	30€/year	41.5M€
Customers interested in personal security software (e.g., personal firewall, etc)	18281920	15€/year	274.2M€
Corporate mobile users interested in mobile protection software	787913	50€/year	39.4M€
Corporate users that adopt the BYOD paradigm, interested in mobile protection software	2412964	50€/year	120.6M€
Companies interested in operator-based protection software (e.g., corporate firewall)	52414	1000€/year	52.4M€
Total			528.1M€

disappear. In fact, the ball is now in the hands of network operators, which prefer not to allow any other entity but themselves to install additional software on the network equipment.

In any case, the Authors believe that long-term plans for control/data plane programmability should allow the software *coming from arbitrary sources* (as proposed in [6]) to change the behavior of the network, either control or data plane, for the same reasons (both technical and economic) presented before. In fact, the future of control/data plane programmability should be the capability to offer to *several parties* (network providers, service providers, end users) the possibility to install and execute *their* software on the network and hence deeply influence its behavior. According to this model, user applications can target both control plane (e.g., changing forwarding paths) and data plane functions (e.g., drop e-mails with malicious attachments).

While the idea of allowing any entity to install its own application on the network may look dangerous because it may break the network itself, we can note that a similar behavior represents the common practice in the computing world, in which the entity that operates the hardware infrastructure (such as in public datacenters) may be different from who installs and operates the applications. In our opinion, it is just a matter of which *permissions* we grant to the users and how we enforce the *control* on their actions, not *if* they are allowed to do so. For example, we foresee that users could install applications such as personal firewalls in the network, operating only on the traffic generated by all the devices of the users itself, and more.

C. The view from the final end user

While the capability to customize the network paths can represent a value for the telecom operator, we believe that in most cases this represents an insignificant *detail* for a typical end user. In fact, end users usually expect their traffic to be delivered to the destination and they do not care about the path traversed by their packets.

Vice versa, they may be interested to install their applications on a programmable data plane, which enables them to relocate some existing functionalities (firewalls, intrusion detection systems, protocol translators, VPNs, etc.) in the network, and potentially add even new applications. From the point of view of the end user, there are two advantages in this model: (i) users have the freedom to install the data plane application they want, without being limited by the ones offered by their network operator, as in it would be in the scenario of Section III-A, and (ii) users do not have to deal with the hardware/deployment details required by those applications, which will be delegated to the network operator.

As a consequence, we believe that a programmable data plane is in general much more relevant than a programmable control plane and that the introduction of user-customizable capabilities in the data plane of the network would be noticed by the end user; hence it may represents an additional selling value for the network operator.

IV. ON THE IMPORTANCE OF THE HARDWARE IN FUTURE DATA PLANE SERVICES

A widespread opinion in the SDN world states that, as most of the intelligence resides in the control plane, the data plane will soon become a commodity, hence almost irrelevant for delivering future services.

However, if we move our focus to data plane services, this conclusion may no longer be valid because the levels of performance required for future data plane services will definitely be a challenge, not achievable with the hardware in use today. For example, telecom operators currently aggregate up to 10-20K ADSL users on a single network node that, taking into account a speed of 20Mbps per user, leads to an aggregated bandwidth of about 200Gbps. It is important to notice *(i)* that this number is expected to grow (more users could be aggregated in the future, and link speed could become larger), and *(ii)* that we are focusing on deeply programmable data plane applications, which may be required to perform custom processing (not just simple forwarding) on each single packet. This seems to suggest that the hardware may become less important for the general public, which will focus on high-value functions and care less about the underlying details (as today in the computing world), but it should definitely evolve in order to support future data plane services.

In fact, if we take a look at the world of computing, which shares many similarities with future data plane services, we can observe that the hardware has evolved considerably even in the last few years, although exploiting the advantages of the scale economies. For instance, thanks to the *standard high-volume* approach, we can use the same hardware in many different application fields (*standardization*) and we increase the number of devices that rely on the a few (sophisticated) components (*high volume*), reducing the overall cost of the system.

In fact, although most people think that current CPUs are just faster than some years ago, among the many innovations

we had in the general purpose processing hardware, we can cite virtualization primitives that allowed to execute virtual machines more efficiently, 64-bit instructions that enabled applications to deal with huge amount of data, efficient locking primitives for shared data; furthermore, dedicated accelerators such as graphics processing units (GPUs) and vector processors, are becoming increasingly common and are increasingly part of the capabilities of a general purpose CPU.

Similarly, we expect that the hardware needed to deliver future data plane services will be very different from today and that future network equipment need to evolve considerably. Particularly, if we assume that the end user can customize the data plane (Section III-B), we should expect a new breed of applications, possibly rather different from today network software, whose requirements could introduce additional pressure on the necessity to evolve the hardware. Probably, designing future network equipment would not be perceived as cool as creating fancy applications, but this would not mean that the hardware will become irrelevant.

V. On the standardizing the Northbound and Southbound interfaces in future data plane services

The standardization of the northbound and southbound interfaces are hot topics in SDN and NSV/NSC. We define the northbound interface (NBI) as the set of APIs that allow developers to create software for a given platform. Vice versa, the southbound interface (SBI) is the set of APIs that allow that code to be instantiated on the physical hardware.

This section presents some thoughts about the standardization of NBI and SBI for future data plane service, introduced by the analysis of how a similar problem has been solved in the world of the general-purpose computing. Although the general purpose programming model may not fit perfectly the necessities of future data plane services, it represents a very strong candidate and, most likely, it could be the model that will be used first.

A. The programming model in general-purpose computing

The programming model in general-purpose computing, which is well understood and has been proved to be quite effective over the years, leverages the different layers and components shown in Figure 2. User *programs*, which implement the application-layer logic, are written in one of the many existing *languages* and make use of additional *libraries* that facilitate the implementation of specific functions. Those three components, with the help of the proper *compilers*, are used to create an executable that, leveraging the services (and, possibly, some virtualization functions) exported by the *operating system*, can run directly on the *hardware*.

If we want to introduce the NBI and SBI concepts in the general-purpose computing model, we could identify the NBI as the set of programming languages and libraries used by programmers, while the SBI is the interface represented by the Operating System API and by the hardware, which includes the CPU instruction set as one of the main components.

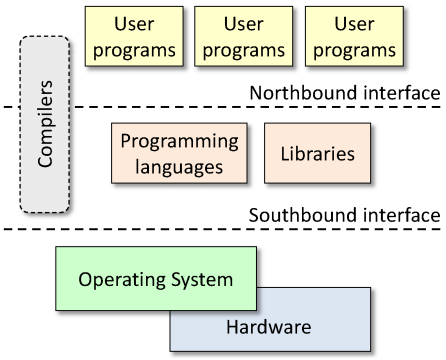


Fig. 2. The programming model used in modern computing environment.

The presence of a standard interface could guarantee many advantages, interoperability among the others, which is one of the reasons why the networking community is pushing for the standardization of NBI and SBI. However, we can note that in the computing world no standards exist for NBI and SBI and this did not prevent the creation of a very active software ecosystem.

Although the Authors believe that the standardization has many clear advantages, particularly in case of large networks where the heterogeneity of the hardware platforms is the common practice, we present in the following some motivations (both technical and economic) that suggest that NBI and SBI may not standardized in future data plane services, similarly to what happened in the general-purpose computing world[3].

B. Northbound interface

The northbound interface should be driven by the applications. Back to the computing world, we invented many programming paradigms (procedural languages, event-driven languages, functional languages, etc.) and many languages (e.g., Fortran, C, C++, Java, Python, SQL, etc.), each one probably representing the best choice in a given condition, such as resource-constrained environments, client-server programming, artificial intelligence applications, etc.. Moreover, since applications (and business necessities) evolve over time, also languages and paradigms evolved, hence new languages were defined, other were abandoned, other where enriched with new functions, and more.

Similarly to what happened in the computing world, we expect not only that the NBI should evolve over time, but that there should be many “northbound interfaces”, each one being the most appropriate for a different business need. On the other way, we believe that the standardization of the NBI could limit the evolution of data plane services and prevent people to exercise their creativity and envision new applications. For instance, the lack of a standardized NBI (and the freedom to invent new “interfaces” when needed) was probably one of the key that enabled the continuous evolution that characterized the computing world.

[3]In fact, another possible outcome is that a standard will be defined but it will not be used in practice, which happened many times in the past.

C. Southbound interface

The southbound interface, being more close to the hardware, is probably less "cool" than the northbound but nevertheless it represents the "workhorse" that allows high-level data plane applications to work.

In the ideal world, a standardized SBI could allow the same data plane application to be executed on any network device. However, the computing world does not have a standard interface here, as each hardware platform has its own characteristics (e.g., CPU instruction set) in addition to the different API exported by the operating system. Nevertheless, we have several options to create portable software, such as source code recompilation, "cross-platform" languages (e.g., Java), interpreted languages, etc. The experience of the computing world suggests that, while a standard SBI interface would be useful for application portability, in practice the same result can be achieved with other technologies. Although the problem of code portability with respect to data plane functions seems to be more difficult to solve because of the heterogeneity of the networking platforms, we have no evidence that the same solutions that work in the computer world would not apply to the networking world as well.

On the other side, there are at least two reasons against the standardization of the SBI. First, from the business point of view, a standard SBI could be against the interest of the (major) network manufacturers, which would lose the possibility to differentiate their products from competitors. Second, on the technical side, a standard SBI could limit the degree of innovation that we can introduce in the hardware/operating system. For instance, the Authors believe not only that the hardware should evolve with the new additional primitives needed to offer a better support to future data plane applications (as presented in Section IV), but that the same applies to the operating system that supervises the network box. This could be achieved much more efficiently if several manufacturers compete to improve their products and are not tied to a fixed and the-same-for-all interface.

D. The role of Openflow in the future SBI

The OpenFlow protocol has been proposed as a possible southbound interface by the SDN community, hence we may wonder if it may be an option for data plane services as well. This section motivates why OpenFlow is not appropriate in case we would like to define a SBI for data plane services.

OpenFlow was defined by the networking community in order to increase the flexibility of current networks. The networking community focuses on the whole *network*[4] and it consider a network device as a black box that takes packets from the input ports and forwards them to the most appropriate output port. According to this network-centered view, a network switch could be modeled by a simple lookup table that, in fact, is what OpenFlow does: a network device becomes a lookup table[5] that can be reprogrammed by an external software, hence achieving very flexible forwarding policies.

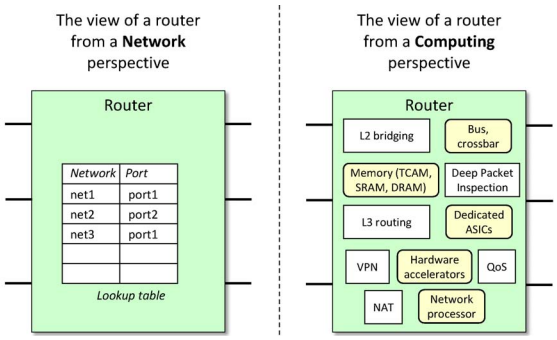


Fig. 3. Different views of a router.

However, if we change our perspective to the one of a computer engineer, the view of the network device would be tremendously different, as depicted in the right side of Figure 3. For instance, the router would be a collection of functions (L2 bridging, L3 forwarding, VPN concentrator, NAT, QoS, etc.) and of functional blocks (CPU, network processors, memories, hardware coprocessors, etc.). When looking from this angle, not only a network device requires a model that is more complex than a simple lookup table, but deriving a suitable model may not be straightforward at all. In essence, we cannot define a standard SBI until we have a suitable model of the network hardware, but the definition of a comprehensive model that satisfies the necessities of data plane applications is still a rather unexplored research topic and no clear solution is visible on the horizon.

As a consequence, the OpenFlow protocol is not a suitable SBI for data plane applications, as it would not be able to exploit the (required) hardware features present in data plane-oriented network devices, although it may be sufficient in some environments where network devices are requested mainly to forward traffic, such as the case of datacenter switches.

VI. Conclusion

This paper presents the opinion of the Authors on some (controversial) aspects that, when examined with a data plane perspective, may look different compared to the the view from the control plane world. Particularly, this paper focuses on three aspects.

First, it argues that, although the control plane received much more attention in the past, a programmable data plane may be more valuable than the control plane because it is more visible by the end users, which may be willing to pay for the possibility to customize its services. Second, it suggests that building the hardware that is needed to provide future data plane services is definitely a challenge, which looks the opposite compared to the vision suggested by the SDN paladins, i.e., that future networking gear will become

[4]Although this statement looks obvious, please do not underestimate its importance, which will be clarified in the following.

[5]In fact, more recent versions of OpenFlow (*gt* 1.0) model a router with a set of lookup tables. While this may be more appropriate for some cases, still does not capture the complexity of a modern router.

commodity. Third, it suggests that the standardization of the northbound and southbound interfaces, which is receiving a great attention in the SDN/NFV/NSC worlds, may not be successful for both business and technical reasons.

Although the Authors agree that some aspects presented in this paper may be controversial, they hope that that their personal opinions will be useful to foster the discussion on programmable data plane issues in future networks.

ACKNOWLEDGMENT

The authors would like to thank the many friends that participated to this discussion; among the other we would like to mention Marco De Benedetto (Embrane) and Pere Monclus (PLUMgrid).

REFERENCES

[1] "Network functions virtualisation," Introductory White Paper, Oct. 2012, work in progress. [Online]. Available: http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf

[2] P. Quinn, J. Guichard, S. Kumar, P. Agarwal, R. Manur, A. Chauhan, N. Leymann, M. Boucadair, C. Jacquenet, M. Smith, N. Yadav, T. Nadeau, K. Gray, B. McConnell, and K. Glavin, "Network service chaining problem statement," Internet Engineering Task Force, Internet-Draft draft-quinn-nsc-problem-statement-03, Aug 2013, work in progress. [Online]. Available: http://tools.ietf.org/html/draft-quinn-nsc-problem-statement-03

[3] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 20–26, Mar. 2009.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[5] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," in *Proceedings of the seventeenth ACM symposium on Operating systems principles*, ser. SOSP '99. New York, NY, USA: ACM, 1999, pp. 217–231.

[6] F. Risso and I. Cerrato, "Customizing data-plane processing in edge routers," in *Proceedings of the European Workshop on Software Defined Networking (EWSDN)*, 2012, pp. 114–120.

[7] J. Martinsy, M. Ahmed, C. Raiciuz, and F. Huici, "Enabling fast, dynamic network processing with clickos," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, 2013.

[8] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: a programmable and high performance platform for data center networks," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 2–2.

[9] J. Whiteaker, F. Schneider, R. Teixeira, C. Diot, A. Soule, F. Picconi, and M. May, "Expanding home services with advanced gateways," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 37–43, Sep. 2012.

[10] J. W. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xomb: extensible open middleboxes with commodity servers," in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ser. ANCS '12. New York, NY, USA: ACM, 2012, pp. 49–60.

[11] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Rfc 6830: The locator/id separation protocol (lisp)," Jan 2013.

[12] R. S. D. Paula, "Market analysis for programmable router," Politecnico di Torino, Torino, Italy, Tech. Rep. MSc thesis, July 2013.

Softwarization of Future Networks and Services - Programmable Enabled Networks as Next Generation Software Defined Networks

Alex Galis, Stuart Clayman, Lefteris Mamatas
Department of Electronic and Electrical Engineering
University College London (UCL), Torrington Place,
London WC1E 7JE, U.K.
{a.galis|s.clayman|l.mamatas@ucl.ac.uk}

Javier Rubio-Loyola
CINVESTAV Tamaulipas
Parque Científico y Tecnológico TECNOTAM, Km. 5.5
carretera Cd. Victoria-Soto La Marina, Ciudad Victoria,
Tamaulipas 87130, Mexico {jrubio@tamps.cinvestav.mx}

Antonio Manzalini
Telecom Italia Lab
via Reiss Romoli 274, Turin, Italy
{antonio.manzalini@telecomitalia.it}

Sławomir Kukliński
Telekomunikacja Polska Orange Labs,
Obrzezna Street, 02-691 Warsaw, Poland
{slawomir.kuklinski@orange.com}

Joan Serrat
Universitat Politècnica de Catalunya,
Campus Nord UPC, Modul D4, C/ Sor Eulalia de Anzizu,
s/n Barcelona, 08034, Spain
{serrat@tsc.upc.edu}

Theodore Zahariadis
Synelixis Solutions Ltd.
10 Farmakidou Av, GR-34100, Chalkida, Greece
{zahariad@synelixis.com}

***Abstract*— The Software Defined Networks (SDNs) and Network Functions Virtualisation (NFVs), as recent separate research and development trends have the roots in programmable / active network technologies and standards developed a decade ago. In particular, they are associated with the decoupling of forwarding from control and hardware from networking software, using open interfaces to connectivity resources. The next phase of R&D would involve novel integration and use of all connectivity, storage and processing resources under new management interacting with control systems for provisioning of on-demand networking and services with continuous update of features. This brings into focus a relatively new and key topics for the next decade: what and how to create the conditions for effective and continuous updating and changing the networking functions without reinventing each time architectural aspects and related components (e.g. Softwarization of Future Networks and Services or Programmable Enabled Networks). This paper presents motivation, architecture and the key challenges in realising such programmable enabled networks as the next generation Software Defined Networks (SDN) focusing on its management plane.**

Keywords—Programmable networks, softwarization of Future Virtual Networks and Services

I. BACKGROUND AND CONTEXT

The current developments in Software Defined Networks (SDNs) and Network Functions Virtualization (NFVs) are highlighting new and critical R&D topics related to what and how create the conditions to effectively and continuously update and change the networking functions (e.g. Softwarization of the future networks and services or *Programmable Enabled Networks - PENs*) without reinventing every time the network architectures. Key software features of the future networks and services are already identified and elaborated in the ITU-T recommendation Y.3001 [4, 5]. These software features include: service diversity, functional flexibility, virtualisation of resources, energy consumption, service universalization, network management, mobility, optimisation, identification, reliability and security would need to be realised as part of the future network services and continuously updated.

The integration of the Internet with software infrastructures and traditional communication / telecommunication technologies has been always a challenge for network and service operators, as far as service deployment and management [7, 8, 9, 11] are concerned.

Different frameworks and architectural approaches have been proposed in the research literature and in commercial work. New approaches and technologies are causing a paradigm shift in the world of network architectures. The motivation behind this shift is the still-elusive goal of rapid and autonomous service creation, deployment, activation, and management, resulting from ever-changing customer and application requirements.

978-1-4799-2781-4/13/$31.00 ©2013 IEEE

Research and development activity in this area has clearly focused on the synergy of a number of concepts: programmable networks, network virtualization, self-managing networks, open interfaces and platforms, and increasing degrees of intelligence inside the network.

The future networks and services need to move from being merely Defined by software to be Programmable by software and must be capable of supporting a multitude of providers of services that exploit an environment in which services are dynamically deployed and quickly adapted over a heterogeneous physical wire, wireless and smart objects infrastructure(s), according to varying and sometimes conflicting customer requirements.

At least three key stages of this technological synergy for the main Software Driven Network concepts could be identified as presented in Figure 1 & 2:

• Programmability in network elements (switches, routers, and so forth) was introduced over a decade ago as the basis for rapid deployment and customization of new services (i.e. first architectural state of the SDN Conceptual View: *programmable networks*) [6].

• Advances in programmable and virtual networks have been driven by the industry adoption of Open-Flow & NFV and a number of requirements that have given rise to a new business model of the same telecom business actors, and roles (i.e. second architectural state of the SDN Conceptual View: *Software-Defined Networks*) [2, 3].

• We are moving away from the centralised control and "monolithic" approaches where systems are vertically integrated toward a component-based approach, where systems are made of multiple components from different manufacturers, interacting with each other through open interfaces to form a service. The result is a truly open management service platform representing a marketplace wherein services and service providers compete with each other [12], while customers may select and customize services according to their needs (i.e. third architectural state of the SDN Conceptual View: *Softwarization of networks* or *Programmable Enabled Networks - PEN*) [1].

The fundamental difference between the envisaged PEN concept and previously proposed SDN technologies is the switch to a connectivity and computation infrastructure which is both a service-aware and a management-aware network foundation, where the network elements have direct support for service lifecycle and built-in support for management functionality. It focuses on developing the management plane for SDN.

This infrastructure utilizing shared virtualised resources, including those in wire, wireless and resource-constrained mobile devices and smart objects. PENs would need to be engineered [1] to facilitate the integration and delivery of a variety of ICT services, Computing and Network Clouds and to enhance integration of the key enabling technologies: programmability, networks, network virtualization and network function virtualisation and self-management.

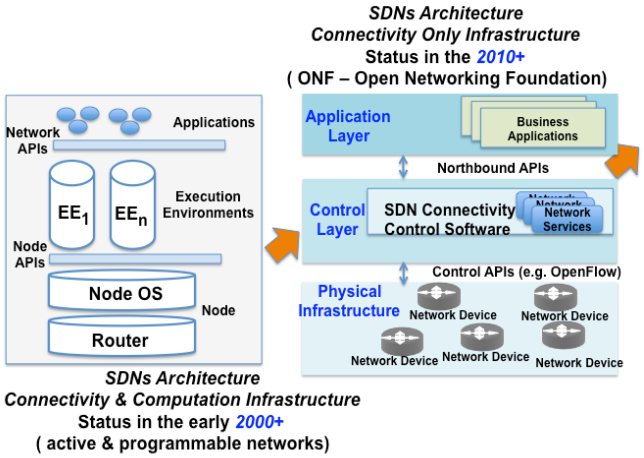


Figure 1 - SDN Evolution - Conceptual View of Networked Systems

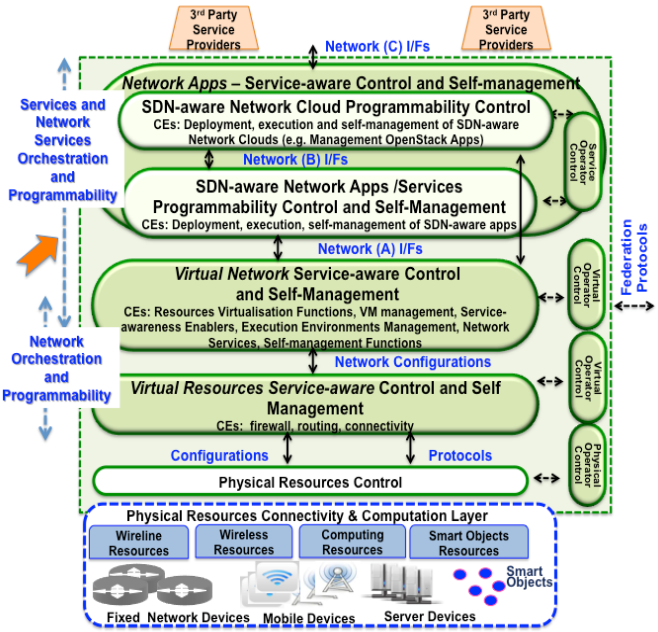


Figure 2 - SDN Evolution (continuation) - Conceptual View of Networked Systems

This paper presents some of the key challenges for realising the Softwarization of Networks / Programmable Enabled Networks (PENs) as the third stage of R&D in SDN.

II. PROGRAMABLE ENABLED NETWORKS (PEN) OVERVIEW AND CHALEENGES

A. PEN Overview

In PEN, the focus is on the service-aware control and management plane, the details of its operation, and the APIs which make it operate. As PEN relies on existing wired and wireless networks and devices, these control elements provide a mapping downwards so there is less emphasis on devising new physical features. This is the main systemic difference to

the traditional programmable networks and the recent activities on Network Function Virtualisation Network Functions Virtualisation (NFV) [3], Network Operating System and Network Orchestration, which are mainly targeted to ONF [2] validation. An important feature of the architecture is a cross-layer approach, i.e. interfaces and mechanisms that enable control and exchange of information between different PEN layers: this provides the ability to push requirements from one layer to the next in a configurable and dynamic way. The proposed functional decomposition simplifies the implementation that is driven by the envisioned functionality. It has to be noted that such an approach is completely different from that of Open Flow which does not decompose network layers into functional blocks.

One key component of the PEN design is the description of services provided by each layer using building blocks defined by an abstract model. PEN does not intend to create new models, but rather to examine and reuse well-established ones, e.g. IETF ForCES, ONF's OpenFlow's switch model and YANG (NETCONF). Accordingly, PEN will extend the chosen model to satisfy the requirements in order to depart from their current 'network function' view and get closer to the 'network service' view.

Composition of services using such a methodology will enable the PEN architecture to have a very fine-grained degree of service programmability as well as to encompass any new future layer primitives. The ability to dynamically insert new layer primitives would be empowered to adapt to future needs. In essence the building-block approach will allow PEN to define, deploy and manage, at runtime, new functionalities and services. These functionalities will be published from bottom-up, whereby each layer publishes to the upper layer the functions that it can provide and ultimately the user will be able to see which services are available. They would be able to be pushed from top-bottom, where the user can request one or more specific services which would then have to be created from existing infrastructure or instantiated at run-time and then published to the user.

The PEN concept is developed according to the features mentioned in the third architectural state of the SDN Conceptual View (Fig. 1) based on a Software Driven/Enabled Networks approach. In opposite to SDN proposed by ONF [2] PEN is a systematic approach towards an integrated connectivity and computation infrastructures. The overall PEN architecture is split into layers depicted in Fig. 2 according to the functionalities described hereafter.

The lowest layer, Physical Resource Layer role is to cope with heterogeneous environments. It has two main functions. It provides a uniform view (via virtualization) of different technological network and computational resources (i.e., providing resource abstraction) and it has intrinsic autonomic and programmable management of the resources, which provides a fast-reaction time for management operations and facilitates scalability of the PEN solution in case of distributed management implementation. The Physical Resource Layer exposes crucial functions to other layers, for example there are monitoring and resource control facilities used by other layers. The information monitoring provides not only a view of the resource health and usage but also of the power consumption, which makes the PEN approach energy efficiency ready. It is assumed that such 'physical resources' can be provided by multiple owners/operators across multiple domains. The deployment of the PEN architecture will be in a form of additional control elements to the wired and mobile environments with adaptation to specific physical resources.

It is worth mentioning that Smart Objects are also part of the architecture. IoT and "Smart Objects" are expected to become active participants in information, social, industrial and business processes where they interact with services and applications while communicate among themselves by exchanging data and information about the environment. In parallel, they are reacting autonomously to the "real/physical world" events and influencing it by running processes that trigger actions and create services with or without direct human intervention.

A set of virtual networks utilizing the underlying physical resources can be created using the mechanisms of the Virtual Network Programmability Layer. These virtual networks adapt their properties to the specific needs of customers and services. The virtual networks have embedded self-managed mechanisms that can control and monitor the underlying physical resources, through utilizing in an intelligent manner the lower-level control and monitoring components of the Physical Resource Layer. The self-management operations include self-configuration, performance optimization, and self-healing. The performance optimization deals with efficient usage of physical resources and cross virtual-network optimizations (traffic management). The manipulation of virtual networks can be programmable using the SDN paradigm. It is assumed that there is support for multiple virtual networks operators. All of these facilities aid in the scalability of a PEN solution.

The end-users and application providers can use specific virtual networks according to their needs in order to create high-quality, personalized, QoS-aware, and secure services. It is assumed in the proposed approach that *programmability of end-user services is provided by the Network Application Programmability Layer.* A simple example would be of a user defining the network topology that he requires from the network along with specific functionalities (e.g., firewall, transcoder, load-balancers) instantiated at specific points in his virtual network. The PEN would be able to create this virtual network and instantiate the requested user's functionalities at the required locations to provide the desired QoS, e.g. minimizing network latency.

It has to be noted that the aforementioned programmability and self-management of different layers of PEN requires the ability to send, execute and monitor the execution code and therefore the management operations should be extended appropriately. In order to do that we need an execution environment that can be centralized or distributed.

The scalability of the proposed architecture is enabled by the following architectural elements: virtualisation of all types of physical resources; the separate mechanisms and mappings

of virtual resources to wire, wireless and smart objects networks; the control elements of the service-aware and management-aware control layer; the northern APIs as depicted in Fig. 1 and by the use of Virtual Machines for the programmability of the service and network components.

III. PEN CHALLENGES

A number of R&D challenges are identified for the realization of programmable enabled networks with solutions hosted by the architectural blueprint presented in fig. 3.

Performant and Safe Network Execution environments: This challenge refers to the network hosting virtual environments and virtual machines to overcome the problem of having several execution environments implemented in various technologies, and providing different abstractions, interfaces, and so on. Network software features would be realised and activated in the network by the creation of specific to the network of execution environment and groups of virtual machines which are managed (creation, change/update, deployment, migration, orchestration, deletion) as one. The advantage of having an explicit notion of a virtual environment is to provide generic means to manage access and resource control on the node-level. While execution environments support the installation, instantiation, and configuration of services code in various ways, the virtual environment puts a uniform management layer on top. This allows external clients to interact with services through the interface of the virtual environment in a generic way, and the interactions will be mapped to specific interfaces of the execution environments. Several execution environments can be attached to a virtual environment, just the same way as other resources. This leads to another aspect of virtual environments: the partitioning of resources. The network provider can set up virtual environments on selected network nodes, and assign them to a particular service provider, in order to offer a virtual network. Access to the virtual environments will be made available to the respective service provider so that it can manage its own virtual network. The resource partitioning implemented among virtual environments will prevent interference with other service providers and, additionally, allow an accounting per service provider.

Several virtual environments belonging to the same service provider but running on different network nodes will form a virtual network to be used by the service provider to deploy services and make them available to customers. In order to know which virtual environments belong to a particular virtual network, the environments are tagged with special network identifiers.

To summarize, the concept of virtual environments enables several aspects:

- A generic way of deploying and managing services independent of the technology of the underlying execution environment;
- A generic way to manage (i.e., monitor and control) nodes for service providers as well as for network providers;
- Partitioning of resources among several service providers;
- Accounting of resource usage per service provider;
- Delegation of service management to the service providers.

Programmability in Future Networks and Services: This challenge refers to solutions for the fast, flexible, and dynamic deployment of new network services. This is aimed to provide easy introduction of new network services by realizing the dynamic programmability of the network and its devices such as routers, switches, and applications servers. Dynamic programming refers to executable code that is injected into the network element in order to create the new functionality at run time. The basic idea is to enable third parties (operators, service providers and other authorised users) to inject application-specific services (in the form of code) into the network. Applications may utilize this network support in terms of optimized network resources and, as such, they are becoming network aware. As such, network programming provides unprecedented flexibility in telecommunications. However, viable architectures for programmable networks must be carefully engineered to achieve suitable trade-offs between flexibility, performance, security, and manageability.

The key question from the public fixed and mobile operator's and Internet service provider's points of view is: how to exploit this potential flexibility for the benefit of both the operator and the end user without jeopardizing the integrity of the network. The answer lies in the promising solutions for:

- Rapid deployment of new services;
- Customization of existing service features; optimisation of network resources
- Scalability and cost reduction in network and service management;
- Independence of network equipment manufacturer;
- Information network and service integration;
- Diversification of services and business opportunities in particular for virtual environments and clouds.

Federation facilities: This challenge refers to the interactions between the different operators and different domains with the same operator. These facilities would include provisioning of:

- *interfaces* that will allow a networking function to federate. Using this interface, the networking function should be able to cooperate in order to provide Interdomain communication.
- *authentication* for other operators, and the two operators confirm with each other the identity of the two consumers of a particular service.
- *mechanisms for communication and programmability of service modules* deployed by different operators for the same service.
- *mechanisms for end-to-end resource management, monitoring, and accounting* should be provided.

Heterogeneous environments: PEN should cope with heterogeneous environments providing uniform view

(virtualization) of different technological networks and computational resources. This functionality is a part of Physical Resource Layer. The research challenges to assess this view with special emphasis on the wireline, wireless and Smart Objects virtual control adaptation.

Control of Virtual wireless resources: this challenge refers to the necessary technology-dependent actions and algorithms for run-time control over local virtual resources in wireless network environments using technology specific operations. This challenge addresses basic configuration functionalities including virtual resource creation, activation, adjustment and termination operations. Dedicated mechanisms and algorithms developed for on-the-fly manipulation of resources in dynamic environments with conflicting requirements according to up-to-date feedbacks from local network monitoring activities are also part of this challenge. These may include adaptive re-allocation of virtual resources according to changing network conditions or service demands. Additionally, this challenge deals with the critical nature of developing autonomous actions that provide network stability and optimizations in absence of higher-level control. This includes for example virtual resource remapping in case of resource scarcity that can be provided internally to the virtual network control.

Mapping virtual resources to the wireless resources: this challenge includes the design and implementation of specific mechanisms and algorithms for optimised mapping of virtual resources onto the physical resources in the wireless environment. Specific optimisation techniques will be developed for efficiently mapping between virtual resources and the physical network infrastructure. In this case of wireless infrastructures, certain characteristics and capabilities have to be considered, e.g. limited bandwidth, processing capabilities, storage, energy (battery), type of interfaces supported of the mobile nodes and mobility, conflicting requirements. As the mapping of virtual to physical resources should be transparent to higher control layers, mechanisms have to be developed that allow the seamless hand-off between different wireless devices. Additionally, algorithms will be identified that optimize the coverage of wireless radio connections to provide access to enough physical resources while avoiding unnecessary energy consumption.

Control of virtual wireline resources: This challenge includes the design and implementation of specific mechanisms and algorithms for run-time control over local virtual resources in wireline environments. OpenFlow environments are considered for representative wireline environments. A major aspect of this challenge is the development of technology-specific methods that enable the provisioning of virtual networks and storage/processing resources over OpenFlow substrate infrastructures. This includes the creation, configuration and tearing-down of virtual resource components, considering both networking and computational/storage resources, e.g. so that link bandwidth or network computation power can be adjusted on-the-fly based on conflicting requirements. By using OpenFlow switch virtualization, networking resources can be re-allocated according to changing network conditions or service demands. Additionally, this challenge considers the development of autonomous actions that provide virtual network stability, performance and optimizations even in absence of higher-level control.

Mapping virtual resources to wireline resources: this challenge includes the design and implementation of specific mechanisms and algorithms for optimised mapping of virtual resources onto the physical resources in wireline environments. Specific optimisation techniques will be developed for efficiently mapping between virtual resources and the physical network infrastructure. Such mapping will involve a wide variety of resources available from the underlying wireline network, including communication, computing and storage capabilities. The mapping will take into account the top-level service/operational requirements such as the demanded QoS requirement and resilience capability to be embedded into the resulting virtual network. By addressing this challenge virtual networks will be customized with optimally allocated capabilities such as virtual nodes (with computing and storage capabilities), virtual links and paths for specific networked services.

Control of virtual resources for smart objects: this challenge will identify and implement the mechanisms required for the discovery, registration and monitoring of virtual and physical resources, configuration and control (including reservation, isolation and release) of virtual resources, and creation of service components in smart objects environments. Taking into account the technology-agnostic requirements of the PEN virtual network control layer, this challenge will identify the technology-dependent control mechanisms needed to meet these requirements.

The control mechanisms will not only be used at this layer/level but they will also need to expose information to the upper layers in order to allow management and control of virtual networks across more than one technology-specific physical domain. It will allow receiving triggers from the upper layers for setting up and tearing resources, as well as adding/removing functionalities and creating service components within the virtual networks which will be accommodated on virtual components residing on smart objects substrates. In this context, an abstract identification model needs to be defined to reference each smart object, as single element or part of a group, for all the control/configuration processes. To realize this, appropriate interfaces need to be defined.

Mapping virtual resources to smart objects resources: this challenge deals with the critical nature of developing the mechanisms and techniques needed to optimize the mapping of virtual resources on physical smart object resources. The objective is to continuously optimize the use of the physical resources (e.g. utilization, energy efficiency) as well as to provide self-organization and self-healing capabilities by appropriately (re)grouping virtual resources and mapping them accordingly to the best set of physical resources. This mapping should ensure that each operation made on a virtual

device has to take effect on the physical object. In fact, all the operations allowed by PEN on virtual instances of the smart objects should then be replicated in a tangible way on real objects. To achieve this each real object must exhibit a set of APIs that enable the interaction with the equivalent virtual object.

This challenge also relates to functions for the setup and control of necessary physical object clusters to support, as an entity, virtual resource requirements in a performance and energy efficient manner. Furthermore, the mapping of virtual to physical resources will support different levels of in-network processing which are needed to provide the best trade-off between computational and networking-related energy consumption in energy-limited smart object environments.

Uniform Autonomic and Optimised Management: This research challenge deals with the critical nature of developing the mechanisms and enablers and systems for autonomic management functions applied not only to the physical resources, but also virtual resources located inside the network. In addition, a unification of all autonomic functions should be realised to enable coordination, orchestration, governance and knowledge closed control loops as applied to all autonomic functions. In this approach the management and control functions would be distributed and located or hosted in or close to the managed network and service elements enabling control of CAPEX and significant OPEX reduction.

Scalable Programmable delivery infrastructures as systems of Inter-orchestration for Big Data and Service Networks: This challenge relates with the critical nature of developing the mechanisms for the transition from current systems designed around discrete and static pieces of uncorrelated silos of content centric information or silos of networks to systems which are more programmable with decentralized control of big data and service networks, incorporating technologies which enable associative orchestration and interactions, and which often leverage virtualisation technologies to provide the capabilities to enable those interactions. In order to integrate such delivery systems, as well as offer new systems to support enhanced composition and correlation - which is what systems of Inter-orchestration is all about, in the end appropriate virtual platform technologies will need to be deployed.

Energy management and optimisation: this challenge relates with the critical nature of developing the mechanisms for Energy- cognisant Internet including optimizing the energy consumption within the limits of a single network and/or a network of networks and /or network of Data Centres and Clouds, based on system virtualization plus the optimal distribution of VMs across the set of networks and servers and providing stabilization of the local networks following electricity demand-response loops.

In Fig. 3 below we have identified and outlined the new closed control loop functionality, which is applicable to energy saving technologies in Future Networks. Fig. 3 shows those logical functions, the information base, and their interactions.

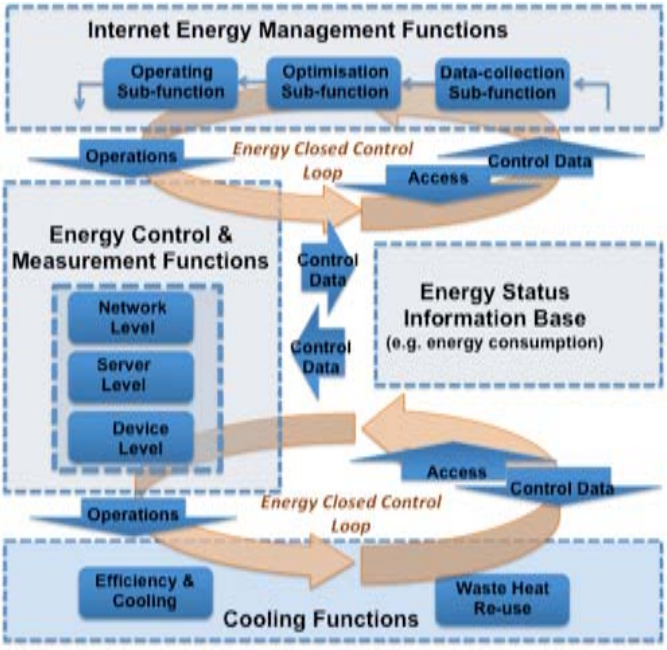


Figure 3– INTERNET SCALE ENERGY CLOSED CONTROL LOOPS

CONCLUDIND REMARKS

This paper discusses the motivation, architecture and research challenges for the next generation Software Defined Networks (SDN) focusing on its management plane. The next generation of Software Defined Networks (SDN) needs to move from being merely *Defined* by software to be *Programmable* by software (e.g. Softwarization of the network) and must be capable of supporting a multitude of providers of services that exploit an environment in which services are dynamically deployed and quickly adapted over a heterogeneous physical infrastructure, according to varying and sometimes conflicting customer requirements.

A programmable enabled network prototype is under development at University College London (UCL) as response to the above new requirements and challenges.

ACKNOWLEDGMENT

The work for this article was partially supported by the European Union UniverSELF No. 257513 and Dolfin No. 609140 projects, Latin American and Caribbean Collaborative ICT Research Federation (LACCIR) Project No. R1211LAC005 and the National Council of Research and Technology (CONACYT) Project No. 185768.

REFERENCES

[1] ETSI 'Software-aware and Management-aware SDN" initiative presented at 3rd ETSI Future Networks Workshop 9-11 April 2013 - http://www.etsi.org/news-events/news/617-2013-fnt-intro)

[2] Open Networking Foundation web site: http://www.opennetworkingfoundation.org

[3] Network Functions Virtualisation (NFV) – ETSI Industry Group started in Jan 2013 http://portal.etsi.org/portal/server.pt/community/NFV/367 & white paper http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[4] Y.3001 ITU-T recommendation – "Future networks: Objectives and design goals" – 2011 http://www.itu.int/rec/T-REC-Y.3001-201105-I

[5] Matsubara, D., Egawa, T., Nishinaga, N., Kafle, V. P., Shin, M. K., Galis, A., -"Toward Future Networks: A Viewpoint from ITU-T" - IEEE Communications Magazine, March 2013, Vol. 51, No. 3, pp: 112 – 118

[6] Galis, A., Denazis, S., Brou, C., Klein, C. (ed) –"Programmable Networks for IP Service Deployment" ISBN 1-58053-745-6, pp450, June 2004, Artech House Books, www.artechhouse.com/Default.asp?Frame=Book.asp&Book=1-58053-745-6,

[7] Chapman, C., Emmerich, E., Marquez, F. G., Clayman, S., Galis, A. "Software Architecture Definition for On-demand Cloud Provisioning" - Springer Journal on Cluster Computing – DOI: 10.1007/s10586-011-0152-0; May 2011; http://www.editorialmanager.com/clus/; on line: www.springerlink.com/content/m31np5112525l67v/

[8] Clayman, S., Clegg, R., Mamatas, L., Pavlou, G., Galis, A., "Monitoring, Aggregation and Filtering for Efficient Management of Virtual Networks", IEEE CNSM conference 2011: 7th International Conference on Network and Service Management www.cnsm2011.org/ - October 2011, Paris, France http://cnsm.loria.fr/

[9] G. Clegg, R., Clayman, S., Pavlou, G., Mamatas, L., Galis, A. - "On the Selection of management and monitoring nodes in dynamic networks"– IEEE Transactions on Computers, 6 March 2012, IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TC.2012.67

[10] UniverSELF EU Project http://www.univerself-project.eu

[11] Galis, A., Rubio-Loyola, J., Clayman, S., Mamatas, L., Kukliński, S. Serrat, J., Zahariadis, T., "Software Enabled Future Internet" - 5th International Conference on Mobile Networks and Management (MONAMI 2013), 23-25 Sept 2013, Cork, Republic of Ireland, http://mon-ami.org/2013/show/home

[12] Open source software for building private and public clouds http://www.openstack.org

Author Index

Table of Contents