NETWORKS 2014

SDN & NFV: The Next Generation of Computational Infrastructure

SDN & NFV: Modern Networking Technologies

2014 International Science and Technology Conference
«Modern Networking Technologies (MoNeTec)»
Moscow, Russia,
October 27–29, 2014

Proceedings

Partners

Applied Research Center for Computer Networks

Skolkovo

IT

Innopolis University

Moscow State University MSU

The Ministry of education and science of the Russian Federation

Technically Co-Sponsored by IEEE computer society

www.arccn.ru

MOSCOW
2014

SDN & NFV: Next Generation of Computational Infrastructure

2014 International Science and Technology Conference
«Modern Networking Technologies (MoNeTec)»

Moscow, Russia,
October 27–29, 2014

Proceedings

МАКС ПРЕСС

МОСКВА – 2014

УДК 004.4
ББК 32.973.202-018.2
У67

Под общей редакцией
д.ф.-м.н., чл.-корр. РАН, профессора *Р.Л. Смелянского*

У67 **Управление и виртуализация в современных сетях (Сети 2014: SDN&NFV):** Труды Международной научной конференции; Москва, 27–29 октября 2014 г. / Под общ. ред. Р.Л. Смелянского. – М.: МАКС Пресс, 2014. – 174 с.
ISBN 978-5-317-04829-7

Международная научно-техническая конференция «Управление и виртуализация в современных сетях» (Сети 2014: SDN&NFV) посвящена двум наиболее перспективным трендам, проявившимся в последние несколько лет - Программно Конфигурируемые Сети (SDN) и Виртуализация Сетевых Функций (NFV). Эти тенденции обещают сформировать эволюцию отрасли Интернета и облачной инфраструктуры в Программно-Конфигурируемые Инфраструктуры. Сетевые операторы и поставщики облачных сервисов уже начали освоение этих технологий, а наиболее смелые уже начали активно испытывать и разворачивать решения на базе SDN и NFV. В конференции приняли участие видные представители международных научных кругов из различных международных проектов, направленных на создание новых способов и инструментов в области SDN&NFV для создания новой инфраструктуры для научных исследований. В трудах конференции представлены доклады, где широко обсуждаются различные аспекты SDN & NFV технологий. Конференция проводится при финансовой поддержке Министерства образования и науки Российской Федерации ГК №14.598.11.0012 и Университета Иннополис (г. Казань). Технический спонсор IEEE Computer Society, рег. конференции №34611.

Ключевые слова: Программно-конфигурируемые сети (ПКС), Сетевая операционная система, Виртуализация Сетевых Функций (NFV), Программно-Конфигурируемая Инфраструктура (SDI), Программно-Конфигурируемые Точки Обмена (SDX), архитектура ПКС контроллера, производительность, масштабируемость, надежность, безопасность, верификация политики маршрутизации.

УДК 004.4
ББК 32.973.202-018.2

SDN & NFV: The Next Generation of Computational Infrastructure: 2014 International Science and Technology Conference «Modern Networking Technologies (MoNeTec)»: Proceedings; Moscow, Russia, October 27–29, 2014 / Alexander Shalimov et al. Editor-in-Chief Ruslan Smelyanskiy. – M.: MAKS Press, 2014. – 174 p.

International Science and Technology Conference «Innovative Networking Technologies: SDN & NFV – The Next Generation of Computational Infrastructure» was dedicated to the Software Defined Network (SDN) and Network Function Virtualization (NFV). Technologies have emerged as the hottest new networking trends of the past a few years. These trends promise to shape the evolution of the Internet and Cloud infrastructure into a Software Defined Infrastructure while transforming the industry. Network and cloud providers are already embracing these trends and early adopters are aggressively trialing and deploying SDN and NFV. The conference gathered prominent international academia representatives from various international projects focusing on SDN/NFV to discuss developing novel ways and tools for network-enabled scientific research. The conference proceedings represent the papers where the broad scope of SDN&NFV are discussed. The Conference is supported by the Ministry of education and science of the Russian Federation GK №14.598.11.0012 and Innopolis University, Kazan, Russia. The Conference is technically co-sponsored by IEEE Computer Society, Conference Record #34611.

Keywords: Software-Defined Network, Network operating system (NOS), Network Function Virtualization (NFV), Software Defined Infrastructure (SDI), SDX – Software Defined Exchange point, SDX architecture, SDN controller architecture, performance, scalability, reliability, security, Forwarding Policy Verification.

Напечатано с готового оригинал-макета

Подписано в печать 07.03.2014 г. Формат 60x90 1/8. Усл.печ.л. 21,75. Тираж 150 экз. Изд. № 215.

Издательство ООО "МАКС Пресс". Лицензия ИД N 00510 от 01.12.99 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 527 к. Тел. 8(985)939-3890/91. Тел./Факс 8(985)939-3891.

Отпечатано в ППП «Типография «Наука»
121099, Москва, Шубинский пер., 6
Заказ №

ISBN 978-5-317-04829-7

Conference Committee

Chairman: Ruslan L. Smelyanskiy

Corresponding member of the Russian Academy of Sciences, Professor (MSU, IEEE member), R&D Director at Applied Research Center for Computer Networks

Co-Chairman: Vladimir N. Vasilev

Corresponding member of the Russian Academy of Sciences, Professor (ITMO, IEEE member), Rector of Saint-Petersburg National Research University of Information Technologies, Mechanics & Optics (ITMO)

Co-Chairman: Alexander P. Kuleshov

Director of the A.A. Kharkevich Institute for Information Transmission Problems (IITP) of the Russian Academy of Sciences

Member: Akihiro Nakao

Professor, University of Tokyo

Member: Ilya Baldin

PhD, RENCI, Department of Computer Science Duke University, IEEE member

Member: Jeffrey S. Chase

Professor, Department of Computer Science Duke University, IEEE member

Member: Nate Foster

Assistant professor, Department of Computer Science Cornell University

Member: Kuang-Ching Wang

Associate Professor (Department of Electrical and Computer Engineering Clemson University, IEEE senior member

Member: Serge Fdida

Professor UPMC - University Pierre & Marie Curie (Paris VI) senior member of IEEE and a Distinguished ACM Member

Member: Ivan Sescar

Assoc. Director WINLAB, Rutgers University, senior member of IEEE

Preface

International Science and Technology Conference «Innovative Networking Technologies: SDN & NFV – The Next Generation of Computational Infrastructure» was organized under financial support of Ministry of Science and Education of Russian Federation (GK №14.598.11.0012) by Moscow State University together with a number of leading Russian universities and scientific centers of the Russian Academy of Sciences (RAS), which are currently joined in the consortium to develop novel SDN-technologies for research and educational environments. It was dedicated to the two main streams in modern Network Architecture – Software-Defined Network and Network Function Virtualization.

Software-Defined Networking (SDN) has emerged as the hottest new networking trend of the past few decade. Network Function Virtualization (NFV) is a complementary trend and together they promise to shape the evolution of the Internet and Cloud infrastructure into a Software Defined Infrastructure while transforming the industry. Network and cloud providers are already embracing these trends and early adopters are aggressively trialing and deploying SDN and NFV. Established and new vendors are busy creating their own SDN and NFV technologies and solutions and are competing for leadership positions in this rapidly growing international market.

The conference brought together SDN and NFV leaders of the international scientific community, research departments of corporations, and industrial enterprises of the Russian Federation, as well as academic institutions and public authorities where they have discussed the most urgent and promising technologies in the area of computer networking, virtualization and cloud computing, to share the latest developments and provide a platform for in-depth discussions on the state of the industry and how to move forward. The conference provided excellent opportunities to interact with and influence the rapidly developing ecosystem of researchers, telecom and cloud operators, vendors and other stakeholders in Russia.
The conference gathered prominent representatives of industry and academia from various international projects focusing on SDN/NFV to discuss developing novel ways and tools for network-enabled scientific research. It was facilitate the exchange of information within individual scientific fields as well as inside interdisciplinary and international collaborations.

The conference agenda has also included two-days School for young scientists, post-graduates and graduate students. Its goal is to develop and enhance the pool of available talent proficient in these technologies and solutions within the Russian Federation.

The papers presented on the conference are collected in this proceeding

October, 2014

Ruslan Smelyanskiy, DrS,
Prof., Cor.-member of
Russian Academy of Science

Table of contents

On real-time delay monitoring in software-defined networks 1

V. Altukhov, E. Chemeritskiy

VERMONT - a toolset for checking SDN packet forwarding policies on-line 7

V. Altukhov, E. Chemeritskiy, V. Podymov, V. Zakharov

Towards SDI-bases Infrastructure for Supporting Science in Russia 13

V. Antonenko, R. Smeliansky, I. Baldin, Y. Izhvanov, Y. Gugel

An Analysis of Approaches to Onboard Networks Design 20

V. Balashov, V. Kostenko, P. Vdovin, R. Smeliansky, A. Shalimov

Towards Load Balancing in SDN-Networks During DDoS-attacks 25

M. Belyaev, S. Gaivoronski

Progress and Challenges in Worldwide Federation of Future Internet and Distributed Cloud Testbeds .. 31

M. Berman, M. Brinn

Network Verication: Calculus and Solvers 37

N. Bjorner, K. Jayaraman

On QoS Management in SDN by Multipath Routing 41

E. Chemeritskiy, R. Smeliansky

Consistent network update without tagging 47

E. Chemeritskiy, V. Zakharov

SDN-based Innovation in New Zealand 53

S. Cotter

The EXPRESS SDN Experiment in the OpenLab Large Scale Shared Experimental Facility 55

S. Fdida, T. Korakis, H. Niavis, S. Salsano, G. Siracusano

SDNI: The GEANT Testbeds Service – Virtual Network Environments for Advanced Network and Applications Research 62

M. Hazlinsky, B. Pietrzak, P. Szegedi, F. Farina, J. Sobieski

Five SDN-oriented Directions in Information Security 68

A. Grusho, N. Grusho, E. Timonina, V. Piskovski

Towards Distributed Hierarchical SDN Control Plane 72

V A. Koshibe, A. Baid, I. Seskar

Selforganizing Cloud Platform 77

V. Kostenko, A. Plakunov, A. Nikolaev, V. Tabolin, R. Smeliansky, M. Shakhova

SDI Defense Against DDoS Attacks Based on IP Fast Hopping Method 83

V. Krylov, K. Kravtsov, E. Sokolova, D. Lyakhmanov

Software-Defined Network Exchanges (SDXs) and Infrastructure (SDI): Emerging Innovations In SDN and SDI Interdomain Multi-Layer Services and Capabilities 88

J. Mambretti, J. Chen, F. Yeh

Program Tools and Language for Network Simulation and Analysis 94

A. Mikov, E. Zamiatina

Application and Device Specific Slicing for MVNO 103

A. Nakao, P. Du

A stateless transport protocol in software defined networks 108

M. Nikitinskiy, I. Alekseev

Controller Failover for SDN Enterprise Networks 114

V. Pashkov, A. Shalimov, R. Smeliansky

Localizing errors in controller applications 120

M. Perevedentsev, V. Antonenko

Data Center Resource Mapping Algorithm Based on the Ant Colony Optimization 127

A. Plakunov, V. Kostenko

Development of educational resource datacenters based on software defined networks 133

P. Polezhaev, A. Shukhman, A. Konnov

OpenFlow SDN testbed for Storage Area Network 140

O. Sadov , V. Grudinin , A. Shevel, D. Vlasov, S. Khoruzhnikov, V. Titov ,
A. Shkrebets, A. Kairkanov

In-kernel offloading of an SDN/OpenFlow Controller 143

A. Shalimov, P. Ivashchenko

Queuing Systems with Multiple Queues and Batch Arrivals for Cloud Computing System Performance Analysis 147

S. Shorgin, A. Pechinkin, K. Samouylov, Yu. Gaidamaka, E. Sopin, E. Mokrov

On the Benefits of 5G Wireless Technology for Future Mobile Cloud Computing 151

S. Shorgin, K. Samouylov, I. Gudkova, O. Galinina, S. Andreev

SDN for network security 155

R. Smeliansky

A network analytics system in the SDN 160

V. Sokolov, I. Alekseev, M. Nikitinskiy, D. Mazilov

Network utilization optimizer for SD-WAN 163

L. Vdovin, P. Likin, A. Vilchinsky

From Federated Software Defined Infrastructure to Future Internet Architecture 167

K.-C. Wang, M. Brinn, J. Mambretti

A Measurement Architecture for Software Defined Exchanges 173

M. Zink

On real-time delay monitoring in software-defined networks

V. Altukhov
Lomonosov Moscow State University
Moscow, Russia
victoralt@lvk.cs.msu.su

E. Chemeritskiy
Applied Research Center for Computer Networks
Moscow, Russia
tyz@lvk.cs.msu.su

***Abstract*—The paper introduces a new loop-based method to measure end-to-end packet delay in software-defined network infrastructures. Although the method generates auxiliary service packets, it does not require any complementary support from the switching hardware. The prototype implementation shows the method is able to provide one-way delay values with microsecond precision on a steady load. Direct application of the method to each data flow in the network is straightforward, but can cause excessive hardware utilization. Thus, the paper proposes an algorithm to improve it by decomposing global end-to-end estimations into the set local ones whereas removing their redundancy. The algorithm makes it practically possible to monitor delay of each data flow in real-time.**

Keywords—One-Way Delay; Measurement; Software-Defined Networking; Quality of Service

I. Introduction

A steady growth in a number of interactive network applications and services originates an increasing demand in advanced control over the quality of connections through the network infrastructure. However, it is a hard problem to compute an appropriate data transmission path and configure network devices along this path to meet the requested end-to-end requirements for the connection. It is even harder to establish such a cooperation of logically independent network devices to enable dynamic provisioning of the requested connections. Furthermore, network hardware evolved without sufficient attention to Quality of Service (QoS) issues, and support of corresponding functionality is often a subject to various restrictions.

Surprisingly, all the listed obstacles have been successfully overcome by the systems focused on end-to-end bandwidth. It is due to its concavity bandwidth is guaranteed to be the minimum among the bandwidths of the links along the connection path. However, the most of the QoS metrics does not have this property, and their calculation cannot be easily decomposed. Quite the contrary, measurement, estimation and attuning of end-to-end delay are naturally hard in any asynchronous distributed system without global clock, and require accurate and precise coordination of network devices. As a result, no modern system for end-to-end delay measurement can improve the precision of a theoretical worst-case estimation, and avoid exotic requirement to the switching hardware.

In this paper we make a first step towards the QoS-aware routing by introducing a new method to measure end-to-end connection delay based on the centralized control and flexible management interfaces for the switching hardware provided by Software-Defined Networking (SDN). Our approach has the following features:

- Measure end-to-end delay on a per-flow basis;
- Precise enough to cover the mutual flow influence;
- Work in SDN with general switching hardware;
- Update results up to several times in a second.

The paper has the following structure. Section II provides a brief review of related works. In section III we introduce a new method to measure packet transmission delay along any route in a network based on header looping. Section IV considers the algorithm to optimize application of our delay measurement method to all routes in a network.

II. Related work

Back in the days of circuit-switched networks end-to-end delay was in a straight dependence on a length of the wire. The compliance with the delay requirements was naturally achieved by searching the network infrastructure for a short enough virtual channel. Since, the problem of delay control has complicated dramatically. With the emergence of packet-switched networks, data flows started to compete with each other for network resources. The development of technology in accordance with Moore's and Gilder's empiric laws gradually shifted the bottleneck of data transmission from the wire to the switching devices. A considerable effort has been made towards the designing of an efficient network switch architecture that could provide maximum utilization to the connected links [1]. In the pursuit of throughput performance a contemporary switch utilizes a multistage engine for packet analysis and a mixture of packet buffers and switching fabric, managed by complicated dynamic packet scheduling algorithms.

Each delay control tool relies on a certain method of end-to-end delay estimation, and there has been suggested quite a number of them. On the one hand, a conservative estimation based on independent computation of the worst-case delay for each network node may be easily implemented and applied to

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and Russian Foundation for Basic Research, project 14-07-00625.

any kind of a network. However, is known to inflate the actual delay value by several orders of magnitude. On the other hand, state-of-the-art achievements in Network Calculus make it possible to compute a tight upper bound for the worst-case end-to-end delay with assumptions of traffic conditioning on the border network switches, fluid data flows, and their FIFO multiplexing [2]. Unfortunately, these limitations as well as a high computation complexity are not insensible. The diverse and intricate operating principles of switching devices obscured network-wide packet scheduling and made it hardly promising to build a method for end-to-end delay estimation with a wide scope, an appropriate precision, and an acceptable computation complexity at the same time.

Inability to estimate network delay pushed forward an intention to measure it. Though, one-way delay measurement in an asynchronous system is challenging. A computation of a one way delay as a bisection of the round trip time seems to be natural, but this approach does not generally work as intended because both routes and network load of the forward and the backward paths of a flow may differ. Although it is possible to compute the one-way delay of a flow with higher precision with help of the complementary software modules installed on the end hosts [3], this method is either applicable only to protocols with some specific features or make the end hosts to generate a lot of secondary traffic.

Less host-assuming approaches address the data transmission only through the network infrastructure. It is a tried-and-true method to bypath the asynchrony by setting up a global clock with Network Time Protocol (NTP), Global Positioning System (GPS), or Code Division Multiple Access (CDMA), and tagging the transmitted packets with timestamp on send. However, it implies each packet has a place for the timestamp in its headers, and the switches are able to handle this timestamp. The prevalent approach is to compute the delay non-intrusively by means of ad hoc service packets and avoid the tagging similar to [4]. However, this modification does not eliminate the need in the dedicated time server and the abilities of the switching devices to synchronize and generate the appropriate service packets automatically.

SDN introduces a concept of a single centralized controller to rule all the switching devices and provided a convenient way to synchronize them. The paper [5] proposes to use this opportunity to measure the delay by the following outline. First, the controller reserves a certain header for the service purposes. Then, it installs a set of forwarding rules to route the packets with this header by the path of the flow of interest. However, the last rule along the path is modified to send outgoing packets to the controller. From time to time, the controller forges a probe packet with the reserved header and a relevant timestamp in its payload, and sends it through the ingress switch of the constructed path. When the packet comes back, the controller checks its timestamp and computes the packet delay.

Packet probes do not require any complementary support from the hardware, nor the synchronization of switching devices. However, the probe comprises not only the route of the real packets, but also the routes from the controller to the ingress switch and from the egress switch back to the controller. Moreover, each probe packet experiences two passes through a network stack of the controller, and a pair of transitions between the Control Plane and the Data Plane at the switches, usually implemented by means of a slow software processing. As a result, the value of the target delay component often becomes smaller than the value of parasitic components, and the method is unable to provide the required precision.

In this paper we propose a novel approach to establish packet probes, which copes the negative impact of the adverse delay components by increasing the share of the target component with packet iteration.

III. One-way delay measurement for a single path

A. Rationale

End-to-end packet transmission delay is equal to a sum of a network infrastructure delay and a delay between border switches and network applications at the ends of the route. It is not possible to measure the latter component due to a lack of information about configurations of the hosts. However, the delay of packet transmission through the network infrastructure is a large part of the end-to-end delay. In this paper we discard the delay between the network and the hosts, and consider the delay of the network infrastructure only.

In SDN packets can pass through the network infrastructure with two types of routes: (1) slow path routes that imply processing of packets at the controller, and (2) fast path routes that are processed solely by the switching devices. In most cases, packets pass through the fast path, therefore, in this work we focus on measuring end-to-end delay for fast path.

We assume each network switch implements Output Queuing and consists of the following components:

- Packet analyzers (one per port),
- Switching fabric,
- Output queues (one per port).

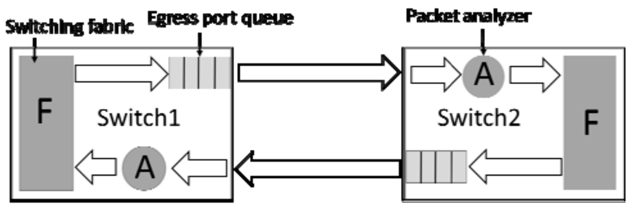


Fig. 1. Scheme of switch interaction.

Packet processing at a switch is organized as follows (fig. 1). Upon receiving a packet, the switch analyzes its headers and produces an instruction to process it. Then, the switch fabric executes the instruction and transmits the packet to an appropriate set of output ports. However, the packet can arrive when the connected channel is already in use by packets from the other ports. In this case the packet is pushed into a FIFO-queue of the port. The queue is polled every time the channel becomes ready to transmit.

We assume the delay of packet processing at analyzers and switching fabrics as well as the delay of packet serialization and propagation depends solely on packet length and some performance characteristics of the networking hardware. Thus, the listed components can be calculated statically without a regard to the network load. Note our assumption does not generally hold and some advanced hardware violates it. However, the value of calculation error is negligible compared to the delay of packet queuing. Thus, our method focuses on measuring of the latter one.

Because of the dependence on mutual influence of the flows, queuing delay cannot be calculated a priori. Our method captures this dependency with help of a service packets forged by a network controller to follow the path of the usual data packets and experience all the appropriate delays. However, instead of making a single run along the path of interest, the packet iterates it back and forth in an endless loop. At the beginning of each iteration the first switch of the loop sends a copy of the packet to the controller as a pulse message.

Note the interval between a pair of consecutive pulses provides a precise estimation for RTT over the path of interest. Its value does not capture any delays cause by interaction with the controller. The first pulse is sent after the service packet is already inside of the data path. Thus, the interval does not include the delay of transmission from the controller to the Data Plane. Next, although each copy of the service packet actually goes from the switch to the controller, the interval value is calculated with a subtraction which annihilates the corresponding delays and reduces their impact to a jitter.

Our method uses aforesaid advantage and derives one-way delay along the path of interest from its RTT. However, direct application of the loop-based measurement results into a heavy load of the controller usually inadmissible in practice. Thereby, we focus on decrease in the performance requirements of the loop-based RTT measurement method in the first place, and consider the ways to divide RTT into one-way delays fairly in the second.

B. Measuring RTT with Packet Looping

Intensity of the pulse packet flow depends on a length of the underlying loop that generates it. The longer the loop, the fewer impulses reach the controller. It is not possible to expand the loop because it is tied to the path of interest. However, the controller can use the headers of a service packet to implement a counter and send pulse messages once per several iterations.

Let a path of interest consists of N>1 switches S1,…, Sn. To set up an appropriate topology loop controller goes through the switches along the path and supplies i-th switch with a pair of forwarding rules to transmit service packets from the switch number (i-1 mod N) to the switch number (i+1 mod N) and back without any modifications. Controller identifies a packet with a predefined value in a certain field of its header (e.g. 0xBEEF in Ethernet type) to be the service one disregarding the other fields. Thus, the installed rules contain a nonempty set of wildcard fields (e.g. Ethernet source and destination addresses).

Controller interprets the values stored a certain subset of wildcarded fields as a encoding of a loop counter. To make the counter run, it selects any switch in the loop and replaces one of its transmission rules with a set of M similar rules that modify the value of stored a counter. The pattern of i-th rule matches the encoding of i while its actions sets the counter fields with the encoding of (i+1 mod M). Thereby, after being sent into a constructed loop, a service packet with a valid encoding of a counter in its headers restores the same set of headers and appears at the same location of a network at every M-th iteration. Note such a combination of packet location and headers is often referred as a packet state [6]. Using this term, it is correct to say the controller sets up a single loop in the space of packet states.

The described approach requires M rules to set up a counter for M iterations and leads to a fast exhaustion of forwarding tables of the switches. Fortunately, it is possible to reduce it by modifying individual fields of a counter at different switches. For example, the switch S1 can increment the first field of a counter encoding and ignore its other fields. The switch S2 can increment the second field of a counter while passing through any packets with non-zero value at its first field. This cascade scheme factorizes the number of required rules. The controller installs M1 rules into the first switch and M2 rules at a second switch and set up a loop with an iteration number equal to their product M1*M2. In general, if the packet has k counter field of a sufficient size, it is possible to set up a loop of M iteration along the path of N≥K switches with K*⌈M^(1/K)⌉+N+1. The number of rules can be reduced even more, if the switches support some advanced actions for a certain set of counter fields (e.g. decrement TTL).

Finally, controller selects any of the counter modification rules that is used by a single iteration of the loop and extends its instruction set with an action to send an appropriate pulse message. As a result, the value RTT can be estimated as an interval between a pair of consequent pulses divided by the number of iterations in the constructed loop.

Upon a loss of a service packet the described method stops the measurement. However, this problem can be solved by injecting of a new service packet to replace the previous one if no pulse message has been received for some period. Also this situation can be used to detect network congestion.

Note a loop over the packet states improves the accuracy of the RTT measurement. Although intervals between the pulses include parasitic jitter of a switch-to-controller communication, its share may be reduced to an eligible value by increasing the length of the state loop. Suppose the switch-to-controller (SC) delay varies from 300 μs to 500 μs, and real RTT is about 5 μs. Then, SC jitter exceeds an actual RTT forty times. If we want the measured value to provide 90 percent accuracy, it is necessary to set up a loop with over 400 iterations. Thereby, we can get a suitable precision even in a network with a high-latency controller.

C. RTT measurement experiments

We implemented our method to measure the RTT along the given path with the state looping as an application for POX controller [7] and validated it experimentally. We used a single hybrid OpenFlow switch NEC PF5200 with 48 1Gbit/s interfaces to create a network with 4 virtual switches (fig. 2).

The experiments were aimed to check the method accuracy in dependence on the network load.

The path of interest is S3, S2, S1.Traffic generators and controller are deployed at a single server with 3 1-Gbit/s interfaces. We used pktgen [8] to generate and send 1000 byte packets over the paths S3, S2, S1 and S1, S2, S4, S3, S2, S1. During the generation, each packet was marked with a corresponding timestamp. Generated traffic was captured with wireshark [9]. A difference between the time of packet capturing and the timestamp inside of its body was considered as a reference approximation of the RTT at the network infrastructure.

Under a steady load the reference delay was in range from 500 to 560 µs with an average of 540 µs. The measurement with a loop running along the path of interest 1024 times estimated the RTT by a range from 500 to 600 µs, with an average of 560 µs. This assessment differs from the average reference estimation by 3.7 percent.

The second purpose of the experiment was to show, that the results of the proposed method reacted the changes in network load. To simulate dynamically changing network load traffic we generated flows of 10000 packets with rate of 600 Mbit/s. Thus, the rate of data transmission in links along the path S3, S2, S1 changed from 0 to 1.2Gbit/s (some packets were dropped).

Measurement results for proposed method showed that delay was in range from 500 µs up to 1.5 ms. Measured delay increase to 700 µs, until output port queues became congested. Upper bound values match packet loss. After output port queues became empty, measured delay decrease to normal value – from 500 to 600 µs.

D. Deriving one-way delay of a route by RTT

The calculation of a one-way delay by bisecting the RTT is often inaccurate. Note we can divide RTT over a single hop with more precision by taking into account the proportion of data transmitted in each link direction.

Consider a pair of switches connected to each other by a link with a bandwidth of C (figure 1). For a given time interval T, X and Y denote a number of bytes, directed to queues Q1 and Q2 of the switches S1 and S2 respectively. Controller can obtain actual values of X and Y by sending appropriate statistic requests to the switches. Note these values are usually measured at the stage of packet analysis. Thus, their accumulated size can exceed the number of bytes transmitted through the channel.

There are three possible options:

1. X/T≤C and Y/T≤C. Thereby, both output queues are empty and one-way delay in each link direction is equal to a half of RTT.
2. X/T≥C and Y/T≤C. Q1 is congested and Q2 is empty. Thus, one-way delay from Switch1 to Switch2 can be calculated as (RTT+(X/C-T))/2 and one-way delay from Switch2 to Switch1 can be calculated as (RTT-(X/C-T))/2.
3. X/T≥C and Y/T≥C. Both Q1 and Q2 are not empty. One-way delay from Switch1 to Switch2 can be calculated as (RTT+(X/C-T)-(Y/C-T))/2 and one-way delay from Switch2 to Switch1 can be calculated as (RTT-(X/C-T)+(Y/C-T))/2.

With these assumptions, we can divide target path into one-hop paths, obtain their one-way delays by an advanced division of RTT and sum them up into a pair of resulted one-way delays. This method has a large overhead, especially if we want to measure multiple paths in the network. However, if the paths of interest have some common parts, it is possible to measure them only once.

IV. Delay measurement for any route

A. Divide and measure

Proposed method allows us to measure RTT of single path in a network. However, the total number of paths depends exponentially on the number of switches and it is not possible to apply the proposed method for each of them directly.

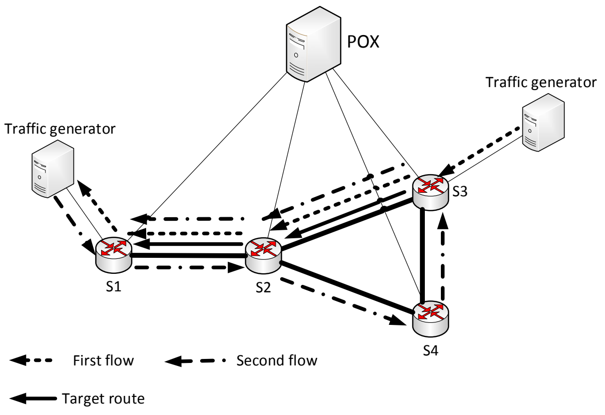


Fig. 2. Delay measurement experiment topology with generated flows and target flow.

Suppose (fig.2) we know delays from S3 to S2 and from S2 to S1. Then delay from S3 to S1, can be represented as sum of one-hop delays: d(3,1)=d(3,2)+d(2,1). Similarly, the delay for any route in network can be split into a sum of one-hop delays and the main target is to measure all one-hop delays in network, or to construct a network delay map - a structure, containing all one-hop delays.

A straightforward approach is to measure all one-hop RTTs, using the proposed measurement method, and obtain one-way delays using the advanced method for RTT separation.

Another approach is to organize so many loop measurements, which will allow obtain network delay map as the result of solving a system of linear equations with loops RTT. We propose an algorithm that construct network delay map and organize measurements with minimal controller load.

B. Algorithm for constructing network delay map

We need to organize measurements with minimal controller load. Header looping measurement method provides two approaches to minimize network load: increase length of the topological loops and increase the number of iterations over the headers. Second approach does not arrange us, because while minimizing number of PacketIn messages, it increases the number of rules installed into the switches. We will use both approaches in proposing algorithm.

The idea of the algorithm is to replace some measurements over single links with measurements over longer paths, and then derive the former from the latter.

We set up the loop construction problem as follows. For a given network graph, find such a set of topology loops as to:

1. Each one-way link must be included in at least one loop;
2. Maximize the accumulated length of the loops in a set;

Assign a variable directed edge in graph. Delay for any path can be calculated from the linear equation, where directed edges will represent each hop in path. Suppose we can measure delay for any path in graph. Then, we can construct such a system of linear equations, solving which will be obtained network delay map. Therefore, we need to find such a set of topology loops that will meet all listed requirements and may be used to construct a system of linear equations solving which will be obtained network delay map.

Let two loops be dependent, if edges set of one loop contain edges set of another loop. Only set of independent loops can be used to construct a system of linear equations.

Let one loop be sum of two another loops, if it's set of edges contain every edge from summand loops and does not contain any other edge.

We will call set of independent loops - *objective*, if it meets all the listed requirements. Any loop of the objective set can be represented as the sum of other loops of smaller lengths (if the objective loop includes more than two directed edges and it does not belong to the graph basis). Then the objective set of cycles can be constructed from the basis of all simple loops of the graph. The construction of simple loops sets requires finding a fundamental set of loops of the graph, which is a union of fundamental sets of all spanning trees of the original graph.

The problem of finding a fundamental set is complicated, because the number of spanning trees of the graph can reach n^{n-2}, where n is the number of vertices in graph. Therefore, to construct the independent set of loops we use an algorithm to find all the simple loops in the graph described in [10]. Its complexity – O((n+m)(c+1)), where c is the number of simple loops in the graph. The resulting set may contain linearly dependent loops and they should be filtered out with post processing.

Next step is to construct objective set from set of basic loops. As mentioned before, any objective loop can be represented as sum of basic loops. We can construct objective set of loops as a linear combination of basic loops. But construction of the objective set of loops with maximum sum of length is a problem that cannot be solved without exhaustive search. Therefore, we propose a greedy algorithm that expands topological loops. In this algorithm, we use only independent simple loops from constructed system. For every loop in system, we try to combine it with other, and if combination is simple independent loop, longer than previous one, we save it. Thus, after every step of algorithm we get a correct (independent) system of loops with total topological length, bigger than the one at the previous step.

Number of loops in the constructed objective set of graph does not exceed its cyclomatic number. Thus, we need to supplement it with more loops (total number of loops must be equals to number of one-way edges in network). To achieve this, we complete the system with measurements using the advanced RTT division.

Now we just need to start measurements for every loop in system. Measuring RTT from this loops and solving linear system will give us network delay map.

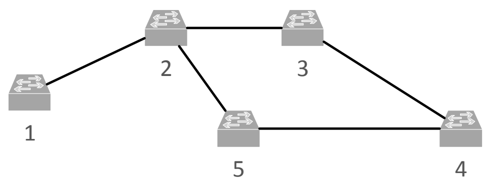


Fig. 3. Delay map construction experiment topology.

C. Delay map construction experiments

Applied to an example network topology showed by figure 3 our algorithm generates a set of seven loops listed in table I. However, there are five links and ten delay values to calculate. Thus, we had to derive one-way delays from the RTT at links 1-2, 2-3, 2-5 (fig. 4).

We have implemented the algorithm as an application for POX controller and have studied its performance in a network simulated by Mininet [11].

Experiments with our method showed the one-way delay for each link has been in range from 16 to 20 µs. For comparison, the value of RTT measured by pinging hosts, connected to switches 1 and 2 (which includes SC delay) is in the range of 40 to 60 µs, so we reached necessary accuracy of measurements. The number of iterations for each loop was 2048. It used two counter-fields, that required to install 96 + k rules per loop (k is number of switches in loop). The number of PacketIn messages in a second is from 60 to 100. Such a low intensity should be acceptable for any modern controller.

TABLE I. SYSTEM OF NETWORK LOOPS, GENERATED BY ALGORITHM

Loop number	Switches in loop
1	1, 2, 5, 4, 3, 2, 3, 4, 5, 2, 1
2	1, 2, 5, 4, 3, 4, 5, 2, 1
3	1, 2, 5, 4, 3, 2, 1
4	1, 2, 5, 4, 5, 2, 1
5	1, 2, 1
6	2, 5, 2
7	2, 3, 2

Num\link	1 - 2	2 - 1	2 - 3	2 - 5	3 - 2	3 - 4	4 - 3	4 - 5	5 - 2	5 - 4
1	1	1	1	1	1	1	1	1	1	1
2	1	1	0	1	0	1	1	1	1	1
3	1	1	0	1	1	0	1	0	0	1
4	1	1	0	1	0	0	0	1	1	1
5	1	1	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	1	0
7	0	0	1	0	1	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	0

Fig. 4. System of linear equations, generated by algorithm.

V. Conclusion

We proposed a flexible method to measure one-way delay of any flow with adjustable trade-off between the accuracy and the load of network infrastructure it imposes. Using of loops in space of packet states allowed us to measure delay through fast path and make switch-controller delay negligible. Proposed method can be used out-of-the-box, and can be easily implemented as module of any SDN controller.

We proposed an algorithm to construct a delay map suitable to estimate the infrastructure delay for all paths in a network with necessary accuracy in real-time. The algorithm allows our delay measurement method to scale without overloading of the controller.

References

[1] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, “Achieving 100% throughput in an input-queued switch”, IEEE Trans. on Communications, № 8, Vol. 47, pp. 1260—1267, 1999.

[2] A. Bouillard and G. Stea, “Exact worst-case delay for FIFO-multiplexing tandems”, Proc. of the 6th International Conference on Performance Evaluation Methodologies and Tools, 2012.

[3] B. Ngamwongwattana and R. Thompson, “Measuring one-way delay of VoIP packets without clock synchronization”, Proc. of the International Instrumentation and Measurement Technology Conference (I2MTC), pp. 532-535, 2009.

[4] S. Shalunov, B.Teitelbaum and A. Karp, “A One-way Active Measurement Protocol (OWAMP)”, Internet Engineering Task Force, RFC 4656, September 2006.

[5] K.Phemius and M.Bouet, “Monitoring latency with OpenFlow”, 2013 9th International Conference on Network and Service Management (CNSM) and its three collocated Workshops - ICQT, SVM and SETM, pp. 122-125, 2013.

[6] Peyman Kazemian, George Varghese and Nick McKeown, “Header Space Analysis: Static Checking For Networks”, Proc. of the 9th USENIX conference on Networked Systems Design and Implementation, 2012

[7] Pox controller. http://www.noxrepo.org/pox/about-pox/

[8] Robert Olsson, “pktgen the linux packet generator”, Linux Symposium, 2005

[9] Arora Himanshu, “Wireshark - The best open source network packet analyzer”, IBM developerWorks, 2012

[10] Donald B. Johnson, “Finding All the Elementary Circuits of a Directed Graph”, SIAM Journal on Computing 4, no. 1, 77-84, 1975

[11] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz and Nick McKeown, “Reproducible Network Experiments Using Container-Based Emulation”, CoNEXT, 2012

VERMONT - a toolset for checking SDN packet forwarding policies on-line

V. Altukhov, V. Podymov, V. Zakharov
Lomonosov Moscow State University
Moscow, Russia
Email: victoralt@lvk.cs.msu.su, valdus@yandex.ru, zakh@cs.msu.su

E. Chemeritskiy
Applied Research Center for Computer Networks
Moscow, Russia
Email: tyz@lvk.cs.msu.su

***Abstract*—In this paper we present a VERifying MONiTor (VERMONT) which is a software toolset for checking the consistency of network configurations with formally specified invariants of Packet Forwarding Policies (PFP). Correct and safe management of networks is a very hard task. Every time the current load of flow tables should satisfy certain requirements. Some packets have to reach their destination, whereas some other packets have to be dropped. Certain switches are forbidden for some packets, whereas some other switches have to be obligatorily traversed. Loops are not allowed. These and some other requirements constitute a PFP. One of the aims of network engineering is to provide such a loading of switches with forwarding rules as to guarantee compliance with the PFP. VERMONT provides some automation to the solution of this task. VERMONT can be installed in line with the control plane. It observes state changes of a network by intercepting messages sent by switches to the controller and command sent by the controller to switches. It builds an adequate formal model of a whole network and checks every event, such as installation, deletion, or modification of rules, port and switch up and down events, against a set formal requirements of PFP. Before a network update command is sent to a switch VERMONT anticipates the result of its execution and checks whether a new state of network satisfies all requirements of PFP. If this is the case then the command is delivered to the corresponding switch. Upon detecting a violation of PFP VERMONT blocks the change, alerts a network administrator, and gives some additional information to elucidate a possible source of an error. VERMONT has a wide area of applications. It can be attached to a SDN controller just to check basic safety properties (the absence of loops, black-holes, etc) of the network managed by the controller. VERMONT may be also cooperated with software units (like FlowVisor) that aggregate several controllers. In this case VERMONT checks the compatibility of PFPs implemented by these controllers. This toolset can be used as a fully automatic safeguard for every software application which implements certain PFP on a SDN controller.**

Keywords*—runtime verification, formal specification, model checking, software defined network, controller, switch, packet forwarding relation, Binary Decision Diagram, network update

I. Introduction

Runtime verification is an approach to computing system analysis and verification based on extracting information, checking required properties and possibly reacting to the violation of some requirements in the course of system execution. Runtime verification can be used for many purposes, such as security or safety policy monitoring, debugging, testing, verification, validation, profiling, fault protection, behavior modification (e.g., recovery), etc. Runtime verification avoids the complexity of traditional formal verification techniques, such as model checking and theorem proving, by analyzing only one or a few execution traces and by working directly with the actual system, thus scaling up relatively well and giving more confidence in the results of the analysis. Runtime verification can be performed when there is no access to the software code of the computing systems to be verified.

No wonder that these nice features of runtime verification make this approach much favor in using it for verification and analysis of the behaviour of reactive systems, such as network protocols. In this paper we present a VERifying MONiTor (VERMONT) which is a toolset for runtime verification of Software Defined Networks (SDNs) against formally specified invariants of Packet Forwarding Policies (PFP). The paper is organized as follows. In Section 2 we introduce a formal model for SDN configurations, and in Section 3 we present a formal language for PFP specification. In Section 4 we discuss three main tasks to be solved for runtime verification of SDNs, namely, model building, model checking, and model updating. In section 5 we describe our runtime verification toolset, its structure and functionality. And, finally, in the Conclusion we compare our tool set with other SDN verification tools.

II. Network Model

In this Section we presented a relational formal model of SDN configurations which is used in VERMONT for SDN data plane runtime verification. This formal model of SDN has been introduced in [15].

Packet header is a binary vector $\mathbf{h} = (h_1, h_2, \dots, h_N)$. All headers have the same length N and the set of all packet headers is denoted by $\mathcal{H}$. Components of a header $\mathbf{h}$ are denoted by $\mathbf{h}[i],\ 1 \le i \le N$.

Switch port is a binary vector $\mathbf{p} = (p_0, p_1, p_2, \dots, p_k)$. Its components are denoted by $\mathbf{p}[i],\ 0 \le i \le k$. If $\mathbf{p}[0] = 1$ then $\mathbf{p}$ is an input port, otherwise it is an output port. All switches in the network are assumed to be identical and have the same number of ports. The set of all (input,output) ports of a switch is denoted by $\mathcal{P}(\mathcal{IP}, \mathcal{OP})$ respectively. The output port $\mathbf{p} = (0, 0, \dots, 0)$ is viewed as a drop port. It is denoted by $drop$; at arriving to this port the packets are dropped. The output port $\mathbf{p} = \langle 0, 1, 1, \dots, 1\rangle$ is the control output port. It is denoted by $octrl$; at arriving to this port the packets are sent to a controller. The input port $\mathbf{p} = \langle 1, 1, 1, \dots, 1\rangle$ is the control input port. It is denoted by $ictrl$; only commands from the controller come to this port.

All network switches are enumerated. The name of each switch is a binary vector $\mathbf{w} = (w_1, w_2, \dots, w_m)$. Its components are denoted by $\mathbf{w}[i],\ 0 \le i \le m$. The set of such vectors is denoted by $\mathcal{W}$.

Let $\mathbf{h} \in \mathcal{H},\ \mathbf{p} \in \mathcal{P},\ \mathbf{w} \in \mathcal{W}$. Then a pair $\langle \mathbf{h}, \mathbf{p} \rangle$ is called a *local packet state*, a pair $\langle \mathbf{p}, \mathbf{w} \rangle$ is called a *vertice*, and a triple $\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle$ is called a *packet state*. The set of all packet states is denoted by $\mathcal{S}$.

A *header pattern* is a vector $\mathbf{z} = (\sigma_1, \sigma_2, \dots, \sigma_N)$, where $\sigma_i \in \{0, 1, *\},\ 1 \le i \le N$. A *port pattern* is a vector $\mathbf{y} = (\delta_1, \delta_2, \dots, \delta_k)$, where $\delta_i \in \{0, 1, *\},\ 1 \le i \le k$. Patterns are used for the selection of appropriate rules from flow tables as well as for the updating of packet headers.

In our model of SDN we consider two types of *actions*: forwarding actions $OUTPUT(\mathbf{y})$, where $\mathbf{y} \in \mathcal{OP}$, and header modification action $SET\_FIELD(\mathbf{z})$, where $\mathbf{z}$ is a header pattern. An *instruction* is any finite sequence of actions.

A *rule* is a tuple $\mathbf{r} = \langle (\mathbf{z}, \mathbf{y}), \alpha, \ell \rangle)$, where $\mathbf{z}, \mathbf{y}$ are header and port patterns, α is an instruction, and ℓ is natural number which is a *priority* of the rule. A *flow-table tab* is a pair (D, β), where $D = \{r_1, r_2, \dots, r_n\}$ is a set of forwarding rules and β is a default instruction. A switch applies rules from its flow-table to those packets which arrive to the input ports of a switch. If all rules from the set D are inapplicable to a packet then the default instruction β takes effect. Usually in practice β just sends the packets to the SDN controller. The set of all possible flow-tables is denoted by Tab.

Unlike the SDN models introduced in [1], [2], [3] our model deals with paths in the data plane routed by forwarding rules (per flow model) rather than individual packets that traverse a network of switches (per packet model). Therefore, the semantics of the SDN model is defined in terms of packet forwarding relations on packet states and vertices. These relations are specified by Quantified Boolean Formulae. To capture the effect of patterns in forwarding rules we use two auxiliary functions $U_\sigma(u, v)$ and $E_\sigma(u)$, where $\sigma \in \{0, 1, *\}$, and u, v are binary vectors, such that

- if $\sigma = *$, then $U_\sigma(u, v)$ is $u \equiv v$ and $E_\sigma(u)$ is 1,
- if $\sigma \in \{0, 1\}$, then $U_\sigma(u, v)$ and $E_\sigma(u)$ are $u \equiv \sigma$.

An action $a = OUTPUT(\mathbf{y})$ sends packets without changing their header to all output ports that match a pattern $\mathbf{y} = (\delta_1, \delta_2, \dots, \delta_k)$. It computes the relation

$$R_a(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \bigwedge_{i=1}^{N} (\mathbf{h}[i] \equiv \mathbf{h}'[i]) \ \wedge \ \bigwedge_{i=1}^{k} U_{\delta_i}(\mathbf{p}'[i], \mathbf{p}[i])$$

on the set of local packet states $\mathcal{H} \times \mathcal{P}$.

An action $b = SET\_FIELD(\mathbf{z})$ uses a pattern $\mathbf{z}$ to modify headers of packets: a bit $\mathbf{h}[i]$ in a header remains intact if $\mathbf{z} = *$, otherwise it is changed to $\mathbf{z}[i]$. This action computes the relation

$$R_b(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \bigwedge_{i=1}^{N} U_{\sigma_i}(\mathbf{h}'[i], \mathbf{h}[i]) \wedge \bigwedge_{i=1}^{k} (\mathbf{p}[i] \equiv \mathbf{p}'[i]) \ .$$

on the set $\mathcal{H} \times \mathcal{P}$.

An instruction α computes the relation R_α which is a sequential composition of the relations that correspond to its actions. If α is empty then a packet by default have to be dropped, i.e. sent to the port $drop$. Therefore, we assume that every instruction always ends with a forwarding action.

A packet forwarding rule $r = (\langle \mathbf{z}, \mathbf{y} \rangle, \alpha, \ell)$ applies the instruction α to all packets whose port and header match the patterns $\mathbf{y}$ and $\mathbf{z}$. Its effect is specified by the relation R_r on the set of local packet states $\mathcal{H} \times \mathcal{P}$

$$R_r(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = PRC_r(\langle \mathbf{h}, \mathbf{p} \rangle) \wedge R_\alpha(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \ ,$$

where $PRC_r(\langle \mathbf{h}, \mathbf{p} \rangle) = \bigwedge_{i=1}^{k} E_{\delta_i}(\mathbf{p}[i]) \ \wedge \ \bigwedge_{j=1}^{N} E_{\sigma_j}(\mathbf{h}[j])$ is a *precondition* of the rule r.

The semantics of a flow-table $tab = (D, \beta)$, where $D = \{r_1, r_2, \dots, r_n\}$ is specified by a binary relation as follows. Let n be the highest priority of the rules from tab. For every $i,\ 1 \le i \le n$, denote by tab^i the set of rules from tab which have priority i: $tab^i = \{r = (\langle \mathbf{z}, \mathbf{y} \rangle, \alpha, i) \ : \ r \in tab\}$. Then define recursively (from n down to 1) the pairs of relations R^i_{tab} and B^i_{tab} as follows:

$R^n_{tab} = \bigvee_{r \in tab^n} R_r$, $B^n_{tab} = \bigvee_{r \in tab^n} PRC_r$;

$R^i_{tab} = \{(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle)$: there exists r in tab^i such that $\langle \mathbf{h}, \mathbf{p} \rangle \notin B^{i+1}_{tab}$ and $(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \in R_r\}$),
$B^i_{tab} = B^{i+1}_{tab} \vee \bigvee_{r \in tab^i} PRC_r$.

Since the missed packets are managed by the default rule β, we introduce also the predicate

$$R^0_{tab}(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) = \neg B^0_{tab}(\mathbf{h}, \mathbf{p}) \wedge R_\beta(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle).$$

Finally, $R_{tab} = \bigvee_{i=0}^{n} R^i_{tab}$; it means that every packet arrived at some port of the switch is either processed by the rule of the highest priority that matches the local state of the packet, or it is managed by the default rule β of the flow-table.

The topology of a network is completely defined by a *packet transmission relation* $T \subseteq (\mathcal{OP} \times \mathcal{W}) \times (\mathcal{IP} \times \mathcal{W})$. In practice T is an injective function. Vertices that are involved in the relation T are called *internal vertices* of the network; others are called *external vertices*. We denote by In and Out the sets of all external input vertices and external output vertices of a network respectively. External vertices of a switch are assumed to be connected to outer devices (hosts, servers, gateways, etc.) that are out of the scope of the SDN controller. Packets enter a network through the input vertices and leave a network through its output vertices.

When a set of switches $\mathcal{W}$ and a topology T are fixed then a *network configuration* is a total function $Net : \ \mathcal{W} \to Tab$ which assign flow-tables to the switches of the network. The semantics of a network at a configuration Net is specified by the *1-hop packet forwarding relation* R_{Net} on the set of (global) packet states S as follows:

$R_{Net}(\langle \mathbf{h}, \mathbf{p}, \mathbf{w} \rangle, \langle \mathbf{h}', \mathbf{p}', \mathbf{w}' \rangle)$ holds iff

- either $(\langle \mathbf{h}, \mathbf{p} \rangle, \langle \mathbf{h}', \mathbf{p}' \rangle) \in R_{Net(\mathbf{w})}$, $\mathbf{w} = \mathbf{w}'$, and $\langle \mathbf{p}', \mathbf{w} \rangle \in Out$ (a packet is forwarded to an outer device connected to an external output port $\mathbf{p}'$ of a switch $\mathbf{w}$),

- or there exists a port $\mathbf{p}''$ such that $(\langle \mathbf{h}, \mathbf{p}\rangle, \langle \mathbf{h}', \mathbf{p}'\rangle) \in R_{Net(\mathbf{w})}$ and $(\langle \mathbf{p}'', \mathbf{w}\rangle, \langle \mathbf{p}', \mathbf{w}'\rangle) \in T$ (a packet is delivered to an input port $\mathbf{p}'$ of a switch $\mathbf{w}'$).

Thus, a formal model of SDN configuration Net is a triple $M_{Net} = (R_{Net}, In, Out)$.

Network configurations alter at the expiry of forwarding rules' time-outs, at the shutting down or failure of links, ports, or switches, and by the *network updating commands* received from the controller. OpenFlow protocol [4] includes network updating commands of the following types:

- $add(\mathbf{w}, r)$ to install a forwarding rule r in the flow-table of a switch $\mathbf{w}$;
- $del(\mathbf{w}, \langle \mathbf{z}, \mathbf{y}\rangle, \ell)$ to remove rules from the flow-table of a switch $\mathbf{w}$: a rule $r = (\langle \mathbf{z}', \mathbf{y}'\rangle, \alpha, m)$ is uninstalled iff $m = \ell$ and the pattern $\langle \mathbf{z}', \mathbf{y}'\rangle$ of the rule matches the pair $\langle \mathbf{z}, \mathbf{y}\rangle$;
- $mod(\mathbf{w}, \langle \mathbf{z}, \mathbf{y}\rangle, \beta, \ell)$ to modify the rules in the flow-table of a switch $\mathbf{w}$: if pattern $\langle \mathbf{z}', \mathbf{y}'\rangle$ of the rule $r = (\langle \mathbf{z}', \mathbf{y}'\rangle, \alpha, m)$ matches the pair $\langle \mathbf{z}, \mathbf{y}\rangle$ and $m = \ell$ then the instruction α in such rule is changed to the instruction β.

As a network updating command is delivered to a switch it changes the flow-table of the switch by installing, removing or modifying the appropriate forwarding rules. Formally, we write $com(Net)$ for the new configuration obtained at the execution of a network updating command com on a configuration Net.

III. Specification Language

Usually a wide range of requirements is imposed upon communication networks to guarantee their correct, safe and secure behaviour. We consider only those requirements that concern the reachability properties. Certain packets have to reach their destination, whereas some other packets have to be dropped. Certain switches are forbidden for some packets, whereas some other switches have to be obligatorily traversed. Loops are not allowed. These and some other requirements constitute a *Packet Forwarding Policy* (PFP). One of the aims of network engineering is to provide such a loading of switches with forwarding rules as to guarantee compliance with a given PFP. Since flow-tables of switches are updated by the controller, this raises the problems of verification of SDN configurations against PFPs. In order to apply formal methods to this problem one needs a formal language to specify forwarding policies.

PFPs refer to properties of network configurations at some stages of the SDN behaviour. These properties mostly concern the paths routed in a network by packet forwarding rules. We choose first-order logic with two variables and transitive closure operator (FO^2[TC] in symbols) to specify the properties of network configurations. As for atomic formulae, we use for this purpose Boolean formulae to specify relationships between packet states and three basic predicates R, I, and O to denote one-hop packet forwarding relation and the sets of incoming and outgoing packet states. Now we consider this PFP specification language in some more details.

Let $Var = \{X, Y\}$ be a set of two variables that are evaluated over the set $\mathcal{S} = \mathcal{H} \times \mathcal{P} \times \mathcal{W} = \{0,1\}^{N+k+m}$ of packet states. A *packet state specification* is any Boolean formula φ constructed from a set of Boolean variables $\{X_i[j] : X_i \in Var,\ 1 \le j \le N+k+m\}$ and connectives $\neg$, $\wedge$, $\vee$. A PFP specification language $\mathcal{L}$ is the smallest set of expressions which satisfies the following requirements:

1) if φ is a packet state specification then $\varphi \in \mathcal{L}$;
2) if $X', X'' \in Var$ then $R(X', X''), I(X'), O(X'')$ are in $\mathcal{L}$;
3) if $\psi(X, Y)$ is a formula in $\mathcal{L}$ and it includes exactly two free variables then $TC(\varphi(X, Y)) \in \mathcal{L}$;
4) if ψ_1 and ψ_2 are formulae in $\mathcal{L}$ and $X \in Var$ then the formulae $(\neg\psi_1)$, $(\psi_1 \wedge \psi_2)$, , $(\psi_1 \vee \psi_2)$, $(\exists X\ \psi_1)$, and $(\forall X\ \psi_1)$ are in $\mathcal{L}$.

A *PFP specification* is any closed formula in $\mathcal{L}$.

The semantics of $\mathcal{L}$ is defined as follows. Let Net be a network configuration, and $\mathbf{s} = \langle \mathbf{h}, \mathbf{p}, \mathbf{w}\rangle$ and $\mathbf{s}' = \langle \mathbf{h}', \mathbf{p}', \mathbf{w}'\rangle$ be a pair of packet states. Then

1) $Net \models R(X, Y)[\mathbf{s}, \mathbf{s}']$ iff $(\mathbf{s}, \mathbf{s}') \in R_{Net}$;
2) $Net \models I(X)[\mathbf{s}]$ iff $\langle \mathbf{p}, \mathbf{w}\rangle \in In$;
3) $Net \models O(X)[\mathbf{s}]$ iff $\langle \mathbf{p}, \mathbf{w}\rangle \in Out$;
4) $Net \models TC(\varphi(X, Y))[\mathbf{s}, \mathbf{s}']$ iff there exists a finite sequence of packet states $\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_n$ such that $\mathbf{s}_0 = \mathbf{s}$, $\mathbf{s}_n = \mathbf{s}'$, and $Net \models \varphi[\mathbf{s}_i, \mathbf{s}_{i+1}]$ holds for every i, $0 \le i < n$.

The satisfiability relation for other formulae in $\mathcal{L}$ is defined straightforward as in the first-order logics.

Some simple examples show that $\mathcal{L}$ is rather expressive to formalize PFPs.

1) No loop-holes are reachable from the outside of the network:
$\neg\exists X\ (I(X) \wedge \exists Y\ (TC(R(X, Y)) \wedge TC(R(Y, Y)))$;
2) Packet flows $flow_1$ and $flow_2$ do not pass the same switch:
$\neg\exists X\ (\exists Y\ (flow_1(Y) \wedge I(Y) \wedge TC(R(Y, X)))\wedge$
$\exists Y\ (flow_2(Y) \wedge I(Y) \wedge TC(R(Y, X))))$,
where $flow_1$ and $flow_2$ are Boolean formulae which specify the aforesaid flows.

There are several reasons to explain our choice of FO^2[TC] for PFP specification language. In the most papers that study verification problem for SDN (see [8], [9], [10], [11], [14]) the authors use temporal logics (LTL or CTL) for PFP specification language. This choice is explicable when per-packet abstraction is concerned since the movement of a packet may be viewed as a process evolving in time. But as soon as our model has a per-flow abstraction level, temporal logics become inadequate formalism. Since we are interested in the relationships between packet states and routs in the network configurations, FO^2[TC] expresses these more properties far more explicitly. Moreover, as it was shown in [6], [7], LTL, CTL, μ-calculus, and PDL can be translated in FO^2[TC]. This fragment of 2-nd order logics is well-suited for model checking. As it follows from the results of [5], model checking problem for FO[TC] is NLOG-complete. The very structure of FO^2[TC] provides a possibility to evaluate it in straightforward manner on any finite model.

IV. Model Building, Model Checking and Model Updating

The aim of run-time verification is to check the correctness of program behaviour in the course of program execution. In the framework of our per-flow abstract model of SDN this problem can be formalized as follows: given an initial network configuration Net_0, a list of PFP formal specifications $\Phi = \{\varphi_1, \dots, \varphi_n\}$, and a sequence of network updating commands $\alpha = com_1, \dots, com_i, \dots$, check that for every i, $i \geq 1$, a network configuration $Net_i = com_i(Net_{i-1})$ satisfies all PFP specifications, i.e. all formulae from the list Φ are invariants of the sequence α.

To cope with this problem one needs some means to solve three individual tasks:

1) build a formal model M_{Net} of SDN configuration by the description of SDN topology T and the content of SDN switch flow-tables Net,
2) check satisfiability $M_{Net} \models \varphi$ of a given formal specification φ on a given formal model M_{Net} of SDN configuration, and
3) update a formal model M_{Net} of SDN configuration Net at the execution of a network updating command com on this configuration.

We briefly discuss our approach to the solution of these tasks.

A formal model of SDN configuration M_{Net} is completely specified by the finite relations R_{Net}, In, Out on the set of binary vectors (packet states and vertices). Finite relations can be represented symbolically by Binary Decision Diagrams (BDDs) that are well-suited for set-theoretic manipulations with such relations (see [17]). Nowadays many software packages for computations on BDDs are available; in our project we used the toolset BuDDy due to its simple and convenient interface.

Using packages for manipulations with BDDs it is quite easy to solve the model building task. To this end it is sufficient to compute step by step in a straightforward way BDDs for all relations involved in the definition of R_{Net} (see Section II), namely for actions, instructions, rules, flow-tables.

As for the second task, model checking, it can be easily solved as well with the help of BDD. Every formula φ from the specification language $\mathcal{L}$ is presented by an Abstract Syntax Tree (AST) T_φ. The leaves of this tree are variables X and Y, whereas the inner nodes of this tree are basic predicates R, I, O of $\mathcal{L}$, Boolean operators and quantifiers, and transitive closure operator TC. To check $M \models \varphi$ it is sufficient to evaluate T_φ on a model M. Vertices marked with basic predicates invoke the corresponding BDDs (with possible variable renaming). If a vertex is marked with a Boolean operator or a quantifier then the corresponding procedures for manipulations with BDDs is used to assign a BDD to this vertex. The only type of vertices that require some specific treatment are those that are marked by transitive closure operator. To build a BDD for $TC(R_0)$, given a BDD for a binary relation R_0, we use the following simple scheme: compute iteratively BDDs for relations

$$R_{i+1}(X, Y) = \exists Z \ (R_i(X, Z) \wedge R_i(Z, Y))$$

until $R_{i+1} = R_i$. This is the most time consuming stage of AST evaluation and much efforts have been made to implement it efficiently. Since every specification formula is closed then BDD assigned to the root of T_φ is a Boolean constant which indicates (un)satisfiability of φ on M.

Some heuristics are used to reduce the cost of AST evaluation. For example, in practice only some fields (VLAN, counters, etc.) in packet headers are subjected to SET_FIELD actions. Therefore, a packet header $\mathbf{h}$ may be split into two components $\mathbf{h} = (\mathbf{h}', \mathbf{h}'')$, where $\mathbf{h}'$ is composed of those bits that are not changed. Then 1-hop packet forwarding relation R_{Net} may be viewed as $R_{Net}(\mathbf{h}'_1, \mathbf{h}''_1, \mathbf{p}_1, \mathbf{w}_1, \mathbf{h}''_2, \mathbf{p}_2, \mathbf{w}_2)$. Such a presentation substantially reduces the size of BDDs.

An efficient solution of the third task — model updating — is crucial for the utility of run-time verification, since the performance of model updating procedure must be adequate to the rate of configuration updatings occurred in real-life networks. In some cases the basic relations in the SDN configuration models can be modified rather quickly.

V. A Toolset VERMONT

An SDN run-time verification toolset VERMONT includes four components (see Fig. 1):

1) a module for intercepting OpenFlow commands and messages (Proxy-Server),
2) a model checking module (Verifier),
3) an initializing module (Feeder),
4) a PFP specification editor (Editor)

While operating the toolset interacts with OpenFlow controller and SDN switches (when carrying out experiments we used instead Mininet — a software system for SDN prototyping [16]).

One of the aims of network engineering is to provide such a loading of switches with forwarding rules as to guarantee compliance with the PFP. VERMONT provides some automation to the solution of this task. VERMONT can be installed in line with the control plane. It observes state changes of a network by intercepting messages sent by switches to the controller and command sent by the controller to switches. It builds an adequate formal model of a whole network and checks every event, such as installation, deletion, or modification of rules, port and switch up and down events, against a set formal requirements of PFP. Before a network update command is sent to a switch VERMONT anticipates the result of its execution and checks whether a new state of network satisfies all requirements of PFP. If this is the case then the command is delivered to the corresponding switch. Upon detecting a violation of PFP VERMONT blocks the change, alerts a network administrator, and gives some additional information to elucidate a possible source of an error. VERMONT has a wide area of applications. It can be attached to a SDN controller just to check basic safety properties (the absence of loops, black-holes, etc) of the flow-tables managed by his controller. VERMONT may be also cooperated with software units (like FlowVisor) that aggregate several controllers. In this case VERMONT checks the compatibility of PFPs implemented by these controllers. This toolset can be used as a fully automatic safeguard for every software application which implements certain PFP on a SDN controller.

To achieve these tasks the modules of VERMONT operate as follows.

Proxy-Server communicates with OpenFlow controller, SDN switches and Verifier Server. It intercepts all commands sent by the controller to SDN switches and all messages transmitted from the SDN switches to the controller. Proxy-Server is managed by the user of VERMONT (network manager) who can turn on and off this module, select its operational mode (SEAMLESS, MIRROR, INTRRUPT), set up and change the operation parameters. Depending on the chosen operation mode Proxy-Server may provide data (OpenFlow messages and commands) to Verifier, suspend some commands sent by the controller to SDN switches and block some of these commands by the results of their verification.

Verifier communicates with Proxy-Server, Feeder and Editor. This module runs three main algorithms:

- an initialization procedure which, given a description of a current network configuration (i.e. network topology and the content of flow-tables of the network switches) Net, builds a BDD representation of 1-hop packet forwarding relation R_{Net};
- a model checking procedure which verifies a set of PFP formal specifications $\Phi_1, \ldots, \Phi_n$ against a formal model of network configuration Net.
- a model updating procedure which, given a BDD representation of 1-hop packet forwarding relation R_{Net} for a current network configuration Net and a network updating command com builds 1-hop packet forwarding relation $R_{com(Net)}$ for the updating of Net.

Verifier as a server receives from Proxy-Server a sequence of OpenFlow network updating commands and messages on forwarding rules time-out expirations, and (depending on the operation mode of the toolset) it checks the correctness of commands w.r.t. given PFP specifications, blocks incorrect commands, and informs the user about the results of the verification. Verifier as a client may send requests to Feeder for the descriptions of a current network configuration. Verifier as a server receives from Editor formal specifications of a current PFP.

Feeder interacts with Verifier and with OpenFlow controller. At the requests from Verifier it communicates with the OpenFlow controller as a client and asks it about the necessary data. At receiving the data on current network configuration Feeder retransmits them to Verifier.

By means of Editor a user of the Toolset may input PFP formal specifications, check their syntactic correctness, and send these specifications to Verifier.

VERMONT admits three modes of operation.

1) SEAMLESS mode. In this mode VERMONT operates like a control flow channel between the OpenFlow controller and the network of switches. Proxy-Server does not invoke Verifier, updating commands are not suspended and they are delivered to corresponding switches without delay. VERMONT proceeds to this mode either by the request from its user (manually), or at the shutting down of communication with Verifier (automatically).
2) MIRROR mode. In this mode Proxy-Server retransmits without delay all OpenFlow commands and messages to the corresponding parties (controller and switches) but the copies of these control flow data are delivered to Verifier. At receiving network updating commands Verifier checks their correctness; it informs a user about the results of the checking, but does not block incorrect commands.
3) INTERRUPT mode. In this mode VERMONT carries out a full-fledged run-time verification of network configurations and handle the flow of network updating commands sent to by the OpenFlow controller to SDN switches. All updating commands and statistics requests that depend on these commands are suspended by Proxy-Server until their verification is completed. The copies of suspended commands are delivered to Verifier. It simulates the execution of every such command on the current network configuration and checks the resulting configuration against the PFP specifications it received from Editor. If all PFP requirements are satisfied then Verifier allows Proxy-Server to sent the command to the corresponding switch. Otherwise, Proxy-Server drops the command and informs the network manager about this event.

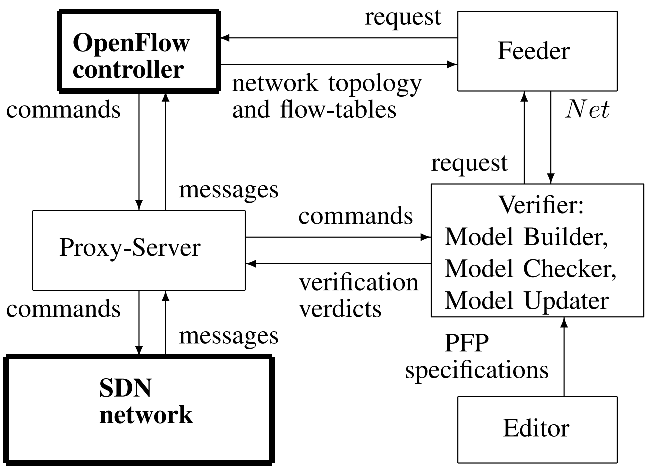


Fig. 1.

VI. Conclusions

We evaluate the performance of our run-time verification toolset VERMONT on the model of Stanford University Backbone Net. This network has 16 switches, and each of them has three flow-tables. Totally, the flow-tables of this net contain more than 750000 forwarding rules. Stanford has made the entire configuration rule set public and it can be found in [18]. This example is used in many papers [8], [9], [10], [12], [13] on network verification as benchmark. The results of comparative analysis of the performance of these tools is presented in the table below (MB — Model Building, MUC — Model Updating and Checking).

Tool	MB (ms)	MUC (ms)	Spec Lang
VERMONT (2013)	4600	100 – 600	FO[TC]
NetPlumber (2013) [12]	37000	2 – 1000	CTL
VeriFlow (2013) [10]	$>$ 4000	68-100	Fixed properties
AP Verifier (2013) [13]	1000	0.1	Fixed properties
FlowChecker (2010) [8]	1200000	350 – 67000	CTL
Anteater (2011) [9]	400000	???	Fixed properties

As it can be seen from this table VERMONT has the most expressive PFP specification language and displays good performance in building initial models of SDN configurations. But some verification toolsets overcome VERMONT in the efficiency of model updating. Nevertheless, we believe that this feature of VERMONT can be substantially improved with the help of new techniques similar to those used in [13]. This is one of the lines of our further research on this topic. Another important task to be solved is the designing and implementation of a new module for generating counter-example in those cases when a network configuration does not satisfies some PFP requirement expressed by a $\forall$-formula of specification language $\mathcal{L}$.

Acknowledgments

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012, and by RFBR Grant 12-01-00706.

References

[1] M. Reitblatt, N. Foster, J. Rexford, D. Walker. Consistent updates for software-defined networks: change you can believe in!. HotNets, v. 7, 2011.

[2] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger. D. Walker. Abstractions for Network Update. In the Proceedings of ACM SIGCOMM, 2012.

[3] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford "A NICE way to Test OpenFlow Applications". In the Proceedings of Networked Systems Design and Implementation, April 2012.

[4] Open Flow Switch Specification. Version 1.3.0, June 25, 2012, http://www.openflow.org/wp/documents/.

[5] Immerman N. Languages that capture complexity classes. SIAM Journal of Computing, v. 16, N 4, 1987, p. 760-778.

[6] Immerman N., Vardi M. Model checking and transitive closure logic. Lecture Notes in Computer Science, 1997, p. 291-302.

[7] Alechina N., Immerman N. Reachability logic: efficient fragment of transitive closure logic. Logic Journal of IGPL, 2000, v. 8, N 3, p. 325-337.

[8] E. Al-Shaer, W. Marrero, A. El-Atawy, K. El Badawi. Network Configuration in a Box: Toward End-to-End Verification of Network Reachability and Security. In the 17th IEEE International Conference on Network Protocols (ICNP'09), Princeton, New Jersey, USA, 2009.

[9] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, R.B. Godfrey, S.T. King. Debugging of the Data Plane with Anteater. In the Proceedings of the ACM SIGCOMM conference, 2011, p. 290-301.

[10] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. VeriFlow: Verifying Network-Wide Invariants in Real Time In the Proceedings of International Conference "Hot Topics in Software Defined Networking" (HotSDN), 2012.

[11] P. Kazemian, G. Varghese, N. McKeown. Header space analysis: Static checking for networks. In the Proceedings of 9-th USENIX Symposium on Networked Systems Design and Implementation, 2012.

[12] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte. Real time network policy checking using header space analysis. In the Proceedings of USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2013.

[13] H. Yang, S. S. Lam, Real-time verification of network properties using atomic predicates. In the Proceedings of IEEE International Conference on Network Protocols, 2013.

[14] S. Gutz, A. Story, C. Schlesinger, N. Foster. Splendid isolation: A Slice Abstraction for Software Defined Networks. In the Proceedings of International Conference "Hot Topics in Software Defined Networking" (HotSDN), 2012.

[15] Chemeritskiy E.V., Smelyansky R.L., Zakharov V.A. A Formal Model and Verification Problems for Software Defined Networks. In the Proceedings of of the 4-th Workshop "Program Semantics, Specification and Verification: Theory and Applications", 2013, Yekaterinburg, Russia, p. 21-30.

[16] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In the Proceedings of the 9-th ACM Workshop on Hot Topics in Networks, October 20-21, 2010, Monterey, CA.

[17] R. E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". IEEE Transactions on Computers, C-35(8):677691, 1986.

[18] Header Space Library and NetPlumber. https://bitbucket.org/peymank/hassel-public/.

Towards SDI-bases Infrastructure for Supporting Science in Russia

V. Antonenko
ARCCN,
Lomonosov Moscow State University
Moscow, Russian Federation
vantonenko@arccn.ru

R. Smeliansky
ARCCN,
Lomonosov Moscow State University
Moscow, Russian Federation
rsmeliansky@arccn.ru

I. Baldin
ARCCN
Moscow, Russian Federation
ibaldin@arccn.ru

Y. Izhvanov
Informika/RUNNet
Moscow, Russian Federation
yli@informika.ru

Y. Gugel
Informika/RUNNet
Moscow, Russian Federation
gugel@run.net

***Abstract*—Modern science presents a number of challenges to the cyber-infrastructure supporting it: heterogeneity of the required computational resources, problems associated with storing, preserving and moving large quantities of information, a collaborative nature of scientific activities requiring shared access to resources, continuously growing requirements for computational power and network bandwidth, and, last, but not least, ease of use. In this position paper we explore a new approach to creating and growing such infrastructure based on the principles of federation, enabled by deep programmability of individual infrastructure elements: Software-Defined Infrastructure (SDI). We describe the evolution of the science infrastructure, open research problems and the concrete steps we are taking towards its realization by building a unique, widely distributed science facility in Russia based on SDI and GENI technologies.**

Keywords—software-defined networking, infrastructure-as-a-service

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and Russian Foundation for Basic Research, project 14-07-00743

I. INTRODUCTIONS

Modern scientific research is impossible without the sophisticated computational and data-processing infrastructure. Different science domains present a variety of challenges to the cyber-infrastructure, which today may consist of desktop computers, small institutional clusters, cloud resources and supercomputers purpose-built to address specific problems. These challenges can be summed up as:

- Infrastructure heterogeneity. Different computational solutions may require differently optimized computational and data-processing architectures. For example part of a given computational workflow may be executed using “map-reduce”, followed by calculations in a tightly-coupled, massively parallel MPI environment. Recent advances in *many-core* systems present new infrastructure optimization points by allowing the use of Intel MIC[1] or GPGPU [2] co-processors targeted at specific computational approaches.
- Data processing. A number of science domains are beginning to encounter a problem that is generically referred to as the “Big Data” problem, where the ability to generate the data by scientific instruments exceeds the ability of the computational elements to process them due to e.g. inadequate network bandwidth and/or insufficient computational power. Added to that are issues with long-term data storage and provenance.
- Distributed science. Today’s progress of science relies on collaborative efforts by multiple institutions and requires cyber resources belonging to multiple organizations. The multi-disciplinary nature of science requires the participation of experts from multiple domains. For example bio-informatics brings together computer scientists and geneticists with the goal of designing efficient genotype processing algorithms.

These challenges are compounded by the reductions in operating budgets of many scientific organizations, which require that the existing or newly constructed cyber-infrastructure is utilized by many collaborative projects across multiple science domains to improve the economies of scale.

In order to answer these challenges we need a new kind of cyber-infrastructure that will simultaneously

- Be deeply reconfigurable, i.e. able to match the requirements of a wide range of computational problems.
- Be economical, i.e. support simultaneous uses by scientists from multiple domains
- Support easy formation of collaborations around available institutional infrastructure by allowing their participants to flexibly ‘opt-in’ portions of their resources without relinquishing control over them.
- Provide for ‘friction-free’ movement of data and computation as determined by the dataset sizes and availability of computational resources.

- Allow for connecting experimental devices and instruments for the purposes of generating or processing data.

In this position paper we explore a new approach to constructing such an infrastructure based on Software-Defined Infrastructure (SDI) technologies that combine performance with the required deep programmability. We also describe our efforts to construct a widely-distributed testbed called "GRANIT – Global Russian Advanced Network InitiaTive" in Russia to help answer the many architectural, deployment and usability questions that we expect to encounter while turning our vision into reality.

II. Related Work

The challenges outlined above have in some form or another been the focus of interest of the research community for some time.

The first systematic approach to address some of them was attempted with the creation of the Grid, standardized via Open Grid Services Architecture (OGSA) [3]. The critical components of the architecture have been realized via Globus Toolkit [4] and Condor-G [5]. The combination of the two allows for distribution of computational tasks over multiple clusters that have Globus interfaces, as is done in many *High-Throughput Computing* (HTC) and *High-Performance Computing* (HPC) facilities. Globus also provides some solutions for data movement via GridFTP software [6] that allows to efficiently move data between clusters by utilizing the available high-bandwidth connections through e.g. multiple parallel TCP sessions.

There have also been attempts to unite under the common grid framework the scheduling of compute resources and on-demand optical network paths that would support large data transfers, as, for example, was done in G-lambda [7] project.

The commonly acknowledged shortcomings of these efforts are in

- The complexity of the configuration and maintenance of the certificate-based grid authorization system.
- The fundamental assumption of the grid computing paradigm that all elements of the grid are continuously connected by reliable high-bandwidth interconnects.
- That the first-class object in the grid architecture is a compute-job, which makes it difficult to attach dedicated networking bandwidth to a particular activity.
- Difficulties in packaging the units of computation with their environments (operating system, data), that lead to productivity lost to adjusting the environments of the executing clusters to run specific applications.
- Difficulties in adapting the grid to conceptually new computational paradigms, such as "map-reduce".

The typical way of achieving data movement in grid-based high-performance and high-throughput environments is by deploying *Data-Transfer Nodes* (DTNs) [20] - specially tuned hardware nodes that have an interface into the public Internet or a bandwidth-on-demand network and also have high-speed access into the shared filesystem of the grid resource. The data transfer software on the DTN (e.g. GridFTP) and the networking stack are specially tuned for sustained high-speed data transfers. It is the responsibility of the user to stage the data in and out of the HTC/HPC resource using the DTNs. The deployment of DTNs represents a form of a static Content Distribution Network (CDN) tuned for super-computing or grid applications.

The *cloud computing* paradigm, which became popular in the recent years also promises to address many of the problems we identified. Originally introduced by several large commercial entities (largely Amazon) as a way to amortize the expense of maintaining their own computational infrastructure, they developed virtualization mechanisms to sell time on that infrastructure during off-peak hours to other customers. It has since become a entire new area of research by commercial and academic research organizations. Among cloud offerings that are of interest to the science community, we should note Amazon Elastic Compute Cloud (EC2) and Simple Storage Service (S3). These initial basic services opened the possibilities for resellers to tailor them to specific customer classes. For example, using EC2, Cycle Computing offers HTC and HPC services for domain scientists struggling to maintain their own infrastructure. Recently they demonstrated a 1.2 peta-flop system built out of EC2 resources for the study of molecular structures of organic semiconductors [8]. Other offerings supporting new computational paradigms include Elastic MapReduce [9]. In parallel to these "public" clouds, new technologies were developed for creating "private" or "institutional" clouds. These include OpenStack[10], Eucalyptus[11], CloudStack[12] and VMWare[13].

Compared to the grid, cloud technologies offer several advantages: deep programmability — since the unit of computation is a virtual machine environment that packages the computation and data together — and massive scale, especially the public clouds, which provide the illusion of infinitely scalable resources. They do, however, have several shortcomings:

- Difficulties with moving data in and especially out of the cloud/vendor lock-in [15]
- Opaqueness of internal cloud topology that does not allow users to predict the performance of network links ahead of time [18]
- Clouds are well suited to "pleasantly parallel" computation paradigms that don't require large amounts of inter-node communications via e.g. MPI
- Difficulties in allowing external users access to specific private cloud resources, which in the grid are resolved using Grid Security Architecture (GSI)

What is required is a combination of some of the best features of grids and clouds with an additional support for programmability, federated access and programmatic control over network resources in a single elastic infrastructure. Due to its deep programmability, this paradigm has been termed "Software-Defined Infrastructure" (SDI) [14], which we examine in the following section.

III. Software-Defined Infrastructure

The move towards SDI began largely with the development of the OpenFlow [16] protocol and the concomitant emergence of the concept of Software-Defined Networking (SDN). Today SDN is understood to be broader than OpenFlow, however this protocol deserves to be called one of the first building blocks towards SDN.

The broadest set of ideas defining SDI today originates from the project named GENI (Global Environment for Network Innovation) [17], initiated by the US National Science Foundation. GENI is an international federation of testbeds by now spanning several countries and offering its resources to many researchers across the globe. GENI resources include cloud compute and storage resources, OpenFlow switches and programmable wireless networks.

GENI control software federates together multiple resource providers and allows users to create "slices" of infrastructure - programmable topologies of resources collected from those providers, connected by various network fabrics which offer bandwidth-on-demand services or higher-level SDN services. These slices are, in effect, independent *virtual systems* or *networks* target-built for specific experimental or computational activities.

The individual slices operate in parallel and independently from each other, with performance isolation provided by the specific virtualization or *slivering* technologies, that allow for programmatic partitioning of various resource types, such as compute hypervisors, individual network paths (VLANs and MPLS) or individually-addressable storage volumes.

Resource *slivers* are mono-typed and user-programmable, i.e. a user may specify, for example, the specific operating system image or file system type in compute and storage slivers, respectively. The slivers within a slice are interconnected into a user-defined topology with dedicated network links of specified performance. Slices may connect to the commodity Internet via a variety of gateway technologies. Taken together, programmable compute, storage and networking form the foundation of the Software-Defined Infrastructure.

We note that the technologies for low-level slivering and programming computational and storage resources are generally well-understood today, although open problems remain when it comes to proper isolation of these sliver types from each other and quantifying their performance in a multi-user system. Network slivers, on the other hand, represent a relatively new idea, which with the emergence of SDN added a new level of programmability, not envisioned before.

Specifically, SDN technologies allow the interposition of user control directly into the network elements within the slice topology, that belong to a specific network provider, thus creating new points for inserting user-specified network traffic forwarding policies. Prior to the introduction of OpenFlow, the level of virtualization and programmability of networks was limited to the creation of bandwidth on-demand paths through one or more network providers [19]. Now users can dictate not only the topologies of the slice interconnects, but actually affect the behavior of the network traffic inside those slices.

These new capabilities coupled with the push by multiple vendors to create OpenFlow-compliant networking equipment capable of serving at different networking layers and different market segments, create new opportunities for extending the notion of SDI deeper into the networking domain. The development and extension of the concept of SDNs into new areas is actively continuing and, along with developing further abilities to orchestrate different resources together into complex slice topologies and behaviors is included as part of our vision in this paper.

The tight coupling of network, computational and storage resources in GENI allows it to become a federated platform unifying multiple independent resource providers for the purposes of solving a variety of computational problems, whose solutions today are limited by their ability to move data, e.g. gene sequencing or processing of high-volumes of other kinds digital data from physics, astronomy, biology or other science domains. For us the problems of data movement within slices in service of science applications represent a critical area for advancement, in order to support the needs of domain sciences.

III. Key Problems for Science Applications and SDI

A. Use Cases

We begin with a few examples of using SDI technologies for scientific computational research. Considering the strong connection between SDI and cloud technologies, one of the easiest areas to apply is to problems that are easily parallelizable. These types of problems are frequently encountered today, for example, in gene sequencing. In part it is because most of the software was originally written for analysis on personal computers, but it is also due to relatively small size of individual genetic datasets, so that they can be packaged together with applications into a virtual machine image.

Using slices created out of institutional private clouds federated using GENI technologies, scientists gain access to a platform that can easily launch large numbers of parallel analyses. However, unlike with public clouds, the users are not limited to using datasets that are baked into the images: their applications may query databases or datasets stored inside or the slice, as necessary and still see predictable data transfer performance. Additionally, the slices may host other auxiliary resources that include, for example, graphical front ends for access by the users. All elements of these infrastructure configurations are completely dynamic, created on-demand as the users see fit.

A more complex example includes a problem that requires elements of parallel computation, but also where some part of the computational process is tightly-coupled and requires MPI on a cluster or supercomputer with an MPI-friendly interconnect. In this case several collaborating organizations can create a virtual system or slice that includes not only cloud computational elements, but also portions or slivers of supercomputers belonging to them, which are connected to the slice when needed. The data needed for processing on the supercomputer can be transferred via the slice network links into the supercomputer and afterwards saved into permanent storage or transferred elsewhere for further processing.

B. Requirements and Open Problems

As we mentioned, any virtual system or *slice* created for a science application consists of three basic sliver types: compute, storage and network. A science application may require heterogeneous computational and storage resources to be present in a slice for solving various steps within the computation. The topology of the slice must support the needs of the application by allowing the movement of data or computation to where it is most efficient, for example, transferring a dataset to available processing compute elements or saving the resulting datasets into permanent storage - all tasks that are difficult today, requiring manual user intervention, as with DTNs and making data sharing, especially in collaborative environments, a difficult task.

In order to provide a predictable high-productivity environment for domain scientists this virtual system must be able to

1. Guarantee the performance of individual slivers. A user may formulate their requirement in the form of an *SLE* (Service Level Expectation), which quantifiably describes their expected behavior. For example for compute slivers it may be specified in the form of expected computational performance, size and I/O speeds for storage, bandwidth and latency for network links.
2. Dynamically control the placement of virtualized slivers into the physical infrastructure of federated providers in order to satisfy the SLE, the indicated topology connecting storage and compute slivers and the anticipated traffic forwarding policies within this topology. Also includes being able to easily recreate a slice configuration whenever a particular experiment or computational activity must be rerun.
3. Be able to efficiently move data or computation, which includes both *provisioning links* with sufficient bandwidth, as well as *providing policies for in-slice control over traffic forwarding* which optimize data movement in response to application objectives.

Some of these questions are being answered in GENI and other research projects. Our focus is on broadening the application of SDI and its extension to new types of resources. Here we describe some of the research directions we plan to take in developing the GRANIT testbed as related to the three requirements formulated above.

C. Heterogeneous Network Virtualization and Bridging

As we mentioned, computational resources in GENI today are represented by cloud-like computational containers capable of launching easily parallelizable tasks. However many computational science domains reliy on tightly coupled models of computation, which require large amounts of inter-node communications via e.g. MPI. The speed and effectiveness of the computations depends directly on the efficiency of the backplane network fabric connecting the nodes. Most supercomputer architectures today rely on InfiniBand interconnects to support these requirements.

InfiniBand is architected for high-performance systems requiring delay-sensitive communications, while Ethernet is firmly entrenched in the data center and even metro-area and some wide-area networks. The popularity of InfiniBand is justified by its relatively low cost and the availability of software solutions that take advantage of it. At the same time, using Ethernet for time-critical communications is still challenging, although the vendor community is working on some converged solutions (like, for example, Data Center Bridging [802.1Qbb, 802.1Qaz etc.]).

In the traditional static computing architectures this dichotomy between InfiniBand and Ethernet does not present a problem, since there is no need to bridge and control the two together - the data is transferred e.g. via DTNs using Ethernet from outside the cluster into the supercomputer filesystem, as we described in Section II, while the nodes inside the supercomputer communicate via InfiniBand.

However, with the ability to federate various resources via GENI technologies, comes the opportunity to tightly interconnect nodes of multiple clusters for the purposes of marshaling their power for addressing a single problem. In this case the InfiniBand fabrics of the supercomputers must be *virtualized* and *joined* together with traditional MAN/WAN mechanisms using Ethernet. This would allow partitions of multiple supercomputers to be joined together in slices with other types of computational or data-generating resources.

Thus *efficient virtualization and bridging of InfiniBand to Ethernet* is one of the problems we plan to address. This fits with the first requirement for satisfying user SLEs. The partitioning of the problem data and codes across the slivers belonging to different supercomputers also represents an interesting area of research we plan to investigate. This could take the form of several tightly coupled models launched on their respective supercomputer partitions and exchanging coupled data in real-time over the bridged InfiniBand/Ethernet fabric. Alternatively, it could be a single problem code and dataset launched on multiple partitions simultaneously as if they were a single supercomputer, with some links experiencing higher latencies due to bridging over Ethernet between different locations. Determining the efficient ways of achieving these partitions, benchmarking the results and devising strategies for determining resource allocation tradeoffs to achieve the best performance are part of our scope.

D. Software-Defined Storage

Software-Defined Storage or SDS represents an important component of the overall SDI system. The key problem to be solved here relates to extremely high expectations placed on data consistency and reliability in the traditional systems. These are a good match for a SAN local to a particular supercomputer, however in a distributed environment due to the CAP (Consistency, Availablity and Partition tolerance) theorem [21] we must find more flexible ways of operating storage in a slice, by making intelligent tradeoffs between those three attributes. Examples of the initial tradeoffs in this area can be seen in the design and deployment of distributed NoSQL databases [22] or distributed storage systems like GlusterFS [23] and CouchDB [24].

These tradeoffs, as applied to block and object storage as well as filesystems in a virtualized SDI environment will involve novel user-programmable behaviors that we plan to concentrate some of our research efforts.

E. Description Languages

A system for supporting scientific experiments and computation must offer its users a way of describing their requirements to the slice. Languages, suitable for describing the experiments are referred to as *domain-specific languages (DSLs)*, and in this case are intended for describing the requests of creating the virtual infrastructure suitable for the experiment. Significant amount of work has been invested in this direction in GENI, where the declarative GENI RSpecs [25] are used for

- Describing the requirements to the desired virtual system
- Describing the resources available within the federation for allocation to virtual system
- Describing the allocated resources back to the user including details of allocation needed for accessing and using the resources.

One key issue lies in the fact that GENI represents a kind of *Infrastructure-as-a-Service* (IaaS), thus focusing on the task of allocating and interconnecting together the resources in the desired topology. However, for serving science the IaaS paradigm may present abstractions that are too low level to be useful. The user must be able to specify not only the necessary resources, but also to some extent pre-define the behavior of the individual slivers by e.g. assigning and configuring applications in them, defining storage access policies for e.g. shared datasets and finally, determining the policies for forwarding traffic within the allocated virtual network, that answer the needs of the application (e.g. prioritize specific types of traffic, route different kinds of traffic in different ways across the topology and so on). All these configurable behaviors fall well within the SDI paradigm, however are not answered by existing GENI technologies.

We plan to *invest significant efforts in designing and implementing a family of domain-specific languages* that answer the needs described above. The related effort would be to design algorithms that operate on these language representations in order to perform the allocation and placement of resources that go into the different virtual systems - a problem that combines aspects of resource scheduling, placement and authorization.

F. Software-Defined eXchanges

The final aspect of this problem area that represents interest to us is the ability to bridge together virtual systems created, for example, by multiple collaborations in order to join their efforts for further collaborative activities and allow the movement of data between their collaborative infrastructures. This requires that multiple virtual systems peer with each other in a controlled fashion.

One of the emergent ideas in this area is the idea of a Software-Defined eXchange [26] - a virtual point location where peering networks can exchange traffic in a way that satisfies their internal policies. The simplest form of an SDX replaces the bulky and expensive to configure BGP protocol with a logical SDN-controlled fabric that can enforce the rules for traffic exchange between the peering partners.

Considering the significant efforts being made by the vendors to extend OpenFlow into new areas of network control and management, we can expect this concept to mature and extend not only to Layer 3 IP networks, but also to Layer 2 and, perhaps, transport networks. The flexibility provided by OpenFlow in terms of traffic matching allows to create sophisticated filtering and forwarding policies at SDXs that are easier to implement and, importantly, validate, than the existing BGP-based solutions.

We plan to investigate the suitability of the SDX paradigm to peering virtual systems serving multiple science applications. This idea also dovetails with our planned research activities into domain-specific languages, as SDXs present a new kind of application running in a slice, which has specific configuration parameters and abstractions that will need to be incorporated into our DSL.

IV. CONSTRUCTING GRANIT

We are in the process of constructing the infrastructure for running and studying distributed science applications in Russia (Figure 1), based on the principles describes in this paper and using SDI and GENI technologies. It has the dual goals of attracting science users to the new type of infrastructure, while at the same time providing a research testbed for addressing the issues of data movement within virtual networks built to serve specific domain science applications.

The testbed is built around a consortium of Russian universities:

1. Lomonosov Moscow State University;
2. Applied Research Center for Computer Networks (ARCCN) ;
3. Saint Petersburg National Research University of Information Technologies, Mechanics and Optics;
4. Moscow Institute of Physics and Technology;
5. The National research university "Higher school of economics";
6. Nizhny Novgorod State Technical University;
7. A. Kharkevich Institute for Information Transmission Problems;
8. Lobachevsky State University of Nizhni Novgorod - National Research University;

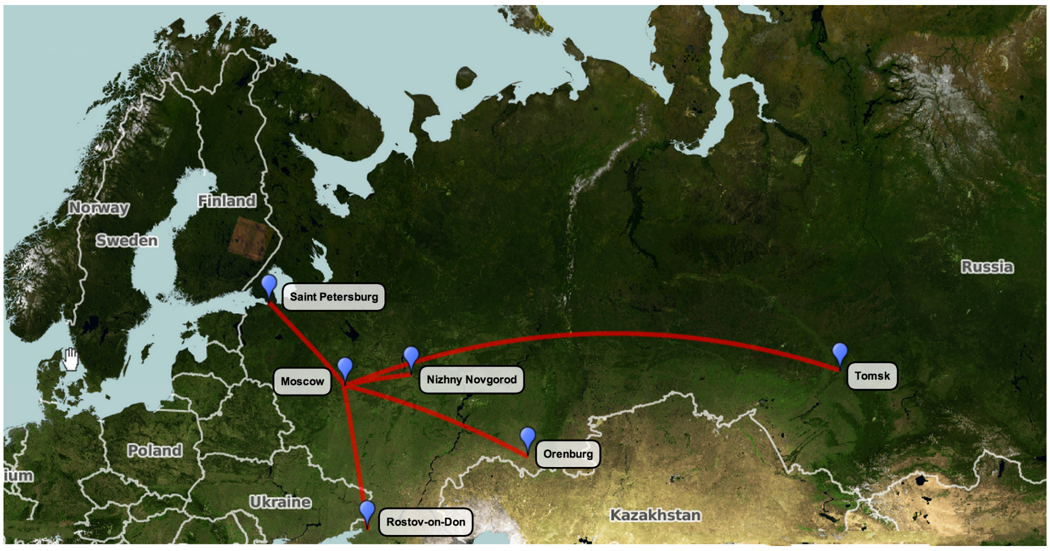


Figure 1: Map of the future testbed.

9. Southern Federal University;

10. Orenburg State University;

11. National Research Tomsk Polytechnic University;

12. State Institute of Information Technologies and Telecommunications.

The core of the infrastructure will be built around cloud components deployed as individual racks of servers, with attached storage and networking capabilities. These racks will be deployed on the campuses of consortium members and interconnected with each other using Russian federal research and education network provider RUNNet. The racks will be controlled using ORCA software suite [27] - one of several control frameworks developed for GENI.

Each rack represents a small private cloud with a selection of SDI technologies available to users and experimenters: compute virtualization via a hypervisor or provisioning of bare-metal nodes, software-sliverable iSCSI storage, onto which we can superimpose programmable high-level distributed storage policies and SDN-capable switches to support the greater degree of control over its networking functions.

Network connectivity between the elements of this system is key to its performance and evolution. This becomes more critical as the scale of the system grows and as it involves more and more members. For example, in the US, the core GENI connectivity depends on Internet2 [28] and its bandwidth-on-demand services AL2S and ION. Internet2 is actively developing SDN-based services for its infrastructure. Similar projects are underway in other research networks around the world: GEANT, NORDUNet, SURFnet [29, 30, 31].

In Russia, RUNNet has points of presence in 63 regions of the country reaching over 2M users. RUNNet partners with a number of commercial providers, like “Rostelecom”, “Transtelecom”, “Megafon”, “Vympelcom” and others to utilize their available fiber and increase its reach.

RUNNet connects to the rest of the world using a number of 10Gbps connections with peerings in Amsterdam and Stockholm with GEANT, NORDUNet, Internet2, GLIF and others. Some parts of RUNNet DWDM infrastructure now run at 100Gbps. RUNNet also peers with commercial providers in- and outside of Russia.

Its infrastructure provides access to high-performance computational resources and instruments that generate large amounts of data in a number of top Russian universities and labs: several Top50 supercomputers, remote robotic protein crystallography stations, instruments for studying synchrotron radiation and so on.

Using this capability we also plan to connect the resources available via RUNNet to our testbed in order to create an environment in which a variety of computational problems with a mix of instruments and computational resources can be addressed. These connections will provide the ability for the users to run their experiments and computations on a heterogeneous set of powerful resources and at the same time enhance our research activities by providing a richer set of problems that need to be addressed.

Selecting an appropriate topology for GRANIT deployment environment depends upon several factors:

1. Scale of project, we planed on first stage to interconnect up to 15 Racks.
2. Amount of traffic expected on the network, base on similar projects in other countries we suppose that 10 Gbps will provide confortable conditions for holding experiments.
3. Budget allotted for the project.

4. Already existed network infrastructure that mentioned above.

Based on all factors we consider to build the hybrid Star-Ring topology. That will provide the scalability without disturbing existing architecture and fault detection and troubleshooting.

Basic workflow for GRANIT user will consist of several steps:

1. Define experiment;
2. Provision the resources;
3. Launch experiment and collect data;
4. Observe and analyze the results.

With the possibility of dynamically adding new computational and storage resources to already launched experiment, GRANIT will become the flexible and powerfully experimental testbad for researchers of natural scientific and computer science profile.

V. Conclusions

In this paper we presented our vision of a future distributed computational science infrastructure built on the principles of federation and using multiple software-defined technologies to support its performance objectives.

We presented a set of open problems we plan to address in order to make our vision a reality. Part of the vision is constructing an advanced testbed connecting deeply reconfigurable compute, storage and networking resources with existing high-performance resource in Russia in a hybrid system that simultaneously serves as a blueprint for evolving the national cyber-infrastructure and a testbed for investigating a number of important problems related to different areas of scientific research.

We are very thankful to GENI Project Office for their openness willingness to help, active support, as well as providing access to GENI toolkits and documentation.

References

1. “Programming and Compiling for Intel® Many Integrated Core Architecture”, https://software.intel.com/en-us/articles/programming-and-compiling-for-intel-many-integrated-core-architecture. Retrieved on June 27, 2014.
2. H. Kim, R. Vuduc, S. Baghsorkhi “Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU).” — Morgan & Claypool Publishers, 2012.
3. OGSA http://toolkit.globus.org/ogsa/
4. Globus Toolkit http://toolkit.globus.org/toolkit/
5. Frey, J., Tannenbaum, T., Livny, M., Foster, I., & Tuecke, S. (2002). Condor-G: A computation management agent for multi-institutional grids. Cluster Computing, 5(3), 237-246.
6. Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., & Foster, I. (2005, November). The Globus striped GridFTP framework and server. In Proceedings of the 2005 ACM/IEEE conference on Supercomputing (p. 54). IEEE Computer Society.
7. Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., Miyamoto, T., ... & Sekiguchi, S. (2006). G-lambda: coordination of a grid scheduler and lambda path service over GMPLS. Future Generation Computer Systems, 22(8), 868-875.
8. MegaRun http://www.cyclecomputing.com/discovery-invention/use-cases/
9. Amazon Elastic MapReduce https://aws.amazon.com/elasticmapreduce/
10. Pepple, K. (2011). Deploying OpenStack. " O'Reilly Media, Inc.”.
11. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., & Zagorodnov, D. (2009, May). The eucalyptus open-source cloud-computing system. In Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on (pp. 124-131). IEEE.
12. Baset, S. A. (2012, October). Open source cloud technologies. In Proceedings of the Third ACM Symposium on Cloud Computing (p. 28). ACM.
13. VMWare http://www.vmware.com/
14. J. Thomas, “A road map to software-defined infrastructure” InfoWorld, November 13, 2013
15. J. Brodkin, “Data movement from Amazon to rival clouds hits speed bump”, ArsTechnica Mar. 20, 2012.
16. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner “OpenFlow: Enabling Innovation in Campus Networks“ ACM SIGCOMM Computer Communication Review, Volume 38, Number 2, April 2008.
17. Elliott, C. (2008, October). GENI-global environment for network innovations. In LCN (p. 8).
18. Wang, G., & Ng, T. E. (2010, March). The impact of virtualization on network performance of amazon ec2 data center. In INFOCOM, 2010 Proceedings IEEE (pp. 1-9). IEEE.
19. Guok, C., Robertson, D., Thompson, M., Lee, J., Tierney, B., & Johnston, W. (2006, October). Intra and interdomain circuit provisioning using the oscars reservation system. In Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on (pp. 1-8). IEEE.
20. Data Transfer Nodes https://fasterdata.es.net/science-dmz/DTN/
21. Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 2 (June 2002), 51-59.
22. Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.
23. Gluster FS http://www.gluster.org/
24. Anderson, J. C., Lehnardt, J., & Slater, N. (2010). CouchDB: the definitive guide. " O'Reilly Media, Inc.”.
25. GENI RSpec http://groups.geni.net/geni/wiki/GENIExperimenter/RSpecs
26. Feamster, N., Rexford, J., Shenker, S., Clark, R., Hutchins, R., Levin, D., & Bailey, J. (2013). SDX: A software-defined internet exchange. Open Networking Summit.
27. Baldine, I., Xin, Y., Mandal, A., Ruth, P., Heerman, C., & Chase, J. (2012). Exogeni: A multi-domain infrastructure-as-a-service testbed. In Testbeds and Research Infrastructure. Development of Networks and Communities (pp. 97-113). Springer Berlin Heidelberg.
28. Internet2 and GENI http://www.internet2.edu/presentations/tip2013/20130116-boyd-fowler-Internet2-GENI.pdf
29. GÉANT SDN research and outlook http://www.ofertie.org/files/2014/02/geant_research.pdf
30. Demonstration: SURFnet SDN/OpenFlow Testbed https://tnc2014.terena.org/core/event/34
31. Integration of NSI and SDN http://apan.net/meetings/Hawaii2013/Session/presentations/APAN-TIP2013-FIT-tanaka-rev04.pdf

An Analysis of Approaches to Onboard Networks Design

V. Balashov, V. Kostenko, P. Vdovin
Department of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
hbd@cs.msu.su, kostmsu@gmail.com,
pavel.vdovin@gmail.com

R. Smeliansky, A. Shalimov
Applied Research Center for Computer Networks
Moscow, Russia
smel@arccn.ru, ashalimov@arccn.ru

***Abstract*—This paper presents a comparison of several approaches to design of data exchange networks for onboard real-time information and control systems (RT ICS). The approaches considered are based on Fibre Channel (FC), Avionics Full Duplex Ethernet (AFDX) and Software-Defined Networking (SDN) technologies. The networks are compared according to the following criteria: ability to guarantee real-time messages transfer; ability to maintain common time in the system; amount of extra hardware resources to ensure the necessary reliability; support for dynamic (during the RT ICS runtime) alteration of message transfer routes.**

Keywords—real-time systems; onboard networks; fibre channel; software-defined networks

I. Introduction

In real-time information and control systems (RT ICS), tasks execution and messages transfer must be performed in strictly defined time intervals. Violation of these intervals leads to RT ICS inoperability. To aircraft onboard RT ICS, besides timing constraints, constraints on weight and dimensions are applied, as well as increased reliability requirements.

Traditionally, point-to-point channels and multiple access channels with centralized control were employed to perform data exchange in onboard RT ICS. This practice resulted in growth of the number of data exchange links according to growth of the number of functional units and subsystems of the aircraft, as well as to increase of requirements to speed and reliability of data transfer. Usage of copper cable for the physical links constrained the onboard network bandwidth and resulted in growth of weight and dimensions of RT ICS. Furthermore, cable shielding was required to protect the network from electromagnetic interference, which led to additional increase of the network weight.

One of the promising approaches to reduction of the number of physical data exchange links is usage of switched data exchange networks based on packet switching. To increase the network bandwidth it is reasonable to widely use optical channels which are by order of magnitude lighter than copper ones and are insensitive to electromagnetic interference.

In this paper we present a comparison of approaches to onboard switched networks design based on Fibre Channel (FC), Avionics Full Duplex Ethernet (AFDX) and Software-Defined Networking (SDN). The networks are compared according to the following criteria:

1) Ability to guarantee real-time messages transfer. This criterion is described in the next section in terms of Service Level Agreement (SLA) requirements.
2) Ability to maintain common time in the system.
3) Amount of extra hardware resources to ensure the necessary reliability.
4) Support for dynamic (during the RT ICS runtime) alteration of message transfer routes without violation of SLA requirements for the other messages.

To ensure the data transfer reliability, it is necessary that two non-intersecting routes for each message are present in the onboard network. For most modern RT ICS this equals to duplication of every physical data exchange channel [1, 2].

Possibility to dynamically alter the message transfer routes is determined by the extent to which the routing tables in the switches can be modified in runtime. In some cases these tables can be modified only before start of RT ICS operation; in other cases, the modification can be performed during the system operation.

Dynamic alteration of message transfer routes is necessary in cases of:

- network equipment and/or computational units failure;
- tasks migration during the RT ICS mode change.

Different RT ICS operation modes involve execution of different, but possibly intersecting, sets of tasks. Tasks migration is relevant for integrated RT ICS in which a common pool of computational resources is shared between different subsystems. Support for tasks migration increases the efficiency of computational resources utilization.

This research is supported by the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60714X0070

II. Data Flows in the Onboard Network

Data flows between network nodes are specified as a set of messages *MSG*={*msg*}. For each message the following attributes are defined:

- $size_{msg}$ – message size;
- for periodic messages: T_{msg} – period of message transfer; for irregular (aperiodic) messages: $\{(s_{msg}, f_{msg})\}$ – set of deadline intervals;
- $src_{msg} \in A$ – message sender node (*A* is the set of all onboard network nodes);
- $\{dst_{msg}\} \subset A$ – set of message receiver nodes;
- J_{msg} – message generation jitter, i.e. fluctuation range for the message generation time in relation to some reference time within the message period or deadline interval.

Message generation jitter emerges because the execution time of the message's source task depends on the values of its input data.

For each message the following SLA requirements are specified to guarantee real time message transfer: For each message the following SLA requirements are specified to guarantee real time message transfer:

- for periodic messages: the message must be transferred no less than once per its period; for irregular messages: the message must be transferred no less than once per each deadline interval;
- τ_{msg} – maximum allowed message transfer latency (duration between message generation on the sender node and message arrival to all receiver nodes);
- J^*_{msg} – maximum allowed message transfer jitter (difference between the maximum and minimum message transfer latencies).

III. AFDX Networks

The Avionics Full Duplex Ethernet (AFDX) standard [3] specifies onboard network design based on the Ethernet 802.3 specification with some modifications to achieve real time operation. According to AFDX, the network consists of the following elements:

- *nodes* which exchange messages;
- *end systems* which provide interface between the nodes and the network;
- *packet switches* connected by data transfer links.

Meeting the constraints on message passing latency in AFDX networks is achieved by allocation of guaranteed bandwidth to connections between pairs of end systems. Such connection can pass through several packet switches and data transfer links. In AFDX, the connection between end systems is referred to as *virtual link*. All data exchange between nodes is performed through virtual links; routes of these links in the physical network are defined in advance. For each virtual link there is one sender end system and one or more receiver end systems. Several nodes connected to the sender end system can send data through the same virtual link.

Reliability of data transfer through an AFDX network is provided by physical sparing. Each end system is connected to two identical independent AFDX networks. The frames are sent to both networks (in each network the frame follows identical routes). If a frame transfer error is detected in one of the networks (e.g. the received frame has incorrect checksum), the duplicate frame is taken from the other network, where there was no error. The receiver end system checks the integrity of the frames, and if a frame was already received from one network, the duplicate frame is discarded.

Routing tables for the AFDX switch are configured for a static set of virtual links defined in advance and covering the set of RT ICS operation modes. Besides routing, AFDX switches perform traffic management and filtering. Filtering includes checking the correctness of frames transfer sequence as well as verification of frames integrity. Traffic management provides guaranteed bandwidth for every virtual link and does not allow the nodes to exceed the bandwidth. To perform traffic shaping in AFDX, the token bucket algorithm is utilized [4]. Bandwidth for each virtual link is specified during the switch configuration before the start of RT ICS operation. Therefore the routing settings for AFDX network, including virtual link routes and bandwidth allocation, are fixed during the RT ICS runtime, and the standard provides no option to dynamically modify the routing tables.

Upon arrival to the sender end system, the messages from a node are split into frames in the link layer; the frames are placed in the appropriate virtual link's queue and then transmitted into the physical data transfer link. Duration of a time interval between consequent frames of the same message (i.e. for the same virtual link) cannot be less than a specific lower bound.

To ensure data transfer determinism, following attributes are specified for a virtual link:

- minimum duration (start to start) between sending of consequent frames into the same virtual link;
- maximum frame size;
- maximum transfer jitter between two consequent frames.

It should be noted that AFDX only accounts for jitter between two consequent frames (of the same virtual link) and does not account for message generation jitter within the message's period. For instance, in the paper [5] a technique is presented for calculating the virtual link attributes according to data flow parameters and constraints on maximum message transfer latency. The paper assumes strictly periodic generation of messages. Operation with irregular messages in

AFDX networks, as well as accounting for message generation jitter J_{msg} within the message period, is not considered.

IV. Fibre Channel Networks

Fibre Channel (FC) standard [6] specifies data exchange protocols for high speed (1 to 20 Gbit/s) data exchange networks. This standard supports the following network topologies: point-to-point, arbitrated loop, switched fabric. In this paper we consider switched fabric topology for RT ICS networks, as point-to-point topology is not suitable for complex onboard networks, and FC arbitrated loop does not support concurrent data exchange between several pairs of nodes.

Let us consider a simplest switched fabric network consisting of a single direct switch and a set of nodes connected to the switch (star physical topology). The statements made below can be generalized for a fabric of multiple switches.

There are following existing approaches to provide guaranteed timings for data transfer over FC network:

1) Master-slave approach in which a single dedicated node supervises data exchange in the network [7]. All the slave nodes transmit data only by command from this master node. This approach guarantees exchange determinism but leads to inefficient utilization of the network bandwidth, as at any time instant only a single pair of nodes can exchange data.
2) Time shared access of nodes to the network according to a static schedule [8]. For each network node there is a data transfer schedule; the schedules are coordinated to avoid access collisions. A node can start data transfer only at time instants specified in the schedule. All nodes have synchronized clocks. The set of schedules allows concurrent data exchange between non-intersecting pairs of nodes. This approach utilizes the inherent concurrency of the FC switched fabric to greater extent than the first one, however is does not support bandwidth sharing between several data flows from the switch to a single node. Furthermore, this approach is not resilient to schedule violation by a single node, or to generation of abnormal data flows.
3) Virtual link-based traffic management implemented in “Fibre Channel – Real Time” (FC-RT) profile which is considered in detail farther on.

As noted above, the approaches 1 and 2 have several drawbacks. Therefore we will concentrate on the FC-RT approach which in fact introduces to FC networks most of the essential data exchange solutions supported in AFDX standard.

According to the FC-RT profile, data are transferred through virtual links with bandwidth control. As in AFDX, for every virtual link there is a single sender node and one or more receiver nodes. The set of virtual links and their routes is fixed for each RT ICS operation mode, but FC-RT provides support for several virtual link configuration tables (configurations) on nodes and the switch, with transitions between configurations by commands from a dedicated *configuration master* node. Transition of the network to a different configuration (e.g. during RT ICS mode change) is initiated by the configuration master via sending a broadcast message containing the number of the new configuration. Consistency of data exchange through virtual links is not guaranteed during the transition between network configurations.

Use of virtual links in FC-RT enables guaranteed message transfer timings. Like AFDX, FC-RT utilizes the token bucket algorithm for traffic shaping, however on the nodes this algorithm operates with whole messages, not with separate frames. Following traffic control parameters are specified for a virtual link in the FC-RT network configuration:

- maximum message size;
- period of message generation (i.e. by an application task);
- message generation jitter;
- parameters for the token bucket scheme used for credit allocation: bucket volume and filling speed.

In contrast to AFDX, FC-RT does not implement “sparse” transfer of multi-frame messages, in which there are constraints on minimum start-to-start interval between consequent frames of a message. In the standard scheme for message transmission to the FC-RT channel, all frames are transmitted sequentially without delay. Interruption of message transmission from a node by another message (without interruption of current frame transmission) is possible only when the second message has higher priority.

Reliability of data transfer is provided in FC-RT network by using two independent identical networks. In case of frame loss in the primary network, the receiver node uses the duplicate frame received from the secondary network. In case of successful arrival of both frames, the first arrived frame is used and the second one is discarded.

To support irregular messages in FC-RT, the priority system can be used. Low priority irregular messages do not interfere with data exchange through virtual links, but the transmission latencies for such messages are hardly predictable. High priorities can be assigned to urgent irregular messages, however transmission of such messages can break data exchange through virtual links.

The FC-RT profile provides a service for time synchronization between the network nodes.

V. SDN Networks

The essence of Software-Defined Networking (SDN) approach is separation of data transfer management (Control Plane) and data transfer itself (Data Plane) in the networked devices. Data transfer is managed from a specific center [9, 10].

One of the approaches to SDN implementation is based on the OpenFlow protocol [11]. In terms of OpenFlow, the network consists of (a) switches responsible for packets transfer according to the routing and switching rules stored in the flow tables, (b) the controller responsible for centralized generation of rules and their transfer to all controlled switches, (c) physical data transfer links, and (d) optional dedicated physical links between the switches and the controller. If no dedicated links are present, regular data transfer links are used to exchange data between the switches and the controller.

The controller itself only provides a layer for interaction with the switches via OpenFlow protocol; network management and rules generation is essentially performed by the applications running on the controller.

The OpenFlow network operates as follows. First packet of each new data flow (or a session) is sent to the controller by the boundary switch (i.e. the first switch of the network to receive the packet), as there is no corresponding record in the flow table of the boundary switch. The controller produces the necessary set of rules for the given flow and sends this set to the switches. All subsequent packets of the same flow are processed by switches according to these rules, bypassing the controller. This operation mode is called active. In the passive mode all rules are stored on the switches in advance and no additional processing on the controller is performed.

In order to manage the onboard network according to SDN/OpenFlow approach, the controller must run a dedicated network application which implements following principles of network control:

- In passive mode the application produces and loads the necessary rule sets to the switches in advance, according to SLA requirements specified for the messages. To ensure data transfer reliability, the rule sets must provide two non-intersecting routes for each message. A message is transferred by the secondary route only in case of transfer errors on the primary route, or to provide redundant transfer, in which case several copies of the message are delivered by different routes. It is not strictly necessary to duplicate the whole network to support redundant transfer; the sufficient solution is to provide at least two non-intersecting routes for each message.

- In active mode each message (whole or header only) is processed by the application running on the controller. The application monitors fulfillment of the SLA requirements (message size, period or deadline interval, jitter, addresses of receiver nodes) and reorders the messages if necessary. The main workload in this mode is assigned to the controller which may become a performance bottleneck. However, according to the analysis presented in [12, 13], the performance of OpenFlow controllers is sufficient for processing the messages transferred with frequencies typical for onboard networks. Furthermore, in some cases there is no need for continuous processing of messages on the controller, as it is sufficient to configure the rules on the switch in order to enable it to check the messages arrival frequency for the given data flows.

Presence of the centralized controller enables dynamic reconfiguration of the network in case of RT ICS operation mode change.

In active network operation mode, a failure of the controller or a link connecting the controller to a switch leads to a failure of the whole network operation. So if the network operates in active mode, duplication of the controller and the links connecting the controller to the switches is critical for reliability of data transfer. If some of the “regular” data transfer links are used to connect the controller to the switches, and there are no alternate routes, these links also must be duplicated.

To maintain unified time on the network nodes, the controller can regularly send time synchronization information to the nodes, e.g. according to PTP (Precision Time Protocol).

As the data flows for most of onboard RT ICS operation modes are predictable or even predefined, passive controller mode looks preferable for onboard SDN networks. Ultimately, in this mode the controller application responsible for configuring the OpenFlow switches must perform following activities:

- construction of the routes for message transfer between network nodes to provide the necessary quality of service, including predictable transfer latency and jitter;

- dynamic adaptation of the routes in case of network failures;

- generation of rules for switches, including:

  - rules for checking the traffic for conformance to the SLA requirements;

  - routing rules;

  - rules for distribution of network bandwidth between data flows.

Another approach to application of SDN technology to onboard networks is integration of AFDX or FC-RT networks with OpenFlow networks to enable dynamic management of switches. In this case there is no need to control the message transfer timings on the OpenFlow controller.

VI. Conslusion

Table I presents a comparison of three above mentioned approaches to design of onboard switched networks. The set of requirements met by a specific approach determines the class of RT ICS to which the approach is applicable.

TABLE I. COMPARISON OF APPROACHES TO DESIGN OF ONBOARD SWITCHED NETWORKS

Requirement to the network	**AFDX**	**Fibre Channel**	**SDN/ OpenFlow**
Support for periodic messages	+	+	+
Support for irregular messages	–	+	+
Ability to maintain common time in the system	–	+	+
Guaranteed maximum transfer latency (τ_{msg})	+	+	+
Guaranteed maximum transfer jitter (J^*_{msg})	for frames only	+	+
Ability to provide reliable data transfer without full duplication of the network	–	–	+ [(a)]
Support for dynamic alteration of message transfer routes	–	–	+

[a.] In active operation mode of an SDN/OpenFlow network, duplication of the controller and the links connecting the controller to the switches is necessary.

AFDX and Fibre Channel networks are widely used in RT ICS for modern aircraft. Use of AFDX is limited to civilian aircraft. A specific of AFDX-based RT ICS is presence of only periodic messages (irregular messages must be simulated as periodic ones, leading to bandwidth wasting). FC networks are used in both civilian and military aircraft, including unmanned ones. Like AFDX, FC networks do not support dynamic alteration of message passing routes without total reconfiguration of the network. Therefore, FC networks can be used only in RT ICS for which the set of modes is defined in advance. Applications of SDN networks are not known to the authors of this paper, however this class of networks is potentially applicable to a wider range of RT ICS than AFDX and FC due to higher flexibility.

REFERENCES

[1] V. Kuminov and B. Naumov, “Space computers: open standards and technologies go to the outer space,” World of Computer Automation, 2002, N. 3.

[2] V.V. Balashov, V.A. Balakhanov, V.A. Kostenko, R.L. Smeliansky, V.A. Kokarev, and P.E. Shestov, “A technology for scheduling of data exchange over bus with centralized control in onboard avionics systems,” Proc. Institute of Mechanical Engineering, Part G: Journal of Aerospace Engineering, vol. 224, N. 9, pp/ 993–1004, 2010.

[3] Aircraft Data Network. Part 7. Avionics Full Duplex Switched Ethernet (AFDX) Network. Aeronautical Radio, Inc., 2005.

[4] R.L. Smeliansky. Computer Networks, vol. 2. Moscow, Academy, 2011, 240 pp. [in Russian]

[5] A. Al Sheikh et al, “Optimal design of virtual links in AFDX networks,” Real-Time Systems, vol. 49, N. 3. pp. 308-336, 2013.

[6] INCITS 373. Information Technology – Fibre Channel Framing and Signaling Interface (FC-FS). International Committee for Information Technology Standards, 2003.

[7] ISO/IEC TR 14165-312. Information technology – Fibre channel – Part 312: Avionics environment upper layer protocol (FC-AE 1553). ISO/IEC, 2009.

[8] INCITS T11/08-013v1. Fibre Channel Avionics Environment – Anonymous Subscriber Messaging (ASM) Amendment 1. International Committee for Information Technology Standards, 2008.

[9] M. Casado, T. Koponen, D. Moon, S. Shenker, “Rethinking packet forwarding hardware,” Proc. HotNets, 2008.

[10] R.L. Smeliansky, “Software-Defined Networks,” Open Systems, N.9, 2012. [in Russian]

[11] OpenFlow Switch Specification, Version 1.3.0. Open Networking Foundation, 2012.

[12] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, “Advanced study of SDN/OpenFlow controllers,” Proc. CEE-SECR'13: Central & Eastern European Software Engineering Conference in Russia, ACM SIGSOFT, Moscow, Russia, October 2013.

[13] P. Ivashchenko, A. Shalimov, R. Smeliansky, “High performance in-kernel SDN/OpenFlow controller,” Proc. 2014 Open Networking Summit Research Track, USENIX, Santa Clara, USA, March 2014.

Towards Load Balancing in SDN-Networks During DDoS-attacks

M. Belyaev
Dept. of Computer Systems and Software Engineering
St.Petersburg State Polytechnical University
Applied Research Center for Computer Networks
St.Petersburg, Russia
belyaev@kspt.icc.spbstu.ru

S. Gaivoronski
Computational Mathematics and Cybernetics dept.
Moscow State University
Applied Research Center for Computer Networks
Moscow, Russia
s.gaivoronski@gmail.com

Abstract—**Software Defined Networks (SDN) are becoming a trending technology in modern Internet. This technology helps to solve a significant number of well-known engineering problems in a effective and elegant way as they provide software-defined centralized network control. An SDN controller can be extended with application that effectively serve for concrete purposes and provide flexible management of network flows. This opens a great number of opportunities for a lot of network security problems such as maintaining of privileges in a proper way, splitting control and data planes, and attacks detection and mitigation. In this work we consider the opportunities of SDN for a "survival" mitigation during DDoS attacks, the load balancing problem. We propose two-level balancing solution in SDN networks, which includes traditional balancing between servers and load balancing between network devices as well. Experiments show that our solution increase "survival" time of a system during DDoS attack in times compared to existing balancing solution in SDN networks.**

Keywords—load balancing; DDoS mitigation; SDN networks;

I. Introduction

Several years ago clouds made a computational revolution in IT world. Clouds also can be considered as logical step of computational evolution, starting from computations on single machines and going through clusters and grids. The idea behind the clouds is to migrate all computational, storage, network, even some specific services requirements to a service-oriented platform using virtual machines at data centers. This idea provides great opportunities for variety of consumers: from independed researchers, small and medium businesses to big organizations. The trend of migrating computations to the clouds continues to grow: according to recent statistics, about 60% of server workloads will be virtualized in 2013 [1], and totally cloud service market is forecast to grow to 18.5% in 2013 [2]. Nowadays, a plethora of big organizations extend their resources for cloud computing: Amazon EC2, Windows Azure, Google Engine, etc.

But every story has two sides. Wide spread of cloud technology leads to a number of interesting research problems, one of which is a load balancing among resources. Load balancing is a problem of resource distribution which guarantees that all available resources are used with maximum utilization. Despite the fact that load balancing problem in cloud considers different types of resources, in current work we focus on traffic balancing and network resources utilization. Network resources utilization typically includes L7 load balancing which is balancing between computing nodes or L4 load balancing which is balancing between network equipment. Existing approaches on load balancing typically describe L7 *or* L4 balancing, but not both.

Over the last few years we also can notice the wide adoption of very new conception in networking - programmable networks, or so-called Software Defined Networks. For example, that technology is already adopted by Google and a number of other significant players. In current work we decided to consider load balancing problem in case of SDN networks. SDN decouples control plane from data plane and gives the functionality of network and resources management to *controller* which can be programmable by user. That leads to such advantages as, for example, flexibility of flows management. In current work we ask ourselves a question: given the opportunities of SDN networks, can we reconsider the problem of load balancing? What can be improved and what can we do better? As a result, we propose load-balancing solution that examines only ip source and destination ip addresses. In common terms, our approach may be considered as L4 solution, but in fact in works at even lower level of OSI model.

One of the interesting application of load balancer that we consider in this work is the balancing in case of DDoS attacks. Speaking of DDoS attacks, load balancing is one of the significant mitigation survival techniques that typically increase maximum capacity of defended system.

The contribution of the paper can be summarized as following:

- We researched the opportunities to apply new concept of networks, SDN, for solving a well-known problem of load balancing during DDoS attacks. We found that ideas that stand behind SDN serve for that purpose natively;
- We proposed an efficient algorithm of two-level load balancing which includes typical load balancing between servers and balancing between network devices for SDN networks. Experiments show that our solution significantly increase survival time of the system under DDoS-attack;
- We implemented our algorithm as the part of DDoS

detection and mitigation system, which is able to detect the presence of attack and start different mitigation solutions automatically.

The paper is organized as following. In section II we provide brief overview of existing load balancing techniques. In section III we describe our load balancing solution for SDN networks, including overview of main differences between traditional networks and SDN networks, detailed description of the proposed algorithms and overview of implementation details. In section IV we provide evaluation results and in section V we summarize our work.

II. Load Balancing Background

A. L7 Balancing

L7 load balancing usually decouples on server cluster load balancing and server load balancing[3]. *Server cluster load balancing* is typically produced between computing nodes. The cluster load can be interpreted as client sessions or running applications. In the first case, TCP sessions are evenly distributed between servers and if some server is overloaded it prohibits new incoming connections. The redirection of already established connections are not usually performed due to the TCP session transfer and applications synchronization overheads. Commonly used techniques for distribution of client sessions between servers are using of DNS server or Network Address Translation (NAT). In the case of running applications servers are clustered by the type of their applications (database server, Web applications, etc.) and every client request is divided between several clusters [4]. The case of *server load balancing* occurs when the system tries to decrease the load of particular server. In our work we will not consider that case.

For both NAT and DNS distribution system should choose the most appropriate server for the next session. All balancing techniques typically fall into the classification, which includes static and dynamic approaches.

Static load balancing algorithms use a-priori information about the system state such that throughput or computation power, or any other performance features of selected nodes. Static approaches ignore current state of the nodes and their load. The main advantage of static approaches is an easy implementation, but the possibility of inefficient balancing is high. Static load balancing typically presented by following techniques [3], [5]: random selection; hash selection, where hash is generally considered as a function of client ip addresses; and (weighted) round-robin which may or may not consider performance of servers.

Dynamic load balancing distributes load between the servers during runtime. Such balancers typically monitor the load of every single server and when imbalance reaches specified throughput they start balancing algorithms [6]. The dynamic algorithms include such simple techniques as selection of server with fastest response time, server with the smallest number of connections, dynamic round-robin techniques and others. More sophisticated algorithms were observed and compared in [7]. It includes description of Honeybee Foraging Angorithm [9] that based on nature algorithm of honeybee self-organization; Biased Random Sampling [8] that uses random sampling of the system domain to achieve self-organization; ACCLB which is load balancing mechanism that base on Ant Colony and Complex network theory [10] and several others.

All described algorithms are suitable for different purposes, typically in Cloud computing environment, but none of them take into assumption the information about load of network devices themselves which is significantly important in case of SDN networks.

B. L4 Balancing

L4 balancing is the balancing between network devices and equipment. In case of SDN network, algorithms describing L4 balancing focused at load distribution from different switches between controllers . Such problem is very significant in terms of reliability of SDN networks but do not has any connection with traffic load balancing.

R. Wang at al [11] describes the load balancing between servers solution under the OpenFlow protocol. That solution may occasionally affect alternate routes between entry point and selected server, but it is not studied in the paper.

The lack of load balancing between switches and alternate routes in SDN networks studies has driven us to cover this gap.

III. Load balancing solution for SDN networks

A. SDN background

Software-Defined Networking (SDN) is a rising approach to networking that allows administrators to manage network services through an abstraction of lower-level functionality. SDN is based on decomposition of the network traffic in two layers: *the control plane* and *the data plane*. The control plane is the distributed system that actually makes decisions on where and how the traffic on the data plane (the usual TCP-IP stack user data) is sent.

SDN networks provide a simple and robust way to access all levels of traffic management in the data layer of the network through a simple interface and using software-based mechanisms exclusively. When it comes to the task of load balancing, this approach does provide some pros and cons, including, but not limited to:

- A single point of failure (the SDN controller) that may become a bottleneck of the whole setup if used in a wrong way;
- There is a number of security concerns regarding the complex interactions between the control plane and the data plane using state-of-the-art SDN implementations;
- The ability to provide means for routing, splitting and controlling traffic streams on all layers of TCP-IP stack using a single software-based solution;
- Multiple ways to provide hardware duplication of used lines and connections without any need to make an account for it on the data layer;
- Ways to balance the traffic on OSI model layers 2 and 3 (as opposed to the usually employed levels 4 and 7).

In this work we are trying to employ the benefits of SDN while also avoiding the possible implications that can be caused by negative effects. The security of SDN networking, which is becoming a big concern the more this type of networking gets widespread and can have negative implications on the way load balancing works, is out of scope for this paper.

B. Proposed solution

The proposed solution to traffic load balancing generally consists of two parts: the L7 load balancing using standard means (DNS/NAT balancing) of splitting the traffic streams between endpoint servers and the L4 load balancing to enable splitting the packets between different paths in the network. The first part does not consider the network which lies between entry point and servers as it is shown at figure 1. Such balancer operates in terms of entry point and servers. The second part takes into consideration all information about local network which includes the network topology (map), current load of channels between different switches, throughput of the channels and other significant parameters, as it shown at figure 2. This approach assumes that the SDN network between the server-level load balancer and the endpoint servers is structurally excessive in order to have different paths to the same servers to begin with. We do not describe the way to do L7 load balancing in this paper, which can be found at [3], [4], [7], [8]. It should be noted, however, that the approach we describe in this section does assume that the traffic streams are already distributed between endpoint servers as the algorithm does not provide any kind of distribution between those itself.

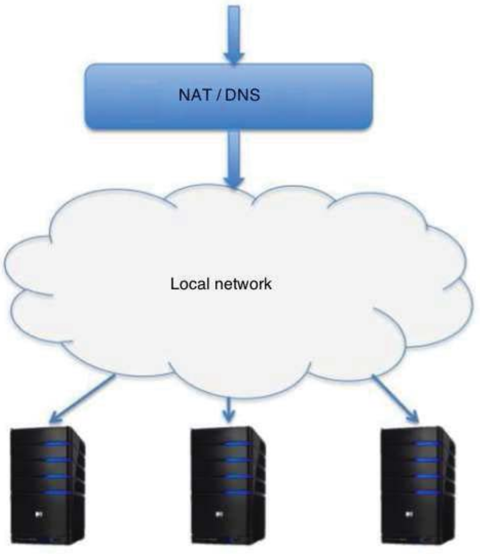


Fig. 1. First level of balancing.

The basic advantage of having these two levels of load balancing is that they can be made sufficiently independent from each other. As the inner balancing algorithm operates on layers 2-4 of the OSI model, it does not care for any of the peculiar properties the outer balancer may introduce, and vice versa. The fact that we use SDN control level to do the job of inner load balancing and do not introduce any additional modifications to the packets themselves, we can be sure that the two levels of load balancing do not interact in any unintended way.

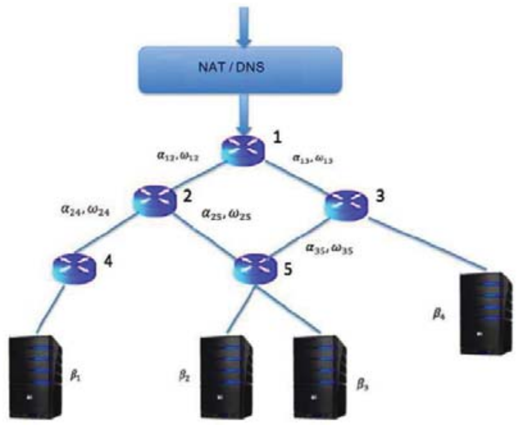

NAT / DNS
1
2
3
4
5
α_{12}, ω_{12}
α_{13}, ω_{13}
α_{24}, ω_{24}
α_{25}, ω_{25}
α_{35}, ω_{35}
β_1
β_2
β_3
β_4

Fig. 2. Second level of balancing. α_{ij} stands for bandwidth of a channel between switches i and j, ω_{ij} stands for current channel load

The algorithm itself is based on the fact that we can use the SDN switch-level flows to redirect traffic based on destination and source IP-address information. This allows for dividing the traffic between different routes in the network regardless of the packets' actual contents. The algorithm goes as follows:

1) Acquire the load and topology information for the network;
2) Override the routing for the network with static routing information acquired by using Bellman-Ford pathfinding algorithm;
3) Iteratively keep splitting (and reapplying) traffic paths for routes that are:
   - Overloaded;
   - Have alternate routes available.

   The splitting is done by using source IP address mask as packet distinguishers.

Let the network contain switches $1 - N$. A *channel* between switches i and j will be addressed as (i, j). We define the *bandwidth* of this channel as α_{ij} and the *current channel load* as ω_{ij}. We say the the channel is ***overloaded*** if $\omega_{ij} + \epsilon \geq \alpha_{ij}$, where ϵ is a constant small load value parameter. In current implementation we define ϵ as an input parameter for the algorithm. The impact of this parameter and the range of its values has not been studied yet and is considered as a direction of futher work.

Let the endpoint servers be defined as $\beta_1 \ldots \beta_K$. Bandwidth matrix $M_{maxload}$ is the matrix of size $N \times N$ containing all the bandwidth values α_{ij}. Load matrix M_{load} is the matrix of size $N \times N$ containing all the current load values ω_{ij}. Matrix of available resources M_{free} is defined as $M_{maxload} - M_{load}$.

Phase 1 of the algorithm need to be executed before the need for load balancing arises (e. g. if we apply the approach to mitigate network attacks, we need to run it in a timed loop without other phases for as long as the attack *doesn't* begin to keep the topology and load information updated). Phase 1 introduces and updates the network load mask M_{load}, where element ω_{ij} corresponds to the number of bytes coming from switch i to switch j during a single update period.

Phase 2 is applied only once to override the default packet routing mechanisms and use statically defined routes we can later modify using network address masks. This is performed by running the standard Bellman-Ford algorithm on the whole network topology graph in order to acquire shortest paths from the network entry point to the endpoint servers.

Phase 3 goes as follows. On the first iteration we build the current path table T_{path} based on the path information acquired on Phase 2. T_{path} is essentially a set of triples $\{ips_{src}, ip_{\beta_i}, path\}$ where each triple denotes a path $path$ from all addresses conforming to the address mask ips_{src} to the address ip_{β_i}, which is the IP address of the server β_i. On the first iteration of phase 3 value of ips_{src} for all the entries in the table is `0.0.0.0/0`, which is a wildcard accepting all the possible IP addresses. On the second and subsequent iterations, this table gets updated along with the corresponding network flows:

1) Update M_{load} and M_{free} with current load information from SDN switches;
2) Find the first overloaded link in M_{load}: the link (i, j) such that $\omega_{ij} + \epsilon \geq \alpha_{ij}$;
3) Find the first path r_q in T_{path} such that it contains link (i, j);
4) For the ip_{β_i} part of r_q, find a new shortest path from entry to the server β_i assuming that link (i, j) is closed in current topology. If there is no such path, we should go back to 3 and find a new path for the same link. If there are no more paths containing this link, we should go back to 2 and select a new link. Let's call the new path $path_q$;
5) Calculate the maximum available additional load for $path_q$. For that, we look up every link in $path_q$ in M_{free}: $al = m_{i_{crit}, j_{crit}} = min\,(m_{ij} : (i, j) \in path_q)$ and note al and (i_{crit}, j_{crit}). If $al < \epsilon$, go back to 4 and find a new path that does not contain (i_{crit}, j_{crit}).
6) Try to calculate the new sets of masks ips_{old} and ips_{new} such that they divide all the address space of ips_{src} into parts with coefficient al/ω_{ij}. Remove the corresponding entry from T_{path}, insert all the entries $\{ips_{old\_k}, ip_{\beta_i}, path\}$ and $\{ips_{new\_k}, ip_{\beta_i}, path_q\}$ into T_{path}.
7) Commit the changes in T_{path} to all the switches across $path$ and $path_q$.
8) Wait for the timeframe and go back to 1.

Of course, it is not generally possible to introduce a set of new network ip/mask pairs for a given one such that it divides all the address space denoted by it to a particular fraction, but it is possible to do it in a discreet manner, up to some number of bits in a network address. For example, if we do the division for 5 significant bits, we can divide the address space into parts that are multiples of $1/(2^5)$. For all practical purposes, this discreetness does not seem to introduce any significant effect on the behavior of the algorithm.

For example, given a ip/mask `9.0.0.0/8` and a factor of $1/3$, we do the following:

1) Introduce the 32 masks dividing the address space given into 32 equal pieces by adding 5 bits to the size of the mask (the ip/mask becoming `9.0.0.0/13`) and enumerating the now valid 5 bits in ip address with values from 0 to 31;
2) Divide the space of 32 address/mask pairs into 2 parts by the ratio: that is, putting first 21 pairs (`9.0.0.0/13`–`9.160.0.0/13`) into first part and last 11 pairs (`9.176.0.0/13`–`9.248.0.0/13`) into the second;
3) Collapse the pairs in each half that can be summarized using a pair with a smaller mask size (e.g. all pairs in (`9.0.0.0/13`–`9.112.0.0/13`) can be collapsed into (`9.0.0.0/9`).

For the given pair `9.0.0.0/8` the result will create 6 address/mask pairs. It can be easily shown that for any N bits used as a discretion factor, any ip/mask pair will produce no more than $N + 1$ new ip/mask pairs in each iteration, thus the growing factor of introduced flows is constant. The ip address space is finite, so this process will always terminate.

It should be noted that we produce flow management that is based at source ip adresses due to the following reasons. First of all, after the phase 1 traffic is already distributed uniformly among available servers. Secondly, we want to distrubute attacking traffic among available routes as uniformly as possible. From that point of view, the worst case to the presented algorithm is the case of DoS attack with one attacking ip address.

C. Implementation

The proposed approach was implemented as a part of a attack detection and mitigation system called `Callophrys`. The system aims at both identifying, detecting and mitigating DDoS attacks at early phases. It uses the Floodlight Openflow controller[12] for both gathering information from the SDN switches and applying the calculated paths and wildcards by deploying them to the switches. Both tasks are achieved through controller's REST API.

`Callophrys` is a distributed software system employing a number of asynchronous agents (actors[13]) communicating using immutable messages both between processes and machines and inside them. This model of computation allows for greater modularity and scalability of the whole system, but also introduces some difficulties for implementation of non-asynchronous algorithms, like the one introduced in the previous section.

The biggest difference for the procedural description of the algorithm found above is the fact that in an asynchronous system there is no need to wait for the next timeframe to come or for the other part of the system (i. e. the SDN controller) to send in the next portion of data to perform useful work. The asynchronous way of implementing the same algorithm is to identify the important events (in this case, keeping the topology and load information updated can be done independently from the rest of the algorithm) and perform actions when they happen. The load balancing algorithm is incapsulated into a single actor (whether it can be further decomposed to employ capabilities of the asynchronous computations is subject to further research) that handles the following kinds of messages:

- *Timeframe* message signaling that a timeframe is reached (sent by the program scheduler):
    1) Send a query message for the current network topology;
    2) Send a query message for the current load information.
- *Topology* message signaling that the topology information has changed (sent by Floodlight interface):
    1) Update the topology information.
- *Load* message signaling that the periodic load information is received (sent by Floodlight interface):
    1) Validate and apply the current load information;
    2) If the balancer is in active mode, start performing a single phase 3 iteration of the algorithm.
- *Alert message* signaling that the attack has started (sent by one of `Callophrys` detector programs):
    1) Turn on active mode;
    2) Force send a *timeframe* message to self.

An important property of an actor in the actor model is the fact that message handlers are never run concurrently. All the message-handling routines, other actions and receiving/sending can be done in separate threads, but the message handlers themselves are always run in an order and thus we don't need any kind of additional synchronization precautions. Thus this asynchronous implementation is very similar to the procedural one described in the previous section.

IV. Evaluation

The prototype `Callophrys` system was evaluated using the Mininet[14] network simulator and Floodlight SDN controller. The simulation used a custom Mininet script for the network topology. The test topology configuration is shown at picture 3. The attacker nodes (signed with ”A”) are generating high traffic towards the target nodes (signed with ”T”) using the `iperf` tool[15], while the SDN switches (the rest of the nodes) try to balance the load between themselves and data links. For the purpose of this experiment, the links that are always overloaded (the links coming directly to target nodes, all the links coming to and from the entry node) are simulated as having infinite bandwidth, while all the other links in the setup have a thorough simulation with limited bandwidth and latency.

The process of evaluated using Floodlight and its built-in network-tracing mechanism. The preliminary experiments using this setup show that the proposed approach does balance the network load between switches, let the traffic take alternate roots and does not overload the switches with static flows.

In this setup, the time between the start of the attack and the full balance of traffic between available roots was from 10 to 60 seconds. Total number of flow rules generated is around 13000, while every single switch is subject to no more than 3000 different rules. Most of existing SDN switches can easily handle number of rules up to hundreds of thousands. A more thorough simulation using different topologies and testing on a real physical network is a subject of further work, as well as introducing more accurate tools to measure the effect of the algorithm.

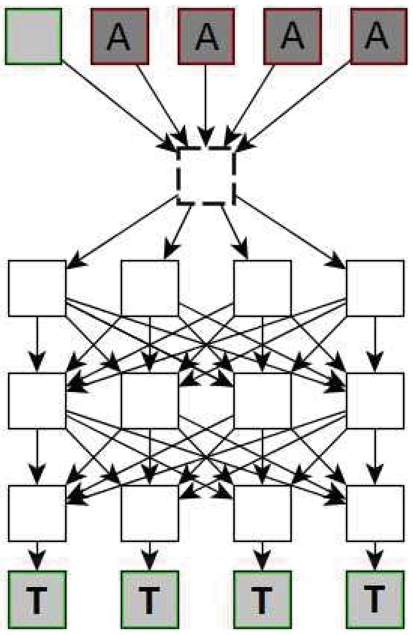


Fig. 3. The test network topology

V. Conclusion

In this work we were focused on studying of possible impacts SDN networks could bring into traditional network security related problems. As DDoS attacks remain one of the most important security problems for a single hosts and data centers as well, we decided to consider DDoS-mitigation solution for SDN networks. DDoS mitigation solutions are typically divided into two classes: active techniques that include filtering of an attacking traffic and ”survival” techniques that include increase of resources under attacks and effective load balancing.

We noticed that all existing load balancing solutions are based on a load balancing between endpoint resources, such as different servers in our case. Despite the fact that those techniques serve well for that purpose in traditional networks, SDN benefits help to increase survival time even more. Typical SDN controllers select alternative route between entry point and end point only when current route is not available. That works well and helps to remain network reliability. Nevertheless, during DDoS attack it does not solve problem at all as all attacking traffic will be forwarded to different route. There is high probability of new route to be overloaded as well. In our work we propose a solution for efficient traffic distribution between all alternative routes in a SDN network.

Experiments show that our solution helps to increase survival time of defended system during DDoS attack, thus it is effective as DDoS mitigation solution in SDN networks, but it can also be used as a general load balancing system.

References

[1] Cloud Hypermarket, *The Cloud Revolution*, http://www.cloudhypermarket.com/whatiscloud/CloudUptake.

[2] Garthner, Inc. *Gartner Says Worldwide Public Cloud Services Market to Total* $131 *Billion.*, https://www.gartner.com/newsroom/id/2352816

[3] Jasobanta Laha, Rabinarayan Satpathy , Kaustuva Dev. *Load Balancing Techniques: Major Challenges in Cloud Computing - A Systematic Review.* IJCSN International Journal of Computer Science and Network, Volume 3, Issue 1, February 2014

[4] Kaushik V. K., Sharma H. K., Gopalani D. *Load Balancing In Cloud-Computing Using High Level Fragmentation Of Dataset*

[5] P. Mohamed Shameem and R.S Shaji.*A Methodological Survey on Load Balancing Techniques in Cloud Computing.* International Journal of Engineering and Technology (IJET)

[6] Malik, S., *Dynamic Load Balancing in a Network of Workstation*, 95.515 Research Report, 19th November, 2000

[7] Rajoriya, Sheetanshu. *Load Balancing Techniques in Cloud Computing: An Overview.* International Journal of Science and Research 3.7 (2014).

[8] Randles, M., Lamb, D. and Taleb-Bendiab, A., *A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing*, 24th International Conference on Advanced Information Networking and Applications Workshops, 551-556, 2010

[9] Padhy, R. P., and Rao, P G. P., thesis entitled *Load balancing in cloud computing system*, Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Orissa, India, May, 2011

[10] Zhang, Z. and Zhang, X., *A Load Balancing Mechanism Based on Ant Colony and Complex Network Theory in Open Cloud Computing Federation*, Proceedings of 2nd International Conference on Industrial Mechatronics and Automation (ICIMA), 240-243, May, 2010

[11] Richard Wang, Dana Butnariu, and Jennifer Rexford. *OpenFlow-Based Server Load Balancing GoneWild.* Hot-ICE11 Proceedings of the 11th USENIX conference on Hot topics. Vol 12

[12] Floodlight Openflow Controller, http://floodlight.openflowhub.org

[13] Hewitt, Carl. *The Actor Model*. MASSACHUSETTS INST OF TECH CAMBRIDGE, 1993.

[14] Lantz, Bob, Brandon Heller, and Nick McKeown. *A network in a laptop: rapid prototyping for software-defined networks*. Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010.

[15] Schroder, Carla. *Measure Network Performance with* `iperf`. Enterprise Networking Planet, 2007.

Progress and Challenges in Worldwide Federation of Future Internet and Distributed Cloud Testbeds

M. Berman, M. Brinn
GENI Project Office
Raytheon BBN Technologies
Cambridge, MA, USA

Abstract—Future Internet and distributed cloud (FIDC) testbeds are rapidly becoming important research and educational resoures worldwide. While FIDC testbeds may be built on diverse technologies, they share the primary capabilities of slicing (virtualized end-to-end configurations of computing, networking, and storage resources) and deep programmability (experimenter programmability of all resources from low level hardware to virtualized components). FIDC testbeds often achieve their deep programmability through software defined networking (SDN) capabilities, which researchers employ both to construct per-application and per-experiment virtual networks, and to intelligently steer traffic throughout the virtual network/cloud environment.

Increasingly, FIDC testbed developers and researchers worldwide are working together to create federated testbed and experiment configurations. Federation holds the promise of greater scale, geographic reach, and technical diversity, while controlling the cost and effort required to create and maintain each individual testbed.

Federation is primarily a human endeavor. In the case of FIDC testbeds, the underlying agreements of trust and resource sharing are implemented technically via trusted identities, policies, and resource managers.

The past two years have seen strong progress and international cooperation in defining many of the key application programmer interfaces (APIs) that enable the technical implementation of federated testbeds, at both the control plane and data plane levels. These APIs, and the implementation of the underlying services, rely on well-understood and open technology, such as public key cryptography, attribute based access control (ABAC), and dynamic circuit networking (DCN).

These cooperative efforts have resulted in a number of exciting demonstrations and specific collaborations. There are a number of remaining challenges. Some of these are technical, (e.g., improving the semantic content of resource representation and exploring policy development and enforcement). However, the largest challenges in testbed federation are still on the human side – defining the best ways to build and share resources to meet the shared goals of the research community.

Keywords — future Internet, distributed clouds, FIDC testbeds, federated testbeds, federation policy.

I. Introduction – Why FIDC Testbeds? Why Federate?

A. Motivation

Future Internet and Distributed Cloud (FIDC) testbeds are rapidly gaining acceptance within the computer science research community. These testbeds create opportunities for experimental research and education that are difficult or impossible to conduct in individual laboratories, commercial clouds, or the public Internet. FIDC testbeds, which began with the Global Environment for Networking Innovation (GENI) project in the US [*1*] and the Future Internet Research & Experimentation (FIRE) project in the EU [*2*], are gaining acceptance. There is now a growing number of national and regional scale FIDC testbeds in use or development worldwide, as shown in **Figure 1**.

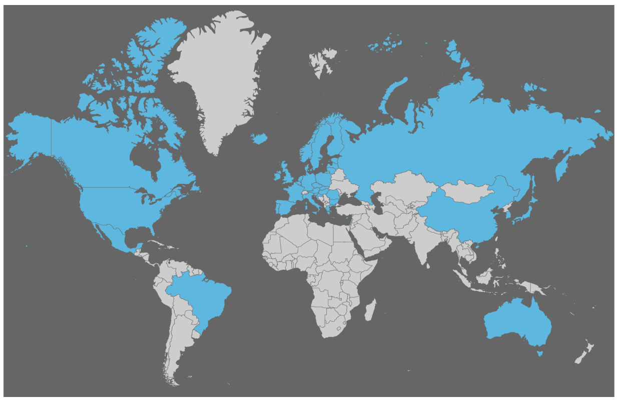


Figure 1: Worldwide FIDC testbed activity

These testbeds were originally conceived in response to researchers' concerns over Internet ossification, a term that refers to the difficulty of performing innovative research within the public Internet [*3*]. For example, novel protocols that do not accord with current Internet standards are, by specification, discarded or ignored within the public Internet. Other experiments might disrupt the normal behavior of the Internet, and are therefore not ethical. Networking researchers found themselves in a dilemma. In general, they were forced to compromise by conducting their novel experiments in simulation or in small-scale, isolated laboratory conditions.

More recently, cloud computing researchers found themselves in much the same situation. Commercial cloud providers do not typically grant researcher access to the

This research is supported by the GENI project. GENI is funded by the US National Science Foundation under cooperative agreement CNS-0737890. Any opinions, findings, conclusions or recommendations expressed in this material are the author's and do not necessarily reflect the views of the National Science Foundation.

internal workings of their data centers, and with good reason – they are in the business of providing cloud services, not supporting research. However, researchers are out in the cold if they seek to conduct experiments affecting the network topology or protocols comprising a cloud computing environment.

These researchers are increasingly choosing to conduct their experiments in FIDC testbed environments. As these testbeds grow in scale and capability, they are supporting a wide variety of research, not only in the original target communities of networking, distributed computing, and cloud computing, but also in computer science education and in data intensive domain sciences.

In the past few years, the FIDC research, testbed developer, and testbed owner communities have begun to build federations of testbeds in order to better achieve their shared goals. A number of straightforward considerations have driven a desire for testbed federation.

- In order to conduct the most realistic experiments, researchers want access to resources around the world, but testbeds are often limited to national or regional scope.
- Federation extends the reach of each community of researchers supported by each participating testbed.
- Federation preserves the unique capabilities of each participating testbed.
- Federation enables participating testbeds to enter into multiple arrangements for specific purposes. ("Federation is not monogamous.")

In fact, federation is often an underlying architectural principle of a national or regional FIDC. GENI, for instance, is built as a federation of participating resource owners, some preexisting, and others deployed during the project [*4*]. Similarly, the Fed4FIRE (Federation for FIRE) effort is federating multiple FIRE testbeds [*5*].

This paper surveys recent progress towards worldwide federation, discusses some of the key technical underpinnings of federation, and identifies some of the important challenges confronting the growing international FIDC testbed community.

B. Definitions

The following terms will be useful to clarify a discussion of FIDC testbeds and federation.

A *slice* is a group of physical and/or virtualized resources, potentially heterogeneous (e.g., computers, networks, and storage), that are reserved and connected into a single configuration on behalf of one or more slice owners. A slice differs from a bag of resources because it is constructed to preserve some degree of isolation that applies to the entire collection, rather than just its individual components. As with any type of virtualization, the quality and performance of isolation will vary according to the implementation. The intent is to give a slice owner the illusion that he or she enjoys exclusive use of the entire interconnected collection of resources.

Deep programmability is the ability of an experimenter to exert programmatic control over all (physical or virtual) resources in a configuration, not limited to computing resources or to resources at the network edge. As with any programming model, the expressive power and performance will vary according to the implementation and is likely to vary across different resource types. The goal is that the slice owner can "program everything" in his or her slice.

A *FIDC testbed* is a shared computing environment that enables future Internet and distributed cloud experiments by implementing the two key capabilities of slicing and deep programmability.

A *resource (or testbed) owner* is the person or entity that has physical and administrative control over a particular resource (or testbed) and is presumptively responsible for its misuse. The various roles (ownership, administration, and accountability) may be separated, but such distinctions are not needed for the current discussion.

A *federation* is a group of testbeds whose owners choose to share resources across their user communities, according to mutually agreed rules and limits. The intent of the federation participants is implemented by testbed hardware, software, and configuration. Federations whose participants are themselves federations are entirely possible.

II. ADMINISTRATIVE AND POLICY STEPS TO FEDERATION

Establishing a federation of testbeds is a fundamentally human endeavor. Before the technical processes get underway, the affected testbed owners and researchers should come to a clear understanding of their various goals in pursuing federation. Often the driving goal for all participants is the straightforward desire to provide more resources and broader geographic scope to researchers from the participating testbeds. However, there are several additional factors to be considered.

A. Participant approval and access policies

Perhaps the greatest benefit that a federation offers to its participants is to act as a trust broker. Resource owners trust the federation to grant access only to qualified users. Similarly, researchers trust the federation to admit only reputable and well-managed testbeds, where their work will be safe. By relying on these trust relationships, the number of agreements among m resource owners and n end users is reduced from the intractable $m \times n$ to a more reasonable $m+n$. Repeating this process to federate a group of existing federations is very much akin to investors trading derivatives – everyone's leverage is increased, but it's important that the participants understand what they are doing.

While end users clearly reap the benefit of an extended collection of available resources, the testbed participants benefit as well. Testbed owners often have a strong interest in expanding their research user base and thereby maximizing the impact of their testbeds. In addition, exposure to a broader group of end users will place greater demands on the testbed, identifying additional potential uses and identifying areas for improvement and expansion.

Potential issues arise when different participants have varying policies for trust. This concern is typically less important for the end user than for testbed participants. An end user who is concerned about the trustworthiness of a particular testbed or resource can generally design slices that simply exclude any suspect resources. Participating testbed owners can face more complex policy challenges.

In some cases, the membership policy of one federation participant may allow end user members who are not acceptable to another resource owner. For example, a testbed whose sponsor limits use to openly disseminated academic research may need to take special steps in order to enter into federation with another that encourages commercial, for-profit use. A similar challenge confronts testbeds that limit end user access by nationality. These challenges are not insurmountable. In fact, they are readily addressed by simple application of control plane policies discussed in section III.A below. However, the participant who admits the end user must collect the information needed to implement the relevant policies (e.g., "Is this researcher an academic?") and must share this data within the federation. Advance planning makes this task much simpler.

Another relatively common event is when different participating testbeds recognize different categories of end user. For example, one testbed may have Principal Investigators, while another has Project Leads. If there is a simple one-to-one correspondence, a policy to map terminology is quite simple. In other cases there is no clear correspondence. For example, one testbed often used for education may have the concept of a Teaching Assistant, who can gain access to student's resources for grading or debugging, while a more research-focused testbed may not share this concept. In such cases, the latter testbed may need to craft a custom policy to recognize the new concept within the federation, or the federation participants may choose to forgo this particular capability.

B. *Resource allocation and limits*

Some testbed resources will be subject to quotas or to special limits to ensure broad availability or to avoid accidental or intentional misuse. Typically the resources in question are either in high demand or "dangerous" in some way. The goal of restrictions on high demand resources is to avoid a "tragedy of the commons" situation, where end users who perceive little or no cost in consuming resources take unfair advantage of lenient policies. Policies for the dangerous resources are intended to prevent undesired consequences and to ensure accountability in the event of an incident.

Many of these policies are straightforward, but some can be quite sophisticated. As with membership policies, implementation is greatly facilitated if the participants are prepared to gather and disseminate the required information. Some example policies are listed below.

- "Only end users with the Administrator attribute may shut down resources."
- "Only researchers whose code has been reviewed and approved may install an OpenFlow controller in the core network."
- "Researchers from testbed X may collectively consume no more than Y% of resources from testbed Z."
- "End users who leave scarce resources idle for X hours will not be permitted to renew their reservation on these resources beyond current expiration time."

III. Control Plane Federation

Armed with a clear understanding of the intent of the resource owners and the testbeds to be federated, it falls to the testbed developers and owners to complete the implementation. A central goal is to automate the largest possible fraction of federation functionality. Thus, capturing federation policy in configuration files is optimal; software is next best; and falling back to human intervention in the form of hardware configuration or administrative procedure is least desirable.

Federating FIDC testbeds requires some level of coordination at both the control plane and data plane levels. Control plane refers to the functions associated with the creation, configuration, and management of testbed resources. Control plane functions are often implemented over the public Internet, to give experimenters ready access to their slices. Data plane refers to the experimental network resources that are allocated to slices. The data plane should generally not be implemented over the standard Internet for reasons discussed in section I.A above.

A. *Policy statements and enforcement*

Most policy should be enforced locally by each participating testbed. There are a number of benefits to this approach. Ultimately, it is the resource owner who is responsible for the correct administration of policy at each testbed. Furthermore, because a testbed may well be participating in multiple federation agreements, and serving its own local users outside of any federation, no other authority can be expected to have full knowledge of the complete set of policies to be enforced. Finally, a testbed will generally already have an existing admission and allocation procedure encoded in software, saving additional development.

A concrete example used in the following discussion is provided by the growing international federation discussed in section V below. This federation has adopted certain central concepts from the GENI architecture. For example, the local testbed software component responsible for access control and allocation is called an aggregate manager (AM). The interactions among key components involved in policy enforcement are shown in **Figure 2** and summarized below. More detailed discussions may be found in [*4*] and [*6*].

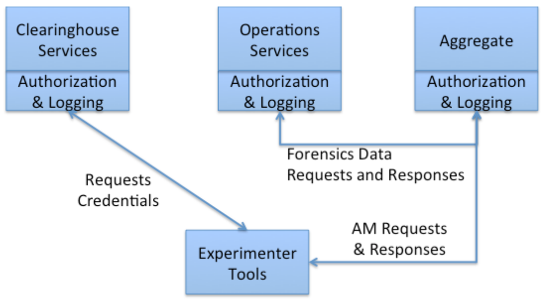


Figure 2: Key AAA Components in Federation Architecture

B. *AAA – Authentication, Authorization, and Accounting*

Three vital services required to achieve adequate trust within the control plane are authentication, authorization, and accounting, collectively known as AAA. *Authentication* is the ability to validate the identity of the person (or entity) who is making a request. *Authorization* is the process of confirming that the person requesting an action is permitted to perform that action. *Accounting* is recording and retaining enough information about each transaction to carry out the business of the federation, which may include such functions as audit, billing, and incident response. Typically, participating testbeds will already have local AAA capabilities in place when joining a federation. These existing capabilities may be sufficient for the needs of the federation, or additional federation-level processes may be needed. In the GENI federation, for example, the GMOC (GENI meta-operations center) receives and retains accounting data logged by various federation participants.

International federation efforts have worked to ensure that each of these functions is implemented using readily available open standards and software technology. This approach has the advantage of making federation accessible to all interested testbeds. Furthermore, using open technology encourages the development of a thriving community of tool developers, who can create software that facilitates end user access to federation resources. In current FIDC federation implementations, authentication is generally provided via X.509 certificates [*7*] and secure socket layer (SSL) [*8*]. Authorization generally follows one of two approaches. Role-based access control (RBAC) describes a person's rights to perform actions on a particular slice and is implemented via the slice federation architecture (SFA) [*9*]. Attribute-based access control (ABAC) enables policy enforcement based on signed statements asserting attributes of a particular person or entity [*10*]. (E.g., "Resource owner X certifies that person Y is a principal investigator.") Accounting data chiefly consists of transaction and resource status information, and may be managed by standard database software. Transaction reports are produced as a side effect of authorization actions. In the case of ABAC, which follows a theorem-proving approach to authorization, each approved transaction can be accompanied by the first-order logic proof that justifies a particular authorization decision. Various resource managers in the federation generate a variety of status information, such as utilization and up/down condition, which can be used for alerting and to inform end users of the state of different participating testbeds and their resources.

C. *Common APIs*

Two key application programmer interfaces (APIs) are used for control plane federation. An international consortium is responsible for specification of these APIs, and open source reference implementations are available. Testbed developers who wish to participate in FIDC testbed federations based on these APIs may either adopt them natively or develop translation code, such as the slice exchange point SEP software [*11*].

The federation API [*12*] is used to coordinate the so-called "clearinghouse services" of the federation. These services are conceptually centralized, but may be implemented in a distributed fashion among federation participants. The federation API defines two types of entity, a *member authority (MA)* and a *slice authority (SA)*, and the relationships between. Briefly, federation participants choose to trust a member authority to make assertions about end users. Similarly, a slice authority manages slice objects and generates credentials for members representing their authority to act on slices and their associated resources. A federation may include one or more of each type of authority.

The aggregate manager API (AM API) is the *lingua franca* that connects end users (or the software tools acting on their behalf) to the aggregate managers that allocate, configure, and manage testbed resources. The central operations supported by this API include resource discovery, resource allocation, resource management, status inquiries, and resource reclamation. Current international testbed federations are implemented using the GENI AM API, with an open source reference implementation available from [*13*]. An internationalization effort is underway; its emerging open source implementation may be found at [*14*].

IV. DATA PLANE FEDERATION

By contrast with control plane efforts, the work to federate FIDC testbeds at the data plane level seems comparatively simple, because it is not plowing nearly as much new ground. The chief challenge in data plane federation is to provide connectivity across the underlying research networks participating in the federation. A few specific capabilities are desirable to maintain and extend FIDC capabilities within the larger federation.

- Data plane connections should be carried over research and education (R&E) networks or other assets suited for FIDC applications, rather than the public Internet.
- In order to enable novel, non-IP-based research, researchers must be able to establish connectivity at layer 2.
- To maintain slice isolation, the federated data plane must preserve a network virtualization model across participating federates.
- To support deep programmability, the federated data plane should support a consistent programmability model for network resources, or at a minimum, not interfere with the network programmability model of diverse resources combined into a single slice.

Fortunately, there are a number of existing technologies that support at least the first three of these requirements. The most accessible and familiar of these is simply to provision VLAN connections to create the data plane connectivity among federated resources. The VLAN approach has several benefits, but also suffers from a few drawbacks. On the positive side, VLANs are a well-understood network virtualization model, and are essentially certain to be supported by networking hardware in each federate. Similarly, most resource owners already have expertise in segmenting their networks by VLAN, so provisioning a distinct group of VLANs on behalf of data plane federation will be relatively easy.

Current FIDC testbed federations make frequent use of VLAN-based data plane federation. In its most rudimentary form, such federation can be achieved by manually provisioning a connection between federated resources. Clearly, this method is undesirable from a scalability viewpoint, because it requires human intervention to create each federated slice. The process of *stitching* the data plane of these slices is greatly accelerated by placing a statically provisioned group of VLANs under the control of an aggregate manager, which allocates them to slices as needed and reclaims them after use. The current implementation of stitching in most of the GENI federation uses this approach. A recent data plane federation effort connecting GENI and FIRE resources is also built on this model, using a group of fifty VLANs connecting iMinds in Ghent with the Manhattan Landing (MAN LAN) exchange point in New York.

The next step evolution of stitching beyond dynamic allocation from a static pool is dynamic circuit allocation under aggregate manager control at slice creation time. One existing implementation of this approach in the GENI federation uses an aggregate manager to request dynamic virtual circuits in the Internet2 R&E network via the Internet2 ION service. A particularly promising technology for international federation using this strategy is the network service interface (NSI) connection service [*15*].

In situations where resource limitations do not permit true layer 2 connectivity via R&E network connections, it is possible to fall back to a tunneling approach. Existing implementations use generic routing encapsulation (GRE) or enhanced generic routing encapsulation (EGRE) to create a tunnel connecting federated resources. Although this tunnel is typically carried over the public Internet, it provides a degree of isolation encapsulates traffic sufficiently to permit non-IP experimentation. However, tunneling remains a less desirable solution, as it generally comes with undesired overhead and can introduce unwanted technical restrictions. Furthermore, when tunneled connections are carried over the public Internet, unpredictable performance variations are likely.

The data plane federation approaches discussed above provide connectivity solutions, but they sidestep the question of deep programmability. In many cases, adequate programmability is achieved within the data plane of the participating testbeds, and the goals of the research end user can be met simply by interconnecting these collaborating resources within a slice. However, in some slice designs, the researcher may wish to program the federation data plane resources as well. There are a number of promising approaches currently in the investigation, development, and deployment stages. One approach is to use software defined networking (SDN) technology, such as OpenFlow, to implement data plane virtualization and interconnect federated resources. In addition to avoiding some of the negative aspects of VLAN-based data plane virtualization, the approach holds out the promise to extend SDN control uniformly throughout the slice, including core network resources. GENI and Internet2 are jointly pursuing this approach, through a two-pronged strategy. The first component is an aggregate manager that provisions virtual circuits within the network's SDN-based core. The second, a "flow space firewall" multiplexes research OpenFlow controllers over the core network's flow space, enabling uniform deep programmability. For the protection of the core network, it is likely that such research controllers will require detailed review and monitoring for the foreseeable future.

Another promising line of inquiry for federation lies in the concepts of software defined exchange (SDX) and software defined infrastructure (SDI). These concepts are relatively new and their definitions, specifications, and implementations are likely to be the topics of debate for some time to come. A recent workshop [*16*] tentatively defined SDX as "a real or virtual 'meet-me' point, where [SDN-enabled] peers meet to communicate, each with its own policies." Similarly, SDI was defined as "the collection of shared [resources] plus networks and SDXs that users / applications can utilize to build end-to-end, multi domain software defined slices." While SDX and SDI are clearly unproven technology, it is clear that any capability that emerges in this area will bear directly at least on data plane federation, and quite probably on control plane federation as well.

V. International Federation – Recent Progress and Upcoming Challenges

Initial application and validation of most technical capabilities supporting FIDC testbed federation take place within national or regional scale testbeds. Progress towards international federation of FIDC testbeds really began with a series of demonstrations of *ad hoc* multi-testbed configurations assembled for specific events. Key venues for such demonstrations included GENI engineering conferences (GECs) and SC (formerly Supercomputing) conferences, beginning in 2012. Understandably, these configurations typically focused on the data plane aspects of federation, validating the ability to assemble transoceanic collections of assets into slices for high performance networking or non-IP future Internet applications. While these experiments were valuable to illustrate the scientific potential of federation, the relative weakness of control plane coordination represented a significant shortcoming. Because little automation was available to support the setup and control of these *ad hoc* configurations, they required significant person-to-person coordination. As a result, these demonstration configurations were not generally available to typical end users of the federated testbeds.

Early discussions on enduring international federation began in July 2012, with a focus on available resources and understanding the requirements for control plane and data plane federation. These plans matured into the July 2013 "breakfast club" meeting, in which participants committed to devote testbed resources and staff time to build an enduring federation based on common APIs. In the year since that meeting, dramatic progress has been made towards an initial federation, although only limited resources have been dedicated to date, and general availability of federated resources is only now emerging.

The trend towards FIDC testbed federation represents an exciting opportunity for testbed developers and the research and educational communities that they support.

However, there is still a long way to go, on both the human and technical sides of FIDC testbed federation. In many ways, the current state of activities is still very much in a Wild West (or perhaps Summer of Love) phase. Resource owners and testbed developers have relatively little experience with complex federation policies, even within at national scales. Existing implementations often make do with the unmodified default policies of individual participating testbeds. In some cases, these policies are trivial or nearly so. As membership grows beyond a tightly knit group of participants, the *status quo* is clearly not scalable and will lead to unintended consequences. Better tools for defining and enforcing federation policies are needed to remedy this situation. While it is tempting to press ahead with purely technical approaches, it is apparent that software implementations must be guided by clear understanding of the policies that are to be enforced. This guidance must come from the resource owners and testbed developers, based on their combined efforts to gain the greatest benefit from FIDC testbeds.

ACKNOWLEDGMENTS

Building federations is by its nature a collaborative activity, and any work to plan, build, and support federated FIDC testbeds relies on the efforts of many participants. Although there are too many for a comprehensive list, some of the key participants in the efforts described above include the following. The GENI architects (who have included Ilya Baldin, Nick Bastin, Andy Bavier, Mark Berman, Marshall Brinn, Jeff Chase, Chip Elliott, Aaron Helsinger, Tom Mitchell, Max Ott, Larry Peterson, Rob Ricci, and Mike Zink) insisted on designing for federation from the start. Early *ad hoc* international federation efforts were supported by collaboration and late-night efforts from many, including Akihiro Nakao, Joe Mambretti, Rick McGeer, Paul Müller, and Niky Riga. Key contributors to the common APIs, their early implementations, and dissemination include Tom Rothe, Brecht Vermeulen, and Timur Friedman. The MobilityFirst and eXpressive Internet Architecture NSF Future Internet Architecture projects served as willing test subjects for early international FIDC federations – thanks to Ray Raychaudhuri, Ivan Seskar, Kiran Nagaraja, Peter Steenkiste, Matt Mukerjee, and David Naylor. The GENI and Fed4FIRE efforts enjoy a very fruitful collaboration, supported by Piet Demeester and cheered on by EC and NSF representatives, including Per Blixt, Jorge Pereira, Suzi Iacono, and Bryan Lyles. Data plane stitching innovations are actively supported by Internet2, including efforts by Eric Boyd, Luke Fowler, and others, while Tom Lehman and Xi Yang implemented controlling software. International connections supported by the NSF international research network connections (IRNC) program were invaluable in building connections – thanks to David Lassner, Julio Ibarra, Jim Williams, and others already mentioned. The initial July 2012 GEC planning group included Tomonori Aoyama, Dave Farber, Serge Fdida, Martin Swany, MyungKi Shin, Mauro Campanella, James Sterbenz, Vasilis Maglaris, Cees de Laat, Cesar Marcondes, and others already mentioned. Larry Landweber convened the July 2013 “breakfast club” meeting, which, in addition to several already listed, included Michael Stanton, Luis Lopez, Jack Hong, Iara Machado, Al Leon-Garcia, Thierry Rakotoarivelo, Carlos Casasus, and Hagen Woesner. KC Wang and Felix Wu, as well as many others, some already mentioned, participated in international outreach.

REFERENCES

[1] Mark Berman et al., "GENI: A Federated Testbed for Innovative Network Experiments," *Computer Networks*, no. 61, pp. 5-23, March 2014.

[2] Serge Fdida et al., "FIRE Roadmap Report 1 – Part II," Future Internet Research and Experimentation (FIRE), 2011.

[3] J.S. Turner and D.E. Taylor, "Diversifying the Internet," in *Global Telecommunications Conference*, vol. 2, 2005.

[4] GENI Architecture Team (Marshall Brinn and Robert Ricci, chairs). (2012, March) GENI Federation Software Architecture Document. [Online]. http://groups.geni.net/geni/attachment/wiki/GeniArchitectTeam/GENI%20Software%20Architecture%20v1.0.pdf

[5] Wim Vandenberghe et al., "Architecture for the Heterogeneous Federation of Future Internet Experimentation Facilities," in *Future Network and Mobile Summit 2013 Conference Proceedings*, 2013.

[6] Marshall Brinn. (2014, June) 20th GENI Engineering Conference. [Online]. http://groups.geni.net/geni/attachment/wiki/GEC20Agenda/IntroToArch/GEC20%20Architecture%20Review%20final.pptx

[7] D. Cooper et al. (2008, May) RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. [Online]. http://tools.ietf.org/html/rfc5280

[8] A. Freier, P. Karlton, and P. Kocher. (2011, August) RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0. [Online]. http://tools.ietf.org/html/rfc6101

[9] Larry Peterson et al. (2009, April) Slice-Based Facility Architecture. [Online]. http://svn.planet-lab.org/attachment/wiki/WikiStart/sfa.pdf

[10] ABAC Development Team. ABAC. [Online]. http://abac.deterlab.net/

[11] Michiaki Hayashi, Toshiaki Tarui, Yasushi Kanada, Shu Yamamoto, and Akihiro Nakao, "Inter-domain Virtualization with Slice Exchange Point," in *10th International Conference on IP + Optican Network*, Tokyo, 2014.

[12] Marshall Brinn et al. (2013, November) GENI Wiki. [Online]. http://groups.geni.net/geni/wiki/CommonFederationAPIv1

[13] GENI Project Office. (2014) Getting GCF and Omni Source Code. [Online]. http://trac.gpolab.bbn.com/gcf/wiki/GettingGcf

[14] Federation AM API Team. (2014) Federation AM API. [Online]. https://github.com/open-multinet/federation-am-api

[15] G. Roberts et al. (2014, June) NSI Connection Service v2.0. [Online]. http://www.ogf.org/documents/GFD.212.pdf

[16] Lawrence Landweber and Jeannie Albrecht. (2014, June) Workshop on Prototyping and Deploying Experimental Software Defined Exchanges (SDXs). [Online]. http://groups.geni.net/geni/wiki/SDXandSDIWorkshop

Network Verification: Calculus and Solvers

N. Bjørner
Microsoft Research, Microsoft Corporation
Redmond WA, USA
nbjorner@microsoft.com

K. Jayaraman
Microsoft Azure, Microsoft Corporation
Redmond WA, USA
karjay@microsoft.com

***Abstract*—We examine calculus and solvers for *Network Verification*. As starting point we take the SecGuru tool that checks network access restrictions in the Microsoft Azure public cloud infrastructure. The tool is based on the Satisfiability Modulo Theories solver Z3. SecGuru is also used for checking *Network Invariants* for data-centers that are deployed using Azure's network architecture. In both cases SecGuru relies on a calculus of network configurations in order to capture intent and check these statically. SecGuru models network configurations using quantifier-free logical formulas over bit-vectors. We recall also other scenarios in the context of network verification. They use other fragments of logic and specialized engines for Datalog and quantifier reasoning in Z3. In each case, correctness assertions can be modeled and solved using logics that are supported in state-of-the art theorem provers.**

Based on our experiences we claim that Network Verification is an important and exciting new opportunity where formal methods and modern theorem proving technologies play an important role. Many formalisms that make it convenient to model scenarios from networking domain are already supported in modern solvers. On the other hand, networking provides an inspiration for additional formalisms that can be supported using new efficient data-structures and solving algorithms.

I. Calculus and Solvers

A. *Network Verification Calculus: Routers, Access Control Lists and Protocols*

Modern data-centers use routers from several vendors, such as Cisco and Juniper networks. They expose different interfaces for configuration and newer routers also ease programmability for open-stack style controller-based software defined networking. The configuration formats are on one hand very low level: the language resembles a bare bones assembly format. On the other hand, configurations are quite expressive, including Access Control Lists (ACLs), Quality of Service contracts, and monitoring directives. Furthermore, in large networks, configurations are distributed among several routers and management devices. The behavior of a full system is the effect of aligning many configurations. Configurations are of course only a means to an end: ACLs exist to enforce security policies and routing policies exist to implement a routing architecture. The task of bridging the actual configurations with the original intent is *inhumane*: the complexity of large scale deployments does not lend itself to manual inspection, even for masters of complexity [1].

We provide selected use cases where important features of modern industrial network systems can be modeled using logical theories capturing the main intent of operators.

Microsoft Research
nbjorner@microsoft.com
Microsoft Azure
karjay@microsoft.com

[1] Thanks to Nick McKeown for this fitting characterization

B. *Solvers for Network Verification*

The use of Satisfiability Modulo Theories, SMT, solvers for software analysis, verification and testing has blossomed in recent years thanks to significant advances in theorem proving technologies coupled with availability of usable SMT tools that match closely the domains useful in software analysis. The SMT solver Z3 [5] is the most widely used SMT solver with applications ranging from symbolic execution and test-case generation [6], program verification [10], symbolic model checking [2] and many other areas.

Network Verification is not unlike software or hardware so it is possible to apply some of the tools developed with other applications in mind for software defined networks. We here provide instances where SMT solvers can be of significant benefit for managing modern software defined networks. We also claim that the networking domain presents its own features that inspire new efficient constraint solving algorithms and data-structures.

II. Access Control Lists

```
remark Isolating private addresses
deny ip 10.0.0.0/8 any
deny ip 172.16.0.0/12 any
deny ip 192.0.2.0/24 any
...
remark Anti spoofing ACLs
deny ip 128.30.0.0/15 any
deny ip 171.64.0.0/15 any
...
remark permits for IPs without
       port and protocol blocks
permit ip any 171.64.64.0/20
....
remark standard port and protocol
       blocks
deny   tcp any any eq 445
deny   udp any any eq 445
deny   tcp any any eq 593
deny   udp any any eq 593
...
deny   53 any any
deny   55 any any
...
remark permits for IPs with
       port and protocol blocks
permit ip any 128.30.0.0/15
permit ip any 171.64.0.0/15
...
```

Fig. 1. An Edge Network ACL configuration

The Azure architecture enforces network access restrictions using ACLs. These are placed on multiple routers and firewalls in data-centers and on the edge between internal networks and the internet. Miss-configurations, such as miss-configured ACLs, are a dominant source of network outages. The SecGuru [8] tool uses Z3 to check contracts on firewall ACLs. It translates the ACLs into a logical predicate over packet headers that are represented as bit-vectors. These predicates are checked for containment and equivalence with contracts that are represented as other bit-vector formulas. SecGuru checks virtually all Microsoft routers on a continuous basis: each router is checked every 30 minutes against a data-base of contracts.

The routers that are dedicated to connect internal networks to the Internet backbone are called Edge routers and they enforce restrictions using ACLs. Figure 1 provides a canonical example of an Edge ACL. The ACL in this example is authored in the Cisco IOS language. It is basically a set of rules that filter IP packets. They inspect header information of the packets and the rules determine whether the packets may pass through the device.

Each rule of a policy contains a packet filter, and typically comprises two portions, namely a traffic expression and an action. The traffic expression specifies a range of source and destination IP addresses, ports, and a protocol specifier. The expression 10.0.0.0/8 specifies an address range 10.0.0.0 to 10.255.255.255. That is, the first 8 bits are fixed and the remaining 24 (= 32-8) are varying. A wild card is indicated by *Any*. For ports, *Any* encodes the range from 0 to $2^{16} - 1$. The action is either *Permit* or *Deny*. They indicate whether packets matching the range should be allowed through the firewall. This language has the first-applicable rule semantics, where the device processes an incoming packet per the first rule that matches its description. If no rules match, then the incoming packet is denied by default.

The meaning of network ACLs can be captured in logic as a predicate ACL over variables *src*, a source address and port, *dst*, a destination address and port, and other parameters, such as protocol and TCP flags. For our example from Figure 1, we can capture the meaning as the predicate:

$$\begin{array}{l} ACL \equiv \\ \quad \textbf{if } src = 10.0.0.0/8 \ \wedge proto = 6 \textbf{ then } \mathit{false} \textbf{ else} \\ \quad \textbf{if } src = 172.16.0.0/12 \ \wedge proto = 6 \textbf{ then } \mathit{false} \textbf{ else} \\ \quad \textbf{if } src = 192.0.2.0/24 \ \wedge proto = 6 \textbf{ then } \mathit{false} \textbf{ else} \\ \quad \ldots \\ \quad \textbf{if } dst = 171.64.64.0/20 \wedge \ proto = 6 \textbf{ then } \mathit{true} \textbf{ else} \\ \quad \ldots \\ \quad \textbf{if } proto = 4 \wedge dstport = 445 \textbf{ then } \mathit{false} \textbf{ else} \\ \quad \ldots \end{array}$$

For ease of readability, we re-use the notation for writing address ranges. In bit-vector logic we would write the constraint $src = 10.0.0.0/8$ as $src[31:24] = 10$, e.g., a predicate that specifies the 8 most significant bits should be equal to the numeral 10 (the bit-vector 00000110).

Traffic is permitted by an ACL if the predicate ACL is true. Traffic permitted by one ACL and denied by another is given by $ACL_1 \oplus ACL_2$ (the exclusive or of ACL_1 and ACL_2). The SecGuru tool uses the encoding of ACLs into bit-vector logic and poses differential queries between ACLs to find differences between configurations. It also checks contracts of ACLs by posing queries of the form $ACL \Rightarrow Property$, where an example property is that UDP ports to DNS servers are allowed. The main technological novelty in SecGuru is an enumeration algorithm for compactly representing these differences. Compact representation of differences help network operators understand the full effect of a miss-configuration. Checking firewall configurations is central to securing networks. Several other tools address checking firewall configurations. These include Margrave [15], which provides a convenient formalism for expressing rich properties of networks and firewalls (but counter-examples are only available for one address at a time), and the firewall testing tool in [4], which builds upon Isabelle/HOL and Z3 for generating test-cases.

III. Routing tables

Figure 2 shows an excerpt of a routing table from an Arista network switch

```
B E    0.0.0.0/0 [200/0] via 100.91.176.0, n1
                         via 100.91.176.2, n2

B E    10.91.114.0/25 [200/0] via 100.91.176.125, n3
                              via 100.91.176.127, n4
                              via 100.91.176.129, n5
                              via 100.91.176.131, n6
B E    10.91.114.128/25 [200/0] via 100.91.176.125, n3
                                via 100.91.176.131, n6
                                via 100.91.176.133, n7
...
```

Fig. 2. A BGP routing table

Similarly to ACLs we can model routing tables as relations *Router* over destination addresses and next-hop ports that can be represented as atomic Boolean predicates. Each rule in the routing table is either provisioned based on static configurations specified in the device, or derived based on BGP network announcements that the device receives.

We here choose an encoding of *Router*, such that for each destination address dst and next-hop address n:

$$\begin{array}{ll} & Router[dst \mapsto \mathsf{dst}, \mathsf{n} \mapsto \mathit{true}] \text{ is true} \\ \text{iff} & \\ & \mathsf{n} \text{ is a possible next hop for address } \mathsf{dst} \end{array}$$

The routing tables have an ordered interpretation, wherein rules whose destination prefixes are the longest applies first. The default rule with mask 0.0.0.0/0, listed first, applies if no other rule applies. For our example, our chosen encoding of the predicate *Router* is of the form:

$$\begin{array}{l} Router \equiv \\ \quad \textbf{if } \ldots \\ \quad \textbf{if } dst = 10.91.114.128/25 \textbf{ then } n_3 \vee n_6 \vee n_7 \textbf{ else} \\ \quad \textbf{if } dst = 10.91.114.0/25 \textbf{ then } n_3 \vee n_4 \vee n_5 \vee n_6 \textbf{ else} \\ \quad n_1 \vee n_2 \end{array}$$

Each Azure data-center is built up around a hierarchy of routers that facilitate high-bandwidth traffic in and out as well as within the data-center. Traffic that leaves and enters the data-center traverses four layers of routers, while traffic within the data-center may traverse only one, two or at most three layers depending on whether the traffic is within a rack, a physical partition called a *cluster*, or between clusters. Routers close to the host machines belong to one of the clusters. Traffic in a correctly configured data-center is routed without loops and along the shortest path for cluster-local traffic. Azure checks these properties as *network invariants*. Sample (slightly simplified from the ones checked for Azure) network invariants are:

Network Invariant 1: Traffic from a host leaf directed to a different cluster from the leaf is forwarded to a router in a layer above. In other words, suppose that *Router* belongs to a cluster given as a predicate *Cluster*, and that *RouterAbove* is the set of routers above *Router*, then

$$dst \notin Cluster \land Router \Rightarrow \bigvee_{n \in RouterAbove} n$$

On the other hand,

Network Invariant 2: Traffic from a host leaf directed to the same cluster is directed to the local VLAN or a router in the layer above that belongs to the same cluster as the host leaf router:

$$\begin{array}{l} dst \in Cluster \land Router \quad \Rightarrow \\ \quad VLAN \ \lor \ \bigvee_{n \in RouterAbove} (n \land n \in Cluster) \end{array}$$

The routing behavior of routers at the same level from the same cluster should also act uniformly for addresses within the cluster (they can behave differently for addresses outside of a cluster range).

Network Invariant 3: Let $Router_1$, $Router_2$ be two routers at the same layer within the cluster *Cluster*, then

$$dst \in Cluster \Rightarrow Router_1 \equiv Router_2$$

IV. Differential Network Reachability

In the previous section we described how SecGuru performs local checks on routers. These local checks often imply global properties of the network. This approach works fine in the context of the Azure architecture, which is fixed and data-centers are deployed in cookie-cutter form. Finding local invariants, however, is an entirely manual process and the approach does not generalize to arbitrary networks (though there is a really good point to capturing and checking architecture based invariants for Azure). The behavior of a router is commonly a combination of ACLs, forwarding rules, and packet rewriting. It is therefore not generally possible to check global network invariants from a fixed set of local network invariants. To check global network properties we developed a specialized tool in Z3 that handles configurations for packet switching networks efficiently.

This time we represent forwarding logic and networks as a set of constrained *Datalog* rules. Suppose that n_r is a predicate representing the current router from our example, and $n_1, n_2, \ldots$ are the names of next-hop routers, represented as predicates, then the rules for representing the routing behavior can be written:

$$\forall dst \ . \ n_1(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \land & dst \neq 10.91.114.0/25 \\ \land & dst \neq 10.91.114.128/25 \land \ldots \end{array} \right)$$

$$\forall dst \ . \ n_2(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \land & dst \neq 10.91.114.0/25 \\ \land & dst \neq 10.91.114.128/25 \land \ldots \end{array} \right)$$

$$\forall dst \ . \ n_3(dst) \leftarrow \left(\begin{array}{ll} & n_r(dst) \\ \land & \left(\begin{array}{l} dst = 10.91.114.0/25 \lor \\ dst = 10.91.114.128/25 \end{array} \right) \\ \land & \ldots \end{array} \right)$$

$$\ldots$$

Constrained Datalog with stratified negation provides logical expressitivity that makes it easy to encode queries over pairs of paths. Thus, one can use Datalog to query for packets that are dropped along one route but not another.

Header Space Algebra (HSA) [9] was introduced to reason efficiently about reachability over sets of headers. The basic data-structure used by HSA is a difference of cubes (DOC) representation of three-valued bit-vectors. Three-valued bit-vectors encode address masks compactly using don't cares. An example DOC is the expression:

$$1 * 110 * * \setminus (*1 * * * 11 \cup *0 * * * 00)$$

It is shorthand for the set

$$\{1011011, 1011001, 1011010, 1111000, 1111001, 1111010\}.$$

In [11] we adapt DOC encodings as an underlying table representations for a Datalog engine in Z3. For a set of benchmarks extracted from Azure and Stanford networks we observed that the DOC representation scales well beyond competing representations, such as BDDs, or SAT based bounded model checking. Model checking techniques for (software defined) networks is actively investigated in several contexts, including the Anteater tool [12] and in [18].

V. Programmable Controllers

Network controller programs operate at their core by receiving packets from routers. The packets are rewritten, forwarded and used to update both local state and routing tables. In [1] we developed a language, VeriCon, capturing core features of network controllers relevant to verification of network controllers. State, local and external routing tables, are uniformly represented as predicates (Boolean arrays). Proving invariants of the controllers turns out to requite a limited expressive logical power close in style to the Bernays-Schönfinkel-Ramsey, otherwise known as Effectively Propositional Reasoning (EPR). EPR formulas are of the form: $\forall \vec{y} \ . \ \varphi[\vec{c}, \vec{y}]$, where $\vec{c}$ is a set of constant symbols, and the formula φ is quantifier-free over equalities and uninterpreted relations over the constants and bound variables. Thus, EPR formulas do not have nested functions.

The VeriCon verification conditions are discharged automatically by Z3, or in case of properties that are not invariants, Z3 provides counter-examples. Furthermore, invariants that were not already inductive are in some cases inductive after conjoining the invariants with their weakest pre-conditions. Weakest pre-condition strengthening is a folklore approach used in variations in deductive-algorithmic model checking. While it is simple to implement it does not scale very well and current efforts include replacing the strengthening by more sophisticated approaches and also ensuring that the assertion language remains within a decidable extension of EPR.

VI. The Logical Power of Networks

A common experience so far has been that network verification is matched well by logics and solvers that exploit how ACLs, forwarding rules and controller programs handle sets of packets the same way: Transitions are guarded by predicates on bit-ranges and state updates copy or update bit-ranges to constant values. In other words, the tools exploit and support packet ranges and how the state of controllers is updated based on a few enumerable attributes. Yet, the underlying algorithms from our experiences are orthogonal. The bit-vector solver used in SecGuru reduces verification to propositional SAT; DNA pairing requires a Datalog engine; controller verification uses invariants expressed over quantified first-order logic so it requires efficient quantifier instantiation. The Z3 SMT solver exposes much richer functionality than the fragments we used here: Z3 supports reasoning about logical formulas using linear integer, linear and non-linear real arithmetic, algebraic data-types and arrays. It contains specialized engines for solving Horn clauses over arithmetic [3], [7], [13] that so far target applications from symbolic software model checking.

We believe the mutual exposure of formal methods to modern packet switched network engineering is a significant area of opportunity for both camps. An indication that this is broadly the case is that we are not the only ones who apply SMT, SAT, QBF, finite state model checking and other verification and synthesis technologies for programmable packet switched networks [14], [17], [19], [16]. More narrowly, the use of SMT solving and other theorem proving technologies for Network Verification offers mutual opportunities to improve scale and reliability of modern (large scale) data-center networks. On the other hand, the applications that emerge from Network Verification inspire new algorithms and data-structures for theorem proving and model checking.

Acknowledgment Our experiences with network verification is based on joint work with several collaborators, including: George Varghese, Mooly Sagiv, Charlie Kaufman, Geoff Outhred, Nuno Lopes, Mingchen Zhao, Jeff Jensen, Monika Machado, Garvit Juniwal, Ratul Mahajan, Ari Fogel, Jim Larus, Thomas Ball, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Michael Schapira and Asaf Valadarsky.

References

[1] Thomas Ball, Nikolaj Bjørner, Aaron Gember, Shachar Itzhaky, Aleksandr Karbyshev, Mooly Sagiv, Michael Schapira, and Asaf Valadarsky. VeriCon: towards verifying controller programs in software-defined networks. In Michael F. P. O'Boyle and Keshav Pingali, editors, *PLDI*, page 31. ACM, 2014.

[2] Thomas Ball, Vladimir Levin, and Sriram K. Rajamani. A decade of software model checking with SLAM. *Commun. ACM*, 54(7):68–76, 2011.

[3] Nikolaj Bjørner, Kenneth L. McMillan, and Andrey Rybalchenko. Program Verification as Satisfiability Modulo Theories. In Pascal Fontaine and Amit Goel, editors, *SMT@IJCAR*, volume 20 of *EPiC Series*, pages 3–11. EasyChair, 2012.

[4] Achim D. Brucker, Lukas Brügger, and Burkhart Wolff. hol-TestGen/fw - An Environment for Specification-Based Firewall Conformance Testing. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *ICTAC*, volume 8049 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 2013.

[5] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[6] P. Godefroid, J. de Halleux, A. V. Nori, S. K. Rajamani, W. Schulte, N. Tillmann, and M. Y. Levin. Automating Software Testing Using Program Analysis. *IEEE Software*, 25(5):30–37, 2008.

[7] Krystof Hoder and Nikolaj Bjørner. Generalized Property Directed Reachability. In Alessandro Cimatti and Roberto Sebastiani, editors, *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2012.

[8] Karthick Jayaraman, Nikolaj Bjørner, Geoff Outhred, and Charlie Kaufman. Automated Analysis and Debugging of Network Connectivity Policies. Technical Report MSR-TR-2014-102, Microsoft Research, July 2014.

[9] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: static checking for networks. In *NSDI*, 2012.

[10] K. Rustan M. Leino. Developing verified programs with dafny. In David Notkin, Betty H. C. Cheng, and Klaus Pohl, editors, *ICSE*, pages 1488–1490. IEEE / ACM, 2013.

[11] Nuno Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. DNA Pairing: Using Differential Network Analysis to find Reachability Bugs. Technical Report MSR-TR-2014-58, Microsoft Research, 2014.

[12] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the Data Plane with Anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, New York, NY, USA, 2011. ACM.

[13] Kenneth L. McMillan. Lazy Annotation Revisited. In Armin Biere and Roderick Bloem, editors, *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2014.

[14] Sanjai Narain, Gary Levin, Sharad Malik, and Vikram Kaul. Declarative Infrastructure Configuration Synthesis and Debugging. *J. Netw. Syst. Manage.*, 16(3):235–258, September 2008.

[15] Timothy Nelson, Christopher Barratt, Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. The Margrave tool for firewall analysis. In *LISA*, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[16] Andrew Noyes, Todd Warszawski, Pavol Cerný, and Nate Foster. Toward synthesis of network updates. In Bernd Finkbeiner and Armando Solar-Lezama, editors, *SYNT*, volume 142 of *EPTCS*, pages 8–23, 2014.

[17] Shuyuan Zhang, Abdulrahman Mahmoud, Sharad Malik, and Sanjai Narain. Verification and synthesis of firewalls using SAT and QBF. In *ICNP*, pages 1–6. IEEE, 2012.

[18] Shuyuan Zhang and Sharad Malik. SAT Based Verification of Network Data Planes. In Dang Van Hung and Mizuhito Ogawa, editors, *ATVA*, volume 8172 of *Lecture Notes in Computer Science*, pages 496–505. Springer, 2013.

[19] Shuyuan Zhang, Sharad Malik, and Rick McGeer. Verification of computer switching networks: An overview. In *ATVA*, 2012.

On QoS Management in SDN by Multipath Routing

E. Chemeritskiy
Lomonosov Moscow State University
Moscow, Russia
tyz@lvk.cs.msu.su

R. Smelansky
Applied Research Center for Computer Networks
Moscow, Russia
smel@arccn.ru

***Abstract*—The Quality of Service (QoS) management is one of the urgent problems in networking which doesn't have an acceptable solution yet. In the paper the approach to this problem based on multipath routing protocol in SDN is considered. The proposed approach is compared with other QoS management methods. A structural and operation schemes for its practical implementation is proposed.**

Keywords—Quality of Service; Multipath Routing; Software-Defined Networks; Network Management

I. Introduction

QoS (Quality of Service) as a term is a general description of the performance of a network connection. This term is treated either as qualitative assessment of the connection performance by a user, or as a set of objective quantitative parameters characterizing the one. Qualitative evaluation of QoS is defined as the degree of satisfaction of a user by communication quality as for example in Skype – the sound quality, the presence of a distortion, the appearance of echo, jitter, quality of the picture etc. There are two basic methods for QoS qualitative evaluation: Mean Opinion Score and Quality of Experience [1]. These methods provide an integrated assessment of all subjective assessment of service.

In this paper we are primarily interested in the second interpretation of the term QoS as a set of the parameters a network connection. Under term QoS requirements we will mean a set of the QoS parameters a network connection has to meet. The term QoS management we will treat as ability of network to maintain a set of connection parameters compliant with the QoS requirements of the application it is due to. Saying "connection" we mean end-to-end (e2e) connection. A set of QoS parameters includes:

- Throughput – a part of the channel bandwidth available to the particular connection;
- End-to-end delay – time is needed to deliver a packet from one source host to a destination host;
- Jitter – a deviation of the end-to-end delay from its mean value;
- Error Rates - the share of packets lost or damaged during a transmission through connection.

Different parameters of QoS play a different role for different applications. For example, multimedia application requires high throughput, videoconferencing and real time simulation – small jitter and end-to-end delay, telemedicine (distance surgery) – high throughput and low error rate.

Providing a connection with an appropriate QoS require a certain network resources. However, the network has only a limited amount of the resources to handle data flows. Thus we get a problem how to allocate network resources to meet QoS requirements of different applications operate at the same time? In practice usually there is problem connected to the previous one - what level of utilization (efficiency) of the network resources under allocation have been made? Thus, a network has to be selective while spreading bandwidths of its channels and capacities of its switching devices over the applications. Thereby, the solution for the quality of service problem we are looking for should meet the following criteria: (1) ensure compliance of granted e2e connections with the QoS requirements of applications, (2) provide a small resource fragmentation, and (3) to be a practical method delivering a suboptimal resource allocation.

Although QoS issue has been addressed since the first attempts to transmit voice over a packet switched network [2], and the community has developed a set of diverse approaches to conquer it, none of them is successful enough to be implemented by default. They are either too expensive to deploy or provide insufficient increase to the admissible utilization of a network. Thereby, the existing practices of the network management advice to obtain the missing resources by a straightforward resource extension, rather than to invest into an intricate piece of hardware, gain better control over the resource distribution and attune the performance in an intelligent way.

In this paper we propose a new approach to QoS management in SDN networks [3] based on Multi Path Routing (MPR) called MPRSDN with the following features:

- MPRSDN refuses resource reservation in favor of their efficient utilization. Thereby, it provides no strict guarantees and implements a best effort approach.
- Although we propose to construct a QoS-compliant resource allocation with a heuristic search, our approach uses a considerably large search space to allocate the resources for each of the requested connections. Thus, if it fails to meet the requirements of a given application, most likely, there are no more suitable resources left.
- It does not require specialized hardware and may be deployed in any SDN network with an appropriate

This research is supported by the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003 and Russian Foundation for Basic Research, project 14-07-00625

control over the switches. The hosts have to be preinstalled with the software agent for multipath routing enabling to involve some idling resources.

In section II we provide the comparative analysis of existing approaches to QoS management. Section III introduces the structural and operational schemes of the proposed QoS control toolset.

II. Related Work

A. Conventional QoS management

There are multiple well-known approaches to the quality of service management. Introduced by the model of Integrated Services (IntServ) [4], signaling protocol RSVP (and later NSLP [5]) provides applications with guarantees over throughput and delay of the granted connection by resource reservation at each router along the flow path calculated by a routing protocol. The reservation restricts schedule of packet handling at each affected router because the allocated resources are assigned to the flow exclusively and cannot be used even if the flow does not fully utilize them at that time. An application has to announce its QoS requirements before the connection setup and cannot modify them until the connection close. Thus, the application is forced to over pledge and reserve resources with a margin for the maximum traffic burst.

IntServ relies on static resource reservation and brakes work-conserving operation of switching devices. This results into an unnecessary resource fragmentation, similar to the one in a computer with paged allocation of RAM. As a result, in some cases network fails to supply the connection with the requested QoS even if accumulative amount of the network resources is enough to make it. The similar problem may be also caused by the independence of the signaling and routing protocols. There might be a bypass route to avoid the overloaded network component, however reservation is separated from routing and cannot take this advantage.

The model of Differentiated Services (DiffServ) [6] proposes to replace an awkward resource scheduling for end-to-end connections with predefined qualities by a local flows grading at the network devices. Each device defines a set of service classes and attributes each class with a certain QoS. Although each flow has a right to request a class with an appropriate service, the model does not provide any guarantees over the provided packet processing quality. Instead, each switch undertakes to share its resources among the flows of different classes in accordance with their relative shares. If there are no flows for a certain class of service then the resources of this class are allocated among the other classes. Thereby, switches are work-conserving and never idle when there are some packets to process. Although the application may specify required class of service for its packets explicitly, it is optional. In practice switching devices often calculate the class of service for a packet automatically by a certain set of its attributes and a mapping preinstalled by the administrator.

Differentiated Services introduce a way to deal with switch-level resource fragmentation and increase the overall network performance. However, it manages only the network resources along the primary route of an application. Thus, some idling and suitable resources away from this route are unavailable. Moreover, the class of service of the flow is set statically for the whole path. Although it is possible to improve granularity by dynamic changing of class of service at some points in the network this interference into the switching logic is beyond the capabilities of the networks of ordinary switching devices without a centralized control.

QoS-routing [7] was intended to improve allocation of network resources by constructing individual data transmission paths for each connection. Such a fine-grained routing is used to balance data flows among several paths, bypass congestion involve idling resources aside from heavy loaded channels, and take into account the QoS requirements of the application. For example, the delay sensitive traffic is usually routed along the shortest path, whereas the other flows may be forced to use the longer paths. However, a practical implementation of this method requires a low-level and centralized control over the switching devices unavailable back in time of its emergence. Moreover, QoS-routing algorithms tried to treat the problem of resource allocation as a global optimization problem with multiple constraints and their implementations were too slow to run on the fly.

B. QoS management with SDN

SDN supplies a complete control over the packet handling rules of each switch in the network, and an SDN controller may easily implement each of the mentioned approaches to QoS management without a regard to a complex distributed exchange algorithms for service data. Controller can mimic resource reservation by dynamic adjustment of traffic shaping parameters at its border switches of the network. It is also capable to collect a comprehensive set of the QoS metrics and implement a relevant QoS-aware routing on a per-flow basis, or improve capabilities of DiffServ with dynamic reassigning the class of service mark for any flow at any point of the network. Unfortunately, neither flexibility, nor convenience of SDN removes the inherent disadvantages of these methods.

SDN provides a technical capability to gather the relevant information about the network, but it is a hard task to construct a comprehensive algorithm to dispose the collected data properly. This algorithm is expected to analyze a set of heterogeneous parameters and synthesize such a set of appropriate forwarding instructions for the switches to achieve a better network performance. It is hardly believable there are real opportunities to construct routing algorithm able to work on the fly [8].

SDN does not give us any advantage to cope the problem of how to transmit QoS requirements from the user application to the Control Plane. However, this problem has been realized. FLARE [9] proposes to enable such an interaction by appending of arbitrary data to the tail of a packet and introducing corresponding handlers for the piggy-backed data at both end-host and switches. PANE [10] considers direct communication of the end-host application and the controller. On the other hand, loosening of the separation between the Data Plane and the Control Plane leads to potential security breach, and there is a lot of skepticism about its overall advantage.

Another reason for controller to avoid interference in applications communication is Internet Architecture Principles [11, 12]. As an evolutionary development of the network architecture SDN should not violate these principles. End to End principle states “The network’s job is to transmit datagrams as efficiently and flexibly as possible. Everything else should be done at the fringes…” [11]. Clark explained this principle with the following words “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)” [13].

C. Multi-Path Routing

An SDN controller has a number of options to provide an application with a connection of an appropriate QoS: controller can route the flow through the underused links, reallocate the resources along the existing routes and/or impose stronger restrictions to the other flows. However, it requires too complicated algorithm to manage all the listed possibilities simultaneously. MPRSDN proposes to decompose this global resource management problem into a set of smaller problems with help of Multi Path Routing.

MPRSDN associate each connection with a simple module to detect violations of its QoS requirements and request the controller to supply additional resources on their occurrences. The controller module handles the requests by constructing of additional data transmission paths through the network. The set of paths granted to a connection is used to balance its packets and gain a larger amount of the resources. If controller provides connection with a path, it has not used before, there is a good chance, this path improves accumulated QoS of the connection.

There are multiple well-known approaches to implement the described splitting and balancing of a packet flow among a set of alternative paths. Routers often use Equal Cost Multi Path (ECMP) [14] to route the traffic addressed to the same destination along the different paths with equal cost. ECMP is simple to implement by distributing of the incoming packets with round-robin. However, such a naive approach to balancing results into packet reordering, the most of TCP congestion-avoidance algorithms treat as a packet loss. As a result, the size of congestion window decreases, and the original non-split connection may even outperform the balanced one. Thereby, practical balancer implementations send all the packets of a single connection along the same route. So, they are often unable to split the “elephant” flows and overcome the problem of fragmentation at the data channels.

In contrast to ECMP, Multi Path (MP) TCP [15] follows the End to End principle and proposes to split a single TCP session into smaller virtual sessions at the end hosts. MP TCP operates transparently for an application. Upon the setting up of the connection, it creates a static set of internal sockets. Each of these sockets is used to establish an individual connection trough the network. MP TCP balances the packets among this set of connections and uses an original congestion-avoidance algorithm to cope inter-connection packet reordering without a significant performance drop.

Although MP TCP implements an automatic adjustment for the packet ordering, it does not provide any means to ensure the allocated internal connection use different paths. Existing implementations of MP TCP send the information about the original connection the packet within an optional L4 field the most of network devices unable to distinguish. Thereby, flows of the same application are most likely to take the same path. This fact cancels all the advantages of a multipath routing, until the sender or/and receiver has multiple interfaces connected to different networks.

Fortunately, flexibility of SDN networks can surmount the disadvantages of MP TCP. Controller may easily detect a new connection is setting up by intercepting its first packet; get any of its attributes including the data stored inside of the payload; find out the original application connection it belongs to, and minimize intersection of its route with the other flows of the same connection.

III. Quality of service in Multi Path SDN

The paper refers a middleware designed to split a single Application Flow (AF) into a set of Sub Flows (SF) and multiplex these SFs into a single AF as a Multi Flow Agent (MPA). For a given AF, we will call the AF degree a number of SFs, carrying its data.

Each SF establishes a connection between a pair of unique L4 addresses: one at the source and one at the destination host. Network switches are supposed to distinguish different SFs by their headers and treat each of them as an ordinary and independent flow. In particular, each SF may attribute its packets with a higher TOS/DSCP mark and get a better service as compared to the other SFs of the same AF.

Although MP TCP agent may be considered as an example of MPA, we imply the latter to be a more general term. Different MPA implementation may go over TCP and provide the similar multi path transmission to other protocols, modify the number and intensity of SFs dynamically without the need to reestablish the parent AF, rate-limit or shape individual SFs with some arbitrary algorithms, and interact with an SDN controller explicitly or implicitly.

To design an efficient implementation of the MPRSDN one should answer on the following questions:

- How to retrieve the QoS requirements for an application?
- How to monitor and properly estimate the quality of the granted connections?
- How to keep connection properties compliant with the QoS requirements of applications by MPA?
- How should MPA and SDN controller interact?

A. Deriving QoS requirements

MPRSDN does not use the greedy approach. It requests extra resources dynamically and only when it founds that there is a risk to violate the QoS requirements. Thus, it allows application to release the sparse part of the previously acquired resources and request the missing resources without reestablishing of the connection. For example, a network video-streaming application may loosen its requirements to the connection, while playing static scenes, and increase them at the moments of active motions.

Thereby, there is an issue, how to retrieve the initial QoS requirements of the application and how to modify them during the MPA operation? There are two options to resolve this problem: (1) make application to specify its QoS requirements through a socket-level API, or (2) derive these requirements from some application profile.

Using of the socket-level API results into a considerable complication of network programming for the application developer. Although this kind of effort may result into a reasonable benefit for applications with severe dependency on the connection QoS, in many cases this functionality will be considered as unnecessary and obscuring.

Transparent deriving of the application requirements does not imply any extra effort by the developers, and has more perspectives to be generally accepted. However, the only connection characteristic that can be estimated transparently is its intensity. This kind of data may be sufficient to derive the required bandwidth, but it does not allow estimate the other QoS characteristics such as a transmission delay.

B. *Monitoring of a connection QoS*

SDN controller has comprehensive possibilities to monitor QoS of an e2e connection. There are some researches devoted to constructing and maintenance of a traffic matrix formed by an enumeration of bandwidths consumed by each of the end-host applications [16] and measurement of one-way delay for an arbitrary flow while it moves through the network infrastructure [17]. However, a comprehensive fine-grained measurement imposes a frequent polling of the devices and results into excessive loading of both network devices and the controller. There are some attempts to reduce intensity of the controller requests to the devices by using the dead reckoning estimation [18]. The idea is to use a simple network model to approximate parameters of interest between the measurements and reduce their total number. However, the simulation of a network with an appropriate accuracy often results into even higher requirements to computation power of the controller.

As a result, controller has to delegate part of its monitoring functions to MPAs. However, monitoring at hosts becomes rather challenging, especially in case of a UDP-like half-duplex connections. UDP sender does not know the amount of packets dropped and both the connected hosts are unaware of an actual network delay value. In practice, this problem is usually moderated by wrapping the raw application data into RTP protocol [19]. It establishes an additional RTCP connection to send periodic statistics backwards from destination to source, and reduces the case of half-duplex connections to the simpler full-duplex one. TCP-like connection allows the hosts to detect bandwidth shortage by the amount of the lost packets and infer a one way delay of the connection from the RTT provided by the underlying congestion avoidance algorithm.

C. *QoS management with MPRSDN*

MPRSDN provides two ways to meet QoS requirements: adjustment of the number of SFs in the AF and individual regulation of their service classes. Upon QoS violation MPA scales AF partitioning and/or steps up the service for some of its SFs. Upon detecting excessive overprovisioning MPA rollbacks the parameters to avoid unnecessary overhead and simplify the AF maintenance.

The listed QoS management means are independent of each other, and may be applied in any order. However, one sequence may be superior in the first set of cases, while the other is more efficient in another set. Thus, it makes sense to develop a set of strategies to regulate the properties of some SFs and adjust their number for different types of requirement violations in a most efficient way. A set of appropriate MPA heuristics may include the following examples:

- When accumulated bandwidth of the SFs subsides, some network channel is likely to become congested. In this case rise in classes of service for the SFs with the lower throughput is usually less efficient than increase in the number of the SFs.
- If the estimated AF delay exceeds the allowed upper limit, MPA should accelerate the slowest of its SFs. One way to accomplish this task is to give up using this SF and reallocate its data among the others.
- If the violation is due to a change in the requirements of an application, there are no reasons to increase the degree of AF partitioning. Thereby, MPA should cover the lack of resources by rising of QoS requirements for some of the existing SFs in the first place, and consider increasing of SF number to be an auxiliary leverage.

D. *Communication between an MPA and and SDN controller*

SDN provides two different ways to install forwarding rules into the network devices: the proactive and the reactive one. The former one implies an SDN controller foresees the need in some paths through the network and sets up appropriate rules in advance. Any packets that match these rules are transmitted by the devices autonomously without further involvement of the controller. Thus, it is unable to track the establishment of new connections directly. The reactive approach implies the border network devices request packet processing instructions from the SDN controller upon receiving a packet without a match among the existing rules.

In order to support multipath routing an SDN controller should identify individual SFs of a single AF and provide them with different paths. This requires the controller to react MPA in dynamic. Thus, the controller either has to provide MPAs with ability to connect it directly through a dedicated channel, or operate in the reactive mode. Since the former one implies mixing of Data and Control planes and requires a fundamental change of the interaction between the host and the network, we give preference to a more practical second option.

While requesting controller for instructions to process a packet of an unknown flow, switching device either provide controller with a set of preprocessed headers, or supplement these headers with the original packet body.

MPAs at the sender and the receiver hosts interact to each other through a certain set of L4 header options. These are used to initiate a new multipath connection, preserve correct relative ordering among the packets sent through the different SFs of a single AF, synchronize opening and closing of certain SFs, etc. Commodity switching devices cannot parse optional headers at a suitable speed. Therefore they are able to identify new data flows, and new SF in particular, but are unable to simplify matching against the existing AFs. Thus, to lower the threshold for the deployment the controller has to request the switches to send a full body of the packet and extract the multipath options from the packet by its own.

After detection and identification, the controller should check validity of the new flow with regards to a certain set of policies. In this paper we restrict the term policy to a scope of QoS management and consider the following examples of the enforced restrictions:

- AF may split into at most 10 SFs simultaneously;
- AF may request at most 5 connections within a second;
- SFs of a certain AF cannot request priority service.
- Accumulated bandwidth of the AF must not exceed 10 Mbps (this kind of restrictions implies monitoring).

Next, the controller should generate an appropriate path to route the SF through the network and take into account the dependencies among the SFs of a single AF.

The path should avoid the points of congestion. Otherwise, the new extra flow will not bring much gain, but subtract some resources from the other flows, who would probably try to take their resources back with their own extra flows. Thereby, the MPAs will compete to each other and request the controller to grant them more and more SFs. This kind of racing reveals no new resources but complicates packet processing and occupies the links with unnecessary headers. Thereby, the controller should banish appending of extra SFs, if there no appropriate path to set it up.

Next, the controller has to minimize the number of links traversed by several SFs of a single AF. If an arbitrary subset of SF has similar paths, the congestion at any of its components is likely to affect both SFs, and the AF multiplexing does not increase its accumulated QoS.

Note the controller should route as flows as SFs without regards to violation of the route restriction to preserve network availability under a heavy load.

Taking into account these remarks, the routing library of the controller may calculate paths using the following logic:

1. Identify the congested links using network monitoring and temporarily exclude corresponding edges from the topology graph.
2. If the flow is not a subsidiary one, route it with some Shortest Path algorithm (such as the Dejkstra one). If there is no appropriate path, route the flow using the original topology graph;
3. Otherwise, generate a set of alternative paths using one of K Shortest Path algorithms (such as [20]) and choose a path with high QoS and minimal intersection with other SFs of the same AF.

If the new SF violates some multipath routing policies or the controller fails to construct an appropriate path to route it, the packets of this SF should be dropped. This behavior of the controller prevents MPA to increase the degree of partitioning for some AF, and the AF will likely violate QoS requirements of some application. However, the requested flow was unable to give more resources to the AF.

IV. Conclusion

MPRSDN method is a novel approach to manage QoS of the connections in SDN networks based on multipath routing. The primary focus of our approach is to meet the QoS requirements of network application. However, it does not coincide with the aims of the IntServ model. The latter considers QoS requirement as a dominant, and does not take much account to the capabilities of the network. As well as the DiffServ model, we tolerate QoS violations in favor of network efficiency. However, we do not rely on convenience of local resource reallocation at the switching hardware. Multipath routing allows our approach to increase the search space for the idling resources dramatically and to result into their better allocation. Although our approach is fully compatible with DiffServ model and they may supplement each other, it can also work independently.

We have also proposed a possible scheme to implement the idea of QoS management with multipath routing in practice. Although the scheme provides conventional network services to any hosts, only the ones with the preinstalled multipath agent are capable to use all its advantages. Note the agent does not provide any interface to manage the host externally, and does not inject any additional security breaches. As for the network infrastructure, our approach does not impose any requirements to the hardware. The only modification of the Control Plane we need is the specialized routing application. Although controller interacts with agents at the hosts and reallocates resources in response to their request, this kind of communication does not break separation between the Control and Data plane.

References

[1] R. Serral-Gracià, E. Cerqueira, M. Curado, M. Yannuzzi, E. Monteiro, X. Masip-Bruin “An Overview of Quality of Experience Measurement Challenges for Video Applications in IP Networks” Proceeding of the 8th international conference on Wired/Wireless Internet Communications (WWIC'10), pp. 252-263, Luleå, Sweden, 2010.

[2] Cohen, D. "Specifications for the Network Voice Protocol (NVP)," IETF Network Working Group, Request for Comments 741, November 1977.

[3] Open Networking Foundation “Software-Defined Networking: The New Norm for Networks” April 2012.

[4] R. Braden, D. Clark, S. Shenker. “Integrated Services in the Internet Architecture: an Overview” IETF Network Working Group, Request for Comments: 1633, June 1994.

[5] Xiaoming Fu, Schulzrinne, H., Bader, A., Hogrefe, D., Kappler, C., Karagiannis, G., Tschofenig, H., Van den Bosch, S. “NSIS: a new extensible IP signaling protocol suite” IEEE Communications Magazine, Vol. 43, Issue 10, 2005.

[6] Kalevi Kilkki “Differentiated Services for the Internet” Macmillan Technical Publishing, Indianapolis, IN, USA, June 1999.

[7] P. Van Mieghem, F.A. Kuipers “On the complexity of QoS routing” Computer Communications, Volume 26 Issue 4, March, 2003, pp. 376–387.

[8] Garroppo, R. G., Giordano, S., Tavanti, L. “A survey on multi-constrained optimal path computation: Exact and approximate algorithms” Computer Networks, 54(17), 2010, 3081–3107.

[9] Akihiro Nakao, “Deeply Programmable Network Through Network Virtualization,” In The 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), August 2012

[10] Ferguson, A., Guha, A., Liang, C., Fonseca, R., Krishnamurthi, S. “Participatory Networking: An API for Application Control of SDN” Proceedings of the ACM SIGCOMM 2013 (SIGCOMM'13), pp. 327-338, Hong-Kong, August 2013.

[11] Architectural Principles of the Internet. RFC 1958. http://www.ietf.org/rfc/rfc1958.txt

[12] David D. Clark, "The Design Philosophy of the DARPA Internet Protocols", Computer Communications Review 18:4, August 1988, pp. 106–114

[13] Saltzer, Reed, and Clark, End-to-end Arguments in System Design, 1984

[14] Thaler, D., Hopps, C. “Multipath Issues in Unicast and Multicast Next-Hop Selection” IETF Network Working Group, Request for Comments 2991, November 2000.

[15] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M. “Improving datacenter performance and robustness with multipath TCP” Proceedings of the ACM SIGCOMM 2011 (SIGCOMM '11), Toronto, Ontario, Canada, 2011, 266-277.

[16] [12] Tootoonchian, A., Ghobadi, M., Ganjali, Y. “OpenTM: traffic matrix estimator for OpenFlow networks” Proceedings of the 11th international conference on Passive and active measurement (PAM’11), 2010, pp. 201-210.

[17] Phemius K., Bouet M. “Monitoring latency with OpenFlow” Proceedings of the 9th International Conference on Network and Service Management (CNSM) and its three collocated workshops, 2013, pp. 122-125.

[18] Ciucu, F., Schmitt, J. “Perspectives on network calculus: no free lunch, but still good value” Proceedings of the ACM SIGCOMM 2012 (SIGCOMM’12), pp. Helsinki, Finland, 2012, pp. 311-322.

[19] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson “RTP: A Transport Protocol for Real-Time Applications” IETF Network Working Group, Request for Comments: 3550, July 2003.

[20] Aaron Bernstein: “A Nearly Optimal Algorithm for Approximating Replacement Paths and k Shortest Simple Paths in General Graphs” Proceeding of the Symposium on Discrete Algorithms (SODA), Austin, Texas, USA, 2010, pp. 742-755.

Consistent network update without tagging

E. Chemeritskiy
Applied Research Center for Computer Networks
Moscow, Russia
tyz@lvk.cs.msu.su

V. Zakharov
Lomonosov Moscow State University
Moscow, Russia
zakh@cs.msu.su

***Abstract*—Designing of network update algorithms is urgent for development of SDN control software. A particular case of Network Update Problem is that of adding a set of forwarding rules into flow-tables of SDN switches (say, to install new paths in the network) or restoring seamlessly a given network configuration after some packet forwarding rules have been disabled (say, at the expiry of their time-outs). Some algorithms provide solutions to these problems but only with the help of tagging techniques. But is it possible to perform a consistent network update without tagging? We study this problem in the framework of a formal model of SDN, develop correct and safe network updating algorithms, and show that in some cases it is impossible to update consistently network configurations without resorting either to tagging or to priorities of packet forwarding rules.**

Keywords—software defined network, network update, forwarding rule, updating command, packet forwarding relation, invariant, post-condition

I. Introduction

Almost as soon as the concept of SDN emerged, a number of projects on the development of languages and tools for SDN programming have been launched: Frenetic [1], Maestro [2], Procera [3], Nettle [4]. Although focusing on different aspects of network management, they exploit the same principal idea of multi-level programming: replace a low-level imperative interfaces of OpenFlow commands and messages [5] with a high-level abstractions for querying network state, defining, updating and combining policies in a consistent way. In these languages the application programs for SDN controllers could be designed in highly abstract terms, like "Maintain the current configuration until an event E occurs", or "As soon as a rate of the flow F exceeds the bound c change seamlessly the route A to B", etc. It is highly desirable to make these constructs modular; then simple policies can be formed in isolation and later composed with other modules to create more sophisticated switch-level and net-level forwarding policies. At the next stage these high-level constructs are translated into appropriate sequences of low-level instructions that are more adapted to OpenFlow protocol; they form some kind of objective code. Finally, a composition of such low-level instructions is transformed into a sequence of OpenFlow commands to be delivered to SDN switches.

Certainly, a great deal of problems have to be solved to implement this approach to SDN programming. Some of them are usual tasks of system programming; in choosing appropriate means to meet the challenge one would probably more suffer from embarras de choix than from desolation. But there are many problems that are totally specific for network control. Some of them arose in the framework of SDN paradigm and, therefore, have not been considered until recently. The study of these network control problems is crucial for the development of SDN programming, since algorithms and techniques for their solution constitute the routine library of high-level network programming systems.

One of such problems innate to SDN control is that of correct and safe modification of network configurations — Network Update Problem. The list of cases when network update is necessary includes flow-table optimization, maintenance of configurations at shutting down switches and links, or the expiry of forwarding rules' time-out, end host migration, load balancing, changing access control lists, traffic monitoring.

In the most general setting Network Update Problem (NUP) is as follows: given a network configuration C – a network topology coupled with an assignment of flow-tables to switches, – and a pair of network specifications – an invariant Φ and a post-condition Ψ – compute a sequence α of flow-table updating commands such that by applying α to C we finally obtain a configuration C' which satisfies Ψ while every intermediate configuration satisfies Φ. In the early papers [6], [7], [8], [9] researchers considered only some special cases of this problem for specific protocols and operational practices to prevent transient anomalies such as forwarding loops, black-holes, disconnection, etc. A far more systematic study of NUP was launched in [10] and continued in [11], [12], [13], [14]. In these papers the authors studied the problem of consistent global SDN update, namely, how to transform a given configuration C into another given configuration C' in such a way that every packet traversing the network is forwarded either by the rules of C or by the rules C', but not by their mixture.

In [12] it was shown that consistent global update can be achieved with the help of a three-phase commitment algorithm which operates with meta-data — tags or labels attached to every data packet. It is assumed that every stable network configuration C is associated with some version number (tag) L_C. When a packet arrives at some ingress port of the net it is stamped with this tag (e.g., it may be stored in a VLAN field). Tag L_C is included in the patterns of all forwarding rules and the switches in configuration C process only those packets that have a set version number. To alter from configuration C to a new network configuration C' the update algorithm first installs the forwarding rules of new configurations C' guarded by the next version number $L_{C'}$ in the middle of the network. At completing the first phase the algorithm enables the new configurations C' by installing rules at the perimeter of the network that stamp packets with the next version number

$L_{C'}$. Finally, the the out-of-date rules of configuration C are removed. In [13], [14] the authors considered the optimization issues of implementation of their algorithms.

Network update problem has been studied in some other papers. An algorithm offered in [11] redirects traffic through the controller, introducing substantial overhead and limiting packet-processing throughput. For the same purpose the authors of [15] suggested to use scheduled execution time to coordinate network updating. Some preliminary studies of other variants of NUP have been carried out in [16] (synthesis of network configurations) and in [17], [18] (flow-table optimization).

Although network update algorithm presented in [12] gives a universal solution to some variants of NUP, its safe and correct implementation requires additional resources — extra fields in packet headers. Therefore, it is also important to study the cases of NUP that can be solved by means of some tag-free techniques. In this paper we consider the case of NUP when a new network configuration C' is obtained from an initial configuration C by adding a finite set of packet forwarding rules into flow tables of some switches. Or, in other terms, this variant of NUP may be viewed as the problem of restoring a SDN configuration after some packet forwarding rules have been disabled. To the extent our knowledge, this problem has not been considered yet in a formal setting. Our contribution to the study of Network Recovering Problem (NRP) is twofold. First, following [12] as the example, we introduce an enhanced abstract model of SDN which is both compatible with OpenFlow protocol [5] and suitable for setting up formally all those variants of NUP that have been studied in [10], [11], [12], [13], [14], [16], [17], [18]. Second, in the framework of the said model of SDN we define formally NRP, give solutions to some of its basic cases, and show that even this simple variant of NUP is not quite trivial.

II. Network Model

To build the formal model of SDN we take OpenFlow protocol [5] as a standard and abstract from the internal structure of packets, forwarding rules and reconfiguration commands: packet headers and ports are considered as atomic entities, match sections and action sections of forwarding rules are specified by predicates on packet states. Unlike [12], we define network semantics not in terms of packet-processing transitions for individual packets in a network (per-packet abstraction) but in terms of packet forwarding relations induced by packet forwarding rules in the flow-tables of SDN switches (per-flow abstraction). The similar approach is used in some papers on network verification (see [19], [20]).

Denote by H the set of all packet headers, by W the set of all switches in a SDN, and by P the set of data flow ports of a switch (all switches are assumed to be of the same type). The pairs from $V = P \times W$ are called *network points*, the pairs from $L = H \times P$ are called *local packet states*, and the triples from $S = H \times P \times W$ are called *packet states*. In every switch two special ports $Drop$ and $Contr$ are distinguished. Packets queued to $Drop$ port ($Contr$ port) have to be dropped (sent to the controller). We denote by L_0 the set $H \times (P \cup \{Drop, Contr\}$.

Network topology is defined by a *topology relation* $T \subseteq V \times V$ such that $(v', v'') \in T$ iff there is a point-to-point link between v' and v''. A point v is an *egress point* if it is not linked to any point in the network. Networks communicate with the outside world (environment) via egress points. We denote by E_T the set of all *egress packet states* $\langle h, p, w\rangle$, where $\langle p, w\rangle$ is an egress point of the network.

A *forwarding rule* is a triple $r = (G_r, A_r, m_r)$, where $G_r \subseteq L$, $A_r \subseteq L \times L_0$, and m_r is a positive integer. A predicate G_r (*guard*) is an abstraction of a match section of the rule, a binary predicate A_r (*action*) is an abstraction of an action section of the rule, and m_r is a priority of the rule. The effect of r is specified by a relation $F_r \subseteq L \times L_0$ such that $(\ell, \ell_0) \in F_r \iff \ell \in G_r \wedge (\ell, \ell_0) \in A_r$.

A flow-table tab of a switch w is a finite set of forwarding rules $\{r_1, r_2, \ldots, r_N\}$. The semantics of tab is specified by a packet forwarding relation R_{tab} as follows. Let k be the highest priority of the rules from tab. For every i, $1 \le i \le k$, denote by tab^i the set of rules from tab which have priority i: $tab^i = \{(G, A, i) \ : \ (G, A, i) \in tab\}$. Then define recursively (from k down to 1) the pairs of predicates R^i_{tab} and B^i_{tab} as follows:

$$R^k_{tab} = \bigcup_{r \in tab^k} F_r, \ B^k_{tab} = \bigcup_{r \in tab^k} G_r;$$

$$R^i_{tab} = \{(\ell, \ell_0) \ : \ \exists\, r\ (r \in tab^i \wedge \ell \notin B^{i+1}_{tab} \wedge (\ell, \ell_0) \in F_r\}),$$
$$B^i_{tab} = B^{i+1}_{tab} \cup \bigcup_{r \in tab^i} G_r.$$

Assuming that missed packets are sent by default to the controller, we introduce also the predicate

$$R^0_{tab} = \{(\langle h, p\rangle, \langle h, Contr\rangle) \ : \ \langle h, p\rangle \notin B^1_{tab}\}$$

and define $R_{tab} = \bigcup_{i=0}^{k} R^i_{tab}$ which means that every packet arrived at some port of the switch w is either processed by the rule of the highest priority that matches the local state of the packet, or sent to the controller.

SDN flow-tables must be unambiguous: no packets match two rules of the same priority in the same table, i.e. for every pair of rules $r_1 = (G_1, A_1, m_1)$ and $r_2 = (G_2, A_2, m_2)$ if $m_1 = m_2$ then $G_1 \cap G_2 = \emptyset$. Denote by Tab the set of all possible unambiguous flow-tables.

A *network configuration* C on the set of switches W is a pair (T, I), where T is a topology relation on V, and $I : W \to Tab$ is a *table assignment function* which maps a flow-table $tab_w = I(w)$ to every switch w. Given a network configuration $C = (T, I)$ we define a *1-hop packet forwarding relation* R_C on the set of (global) packet states S as follows: $(\langle h, p, w\rangle, \langle h', p', w'\rangle) \in R_C$ if one of the following requirements holds

1) there exists such a port p'' that $(\langle h, p\rangle, \langle h', p''\rangle) \in R_{I(w)}$ and $(\langle p'', w\rangle, \langle p', w'\rangle) \in T$ (packet transmission to the next switch);
2) $w' = w$, $(\langle h, p\rangle, \langle h', p'\rangle) \in R_{I(w)}$ and p' is an egress port (packet transmission outside the network);
3) $w' = w$, $(\langle h, p\rangle, \langle h', p'\rangle) \in R_{I(w)}$ and $p' = Drop$ (packet drop);

4) $w' = w$, $(\langle h,p\rangle, \langle h',p'\rangle) \in R_{I(w)}$ and $p' = Contr$ (PacketIn message to the controller).

The relation R_C provides the semantics of a network configuration C and completely specifies the behavior of packets in a network. Nevertheless, sometimes only the connectivity between egress ports via data paths may be of particular importance. A sequence of packet states $path = s_0, s_1, \dots, s_i, s_{i+1}, \dots$ is called a *path* in a network configuration C iff $s_0 \in E_T$, and $(s_i, s_{i+1}) \in R_C$ holds for every i, $i \geq 0$. If a path ends with a packet state $s = \langle h,p,w\rangle$ such that either $s \in E_T$, or $p \in \{Drop, Contr\}$, then it is called a *complete path*. Denote the set of all complete paths in a network configuration C (that begin with an egress packet state s) by $Path(C)$ (respectively, $Path(C,s)$).

Network configurations alter at the expiry of forwarding rules' time-outs, at the shutting down or failure of links, ports, or switches, and by the commands received from the controller. OpenFlow protocol [5] includes reconfiguration commands of the following types:

- $add(w,r)$ to install a forwarding rule r in the flow-table of a switch w;
- $del(w, G_0, m)$ to remove rules from the flow-table of a switch w: a rule $r = (G, A, m)$ is uninstalled iff its guard G is subsumed by a predicate G_0.

We denote by $com(C)$ the result of application of a reconfiguration command com to a configuration C. Given a sequence of commands $\alpha = com_1, \dots, com_k$ we define $\alpha(C) = com_k(\dots, com_1(C)\dots)$.

Since SDN is a completely asynchronous distributed system, any pair of reconfiguration commands (even though addressed to the same switch) can be executed in an arbitrary order. OpenFlow protocol provides some synchronization means to regulate partially the order of command execution. Without going into details of such means we will assume that every finite set of reconfiguration commands Com (in what follows we call it a *reconfiguration batch*) is supplied with a partial order $\prec$: if $com' \prec com''$ then at every run of the batch Com the command com'' is executed after the completion of com'. Given two reconfiguration batches $(Com_1, \prec_1)$ and $(Com_2, \prec_2)$ we write $(Com_1, \prec_1); (Com_2, \prec_2)$ to denote their sequential composition which is a batch $(Com_1 \cup Com_2, \prec)$ such that any pair of commands com', com'' is ordered as $com' \prec com''$ iff either both com' and com'' are in the same set Com_i, $i = 1, 2$, and $com' \prec_i com''$, or $com' \in Com_1$ and $com'' \in Com_2$.

III. Network Recovery Problem

There are many ways to specify formally packet forwarding policies. Since a pair (S, R_C) may be viewed as a Labeled Transition System, one may specify a desirable network behavior by using Temporal Logics, μ-calculus, or some fragments of the first-order logic (see [10], [12], [16], [19], [20]). Alternatively, a language of extended regular expressions introduced in [21] may be used to specify packet forwarding policies in terms of sets of paths in admissible network configurations.

In the most general case Network Update Problem (NUP) can be set up as follows. Let Φ (invariant) and Ψ (post-condition) be some formal specifications of packet forwarding policies, and C be a network configuration. Then NUP is the problem of computing a reconfiguration batch $(Com, \prec)$ such that for every linearization α of the partially ordered set $(Com, \prec)$ the following requirements hold: 1) $\alpha(C) \models \Psi$, and 2) $\beta(C) \models \Phi$ for every prefix β of α.

Network Recovery Problem (NRP) is a particular case of NUP. A network configuration C' may spontaneously turn into a configuration C upon removing some forwarding rules from flow-tables, say, due to the expiry of their time-outs, erroneous execution of reconfiguration commands, switch faultiness, etc. Let C' be an arbitrary configuration, and forwarding rules $r_1, r_2, \dots, r_n$ are in the flow-tables of switches $w_1, w_2, \dots, w_n$ respectively. Suppose that time-outs of all these rules expired, the switches uninstalled them and notified the controller about these events. Thus, the configuration C' degraded to configuration C. The aim of the controller is to restore seamlessly C' from C. Here the "seamlessness" requirement means that in the course of network recovery any outside observer (an end host or the controller) detects only minimal necessary changes in the behaviour of the network. Namely, only those complete data paths that are in $Path(C)$ but not in $Path(C')$ dissolves, and only those paths that are in $Path(C')$ but not in $Path(C)$ arise. In fact, a seamless network recovery assumes that a reconfiguration batch should operate by the principle "push-and-forget": after being pushed to the SDN control flow channel it neither cause network to transmit any unexpected PacketIn message to the controller, nor generate any redundant data routes in the network. In more precise terms this problem may be regarded as a variant of NUP, where post-condition $\Psi(X)$ is $X = C'$, and invariant $\Phi(X)$ is specified by the conjunctive formula

$$\begin{aligned} &Path(C) \cap Path(C') \subseteq Path(X) \wedge \\ &Path(X) \subseteq Path(C) \cup Path(C') \end{aligned}$$

When configurations C and C' are fixed we denote this variant of NUP as (C, C')-NRP.

We begin with the consideration of the case when all rules in C and C' have the same priority. It should be noticed that sometimes it is impossible to restore seamlessly the configuration C' just by installing the lost rules in some appropriate order in the corresponding flow-tables. Consider a network configuration C' depicted on Fig. 1. Here $H = \{g, h\}$, and the rules r_1 and r_2 serve the packets of both types g and h. Notations $r_1 : g, h$ and $r_3 : g$ mean that a rule r_1 is applicable to the packets of both types g and h, whereas a rule r_3 is applicable only to the packets of the type g. Clearly, there are 4 complete paths in C':
$p'_1 = (h, v_{11}), (h, v_{31}), (h, v_{21}), (h, v_{41}), (h, v_{43})$;
$p'_2 = (g, v_{11}), (g, v_{31}), (g, v_{32})$;
$p'_3 = (g, v_{22}), (g, v_{41}), (g, v_{12}), (g, v_{31}), (g, v_{32})$;
$p'_4 = (h, v_{22}), (h, v_{41}), (h, v_{43})$.
Suppose that the rules r_1 and r_2 disappeared and C' degraded to a configuration C. Since no rules serve the packets of the types h and g in the switches w_1 and w_2, these packets are sent to the controller by default. Thus, there are 4 complete paths in C:
$p_1 = (h, v_{11}), (h, Contr)$; $p_2 = (g, v_{11}), (g, Contr)$;
$p_3 = (h, v_{22}), (h, Contr)$; $p_4 = (g, v_{22}), (g, Contr)$.
But an attempt to restore C' by adding first the rule r_1 to the flow-table of w_1 (see Fig 2.) brings a complete path $p_0 = (h, v_{11}), (h, v_{31}), (h, v_{21}), (h, Contr)$ which is neither

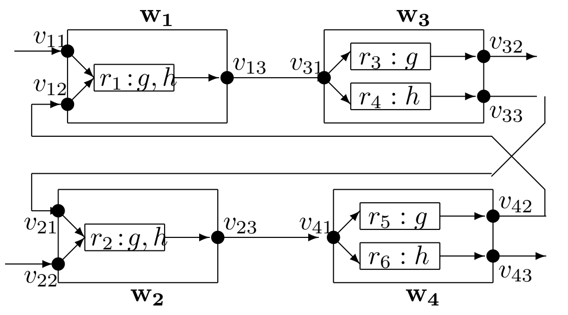


Fig. 1.

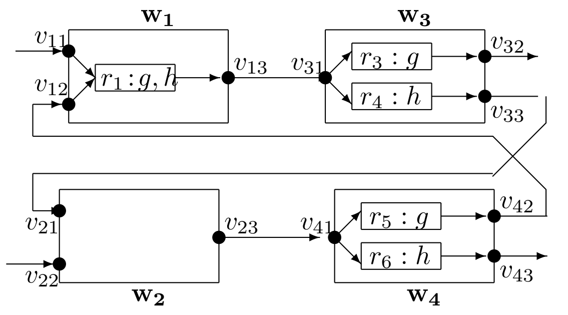


Fig. 2.

in $Path(C)$, nor in $Path(C')$. A similar situation holds when one tries to install first the rule r_2.

This effect is due to the mutual dependency between the rules r_1 and r_2: each of them is able to process packets that have been forwarded by the other rule. To restore a configuration seamlessly we need to break the loops in the dependence of one rule upon another in a given configuration. To this end we introduce auxiliary predicates on the set L of packet local states.

Let Z be a arbitrary network configuration and $r' = (G', A', m')$, $r'' = (G'', A'', m'')$ be a pair of rules from the flow-tables of switches w' and w'' respectively. Denote by R_Z^+ the transitive closure of 1-hop packet forwarding relation R_Z. Then a *dependency predicate* $\theta_{r'r''}$ on L is specified by the formula

$$\begin{aligned}\theta_{r'r''}(x) = \;& \exists s \in S, \ell \in L_0\ [E_T(s) \wedge R_Z^+(s, \langle x, w'\rangle) \wedge \\ & G'(x) \wedge R_Z^+(\langle x, w'\rangle, \langle \ell, w''\rangle) \wedge G''(\ell)]\ .\end{aligned}$$

which, intuitively, holds iff there exists a complete path $s \xrightarrow{r_0} s_1 \rightarrow \cdots s_i \xrightarrow{r'} s_{i+1} \rightarrow \cdots s_j \xrightarrow{r''} s_{j+1} \rightarrow \cdots$ such that the rule r' processes packets at the state $s_i = \langle x, w'\rangle$, and the rule r'' processes the same packets but *after* the rule r'. Dependency predicates give rise to a binary relation $\sqsubseteq_Z$ on the set of forwarding rules in the flow-tables of configuration Z: a rule r'' *depends* on a rule r' ($r'' \sqsubseteq_Z r'$ in symbols) iff $\theta_{r'r''} \not\equiv false$. The relation $\sqsubseteq_Z$ in its turn induces a binary relation $\prec_Z$ on the set of reconfiguration commands: $add(w'', r'') \prec_Z add(w', r')$ iff a forwarding rule r'' in the flow-table of a switch w'' depends on a rule r' in the flow-table of a switch w' in a network configuration Z.

The most simple case of (C, C')-NRP is that of partially ordered dependence on the set of removed rules.

Theorem 1. *Suppose that a network configuration C' turns into a configuration C as some forwarding rules $r_1, r_2, \ldots, r_n$ have been disabled in the flow-tables of switches $w_1, w_2, \ldots, w_n$. Suppose also that $\sqsubseteq_{C'}$ is a partial order on the set of rules $\mathcal{E} = \{r_1, r_2, \ldots, r_n\}$. Then a reconfiguration batch $(Com, \prec_{C'})$, where Com is the set of commands $add(w_i, r_i) : 1 \le i \le n$, provides a solution to (C, C')-NRP.*

If a dependency relation $\sqsubseteq_{C'}$ is not a partial order on the set of removed rules then priority mechanism becomes crucial for the seamless restoration of C'.

Theorem 2. *Suppose that network configurations C and C' are the same as in Theorem 1 and $\sqsubseteq_{C'}$ is not a partial order on the set of removed rules $\mathcal{E}$, i.e. $r' \sqsubseteq_{C'} r''$ and $r'' \sqsubseteq_{C'} r'$ hold for some pair of rules r', r'' in $\mathcal{E}$. Then any reconfiguration batch $(Com, \prec)$ composed of reconfiguration commands that operate with forwarding rules of the same priority is not a solution to (C, C')-NRP.*

To resolve (C, C')-NRP for an arbitrary $\sqsubseteq_{C'}$ we will use auxiliary forwarding rules with multiple priorities. For every forwarding rule r_i, $1 \le i \le n$, such that $r_i = (G_i, A_i, 1)$ we split this rule against the rules of the set $\mathcal{E}$ as follows: for every binary tuple $\sigma = (\sigma_1, \ldots, \sigma_n)$ we define a guard $G_{i,\sigma} = G_i \wedge \bigwedge_{j=1}^{n} \theta_{ij}^{\sigma_j}$, where $\theta_{ij}^1 = \theta_{ij}$ and $\theta_{ij}^0 = \neg\theta_{ij}$, and form a set of forwarding rules $Rules(r_i) = \{r_{i,\sigma} = (G_{i,\sigma}, A_i, 2) : \sigma \in \{0,1\}^n, G_{i,\sigma} \not\equiv false\}$. Let Z be a configuration obtained from C by inserting all rules from the set $Rules(r_i)$ to the flow-table of every switch w_i, $1 \le i \le n$. Clearly, Z is an unambiguous network configuration which has the same 1-hop packet forwarding relation as C'.

Lemma 1. *If a network configuration C' is free from forwarding loops (i.e. $R_{C'}^+$ is a partial order relation on the set of packet states S) then the dependency relation $\sqsubseteq_Z$ is a partial order on the set of rules $\bigcup_{i=1}^{n} Rules_i$.*

Now we are able to introduce a three-phase solution to (C, C')-NRP. Let Com_1 be the set of reconfiguration commands $\{add(r_{i,\sigma}, w_i) : r_{i,\sigma} \in Rules_i\}$, Com_2 be the set of reconfiguration commands $\{add(r_i, w_i) : 1 \le i \le n\}$, and Com_3 be the set of reconfiguration commands $\{del(w_i, G_i, 2) : 1 \le i \le n\}$.

Theorem 3. *Suppose that configurations C and C' are free from forwarding loops. Then a reconfiguration batch $(Com_1, \prec_Z); (Com_2, \emptyset); (Com_3, \emptyset)$ provides a solution to (C, C')-NRP.*

This reconfiguration batch operates as follows. At first it installs in the appropriate order the high-priority split rules from $Rules(r_i)$, $1 \le i \le n$, and, thus, seamlessly restores all complete paths of C'. Next it installs (in an arbitrary order) the low-priority rules from $\mathcal{E}$. Since every low-priority rule r_i from $\mathcal{E}$ is "locked-out" by the set of high-priority rules $Rules(r_i)$, this does not affect the 1-hop packet forwarding relation. Finally, the split high-priority rules $r_{i,\sigma}$ are deleted (in an arbitrary order). At the disabling of every such rule $r_{i,\sigma}$ its functionality is immediately passed to the corresponding low-priority rule r_i. As the result the 1-hop packet forwarding relation does not change in the course of such deletions.

Clearly, the solution provided by Theorem 3 is not optimal since it requires to split *all* forwarding rules to be reinstalled. A

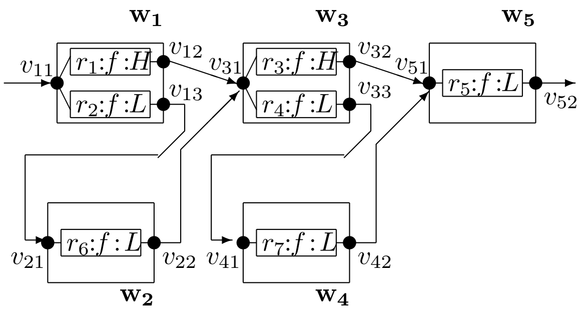


Fig. 3.

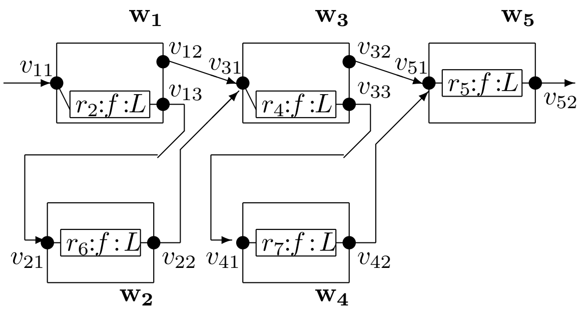


Fig. 4.

more careful treatment of the dependency relation $\sqsubseteq_{C'}$ makes it possible to reduce considerably the size of reconfiguration batch.

Let $\mathcal{E} = \{r_1, r_2, \dots, r_n\}$ be a set of forwarding rules to be reinstalled to the flow-tables of the network. Consider a transitive-reflexive closure $\sqsubseteq^*_{C'}$ of the dependency relation $\sqsubseteq_{C'}$ on the set of rules $\mathcal{E}$. It is well known that $\sqsubseteq^*_{C'}$ is a quasi-order, and the set $\mathcal{E}$ can be partitioned into equivalence classes $\mathcal{E}_1, \dots, \mathcal{E}_k$ w.r.t. $\sqsubseteq^*_{C'}$. Occasionally, some $\mathcal{E}_i$ may coincide with the whole set $\mathcal{E}$, but typically non-trivial equivalence classes are rare and small. Now for every non-trivial equivalence class $\mathcal{E}_j,\ 1 \le j \le k$, we split each rule r_i from $\mathcal{E}_j$ but only against all rules from *the same equivalence class* $\mathcal{E}_j$. Thus, we obtain the set of auxiliary rules $\widehat{Rules}(r_i)$ for every rule $r_i,\ r_i \in \mathcal{E}$, and this set is typically far smaller than $Rules(r_i)$.

Lemma 2. *Suppose that a configuration $\widehat{Z}$ is obtained from C by formally inserting all forwarding rules from $\widehat{Rules}(r_i)$ to the flow-table of every switch $w_i,\ 1 \le i \le n$. Suppose also that a network configuration C' is free from forwarding loops. Then the dependency relation $\sqsubseteq_{\widehat{Z}}$ is a partial order on the set of rules $\bigcup\limits_{i=1}^{n} \widehat{Rules}_i$.*

Following this lemma we may form a reconfiguration batch in the same way as in Theorem 3.

However, if configurations C and C' include packet forwarding rules with multiple priorities it is impossible to guarantee the solution of (C, C')-NRP.

Consider a configuration C' depicted on Fig. 3 which includes rules of high proirity (e.g., $r_1 : f : H$) and low priority (e.g., $r_2 : f : L$). Let $H = \{f\}$. Since the high priority rules suppress the low priority rules, there is only one complete path in C:

$$p_1 = (f, v_{11}), (f, v_{31}), (f, v_{51}), (f, v_{52}).$$

Suppose that high priority rules r_1 and r_3 were removed and the configuration C' degraded to a configuration C depicted on Fig. 4. Now the low priority rules in the switches w_1 and w_3 recover their capablity and the path

$$p_2 = (f, v_{11}), (f, v_{21}), (f, v_{31}), (f, v_{41}), (f, v_{51}), (f, v_{52})$$

becomes active. Clearly, this is the only complete path in configuration C. But there is no way to restore seamlessly C' from C by means of ordinary reconfiguration commands.

Theorem 4. *Suppose that configurations C' and C are those as depicted on Fig. 3 and 4 respectively. Then (C, C')-NRP doesn't have a solution, i.e. for every reconfiguration batch α there exists a prefix β of α such that $Path(\beta(C)) \not\subseteq Path(C) \cup Path(C')$.*

IV. Conclusions

We introduced an abstract model of SDN as a uniform framework for formal study of Network Update Problems and consider one of the most elementary case of NUP — the Network Recovery Problem. But as it can be seen from the list below many other variants of NUP can be formalized in the same way.

1). *Network Configuration Synthesis*. Given a post-condition Ψ build a configuration C' such that $C' \models \Psi$. An initial configuration C is of no importance since to achieve C' it suffices to uninstall all previous rules and then insert new rules of a target configuration C'. The study of this variant of NUP has been initiated in [16].

2). *Global Consistent Network Update*. Transform a network configuration C into a configuration C' in such a way that every packet traversing the network in the course of update is processed either only by the rules of C, or only by the rules of C'. In this case the post-condition $\Psi(X)$ is $X = C$, and the invariant $\Phi(X)$ is specified by the formula

$$\forall\ s\ (E_T(s)\ \rightarrow\ ((Path(X, s) \subseteq Path(C)) \vee (Path(X, s) \subseteq Path(C'))) .$$

Some algorithms for Global Consistent Network Update has been developed and studied in [10], [11], [12], [13], [14].

3). *Local Consistent Network Update*. Transform a network configuration C by safely installing a set of complete paths P_{add} and uninstalling a set of complete paths P_{del}; the safety constraint requires that in the course of updating only new paths from P_{add} may appear and only paths from P_{del} may dissolve in the intermediate configurations. In this case $\Psi(X)$ is specified by the formula $Path(X) = (Path(C) \setminus P_{del}) \cup P_{add}$, and $\Phi(X)$ is specified by the formula

$$Path(C) \setminus P_{del} \subseteq Path(X) \subseteq Path(C) \cup P_{add}.$$

This problem has been considered in [6], [7], [8], [9].

4). *Network Configuration Optimization*. Given a configuration C and some numerical characteristics of network configurations $f(X)$ (i.e. the total number or the maximal number of rules in the flow-tables) build an optimal configuration C' which has the same 1-hop packet forwarding relation. In this case $\Psi(X)$ is $(R_X = R_C)\ \wedge\ \forall\ Y\ (R_Y = R_C\ \rightarrow\ f(X) \le f(Y))$, and $\Phi(X)$ is $R_X = R_C$. Some preliminary studies of this problem have been made in [18].

While studying Network Recovery Problem we confine ourselves with the case of network configurations that include only forwarding rules of the same priority. Restoration of network configurations that include forwarding rules with multiple priorities is a far more complicated task. In a simple case, when the removal high-priority rules does not uncover hidden low-priority complete paths, it can be solved with a help of techniques similar to that in Theorem 3. But in the most extreme case the removal of high-priority rules may bring a configuration C' to an arbitrary configuration C such that 1-hop packet forwarding relations of C and C' have nothing in common. In this case Network Recovery turns into Global Consistent Network Update.

It is worth noticing that to restore network configuration seamlessly it is inevitable to check dependency relation $r' \sqsubseteq_{C'} r''$ between the forwarding rules r', r'' to be reinstalled. The way to do this is through the using of the tools for verification of network forwarding policies [19], [20], [21]. Thereby, the designing of efficient network updating algorithms relies upon the development of efficient network verification techniques.

Theorems 1 and 3 show that sometimes consistent network update may be achieved merely by the use of forwarding rule prioritization which is a far less restrictive reconfiguration means than those brought into play in [10], [11], [12], [13], [14]. We think that it is a challenging mathematical task to give a complete characterization of all those cases of NUP that could be solved by a particular network reconfiguration means. Such classification would be very much helpful in the development of efficient low-level procedures for SDN control.

V. Acknowledgments

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012, and by RFBR Grant 12-01-00706.

References

[1] N. Foster, M. Harrison, M.J. Freedman, et al., Frenetic: A Network Programming Language // Proc. of the 16th ACM SIGPLAN International Conference on Functional Programming, 2011, p. 279-291.

[2] T. S. E. N. Zheng Cai, A. L. Cox. Maestro: A System for Scalable OpenFlow Control // Tech. Rep. TR10-08, Rice University, 2010.

[3] A. Voellmy, H. Kim, N. Feamster. Procera: A Language for High-Level Reactive Network Control // Proc. of the First Workshop on Hot Topics in Software Defined Networks, 2012, p. 43-48.

[4] A. Voellmy, P. Hudak. Nettle — a Language for Configuring Routing Networks // Proc. of the IFIP TC 2 Working Conference on Domain-Specific Languages, 2009, p. 211-235.

[5] OpenFlow Switch Specification. Version 1.4.0, October 14, 2013, https://www.opennetworking.org.

[6] P. Francois, M. Shand, O. Bonaventure. Disruption-free topology reconfiguration in OSPF networks // IEEE INFOCOM, May 2007.

[7] P. Francois, P.-A. Coste, B. Decraene, O. Bonaventure. Avoiding disruptions during maintenance operations on BGP sessions // IEEE Transactions on Network and Service Management , v. 4, N 7, 2007, p. 1-11.

[8] S. Raza, Y. Zhu, C.-N. Chuah. Graceful network state migrations // IEEE/ACM Transactions on Networking, v. 19, N 4, 2011, p. 1097-1110.

[9] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, O. Bonaventure. Seamless network-wide IGP migration // ACM SIGCOMM Computer Communication Review - SIGCOMM '11 , v. 41, N 4, 2011, p. 314-325.

[10] M. Reitblatt, N. Foster, J. Rexford, D. Walker. Consistent updates for software-defined networks: change you can believe in! // HotNets, v. 7, 2011.

[11] R. McGeer. A safe, efficient update protocol for OpenFlow Networks // Proc. of the First Workshop on Hot Topics in Software Defined Networks, 2012, p. 61-66.

[12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger. D. Walker. Abstractions for Network Update // Proc. of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2012, p. 323-334.

[13] N. P. Katta, J. Rexford, D. Walker. Incremental Consistent Updates // Proc. of the Second Workshop on Hot Topics in Software Defined Networks, 2013, p. 49-54.

[14] R. McGeer. A Correct, Zero-Overhead Protocol for Network Updates // Proc. of the Second Workshop on Hot Topics in Software Defined Networks, 2013, p. 161-162.

[15] T. Mizrahi, Y. Moses. Time-based updates in software defined networks // Proc. of the Second Workshop on Hot Topics in Software Defined Networks, 2013, p. 163-164.

[16] A. Noyes, T. Warszawski, P. Cernyand, N. Foster. Toward Synthesis of Network Updates //Proc. of the Second Workshop on Synthesis, July 13-14, 2013, Saint Petersburg, Russia.

[17] A. X. Liu, C. R. Meiners, and E. Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs // IEEE/ACM Transactions on Networking , vol. 18, 2010, p. 490-500.

[18] K Kogan, S.I. Nikolenko, W. Culhane, P. Eugster, E. Ruan. Towards efficient implementation of packet classifiers // Proc. of the Second Workshop on Hot Topics in Software Defined Networks, 2013.

[19] E. Al-Shaer, W. Marrero, A. El-Atawy, K. El Badawi. Network Configuration in a Box: Toward End-to-End Verification of Network Reachability and Security // Proc. of the 17th IEEE International Conference on Network Protocols (ICNP'09), Princeton, New Jersey, USA, 2009.

[20] E.V. Chemeritsky, R.L. Smeliansky, V.A. Zakharov. A formal model and verification problems for Software Defined Networks // Proc. of the 4-th International Workshop "Program Semantics, Specification and Verification: Theory and Applications", 2013, Yekaterinburg, Russia, . 21-30.

[21] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, S. Whyte. Real Time Network Policy Checking using Header Space Analysis // 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2013).

SDN-based Innovation in New Zealand

S. Cotter
REANNZ Ltd.,
Research and Education Advanced Network
New Zealand
steve.cotter@reannz.co.nz

***Abstract*—The innovation ecosystem is often seen as a 'pipeline', or in other words, a linear commercialization model that goes from idea to the marketplace with stages in research, proof-of-concept development, prototype, product beta-testing and market launch. This often works for 'technology push' projects that commonly begin not with a discovery, but with the identification of a market need that triggers industry-led innovation.**

REANNZ, the Crown-owned research and education network supporting New Zealand's universities and research institutes, has been partnering with researchers and industry to identify market opportunities and develop SDN-based technologies to address them. In this talk, I will highlight the experiences and lessons learned on the following projects:

- **SDN-Secured ScienceDMZ that uses ACLs on an OpenFlow edge switch to whitelist authorized traffic, and configures this through a web frontend. Recently deployed at a NZ university.**
- **Distributed Routers that split a single control plane over multiple data planes in different locations – a mesh of devices behaving as a single router – in this on either side of the Pacific Ocean.**
- **SDN Internet Exchange Point deployed by a commercial carrier using OpenFlow-enabled devices. Intent is to have several across the country acting as a distributed exchange.**

Keywords—SDN; OpenFlow; ScienceDMZ; Distributed Router; Exchange Point

I. SDN-SECURED SCIENCEDMZ

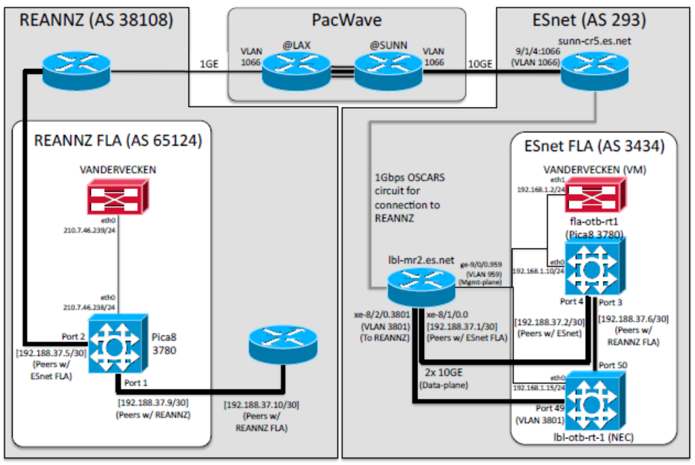
Campus infrastructures designed to support backend office systems and implementing strict firewall policies at the border are often incompatible with researchers' needs to regularly move large files. The "ScienceDMZ" architecture moves data-transfer odes outside the firewall, solving part of the high-speed data transfer problem, but reintroducing security problems that were "solved" by the firewall. Numerous approaches attempt to address these issues, from hardening the servers within the ScienceDMZ, to using short-term virtual circuits (but only if these are supported by the network provider).

REANNZ has developed a service for research-focused institutions that uses an OpenFlow-enabled device at the edge of a ScienceDMZ with a clean web interface that lets researchers poke small holes into the ScienceDMZ as they need them. Using the POX OpenFlow controller, and Django, the open source web framework, this service provides many of the security benefits of virtual circuits without requiring support from the WAN. The end result markedly reduces the attack surface, while allowing data-intensive science to enjoy maximum use of the network infrastructure.

II. DISTRIBUTED ROUTERS

REANNZ worked with researchers from Google, the US Dept of Energy's ESnet, and Victoria University of Wellington to demonstrate an OpenFlow-controlled hardware forwarding router handling a number of layer 3 routes that "challenged" the hardware and control plane (exceeding native hardware flow table resources so that FIB compression/other FIB management strategies had to be used). The primary goal was to demonstrate RIB/FIB scaling properties for future WAN projects, SDN-IX, edge router, etc. as a practical implementation.

Figure 1: Distributed Router Setup



The project leveraged community open-source packages, RouteFlow and Quagga, to establish the first BGP peering using SDN in production between two national-scale research networks. With the REANNZ switch in New Zealand and the ESnet switch in California, innovative FIB compression enabled commodity OpenFlow switches to achieve ~40% FIB compression: 13215 uncompressed routes, 7757 compressed routes.

III. SDN Internet Exchange Point

REANNZ worked with researchers from Victoria University of Wellington, Waikato University, CPqD in Brazil, and commercial Internet exchange point operator CityLink on a pilot deployment of a RouteFlow distributed router. This deployment connected the REANNZ offices to the Wellington Internet Exchange (WIX) in a production environment. RouteFlow was used to control two OpenFlow switches connected via a dark fibre link, located at border of each network. The controller was located at a third location, connected to both switches by an out-of-band layer 2 VLAN. BGP peer sessions were established with a router running within REANNZ and all WIX participants. Routes to the REANNZ network were advertised onto the WIX and traffic was forwarded through the two switches.

Commercial carrier CityLink is continuing development using NoviFlow devices. Their intent is to deploy new features like application-specific peering or certain types of traffic engineering across a distributed exchange that spans the country.

Author Biography

Steve joined REANNZ from the US Department of Energy's Energy Sciences Network (ESnet) in 2011. As the head of ESnet, he was responsible for providing high-performance networking to thousands of researchers and scientists at more than 40 national laboratories and research institutions. Prior to ESnet, he served as Google’s network deployment manager for Europe, the Middle East and Africa. Before being hired by Google, Steve worked for Internet2 as Deputy Operations Officer and was CEO & President of their wholly owned subsidiary, FiberCo. Steve earned a bachelor’s degree in aeronautical engineering at the US Naval Academy and an MBA from Boston University. He also served eight years as an officer and helicopter pilot in the US Marine Corps.

The EXPRESS SDN Experiment in the OpenLab Large Scale Shared Experimental Facility

S. Fdida(1), T. Korakis(2), H. Niavis(2), S. Salsano(3), G. Siracusano(3)
(1) UPMC Sorbonne Universités & CNRS (2) University of Thessaly (3) CNIT/University of Rome Tor Vergata

Abstract— **In this paper we describe the design and implementation of an experiments dealing with SDN for Wireless Mesh Networks over the OpenLab Facility. The experiment is called EXPRESS: "EXPerimenting and Researching Evolutions of Software-defined networking over federated test-bedS". EXPRESS aims at designing and evaluating a resilient SDN system able to operate in fragmented and intermittently connected networks as needed in a Wireless Mesh Networking environment. The experimental dimension of EXPRESS is to deploy the designed SDN infrastructure over a federation of three testbeds (PlanetLab, NITOS and w-iLab.t) from the OpenLab federation. The experiments consist in the evaluation of a designed solution for the selection of the SDN controller by the Wireless Mesh Routers in intermittently connected networks. The experiment is executed through the OMF framework (cOntrol and Management Framework). OMF provides the ability to describe the distributed experiment spanning over different physical testbeds. Following the experiment description, the OMF framework realizes the configuration of the resources (in our case the Wireless Mesh Routers) and their interconnection, runs the experiment and collects the results.**

Keywords—Software Defined Networking, Open Testbeds, Distributed Tesbeds, Testbed Federation, Wireless Mesh Networks

I. INTRODUCTION

SDN is becoming a preferred networking paradigm for Enterprise Networks and Data centers. Since, the networking community is pushing the envelope of SDN to use it in many other type of environments. The expected benefit is mostly related to the possibility to perform dynamic traffic engineering.

This paper explores the possibility to use SDN in a dynamic heterogeneous environment, such as fragmented and intermittently connected networks. The solution should be able to easily organize together isolated networks, as it may be needed in a dynamic Wireless Mesh Networking (WMN) scenario. We designed EXPRESS, which integrates the basic solutions necessary to discover the network topology and operate the routing protocols in WMNs with an SDN architecture meant to support advanced services (e.g. dynamic traffic engineering).

The complexity of the environment under study makes the evaluation of EXPRESS a complicated task. The OpenLab facility has been used for that purpose. Indeed, the experimental dimension consists in deploying the proposed SDN infrastructure (and the implemented software modules) over a federation of three testbeds (PlanetLab Europe, NITOS and w-iLab.t) from the OpenLab federation and collect performance measurements. OpenLab exploits the concept of federation of testbeds that allows a simplified access to diverse set of heterogeneous resources to the experimenter. The paper briefly presents the basic components and benefits of using OpenLab. It describes the experiments that were carried out and a set of results that demonstrate the ability of OpenLab to provide insight into the designed solution.

II. THE OPENLAB FEDERATED TESTBED

OpenLab comes from a vision originated in 2005, built on several issues related to experimentally driven research. The networking community was facing a few successes in its ability to build testing tools (like PlanetLab or Emulab) but many more failures due to well-identified causes. In addition, a challenge that is still open to our community is to develop reproducible research, meaning that one should be able to reproduce the results that are published and supports a discovery.

This vision considers that an experimenter, namely, the one that uses the facility, should have access to an ecosystem or a "market" of various resources managed by different authorities. For this purpose, the experimenter will register to one such authority that will act as a mediator towards its peers.

The beauty of this model is grounded on the observation that there exist plenty of valuable resources out there that one can benefit if an open access is provided. Some of these resources might be unique, or the sum, or combination of them might be valuable. In addition, it became quickly evident there is not a single testbed that fit all needs and that, solely, a federated model will succeed to embrace the vision.

Enabling this vision requires to define an architecture that supports the underlying concept of federation that was originally introduced in the OneLab EU funded project in 2006. Therefore, it became instrumental to address the following questions:
- What is the right level of abstraction, the minimum set of functionalities to be adopted to share resources owned by various authorities?
- What is the governance model that best supports subsidiarity?

A. The Architecture for enabling an Internet of Testbeds

We benefited from the experience in architecting the Internet to design our model. It is grounded on two principles:
- The “Hourglass” model of the Internet that identifies the IP protocol as the convergence layer. We’ll define one such convergence layer for the Federation of testbed resources,
- The peering model of the Internet that relies Customer sand Providers and define peering agreements in a way that there is not a single point of control. Here, we will clearly identify Experimenters, Testbed owners or providers and the Facility itself that rule them all.

We therefore have defined the following abstractions:
- Resource: Testbed ensures proper management of nodes, links, switches, ...
- User/Experimenter: Testbed guarantees the identity of its users
- Slice: A distributed container in which resources are shared (sharing with VMs, in time, frequency, within flowspace, etc.). It is also the base for accountability.
- Authority: An entity responsible for a subset of services (resources, users, slices, etc.)

SFA (Slice Facility Architecture) was designed as an international effort, originated by the NSF GENI framework, to provide a secure common API with the minimum possible functionality to enable a global testbed federation.

The fundamental components for testbed federation were built incrementally, as the understanding about the requirements matured. The first international realization of federation arose in 2007, as a mutual investment from PlanetLab Central, managed by Princeton, and PlanetLab Europe, established by UPMC and INRIA in Europe. It then became of utmost importance to enlarge and extend the federation principle to other type of resources, a more scalable model of federation and an increased ease of use. In parallel, started the important effort to complement and populate the architecture with components mandatory for the entire experiment life cycle.

The experiment lifecycle comprises the following steps:
❶ User account & slice creation
❷ Authentication
❸ Resource discovery
❹ Resource reservation & scheduling
❺ Configuration/instrumentation
❻ Execution
❼ Repatriation of results
❽ Resource release

Step ❶ is handled by the Home Authority of the User, the one the user has registered with. Steps ❷ to ❹ and ❽ concern all involved authorities. Steps ❺ to ❻ are not in SFA but other components such as OMF have been developed for this purpose. OMF is a control, measurement and management framework that was originally developed for the ORBIT wireless testbed at "Winlab, Rutgers University". Since 2008, OMF has been extended and maintained by NICTA as an international effort.

SFA provides a secure API that allows authenticated and authorized users to browse all the available resources and allocate those required to perform a specific experiment, according to the agreed federation policies. Therefore, SFA is used to federate the heterogeneous resources belonging to different administrative domains (authorities) to be federated. This will allow experimenters registered with these authorities to combine all available resources of these testbeds and run advanced networking experiments, involving wired and wireless technologies. The SFA layer is composed of the SFA Registry, the SFA AMs and drivers. The SFA Registry is responsible to store the users and their slices with the corresponding credentials.

MySlice[1] was introduced by UPMC as a mean to provide a graphical user interface that allows users to authenticate, browse all the testbeds resources, and manage their slices. This work was important to provide a unified and simplified view of many hidden components to the experimenter. The basic configuration of MySlice consists on the creation of an admin user and a user to whom all MySlice users could delegate their credentials for accessing the testbed resources. In order to enable MySlice to interact with heterogeneous testbeds, MySlice has to be able to generate and parse different types of RSpecs (Resource Description of the testbeds); this task is performed by plugins.

B. The OpenLab facility

The OpenLab[2] federation of testbeds was launched in august 2014 under the brand of OneLab Facility in order to avoid confusing the Openlab EU funded project that ended in august 2014 with the Facility that we expect to be sustainable). For the sake of clarity, we continue to use OpenLab as the name of the facility in this paper. OpenLab started with the following set of initial federated testbeds:

- Internet-overlaid testbeds: The public fixed-line Internet, at a global scale.
PlanetLab Europe[3], a platform offering virtual machines on over 300 servers located at over 150 locations across Europe.

- Wireless, sensing, and mobility testbeds: Internet of things testing environments.
These platforms offer both fixed nodes and mobile nodes with controlled mobility via robots or model trains. The first testbeds to fit this category are FIT-IoTLab[4] (a French testbed

[1] http://www.myslice.info
[2] http://new.OneLab.eu
[3] http://www.planet-lab.eu
[4] https://www.iot-lab.info

funded by ANR) and the NITOS[5] testbed from the University of Thessaly. The w-iLab.t[6] testbed from iMinds was added for the purpose of the EXPRESS experiment.

As the need for networking research evolves, new testbeds appears or new requirements were expressed. This has been the case for instance for the OpenFlow/SDN developments that trigger new needs and emerging testbeds (such as OFELIA[7] in Europe). The OpenLab project quite early developed a solution named OpenFlow in a Slice that provides the ability to run Openflow vswitches in a slice of a PlanetLab Europe set-up. Experimenters were then able to create an OpenFlow overlay[8] network by specifying the links between PLE nodes, benefiting from the large number of PLE nodes deployed.

Finally, the OpenLab Portal[9] was developed and provides the generic access to the facility. The portal is implemented by the MySlice software component, which allows the users to manage their slices through SFA. The NOC (Network Operation Center) has been installed in the premises of UPMC and allows a full access to the federated testbed, users and experiments managed by the facility. OpenLab is freely accessible to the community at large.

III. THE EXPRESS EXPERIMENT: SDN FOR WIRELESS MESH NETWORKS

The EXPRESS experiment has been selected for funding in the 2nd OpenLab competitive call for additional project partners. EXPRESS stands for "EXPerimenting and Researching Evolutions of Software-defined networking over federated test-bedS" and it includes two main dimensions: scientific and experimental. The scientific dimension considered the design of an innovative, resilient SDN system able to keep operating in fragmented and intermittently connected networks. Such a system should be able to easily glue together isolated networks, as it may be needed in a dynamic Wireless Mesh Networking (WMN) scenario. EXPRESS integrates the basic solutions necessary to discover the network topology and operate the routing protocols in WMNs with an SDN architecture meant to support advanced services (e.g. dynamic traffic engineering). The experimental dimension consists in deploying the proposed SDN infrastructure (and the implemented software modules) over a federation of three testbeds (PlanetLab, NITOS and w-iLab.t) from the OpenLab federation and collect performance measurements.

[5] http://nitlab.inf.uth.gr/NITlab/

[6] http://ilabt.iminds.be/wilabt

[7] http://www.fp7-ofelia.eu/

[8] https://www.planet-lab.eu/doc/guides/user/practices/openflow

[9] http://portal.OneLab.eu/

A. Scientific questions and technicall challenges

The main scientific question behind the experiment is whether the SDN paradigm can be applied to networking scenarios where: 1) it is not feasible or reasonable to implement a separate *out-of-band* signaling infrastructure among nodes, therefore SDN signaling will be intermixed at packet level with user data flows following an *in-band* approach; 2) there is a relatively high probability of link failure, the network can become partitioned in disconnected set of nodes, the partitions can later merge back into larger partitions. These conditions may occur in Wireless Mesh Networks (WMNs), like Community Networks ([14]), in which some parts of the network are interconnected by long links that may temporary fail. The reference scenario for our work is shown in Fig. 1, as an example the link between the Wireless Mesh Routers A and B can partition the network in two parts if it goes down. Let us now consider the advantages and the criticalities of using SDN in WMNs.

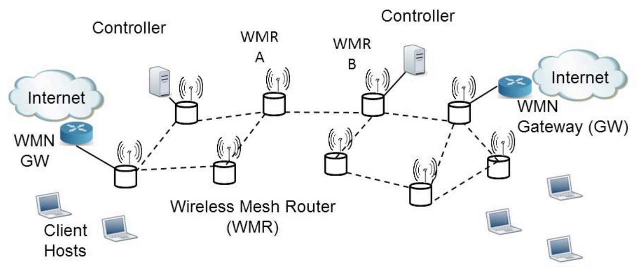


Fig. 1. Wireless Mesh Network reference scenario

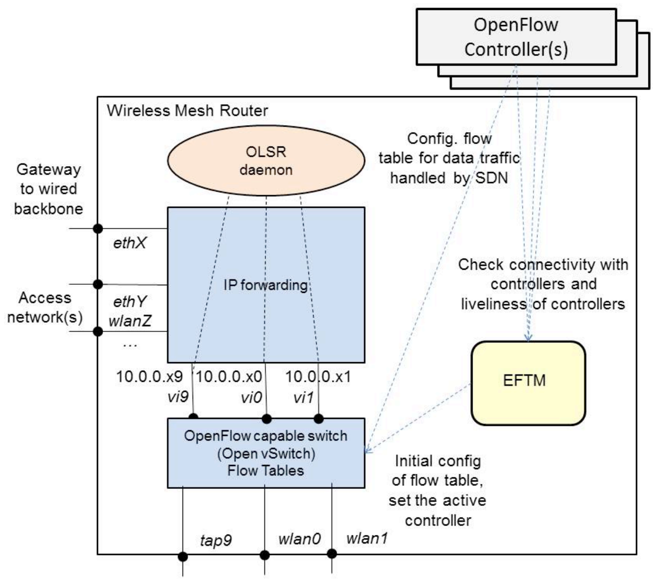
The advantage of introducing the SDN paradigm in such environment are mostly related to the possibility to perform dynamic traffic engineering to optimally distribute the traffic over the wireless resources and across the different gateways towards the Internet that could be available in the WMN. The IP best effort routing based on distributed shortest path (e.g. with OLSR [15] or OSPF routing protocols) may lead to poor utilization of the available capacity, with bottlenecks constituted by congested wireless links or gateway nodes. We expect that, using the SDN paradigm will make possible to optimally allocate the user traffic with the needed level of granularity.

On the other hand, using SDN in the considered WMN scenarios has some criticalities. We have identified two main challenges. As for the first challenge, a SDN based approach requires a control connection between the controlled network nodes and the SDN controllers. In a fixed networking environment the *control-plane* communications between the switching nodes and the SDN controllers typically run over *out-of-band* channels, separated from the *data-plane* traffic. For example VLANs can be used in a layer 2 Ethernet network to establish a "signaling" network that will operate independently from the SDN mechanisms used to manipulate the data-plane traffic. Replicating this approach in the WMN scenario will not work, because: i) VLANs are not typically used in WiFi networks; ii) the basic connectivity among nodes of a WMN (referred to as WMR, Wireless Mesh Routers) is established using layer 3 routing protocols. The first challenge is therefore to design a SDN solution suited to the characteristics of WMNs.

The second challenge that we addressed concerns the applicability of SDN in network partitioning and merging scenarios. Assuming that a SDN controller runs over a set of WMRs, if the network becomes partitioned a subset of WMRs will disconnect from the controller and will need to associate to a different controller (if available in the partition). On the other hand, if two network partitions under the control of two different SDN controllers merge into a single partition, it is desirable that all WMRs fall under a single SDN controller. Clearly, the service logic in the different SDN controllers needs to be coordinated, but as prerequisite we focused on the issue of the establishment of the connection between the WMRs and the most appropriate SDN controller. We can restate the second challenge as "SDN controller selection under network partitioning and merging scenarios". The problem of assigning a SDN controller to each switch in a network with different SDN controllers has been already faced when considering "distributed" SDN solution with multiple controllers, see for example [2]. According to the OpenFlow specifications [4], when a switch is connected to multiple SDN controllers, one of these controllers can act as *master*. The procedure to select the master controller for a given switch is typically referred to as master *election*. The reason is that the procedure is distributed among the controllers that coordinate with each other in order to *elect* the master. The switch is slave in this approach and will be notified by the winner of the election. This procedure works well assuming that there is a stable connectivity among the controllers (in fact in the typical use case the procedure needs to elect a master controller among a set of "replica" controllers operating in the same data center). Using this procedure in the considered WMN scenario may easily lead to inconsistent results. The convergence time of routing protocols used in WMNs is in the order of seconds. During transient phases the different controllers may have different visions of network connectivity. For example two controllers could both believe to be the best candidates to take mastership of a given switch and can both start acting as master for the switch.

A. Solutions to 1st challenge (SDN in WMNs)

In order to address the first challenge identified above, the designed solution foresees to use the OLSR routing protocol [15] to establish the basic IP connectivity in the WMN. Coexistence mechanisms are defined between packets routed using classical IP routing tables (including the OLSR packets) and packets routed using the SDN approach under the instructions of SDN controllers. The forwarding of SDN signaling packets follows an *in-band* approach, i.e. the packets between the switching nodes and the SDN controllers are sent on the same network on which the data plane packets are sent. The signaling packets belonging to the SDN control plane (among WMN nodes and SDN controllers) are forwarded using the basic IP routing information established using OLSR, while the data packets can be forwarded using the basic IP routing or using arbitrary routing under the control of the SDN controller.



OpenFlow Controller(s)
Wireless Mesh Router
OLSR daemon
Config. flow table for data traffic handled by SDN
Gateway to wired backbone
ethX
IP forwarding
Access network(s)
ethY
wlanZ
...
Check connectivity with controllers and liveliness of controllers
10.0.0.x9
vi9
10.0.0.x0
vi0
10.0.0.x1
vi1
EFTM
OpenFlow capable switch (Open vSwitch) Flow Tables
Initial config of flow table, set the active controller
tap9
wlan0
wlan1

Fig. 2. wmSDN node architecture

The architecture of a node implementing the proposed solution is reported in Fig. 2, the solution was first proposed in [3], referred to as wmSDN (wireless mesh SDN). Then it has been improved and extended in [5] taking into account the OSHI IP/SDN framework [6]. We refer the reader to [5] for the technical details of the solution. In the context of EXPRESS, we have ported and deployed the solution on real wireless nodes in the NITOS and w-iLab.t testbeds.

B. Solution to 2nd challenge (SDN controller selection in network partitioning and merging scenarios)

Coming to the second challenge, the EXPRESS experiment has designed and implemented a solution for SDN controller selection by the Wireless Mesh Routers. The main idea is to assign more responsibility to the controlled nodes (WMRs), letting them take the decision about which switch has to take mastership of the node. Therefore we named this procedure controller *selection* rather than master *election*. The nodes will monitor a set of SDN controllers that can potentially assume the master role and will implement a selection algorithm to choose the preferred controller among the set of reachable controllers (see Fig. 3). Note that WMRs and controllers have the same information about the status of the network (excluding transient conditions), because they share the OLSR vision of the topology. In particular, the WMRs are directly involved in the OLSR topology dissemination while the controller extracts the topology information from a nearby WMR. Therefore, from the topology discovery point of view the WMR acquires topology information even before the controller. Moreover, a WMR can directly check the connectivity with potential controllers trying to establish TCP connections towards them (or monitoring the liveliness of established TCP connections).

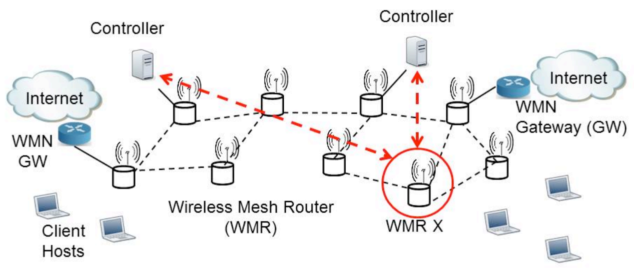


Fig. 3. Controller selection by a Wireless Mesh Router

In the designed procedure a WMR connects only toward a single controller at a given time. This is different from the classical approach where a switch connects in parallel with several controllers. The procedure is performed in the WMR with the help of the EFTM (External Flow Table Manager) entity shown in Fig. 2. The EFTM entity is in charge to perform the master selection procedure and will instruct the switch to connect to the selected controller at a given time.

Performing the master selection on the WMR side has some advantages in our scenario. The first advantage is that each OpenFlow switch will be connected with a single controller at a time, and no conflicting rules can be injected. The second advantage is that a run-time coordination mechanism among controllers is not needed, each controller can operate on its own, obviously all the controllers should follow a consistent service logic.

IV. Experimenting over the OpenLab Testbeds

As described above, we designed and developed novel algorithms and procedures in order to address the aforementioned challenges, thus we needed to test them under real world settings and in the largest scale possible. To this end, we took advantage of the OpenLab facility that provides the unique capability to deploy and evaluate experiments easily in a mid-scale environment exploiting a plethora of different kinds of resources.

A. Taking advantage of the tools provided by OpenLab

The main issue when you conduct an experiment involving several heterogeneous resources is burden related to their control and configuration, as well as their synchronization during the experiment. This was addressed easily by using the OMF framework, which enabled us to configure and control the different kinds of nodes that were part of the experiment, through a single script. In this script, which is written in a simple, domain-specific language provided by OMF, namely the OEDL [7], we described the required initial configuration of the nodes and specified a list of events and associated tasks, as well. The list of tasks includes the drop of a link, the sleep for a specified time period or measurement points for collecting data.

Another big issue that OpenLab tools assisted us to address is the gathering of the measurements generated during the experiment, in a unified way. The collection of all those data is handled by the OML [8], which is again provided by OMF. In this way, we defined measurement points in the experiment description script and OML handled the collection of the experimental results and their storage in a database for further processing.

The necessary steps for conducting the experiment on the federated testbeds and the tools we used are:

- The reservation of the resources through mySlice portal,
- The development of the experiment description using OEDL,
- The development of the experiment scripts through OMF,
- The execution of the experiment through a single script and the collection of the measurements through OML.

In the following subsection, we describe the main challenges faced and the methods followed towards the successful deployment of the EXPRESS experiment on the OpenLab federation.

B. Main challenges during the deployment

The experiment is performed across three different OpenLab testbeds, two wireless testbeds (NITOS [9] and w-iLab.t [10]) that supports different Wireless Mesh Networks and a wired testbed (PLE - Planetlab Europe [11]) that is used to emulate a "backbone" link interconnecting the two Wireless Mesh Networks. The backbone link was implemented through the establishment of an Ethernet over UDP tunnel across PlanetLab testbed (actually two UDP tunnels bridged with a Virtual Switch on a PLE node, as shown in Fig. 4).

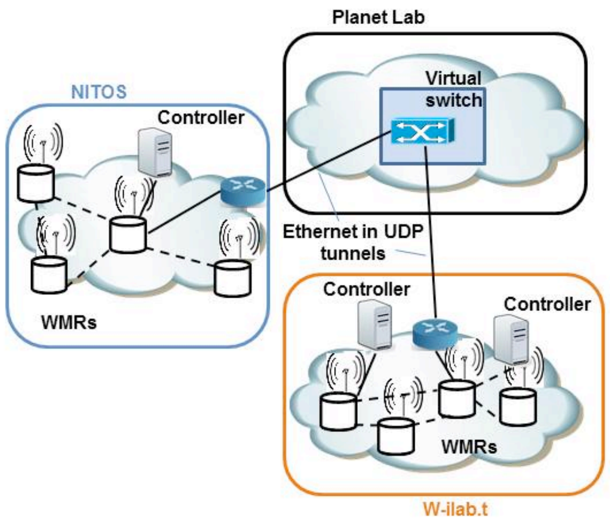


Fig. 4. Experiment over PlanetLab, NITOS and w-iLab.t

In most cases, the description of the experiment in the OMF language is a straightforward procedure, so was in our case. On the other hand, one of the difficulties we faced during the deployment is that most of the wireless nodes in the testbeds are inside the coverage area of all the other wireless nodes in their testbed, making difficult to emulate topologies where nodes lie more than two-hops away from each other. In order to face this difficulty, we filtered the packets at the receiving node in order to emulate the desired experimental topology – thus all packets received by nodes that are not in the mutual communication range were discarded. Further details on the aforementioned procedure can be found in the OpenLab Deliverable D3.11 [12].

Another problem we had to face is the private IP addressing that NITOS and w-iLab.t use for their wireless nodes. Taking under consideration the advantages and disadvantages of all the different options for dealing with this problem, we concluded that a solution based on NAT is the most appropriate and applicable one for our situation. Further details on the aforementioned procedure can be found in the OpenLab Deliverable D3.10 [13].

Since we successfully implemented all challenges during the deployment phase, we designed and developed two different types of experiments.

C. *Description of the experiments*

In the combined OpenLab testbed, we run two types of experiments, respectively denoted as "network merging" and "network partitioning" experiment. In the network merging experiment we start with two mutually disconnected network sets (each one with a SDN controller) and then reactivate a wireless link between two WMRs residing in different sets. In the network partitioning experiment we deliberately deactivate a wireless link that interconnects two sets of the networks, each one including (at least) a SDN controller. In both experiments the following simple control logic is run by the WMR nodes. Each WMR node has a list of SDN controllers that can potentially take control of the node, ordered by priority. The SDN controllers are listed with their IP address. The WMR will periodically check which controller IP addresses are reachable looking at the IP routing table established by means of the OLSR protocol. The WMR will try to connect to all reachable controllers (and will check the liveliness of the connection for the currently selected master SDN controller). Then it will select the highest priority controller among the ones to which it has successfully established a connection. In the experiment, the priority list of the preferred controllers was simply preconfigured in the nodes (using a configuration file). In a real life implementation the priority list could be transferred by a SDN controller to the WMR node and updated when needed.

In both types of experiments, we measured the time needed for the WMRs to connect to the highest priority controller after the event that determined the network merging/partitioning. In the network merging experiment, this time interval can be decomposed into two phases: in the first phase the IP routing (OLSR) will properly reconfigure the connectivity in the control network (OLSR routing procedure). In the second phase our proposed controller selection algorithm will operate to select the highest priority controller (controller selection procedure). In the network partitioning experiment, the WMR node does not rely on OLSR to detect that a controller is no longer reachable, but it will perform its own connectivity check, achieving a faster reaction time.

The rationale of the two experiments is to demonstrate that the controller selection procedure operates within the same time scale than the OLSR restoration procedure. If we accept the performance of OLSR in routing packets over the WMN, we will likely accept the performance of the proposed SDN based approach offering traffic engineering services in the WMN.

D. *Experiment setups*

To perform the network merging experiment, the tunnel between the two wireless testbeds is initially not active, and the WMRs are connected to their respective controllers available within their own local testbed. As soon as the tunnel across Planet Lab Europe is activated, messages start to flow from one wireless testbed to the other and the WMR nodes belonging to w.iLab-t testbed learn the IP route towards the remote controller in NITOS. The EFTM entity implemented into each of the WMR checks if a controller is actually active at that IP address by trying to establish an OpenFlow protocol connection. When this check is positive, the entity chooses to connect to the highest priority controller, in this case the one in the remote wireless testbed (NITOS).

To perform the network partitioning experiments we drop the tunnel established through PlanetLab and measure the time needed by WMRs in each wireless testbed to connect to their local controllers.

V. Experiments Results

In this section we present the experiment results. We do not aim to provide an in depth technical analysis of the results. We only illustrate what results we obtained and show how they can support the answers to the questions we have identified in section III.

A. *Network merging experiment*

In this experiment we evaluate the time needed for the WMRs to connect to a higher priority controller after the merging of two network partitions. As shown in Fig. 5, this time is decomposed in two phases, network connectivity and master selection. The former one considers the time needed for the routing protocol to setup the IP routes in all WMRs taking into account the merged network topology. We measure it by trying to send ping requests from a WMR to the controller that was not reachable before the merging event. In our experiments it averages to 15 seconds. In fact, according to the OLSR routing protocol mechanisms, three "Hello" messages need to be received in order to declare a link up and the default interval for sending OLSR Hello messages is 5 seconds. Starting from this time instant we measure the interval needed for the WMRs to disconnect from old controller and connect to the one with higher priority. In our implementation the EFTM periodically tries to establish a connection with all controllers that have been discovered, starting from the highest priority one. The polling period is 3 seconds. In the experiment we measured an average of more than 1.5 seconds for the latest connected WMR, which is consistent with the expectations.

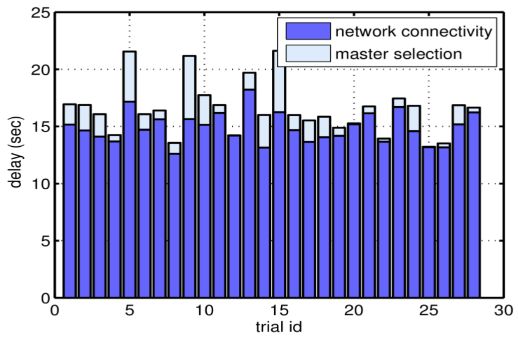


Fig. 5. Network merging experiment

B. *Network partitioning experiments*

In this second set of experiments we consider the partitioning of the network: starting from a connected network as shown in Fig. 4, we disconnect one of the tunnels across PlanetLab Europe. In Fig. 6 we report the evaluation of the time needed by the WMRs in the w-iLab.t testbed to disconnect from the remote controller in NITOS and connect to the local controller. In this case the WMRs does not rely on OLSR to discover that a controller is not reachable, as it would require more than 15 seconds considering the default OLSR configuration (3 Hello intervals of 5 s needed to declare the link down). The ETFM periodic controller polling procedure (running with a 3 seconds period) considers a 2 seconds timeout before declaring that a controller is down. With this procedure, an average master selection delay of 5.5 seconds is measured.

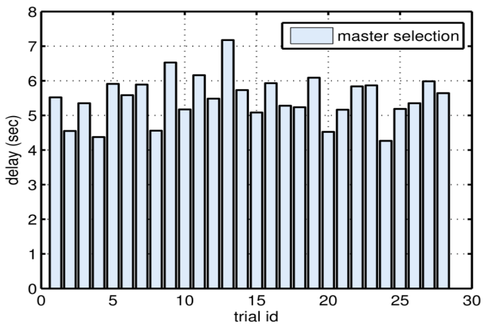


Fig. 6. Network partitioning experiment: master selection delay

VI. Conclusions

This paper presents two major contributions. The first one is related to the question whether SDN can be efficiently used in a dynamic environment with intermittently connected networks. A solution has been designed for this purpose. Nevertheless, it is critical to evaluate such a proposal in a practical setting. The second value of the paper is to demonstrate that the OpenLab federation of heterogeneous testbeds provides the mean to configure and experiment the solution derived in order to assess its performance. The experiment was conducted with a reduced effort thanks to the tools provided by the OpenLab facility. Despites the fact that the experiment involved three different and heterogeneous testbeds, the performance of the proposed solution has been captured and the results helped to answer the suitability of SDN for this type of complex environments.

Acknowledgments

This work has been supported by the OpenLab project, "OpenLab: extending FIRE testbeds and tools ", funded by the EC under FP Grant agreement no: 287581.

References

[1] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, “OMF: a control and management framework for networking testbeds”, SIGOPS Oper. Syst. Rev. 43, 4 (January 2010), pp. 54-59

[2] A. Dixit, F. Hao,S. Mukherjee,T.V. Lakshman, R. Kompella, “Towards an Elastic Distributed SDN Controller”, ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), August 16, 2013, Hong Kong

[3] A. Detti, C. Pisa, S. Salsano, N. Blefari-Melazzi, “Wireless Mesh Software Defined Networks (wmSDN)”, CNBuB Workshop at IEEE WiMob, France, Lyon, 7 October 2013

[4] Open Networking Foundation (ONF) “OpenFlow Switch Specification”, Version 1.4.0, October 14, 2013

[5] S. Salsano, G. Siracusano, A. Detti, C. Pisa, P. L. Ventre, N. Blefari-Melazzi, “Controller selection in a Wireless Mesh SDN under network partitioning and merging scenarios”, submitted paper, available at http://arxiv.org/abs/1406.2470

[6] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, M. Gerola, E. Salvadori, “OSHI - Open Source Hybrid IP/SDN networking (and its emulation on Mininet and on distributed SDN testbeds)”, EWSDN 2014, 1-3 September 2014, Budapest, Hungary

[7] OEDL - The OMF Experiment Description Language http://mytestbed.net/projects/omf54/wiki/OEDL-5-4

[8] OML - Measurement Library http://mytestbed.net/projects/oml/wiki/

[9] NITOS testbed http://nitlab.inf.uth.gr/NITlab/index.php/testbed

[10] w-iLab.t testbed http://ilabt.iminds.be/wilabt

[11] PlanetLab Europe (PLE) testbed http://www.planet-lab.eu/

[12] S. Salsano (editor) “EXPRESS final evaluation and overall report”, Deliverable D3.11 of EU FP7 Project “OpenLab”

[13] S. Salsano (editor) “EXPRESS – Final architecture and design, evaluation plan”, Deliverable D3.10 of EU FP7 Project “OpenLab”

[14] http://en.wikipedia.org/wiki/Wireless_community_network

[15] T. Clausen (Ed.), P. Jacquet (Ed.), “Optimized Link State Routing Protocol (OLSR)”, IETF RFC 3626, October 2003

SDNI: The GEANT Testbeds Service – Virtual Network Environments for Advanced Network and Applications Research

M. Hazlinsky
CESnet
Prague, CZ
hazlinsky@cesnet.cz

F. Farina
GARR
Milan, IT
fabio.farina@garr.it

B. Pietrzak
PSNC
Poznan, PL
blazej.pietrzak@man.poznan.pl

J. Sobieski
NORDUnet
Kastrup, DK
jerry@nordu.net

P. Szegedi
TERENA
Amsterdam, NE
szegedi@terena.org

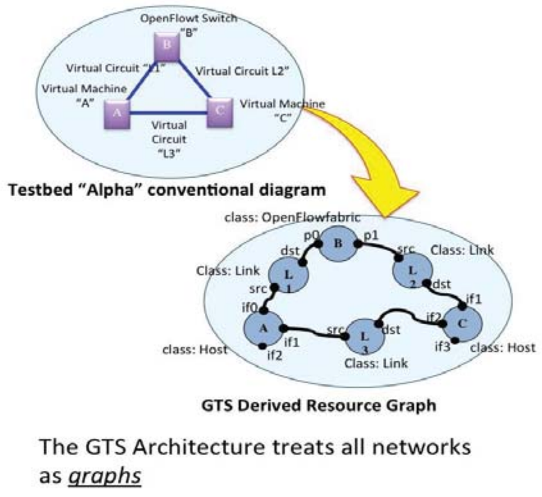
Abstract—The last decade has produced many important advances in virtualization of IT infrastructure. Virtual Machines, once a novelty, have become an integral component of production web applications and the foundation for cloud services. Virtual Circuits have been around even longer and are now seeing the deployment of multi-domain provisioning services. The conventional core internet routing and switching platforms are giving way to virtualized facilities including virtual routers, virtualized network functions, and even virtual switching and forwarding fabrics.

These technologies have been developed traditionally in dedicated labs or testbeds constructed from hardware and software specifically engineered to address particular notions or research topics. The cost and lead time required to construct a conventional experimental network testbed is high, and doing so at scale requires not only advanced engineering skills, but administrative, operational, and contracting skills not typically available among research teams. Such network research typically evolves very quickly which requires constant re-engineering and re-configuration of the underlying experimental facilities. And integrated environments – where new technologies can be tested with other emerging concepts - are even more elusive in this anachronistic model of network experimentation.

But this is changing. Projects such as Planet Lab have explored the prospect of providing testbeds on a global scale for research projects, and more recently the US GENI effort pushes these concepts further in areas such as virtualization, software defined networks, and wireless. Likewise, projects within the European research community have been exploring, formalizing, and generalizing the architectures needed to address advanced network research: Projects such as FEDERICA, GEYSERS, Novi, and OFELIA have been at the forefront. And the interest in SDN and OpenFlow technologies are posing still other novel challenges and opportunities for experimental and virtualized services.

GEANT, the pan-European research and education network, has been involved in and/or supported these efforts. In the current generation of GEANT, the network has developed a new production service called the GEANT Testbeds Service ("GTS"). GTS provides a virtualized network environment that is formally defined, dynamically scheduled and provisioned, and user controled. The GTS facilities are geographically distributed across Europe, co-located with the GEANT core network nodes. This service leverages the knowledge and experience gained from these prior research efforts and has developed a formal architecture for constructing comprehensive virtual networks, at scale, to support the network research community. This paper will describe this architecture and the present status of the GEANT Testbeds Service, key technical capabilities of the Service, and the long term road map for the feature set. This paper will close with some thoughts on the strategic international collaborations we believe will be necessary to create a ubiquitous global standard for delivering virtual network environments into the future.

I. The GEANT Testbeds Service Overview



The GEANT Testbeds Service ("GTS") is a new production service introduced by the GEANT Network to address experimental needs of the networking and distributed applications research community. GTS allows network researchers to formally describe an experimental network environment using a domain specific grammar and then instantiate and manage that environment though its life cycle. Testbeds can be scheduled, i.e. set up at a pre-arranged time in the future, and they can be provisioned across geographically distributed locations. The Service model is generic and highly extensible so that it can incorporate a wide array of experimental components. This Service dramatically reduces costs, complexity, and lead times required to field experimental networks at scale and allows the research community to focus on the research topic rather than the scaffolding associated with conventional network engineering, contracting, procurement, installation, and operations of the physical facilities.

Section 2 of this paper describes the GTS architecture – the conceptual model, and the processes and components that deliver the service and manage the underlying infrastructure upon which it operates. Section 3 will address the specifics of GTS version 1.0 – the key feature set and the current deployment status. Section 4 will provide a features and capabilities roadmap for GTS covering both the near term version 2.0 and the version 3.0 anticipated in early GEANT 4. Section 5 will address the longer term strategy towards interoperability and global scaling.

II. The GEANT Testbeds Service Architecture

The GTS architecture and service semantics are based in the proposition that all networks can be represented as graphs. In conventional graphs, the graph vertices represent network processes that source or sink data flows, or make forwarding decisions about the data as it transits the vertex. The graph edges represent data transport processes (e.g. circuits or links) that simply move data transparently and unmodified from one network vertex to the next. However, GTS reduces and generalizes this conventional network graph one step further to create a *derived resource graph* (DRG.) In the DRG, all data plane functions – transport, switching, forwarding, acquisition, storage, etc - are represented as "resource" objects, and these resources make up the vertices (the nodes) in the DRG. The topology of the network is expressed separately through a set of "adjacency" relationships that constitute the edges in the DRG. This DRG representation allows GTS to treat all data plane functional components in a consistent object oriented fashion, and it allows the juxtaposition of those data plane components to be described separately via the adjacencies.

Just as in the conventional graph, the DRG model recognizes that any particular resource object may have multiple links to/from other resources, perhaps even multiple links between the same two resources. There must be a means to identify and differentiate these links within the local context of each resource. The DRG model defines "port" constructs to enumerate the I/O interfaces associated with a resource thus allowing each I/O interface to have its own unique port id within each resource. The connectivity among resources, and therefore the topology of the network, is described by the adjacency relationships that indicate which resource-ports are connected to which other resource-ports. The port construct allows attributes of each interface to be described separately from the attributes of the resource itself.

With these three basic constructs – resources, ports, and adjacencies – GTS can fully describe a network topology. And the DRG model neatly organizes and associates attributes of each of these key components in a consistent and generic manner regardless of the specific resource functionality. This model simplifies many aspects of network analysis and management, enhances the extensibility of the overall service model, and aids in the reliability and maintainability of the software.

Resources are not uniform – obviously a data transport circuit differs significantly in functionality and relevant attributes from, say, an OpenFlow switching fabric. The GTS model organizes resources by their Type or class. Resources of a given Type will have a common set of attributes or parameters that constrain or define resources of that type. For example, a resource definition of type "Host" may have a parameter "cpuClockRate" that defines the cpu speed of that class of resource. Likewise, a reference to resource of that class may use cpuClockRate as a constraint that must be met when searching for (or manufacturing) a resource instance to satisfy a resource request. Further, even resources of the same class may differ somewhat in their respective attributes within the context of a particular network. For instance, two switching nodes that otherwise are identical in functional characteristics may have different numbers of ports. These differing characteristics of a resource instance do not change the fundamental capabilities or function of the resource type or class.

So GTS supports a resource Class Definition that defines the fixed public *attributes* of a resource and the parameters that can be set/chosen by the user when requesting such a resource.

The user interacts with the GTS service though a set of request/response messages that carry functional primitives (basic commands or requests) between the "user" agent and the "provider" agent. The specific primitives themselves are defined as part of each resource class definition. GTS requires all resource classes to support a base set of primitives: *Reserve* and *Release* of the resource reservation, *Activate* or *Deactivate* of the resource (during its reserved window), and a *Query* primitive that returns the state of the resource. In addition to these base primitives, the GTS model allows resources to define additional resource specific primitives that perform control function(s) that may be unique to a particular class of resource. For instance, a VM resource may provide a “coldStart” primitive that reboots the VM as if the power had been cycled.

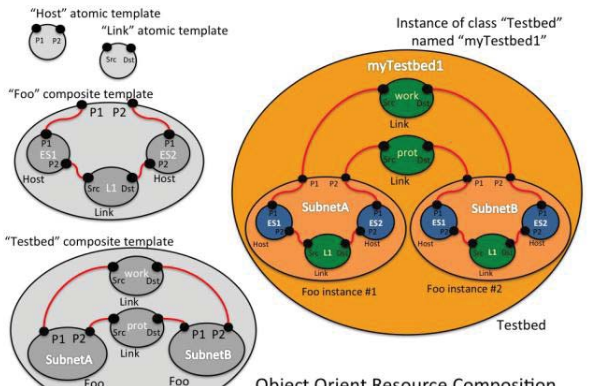
The GTS software suite includes a web based GUI that acts as a user agent. From the GUI, a user can login to the Service, import resource type definitions, instantiate one or more of those resources, and then access those resource instances via other control primitives or via console proxy or some another means (e.g. SSH). The GUI can also display resource state information for a single resource or an entire testbed network

TESTBED RESOURCES

At a very high level, the GEANT Testbeds Service (GTS) implements a simple *virtual resource* model that creates resource instances from underlying *infrastructure* facilities according to resource class descriptions. The Service allocates these virtual resources to the user’s Project for a fixed time period (the reservation window.) The GTS virtualization model is highly scalable, secure, and extensible, and it allows GTS to efficiently share the underlying infrastructure among many potential users.

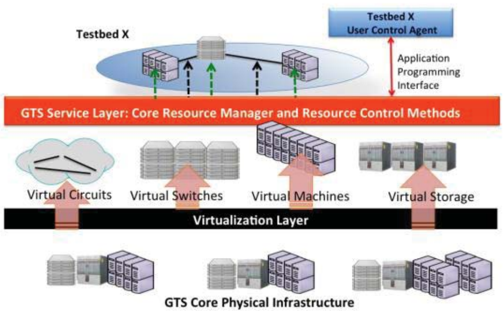
From the user’s perspective, their testbed is comprised solely of virtual resources instantiated within an abstracted virtual private universe. Resource instances have a lifecycle that extends from when the resource is first referenced as part of a Reserve() request until that reservation is Released() and the virtual resource is decomposed and returned to the infrastructure pool. The initial GTS v1.0 supports four key resource types: Composite resources are logical containers that contain other resources. Composites enable an object oriented approach to testbed construction; Host resources are Linux virtual machines running on Dell server hardware and managed by the OpenStack software; Link resources are ethernet framed data transport circuits provisioned over NSI circuit services; And OpenFlowFabric resources are OpenFlow capable data plane switching fabrics provisioned over HP 5900 hardware platforms. These basic resource classes will be enhanced over time to provide more flavors of each resource type. In general, the virtualization process simply allows the underlying physical infrastructure to be partitioned or timeshared and does not otherwise obscure access to the actual underlying physical hardware performance.

A user creates a “Testbed” by first defining a composite resource class to contain all other resource, port, adjacency, and attribute specifications required to define the testbed. When the user Reserves this top level (“root”) resource, the GTS service agent will process the root Testbed description, recursively locate and process the class descriptions for any composite children resources encountered, until only “atomic” resources are remaining. “Atomic” resources are resources that contain no other resource references. Thus a *resource tree* is constructed that contains all of the information required to satisfy the user’s resource reservation request. Once the resource tree has been constructed, and all port adjacencies resolved, the tree is processed once again to actually reserve the constituent atomic resources. This resource tree is the complete internal representation of the user’s testbed and is stored in a persistent Resource DataBase.



Object Orient Resource Composition

The virtual resources in GTS are manufactured as needed from underlying *infrastructure* objects. It is important to differentiate the Service layer virtualization processes from the Data Plane virtualization processes. Functions such as API and primitive processing, resource allocation and mapping, life cycle management, book-ahead scheduling, policy application, etc are performed by the Service layer software. These service layer functions typically only manage and orchestrate the resources from the sidelines, and are not involved directly in realizing the virtualized entities themselves. Data Plane virtualization refers to processes that realize the virtual resource at the data plane. Examples of these data plane processes include the hypervisor that allocates cpu cycles to VM instances and emulates low level VM systems calls, or the encapsulation processes in network elements that sort traffic into virtual circuits or shape that traffic accordingly as it transits a switch.


Testbed X
Testbed X User Control Agent
Application Programming Interface
GTS Service Layer: Core Resource Manager and Resource Control Methods
Virtual Circuits
Virtual Switches
Virtual Machines
Virtual Storage
Virtualization Layer
GTS Core Physical Infrastructure

Virtualisation and Services Layers

“Dynamic” resources are manufactured on-demand from the infrastructure when a Reserve request is received by GTS core service agent. For some commonly requested resources, the resources can be provisioned in advance with a fixed “static” configuration. Such static resources sit in an available “running-but-idle” state until needed. These static resources can be allocated and Activated very rapidly at the cost of some configuration flexibility.

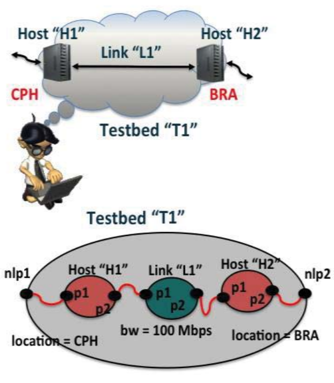
Data flow among resources – and between the internal virtual testbed universe and the external real world- are explicitly defined using the resource port constructs. The GTS virtualization model does not allow data to enter or exit a resource except through one of these explicitly defined ports. This feature allows the Service to identify all I/O flows, and very carefully monitor interfaces that could [potential] pose a threat to external systems or services. If/when necessary, the GTS port monitoring can trigger remedial action if those flows exceed certain thresholds. As long as the data flow activities of a testbed remain internal, among the resources of that testbed, the Service does not interfere with the user's experiment.

GTS testbeds and all associated resources are described using a *domain specific language* – referred to as the GTS "DSL". The DSL is an object oriented grammar, implemented in the Groovy programming language, that provides a concise and very powerful means of describing complex resources. The DSL provides a number of novel capabilities such as *iterators* that allow the grammar to describe large networks using programmatically constructed resource specifications. The DSL inherits many features of modern programming languages. In this respect, it empowers the researcher to develop parametric network design specifications that can dynamically shape target testbeds to address goals of individual experiments.

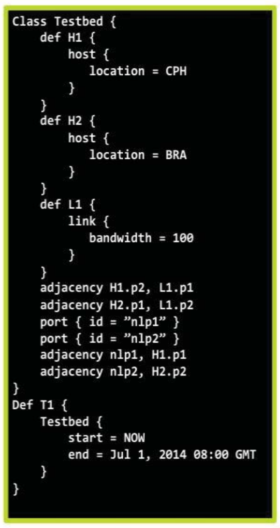
The GEANT Testbeds Service is a work in progress. The GTS architecture is the product of much prior research and the developers believe this prior work is now showing benefit both in its simplicity for new users to be productive and in its adaptability and power to address new requirements into the future.

III. Key Features of GTS version 1.0

The GEANT Testbeds Service first became available for users as a production service in August 2014. It represents an initial implementation of a strategic vision for how future advanced network research can co-exists with and leverage shared production networking infrastructure.



The GTS Domain Specific Language
A grammar tailored for testbeds



```
Class Testbed {
    def H1 {
        host {
           location = CPH
        }
    }
    def H2 {
        host {
           location = BRA
        }
    }
    def L1 {
        link {
           bandwidth = 100
        }
    }
    adjacency H1.p2, L1.p1
    adjacency H2.p1, L1.p2
    port { id = "nlp1" }
    port { id = "nlp2" }
    adjacency nlp1, H1.p1
    adjacency nlp2, H2.p2
}
Def T1 {
    Testbed {
        start = NOW
        end = Jul 1, 2014 08:00 GMT
    }
}
```

The initial deployment of GTS v1.0 offers some basic capabilities and resource types that act as building blocks – allowing new users to experience the service and to become productive very quickly. As mentioned previously, version 1.0 introduces Composite resources, Host and Link resources, and OpenFlowFabric resources. The Composite, Host, and Link classes are general classes and will provide a broadbased capability for the user. The OpenFlowFabric resource class is more specialized and is a good example of how the architecture can accommodate such novel resource classes.

GTS v1.0 provides the researcher with a web based Graphical User Interface that interacts with the GTS service and allows the researcher (the user) to easily manage their testbed resources. This is a basic functionality in v1.0 that allows the user to upload the DSL testbed descriptions, reserve resources, activate/deactivate resources, release resources, and display state and informational attributes of resources. Since one important feature of GTS is its ability to create object oriented and/or geographically distributed virtual environments, the GUI can present the testbed details as either as a tree structured list (similar to a file directory listing) or project it onto Google Earth for geographical presentation. The GUI also provides service administrators with menus to create and manage projects and users associated with each project.

The Testbeds Service supports book-ahead reservations for all resources. This allows users to coordinate their experiments with other external events or to simply coordinate access to diverse resources within the Service itself. GTS treats a resource reservation as an important commitment to the user and does not overbook, prioritize, or preempt confirmed reservations except in the event of unavoidable and unanticipated hardware failures. Even then, in many cases a testbed reservation can be remapped to other infrastructure before the reservation is activated. Book-ahead reservations also provide a not-so-obvious feature: they allow users to validate testbed or resource descriptions by reserving them in the far future – the service will book everything but will not actually activate the testbed. This can reveal specification errors that might be related to resource constraints, topologies, scaling, authorization policy, etc. allowing the user to modify or correct the testbed description before it is actually required.

The Testbeds Service will construct a testbed by interconnecting resources according to the DSL description. But interestingly, GTS does not actually produce a "network" per se. GTS views networking within the testbed environment as an integral element of the researcher's experiment and so GTS prefers to leave this alone. However, the Service recognizes that even disruptive experimental testbeds still need to allow the researcher to access their resources. IP networking is the defacto means of doing so. Therefore, GTS v1.0 constructs a general purpose IP subnet for each GTS project. This subnet is assigned a dedicated VLAN and IP CIDR block. This subnet can be used by any/all resources reserved under that Project. The Testbeds Service configures a gateway router on this subnet, provides DHCP services, NAT'd access to the real world Internet, and provides VPN server so that users can access the subnet from outside. In v1.0, "Host" resource instances (VMs) assign their eth0 interface to this general purpose subnet. Other resources may also attach to this subnet depending on their nature.

A network File System is created for each Project as well. This file system resides on a GTS managed NAS server. This file system can be mounted by resources within the testbed – most notably the virtual machines. This persistent networked storage allows the user to save testbed state and be able to access and/or restore this state across testbed reservations. The file system can also be mounted externally to allow researchers to pre-position software or data into their testbed environment or to extract data or logs from their testbed.

IV. DEPLOYMENT STATUS

The GEANT Testbed Service version 1.0 is currently deployed in four European cities – Copenhagen, Amsterdam, Bratislava, and Ljubljana. Three additional locations - Paris, Hamburg, and Milan - are in progress and will be online in the fall of 2014. As users migrate to the GTS environment from the predecessor GEANT3 OpenFlow test facility, the infrastructure serving this facility will be absorbed into the GTS expanding the reach to include Frankfurt, Zagreb, and Viena. Indeed, as the service matures we expect to recreate the GOFF *within* GTS in order to simplify user migration.

GTS is establishing data plane connectivity to the US GENI project. The [initial] GTS demarcation point is planned to be at the MANLAN open exchange in New York City. GTS and the ExoGENI project have begun initial efforts to construct Testbeds across these two facilities.

The Service is currently in an introductory phase during which new users are invited to self register (*gts.geant.net*) and "kick the tires" by creating simple testbeds. Introductory users are constrained to small testbeds and short reservations. Users wishing to construct larger and more persistent testbeds should contact and register with the GTS Service Management.

V. THE GTS EVOLUTION ROADMAP

The development of GTS v1.0 defined the GTS Architecture and developed the initial key components and resources.

Version 1.1, due in fall 2014, will provide a number of background enhancements such as restart and migration tools, additional local storage for VMs, enhanced operational monitoring capabilities, and "lightweight" VM resources (to enable substantially more "basic best effort" VMs to be made available.)

Version 2.0 is targeted for deployment in early 2015-Q1. This release will deliver several key new capabilities: 1) "Multi-Domain" GTS capabilities. MD-GTS will allow a user's testbed to seamlessly span multiple GTS domains. 2) A much improved graphical user interface will provide drag-and-drop graphical testbed editing. The new GUI will let users graphically manage testbed topology, and configure or query attributes through interactive dynamic dialogs. 3) Active in-situ modification of testbeds will be available allowing resources to be added or removed without releasing and re-reserving the entire testbed. 4) New operational monitoring capabilities will provide more detailed insights into service state. These tools will be integrated into the GUI and will provide greater visibility and more detailed control of the service for the GEANT NOC and for users.

GTS 2.0 will introduce several additional resource classes including "Bare metal servers" that provide a user with an entire physical server as a resource, and VLAN-delineated virtual circuits which will enable multiple virtual circuits to be provisioned over an individual physical Ethernet interfaces. The latter, VLAN delineated VCs, will also introduce increased VC capacity to 10 Gbps [or possibly 100]. The GTS v2.0 Service will peer with the GEANT Bandwidth-on-Demand service and will be able to leverage the GEANT BoD connectivity to deliver virtual circuit resources globally. Global reach for virtual circuits are key to delivering Multi-Domain GTS testbeds. Version2.0 is planned to demonstrate an initial "GTS-to-Cloud" integrated service capability that is hoped will enable GTS to acquire VM resources from cloud data centers and to provision virtual circuit resources into those cloud facilities to realize user topologies.

The next stage, GTS version 3.0, will be developed as part of GEANT4 Phase 1 and will be begin development in 2015-Q2. Version 3 features are not fully defined as of this writing. However, some interesting new features are being discussed: "Smart" resources can react to events in the testbed – such as the addition or deletion of a resource instance somewhere in the topology. A smart resource could, for example, analyze and reconfigure the topology to meet user resiliency goals. "Soft" resources do not have a particular hardware analog. Soft resources might include encryption/decryption modules, or framing adaptors/interworking modules, or a BGP routing instance. There is interest in advanced timing capabilities as a virtual resource, and a new family of "optical/photonic" testbed resources to manage long haul optical/photonic infrastructure and make photonic testbeds facilities easily available to the research community. Similarly, there is interest in wireless and mobile resources. The final v3.0 feature set requires additional consideration which will also be weighed in terms of the growth of the user bases served as GEANT4 begins.

V. THE VISION

The long term vision is a ubiquitous and easily accessible service capability that can construct experimental networks spanning the globe. Recent research efforts have developed a number of promising approaches over the last several years. However, interoperability and global scaling were not the primary research objectives, and so these frameworks exhibit only limited (if any) interoperability. This is not atypical of early very advanced work in a field. The most expedient approach to interworking two dissimilar systems is to develop software translation, or *interworking*, of the two service models. Such translational approaches do not scale well when the interworking must address more than a very few different service models. Interworking often is just unable to convert service features from one model to the other – simulacrums do not exist. The results are often only partially successful, or require excessive manual intervention to deliver services. Even lack of common end-to-end operational continuity can make interoperability sketchy at best. A more strategic and formal approach is needed.

What is needed now is a *common canonical service model* that all agree supports the necessary basic service concept, and that all agree to support and adopt. Such a common model must identify the fundamental service semantics, normalize terminology, define principles for security, privacy, and scalability, identify common data objects/constructs, and agree to a set of common service primitives. Operational implementation of such a common service model requires agreement on policy – at least at a high level – such that administrative control is retained in the domains offering the service, and service interconnections and peering relations must be worked out.

Such a consensus service paradigm need not invalidate existing service models. Indeed, the common model will likely have a strong resemblance to these predecessors. But a canonical model means that existing service domains need only develop a single interworking process between the canonical model and their existing local model. Inter-domain interoperability is achieved using the common canonical model, while internally each domain can retain their existing intra-domain service model. The canonical model may also be implemented directly within a domain making the interworking unnecessary altogether.

Agreement on a common canonical model will take time. There will inevitably be many hard discussions as to which concepts should be implemented, how they should function, what terminology will be used, etc. This is best progressed in stages. Simple shared concepts first, then more sophisticated features. Time allows compromise, e.g. a topic that cannot gain sufficient support now can be held for later re-consideration thus avoiding alienation of the contributors and losing a potential supporter for the overall effort.

The GEANT Testbeds Service will continue to evolve to incorporate more useability features, to provide the reliability of a production service, and to integrate into a global consensus service model. Long term commitment to services such as GTS will entice users to incorporate these capabilities into their ongoing research efforts, and this increasing usage lead to wider adoption and deployment. The result will be a ubiquitous global facility for advanced networking and networked applications development.

Five SDN-oriented Directions in Information Security

A. Grusho, N. Grusho, E. Timonina
The Institute of Informatics Problems of the Russian Academy of Sciences
IPI RAS
Moscow, Russia

V. Piskovski
Non-profit Partnership «Applied Research Center for Computer Networks»
ARCCN
Moscow, Russia

***Abstract*—the concept of Software Defined Networks (SDN) originates new approaches in Information Security. The paper presents how to make use of SDN to guard hosts and networks. Also we consider newly opened perspectives in information security**

Keywords—Software Defined Networks; Information security; Artificial Intelligence; Cryptography

I. Introduction

The work has been granted by the Russian Foundation for Basic Research, project 13-01-00215.

Now access restriction methods mostly are based on the idea to isolate domain. There are such a kind of mechanisms built in OS and DBMS to provide them with discretionary access control. Switches, routers and firewalls implement domain isolation in networks. The concept of Software Defined Networks (SDN) opens new approaches in this area.

The paper presents how to make use of SDN to guard hosts and networks. Also we consider newly opened perspectives in information security.

SDN concept is to put network management functions into physically decoupled control plane. This enables a wide range of technical software and hardware means including artificial intelligence to enforce proactive guarding of selected nodes, LAN segments and IT infrastructure. Some of these means are as follows:

– SDN allows to implement an architecture to secure distributed virtual systems. We consider principles of this idea in next chapter. One of practical use of such a secure architecture is a realization of so called process approach in information security. In other words that is to use patterns and dependencies between data flows, which are predefined by IT. So there is a new chance to build security facilities based upon information processes.

– On-line tailoring security policies in Big Data and OLAP systems. Relations between data flows, its content and the subject can dynamically control an access to information and support multilevel security policy and mandatory access control.

– The use of computationally complex, heuristic algorithms along with AI[1] approaches, implementing a kind of "man-in-the-middle" systems enable on-line traffic analysis and preprocessing. This empowers withstanding to DDoS attacks and traffic anonymizing.

– Parallel processing along with obfuscating in order to store information in a number of isolated independent domains. This essentially complicates unauthorized access. The approach is a new direction of information security.

– Cryptography as a mean to authenticate and isolate domains. The problem resides in crypt algorithm implementations and key management system as well.

II. The Secure Architecture

First of all we define what we mean under the architecture of a distributed virtual system

If we denote V as a set of components belonged to this distributed virtual system, and E as a set of interactions between components, then $G = (V, E)$ represents the architecture of the system. It's also possible to consider time-dependent architecture as $G(t) = (V(t), E(t))$. We consider time-independent systems just for the sake of simplicity.

If we consider interactions E as communications in pairs then we can represent the architecture as an oriented graph where an edge identifies the data flow from a component to another one. Generally the interactions can be denoted as a complex architecture, which contains its own components with its own interactions in its turn. Similarly components can be also a combined structure of elements, which enclose other components with its interactions as well.

We define the concept of the secure architecture as follows.

Information security considers threats as a result of technical issues like vulnerability and exploits and social matters like classes of probable attackers and their potential targets. All of these issues

[1] Artificial Intelligence

We define on the set V a function f: $V \rightarrow L_1$. The function maps components into a scale L_1 defining the level of a risk for a component to suffer from an attack for example with malware. Here L_1 is a semiordered categorical hierarchy (a set) for determining of the risk (scalar value, i.e. a kind of a norm) mentioned above. Thus some components have got a lower risk, some ones map to a larger risk, e.g. components connected to Internet. Similarly we can introduce a function h: $E \rightarrow L_1$, which maps the set of interactions into categorical risk hierarchy L_1, e.g. a channel inside a protected perimeter has a lower risk to be eavesdropped then a channel through-passing an open unsecure territory.

Also we categorize all components according their values by means of a function g: $V \rightarrow L_2$, where L_2 is a hierarchy structure, reflecting component value from the perspective of confidentiality, integrity and availability. For instance components containing crypto keys or digital signatures are more valued then components with open background materials.

Therefore the function f, the norm of L_1 representing a scalar risk and a given threshold defines a subset V_1 of components in some distributed (and/or virtual) system. We can assume those components have got heightened risk of unsafe impact. Alike we described above the function g, a threshold applied to V, L_2 defines a subset V_2 of hosts or components containing valuable information needed to be protected. We determine the system architecture as secure when there are no direct interactions between V_1 and V_2 elements.

If there is a need in such interactions then we have to put an interface called SecS (Security Server)

The Security Server is both to lower a risk of hazard effect on valuable hosts and to prevent of rising value level for risky hosts.

Thus we've defined the model of a secure architecture for distributed (either virtual) system. This model can stand independently of other known concept in information security.

One of the central requirements to distributed or virtual information systems is quick modernization, developing and implementing of new information technologies. It means a need for quick upgrading architecture of such systems and it's possible that new components and requirements can be revealed. In order to keep the architecture secured it's necessary to calculate functions f and g for these components and to check out the requirement mentioned above to have no interactions between subsets V_1 and V_2.

One of solutions to this problem is to make use of SDN concept because it's much faster and efficiently to customize configurations programmatically other than tuning up of traditional routing systems.

Next, we deliberate problems of analyzing the secure architecture of systems considered in this chapter. There is too early to treat, say, some deliberated architecture as absolutely or assured secure one, but it's worth to compare several architectures as more secure or less secure. Also it's to build more secure systems from less secure.

As an example of systems with different levels of security we appeal to a PC with Hyper-V processor feature. Also we assume the PC runs under a hypervisor with its manager and two virtual machine with OS's working under this hypervisor. And a user can switch between those two OS's as he (or she) wants.

Then we assume, that a virtual machine (VM I) has connected with Internet, and the other (VM II) is not allowed to connect to Internet. A user can work with his confidential data on a VM II and has to switch to VM I in order to send these packages. Here we can state that just described architecture is more secure in comparison of situation when both virtual machines have been accessed to Internet.

On like occasions of traditional architectures it's worthwhile to build safety architectural elements. For instance, when either a component with high risk and a component with high value access to common resources it's worth to provide every component with its own "common" resource with previously doubled information. Also these resources have to be placed into isolated domains along with its components. Correspondingly "common" resources have to synchronize data regularly.

Enterprise information systems automate technological and/or business procedures in part or entirely. One of the first stages at implementing a system is a business analysis and modelling. This has data flow formalism as an artifact, describing traffic between servers, nodes, system components and workplaces. We can consider a business model as a number of graphs with vertices as information processors and edges as data flows. That means we consider an architecture which depends on time. Thus we can describe network security model as a domain of legal traffic at every moment of business processes. We treat legal traffic in general irrespective of professional area like industrial control systems or an office infrastructure. This security model along with legal traffic domain is being adjusted during implementing procedures.

Having efficient high performance technical means placed in the control plane to analyze and control traffic of SDN we can manage on-the-fly alternating data flows in accordance with dynamic security policy and system informational model. This reflects the sense of process approach in information security methodology.

III. Information Security in OLAP and Big Data

Enterprises have rigid requirements to guard confidential data when users work with analytical systems such as OLAP or Big Data. Very often these requirements imply the discretionary access policy for a certain type of aggregated data and monitoring of access policy fulfillment in real-time. It means a user or a software agent can have whether a full access to the whole data source (e.g. hypercube) or nothing. Populating a control plane with special technical means like DPI along with AI technology allows us to realize flexible security policy. This policy can on-line combine current data confidentiality, users' credentials and his activity profile.

Let’s assume a user is collecting accessible structured (OLAP) and unstructured (Big Data) data from corporative sources. All facts and information fragments have markers, i.e. structures of information properties including confidential level. Aggregation procedures recalculate markers according to predefined rules. Also logged user (or program agent) activity results into operational recalculating of user’s credentials, which defines first of all his access level. Thus applications located in the control plane constantly recalculate information markers and users credentials in order to limit or grant an access on-line matching markers and user credentials.

IV. Web-traffic content analysis

As common practice shows the usage of web applications and web-services can contain a lot of security threats. One of methods to withstand is web-traffic analysis. To implement it we have to solve two problems:

– To get an access to the object of analysis, i.e. data flow from a web-service to a client, e.g. by “men-in-the-middle” method, realized via any available way.

– To analyze data flow and make a decision on its content during the period defined by technological and business requirements. This problem needs effective fast algorithms and heuristic approaches accompanied with elements of AI technique.

Both problems can be solved as SDN controller applications with proper functionality and performance. Realization of such an approach allows resisting DDoS attacks and anonymity in internet sessions.

Such a kind of solutions regarding content-analysis shows up in firewall products of new generation and anti-bot systems

V. SDN as a Mechanism to Guard

When attacking computer systems with help of malware the toughest problems are to get an access to protected information and its processing. To create isolated domains of storing and processing data is a solution to defend systems against these attacks. For instance, modern OS process data in an isolated domain such as the kernel of OS.

Nowadays techniques of data storage and virtualization provide a number of means of distributed storing and processing. Applying of SDN extends mentioned above solution with an ability to decouple a way to define storing and processing from storing and processing itself. Such a kind of detachment opens a perspective to develop new mechanisms to defend against malware attacks.

Let’s consider the essence of this idea in details. Technically speaking SDN arranges a dynamically isolated domain for any task to link nodes. If this domain contains no built-in malware then an attacker needs to look over all components for valuable data. If this dynamically isolated domain contains a lot of components (irrespective they are virtual or real) of storing and processing data and malware has no access to SDN controller managing the domain then malware has to perform an exhaustive search. Nowadays malware can’t perform such a kind of search and it’s hardly possible that malware could do it in any future perspective.

From another side, if a malware is already running in SDN controller but it has not connected with malware of data plane, then it also can’t arrange a dedicated, object-oriented search.

For example, we have to protect a text of a length L. We can divide the text into sequential fragments of 512 bytes long, what are about a hundred words in Russian. Such a long fragments length can ensure a context search within bulk of data, but there are few chances for a deep inspections and data mining if the text is comparatively long.

Our suggestion is (i) to furnish every fragment with two cryptographically protected marks; (ii) to store crypto keys for these marks in control plane; (iii) to place fragments into randomly chosen memory slices. The first mark contains encrypted information about the address of previous fragment; the second mark contains the address of next one. Thus we can obfuscate all fragments and leave a possibility to look for information via open parts of fragments at the same time.

For to retrieve obfuscated text as a whole a user has to get through an authorization at SDN controller and only then to place his request to obtain the text. In response a controller can swiftly restore a full text as a linked list according to requested fragment. From another hand malicious code or an unauthorized user have to complete an exhaustive search through whole memory storage with no guarantee to restore original information.

As a result SDN can arrange storing information irrespective of computer systems which actually keep data, i.e. storing data system cannot effect on how and when to store either it cannot gather up the thread of obfuscated text. Also malicious code in control plane can’t help malicious code in data plane to reduce an exhaustive search as long as they are not connected with each other.

Thus we’ve got a solution to a long-standing problem to build guaranteed secure system with unsecure, distrusted components. Such a type of protection is a new direction in information security.

VI. SDN and Cryptography

An enhancement of computing and intellectual power to process traffic by means of control plane facilities increases an efficiency of cryptographic security. This is grounded on two ideas. The first idea is that a secure usage of cryptography depends on the quality of isolating domain with cryptographic functions running. As mentioned above such isolation can be achieved by the instrumentality of SDN even in a case of components with built-in malware. The second idea regards with a need to carry out bulk computations at instantiating cryptographic protection of high quality. Applying specialized high performance technical means in control plane allows meeting the requirement of using dissimilar crypto keys in different data flows, generating these keys with usage of compound protocols of key sharing and fault protecting crypto systems.

VII. Conclusion

SDN initiates new perspectives in the area of information security. The possibility to populate a control plane with high performance intelligent facilities opens new directions in managing data flows and developing adaptive algorithms to protect computer systems.

References

1. Martin Casado, Tal Garfinkel, Aditya Akella, Michael J. Freedman Dan Boneh, Nick McKeown, Scott Shenker. SANE: A Protection Architecture for Enterprise Networks // 15-th Usenix Security Symposium, Vancouver, Canada, August 2006.
2. Ruslan Smelyanski. Software Defined Networks// «Open Systems», № 09, 2012 (in Russian)
3. Check Point extends the functionality of products. http://www.securitylab.ru/blog/company/besecure/30615.php (in Russian)
4. A. Grusho, Ed. Primenko, E. Timonina. Theoretical Basis of Computer security, Academia, Moscow, 2009 (in Russian).
5. Grusho A.A., Grusho N.A., Timonina E.E. Information security methods from attacks with the help of covert channels and hardware-software agents in distributed systems // RGGU Bulletin. Scientific journal: "Information science. Information security. Mathematics" Series, 2009. – 10. – pp. 33-45 (in Russian).
6. Kaspersky DDoS Prevention.

   http://www.kaspersky.ru/products/business/services/ddos
7. Norton AntiBot.

   http://en.wikipedia.org/wiki/Norton_AntiBot
8. Timonina, E.E. The analysis of threats of the covert channels and methods of creation of guaranteed protected distributed automated systems // D.Sc. Dissertation, 2004 (in Russian).

Towards Distributed Hierarchical SDN Control Plane

A. Koshibe, A. Baid and I. Seskar
WINLAB, Rutgers University
671 Rt. 1 South, North Brunswick, NJ, USA
{akoshibe,baid, seskar}@winlab.rutgers.edu

***Abstract*—This paper presents a concept for hierarchical distributed control of SDN networks. The proposed architecture is based on a heterogeneous control plane with a hierarchical structure that offers a general framework for building non-classical SDN deployments. This control plane organizes a group of controllers nodes, into a hierarchy, with each tier containing one or more interconnected controllers. The hierarchical approach is intended to improve scalability and increase service flexibility by distributing functionality between multiple controllers. A proof-of-concept implementation using the Floodlight SDN controller platform is described, and performance results demonstrating basic feasibility are given.**

***Keywords*—Software Defined Networking (SDN), Next generation networks, control plane , distributed controller**

I. Introduction

The architecture of the Software Defined Networking (SDN) control plane has been a point of interest since its beginnings in OpenFlow, with the reference design [1] featuring a single physical controller. However, to manage larger or more disparate networks, a single controller can quickly become a point of failure, route inefficiency, and processing bottleneck. The ideas on distributed implementation of the control plane followed naturally in the SDN community to tackle these problems. This is typically achieved through the distribution of the controller across multiple compute resources as clones sharing a synchronized view of the network state. Figure 1 shows the organization of such a logically centralized, distributed control plane. Distributed control platforms, such as [2, 3, 4, 5, 6], often follow this archetype to scale to larger networks and traffic loads and/or to reduce flow setup latencies by reducing the distance between any given switch and the closest available controller. Most early efforts on the distributed control plane architectures have gone towards a functionally homogeneous layer of controllers. Every controller in such an architecture performs the same set of tasks, although for different parts of the network, or in different fail-safe conditions. A functional distribution of the control plane tasks, in addition to the topological distribution, is arguably a harder problem and the subject of this paper.

Due to tighter delay bounds and rapidly changing network conditions, the single vs. distributed control plane debate is even more important for wireless networks. Efficiently operating a large wireless network requires the adaptation of numerous parameters (e.g., channel assignment, data transmission rate, client association, and transmit power) to network conditions. While the algorithms controlling some of the parameters are run directly on the APs, others are run on centralized controllers. Yet other functionalities such as policy management, authentication, and security needs to be implemented at the junction of the wired and wireless networks in order to ensure the same set of rules are used throughout the network. In such situations a single controller or a homogeneous set of controllers with an identical set of functionalities is clearly inefficient.

For this deployment scenario and several others that we expand upon in the next section, a functionally distributed control plane is a better fitting solution. In this paper we propose a heterogeneous SDN control plane with multi-tier hierarchical structure through which different control plane

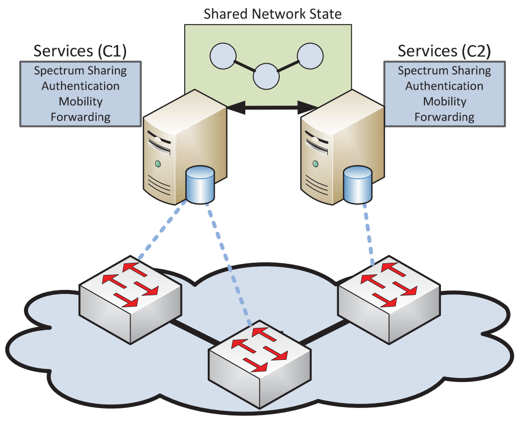


Figure 1: Distributed SDN Control Plane

functions can be distributed between separate physical controllers. This architecture is lean and efficient in terms of the overhead of distribution and transparent to the underlying data plane.

II. Hierarchical Heterogeneous Control Plane

Current SDN control planes were not designed for network deployments that do not uniformly centralize their control logic. A network stack, run in one or more controllers as a set of applications, typically assumes that it is the sole source of network control in a deployment. Several situations break down this assumption, including:

• *Shared networks*: Multiple administrative groups may configure various aspects of a service-rich network. Particularly, in active infrastructure sharing, multiple service providers share network elements such as switching, routing, and other telecommunications equipment, or, in case of wireless, shared access to spectrum for unlicensed bands, and individually provide support to their customer bases. Notably, there is interest for these providers to allow their customers (or controllers) to interact with one another to increase service utilization. In these cases, each administrative group or service provider may wish to maintain their own controllers, and in the latter case, have them coordinate with the global controller for the shared infrastructure.

• *Heterogeneous networks*: Modern networks are built with a mixture of technologies with various properties and feature sets that must coordinate seamlessly. For example, typical LANs combine wired and wireless components, which frequently require different traffic handling and host admission schemes. In another vein, service providers with adjacent infrastructures may wish to negotiate the handling of each other’s customers, e.g. different charging policies for roaming or negotiating traffic routing. Both cases involve potentially disparate service stacks that interact as collaborative peers.

• *Large-scale global policies* applying to multiple spheres of influence: large internetworks, even when managed by a single organization, will often be administered in pieces. Additionally, these local administrative domains may also implement their own local policies and services. This is seen in campus area networks, in which host authentication may be controlled by a set of global policies that span across multiple (W)LANs administered on a per-department/facility basis.

In order to address these issues, we propose that a heterogeneous control plane with a hierarchical structure offers a general framework for building these non-classical SDN deployments. This control plane organizes a group of controllers, or nodes, into a hierarchy, with each tier containing one or more possibly interconnected (peer) controllers. The nodes of each tier behave as clients to the nodes in the tier above, and servers to the nodes in the tiers below. The client-server relationship stems from the notion of a service process chain, in which network events are ferried from low to high tiers. Events are processed incrementally at each hop, implementing network functions as they are handed off across the controllers. The nodes within the same tier host services that serve the same function or share operational scope; in this respect, a single tier of a heterogeneous control plane is roughly analogous to the classic distributed control plane. Figure 2 illustrates this with a simple network stack of three tiers and three categories of services, A, B, and C, and two peer links. The client and server relations are labeled with respect to node B1’s perspective. We point out that the service process chain corresponds to a network stack, and each service

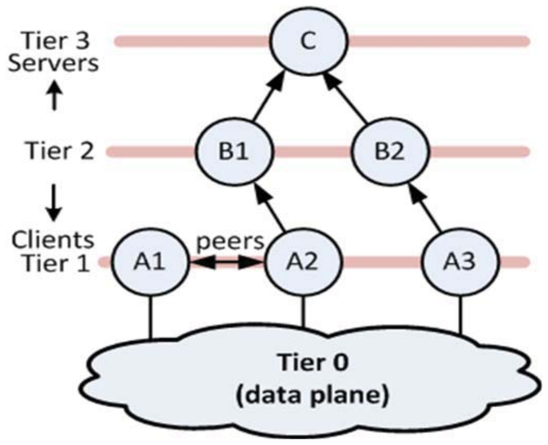


Figure 2: A generic representation of a hierarchical, heterogeneous SDN control plane.

category is its subcomponent; that is, the full set of applications within a typical controller is dispersed across hierarchically arranged controllers.

The ability to break network functions (service set) down into modular components introduces flexibility in terms of what functions are being used to handle a given event. The event process chain enables the control plane to 1) host functionalities from multiple network stacks without the need to rely on assistive tools such as hypervisors [7], and 2) allow services to share their functionalities with other interested parties, including those of other service sets. For 1), event process chains may exist side-by-side as higher tier services invoked on disjoint sets of events. For 2), events may be handled by process chains that incorporate nodes from separate service sets that reside in a hierarchy organized to merge the sets (i.e. different administrative domains, etc.). In other words, 2) enables the coordination of event handling between otherwise functionally disjoint control planes.

There are several challenges in the design of such hierarchical control plane architecture including the fact that the distribution of functionality between different control plane elements has to be relatively transparent to data plane devices.

III. Related Work

Architectures in which varying controllers coexist on a single network have been rarely addressed outside of the topic of network virtualization. However, we acknowledge that the design of this control plane is influenced by several prior works.

FlowVisor [7] is a specialized controller that serves as a virtualization layer between the data plane and multiple, functionally different controllers by overwriting control message contents to present a controlled portion of network resources to each controller. Save for specifically configured mirror slices with read-only access to other slices, controllers behind a FlowVisor are largely unaware of one another.

Kandoo [8] is a hierarchical control plane that separates its service set into two tiers. Although closest to proposed architecture, Kandoo does not allow controllers within a tier to communicate directly with one another, and limits usage of the second tier to services requiring a global network view.

Onix [9] is capable of limited federated operation, in which two Onix instances may share summarized views of the networks that they have control over. This sharing of information is contained to instances under the same authority, and serves to allow the compact representation of massive data planes within the Onix NIB.

IV. Control Plane Implementation

There are several implications to a hierarchical design. The control plane must have a messaging scheme that member nodes can use to pass events amongst themselves. The scheme must also allow nodes to propagate service information that would allow others to determine available event handlers and the means to reach them. In addition, it is assumed that service sets can be broken down into functional subsets according to some policy.

We recognize the control plane to have three distinct layers:

• *control channel handler* : The control channel handler interfaces the control channel, and is directly responsible for sending and receiving network control packets and listening for incoming connections from the data plane. We focus on OpenFlow as a widely-supported control protocol.

• *event dispatcher* : The event dispatcher translates between control channel messages and controller-internal events meaningful within the controller and to its services, and serves as an event dispatch/scheduling mechanism that passes events to various interested services.

• *applications* : Applications implement the various services that add functionality and usability to the controller. These functions range from network stack functions such as topology mapping and packet forwarding to interfaces such as RESTful APIs. Applications may also provide specialized functions such as synchronization elements of distributed controllers.

In the following, we assume that our control plane would be given a dedicated control network physically separate from the data plane. This allows us to avoid the bootstrapping issues associated with in-band control. We also assume that state distribution can be performed through synchronization mechanisms well-explored by logically-centralized distributed control planes. This allows us to focus on the components and mechanisms that provide the features necessary for implementing the heterogeneous hierarchical control plane.

A. Inter-controller (Control-plane-level) functions

Each node in the control plane is identified by three attributes:

1. A unique node identifier (UUID), a value assigned to each node to serve as an address in the control plane
2. A service identifier (SID), a value assigned to the services hosted at a node
3. Event subscriptions, the set of network events that a service is capable of handling and that a server will subscribe to a client for.

1) Controller Initialization and Discovery

The nodes participating in the control plane must be able to find one another in order to form the hierarchy. Our approach mirrors that of the OpenFlow control channel. Upon startup, a

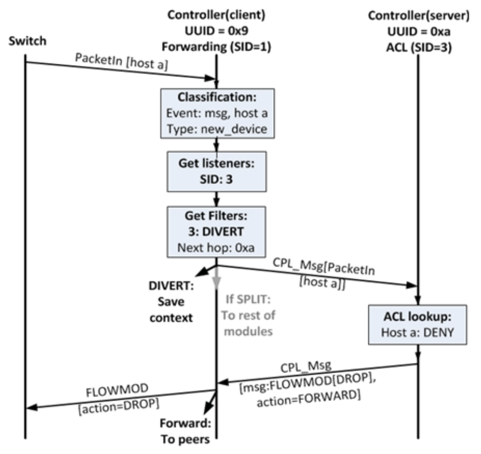


Figure 3: A timing diagram showing the handling of a message event across a two-tiered, two-service event process chain.

server begins listening for client connections on a pre-established port, with clients periodically attempting to connect to them. For the purpose of simplicity, our implementation of the client is supplied with a list of servers to which it must attempt to connect to. The client attempts a limited number of reconnects if its server is initially unavailable. A successful connection attempt is followed by a handshake, in which the server informs the client of its event subscriptions. Peer links are bidirectional, containing two links connecting both ends as clients of the other. Controllers may make use of services similar to portmap or UPnP to facilitate more sophisticated forms of discovery.

2) Service propagation

Each node must advertise the services that they provide, so that others may subscribe to them. This information is propagated in configuration messages that encode a controller's attributes and location within the hierarchy. For the sake of simplicity, we implement a simple RIP-like distance vector algorithm with split horizon [10] which uses hop count as the distance metric. This allows recipients of these messages to build maps of service locations within the control plane in a structure similar to a route table.

B. Context preservation

Once the control plane enters the operating state, a node's ability to preserve the context of an event being handled across the multiple hops in an event process chain becomes important. In situations where client nodes rely on results generated by nodes later in the process chain, each server in the process chain must be able to identify the client that it had received the event from, so that it may return the message to the correct client. We simply rely on the service map to route the messages back down the process chain.

C. *Event process chain execution*

In the initialization of client-server connections, the client's local packet process chain is essentially configured by its servers so as to incorporate, and properly execute, their services. A client node's service map contains mappings between services and server event subscriptions. An OpenFlow message received at a first-tier node is translated into an event, which is then used to search its map for matching entries. Matches return one or more SIDs that can be used to determine the next node in the process chain, to which the event should be dispatched. The current implementation attempts to handle an event locally if no match is found. A more sophisticated action may follow with a discovery process for servers capable of handling the event. In addition to SIDs, we define a set of actions that a client should take when it finds a match:

- DENY: do not process the event further, returning a DROP FlowMod to the sender switch if necessary.
- ALLOW: handle with local packet process chain, bypassing other event subscriptions associated with the event
- DIVERT: dispatch the event to the service and halt the process chain until the service returns a response
- SPLIT: dispatch the event to the service, and continue process chain execution

One or more of these directives are assigned by the server to its event subscriptions. Given the example of the two-tiered control plane in Figure 2, a PacketIn triggered by a new device's traffic will prompt the execution of a process chain containing both forwarding (client) and authentication (server) nodes. In this situation, authentication must first determine whether or not the host is allowed on the network before its traffic can be handled by the forwarding service. The configuration messages sent by the authentication module

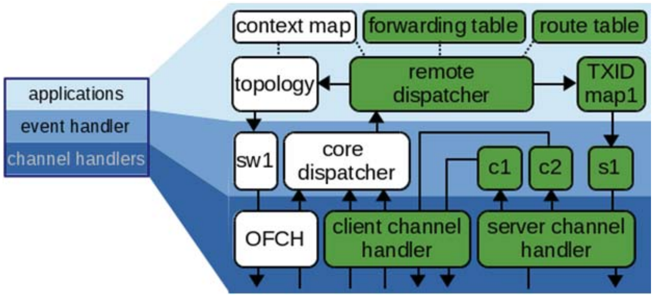


Figure 4: A Hierarchical Controller with Floodlight.

pairs message events, associated with new devices on the network, with the DIVERT action, causing the forwarding node to wait on the server to allow or deny the processing of the host's traffic. Figure 3 illustrates this flow of execution for case of an unauthorized host. The event reaches the tier 1 node, where it matches against the subscriptions of the authentication service, found to reside on the server with the UUID of 0xa. The client blocks on the DIVERT directive, waiting for the server response - in this case, a FlowMod to drop traffic associated with the host. The result, returned via the client node, is forwarded to the client's peers as a rule to apply throughout the network.

We develop the components of our controller as a series of modules and services on the Floodlight [11] SDN platform. Figure 4 shows the layered model of the architecture of our modified version of Floodlight, with our additions shown in green. Our implementation complies with the Floodlight API and is fully compatible with the base platform, and can be treated like any other Floodlight application.

V. Performance and Functional Evaluation

This section presents some of the performance evaluations that were done to test the functionality of distributed hierarchical, heterogeneous control plane. We run our tests on the ORBIT [12,13] network testbed. The nodes used for the evaluations have 8GB of RAM and Core i7 CPUs. Each node has two network interfaces, connected to dedicated VLANs that can be used to separate control and data plane traffic. Each interface connects to an aggregation switch with gigabit Ethernet links.
We consider several factors when evaluating the performance of proposed distributed hierarchical control plane:

1. Number of switches. Control packets and channels are tracked in terms of switches and client nodes. As this value increases, a controller will have larger amounts of transaction and context mappings to maintain if it needs higher tiers to handle the events.
2. Number of hosts. Unique host traffic that generates misses at the switch flow table trigger message events. More hosts intuitively generate more events, as do hosts communicating across multiple datapaths.
3. Number of control plane hops. Even with high-throughput, low-latency links, a longer route to a particular service will incur processing delays in the form of network overhead (propagation, queueing, kernel buffer etc).
4. Number of services. Larger numbers of services (modules) handling a given event naturally incurs more processing overhead.
5. Listener policy. Event traffic volume above tier 1 in the control plane is directly related to listener subscriptions; a listen-to-any policy will dispatch every event from the data plane, whereas more stringent policies will dispatch events less frequently. Listeners may also request to divert process chains, in which case the dispatcher must wait for the listener reply before continuing to process an event.

A. *Overhead Analysis*

The most prominent feature of this control plane is its hierarchical control network and we measure the effect of control plane complexity on control packet processing times, focusing on the impact of route hop count and service subscriptions with DIVERT versus SPLIT directives.
Using a custom OpenFlow client, a stream of PacketIns is injected into the control plane. The time between PacketIn transmission and the reception of the corresponding PacketOut

were measured at tier 1 using Tcpdump, and at the client using its logging functions. Two control plane topologies with three controller configurations were tested.

For the first, a variable-length chain of controllers associated with each other as tiers was configured so that only the highest tier hosted the PacketOut service, and the rest escalated PacketIns to the service. In the second, a two-tiered topology

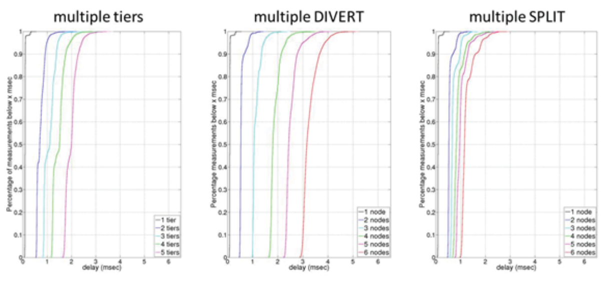


Figure 5: Processing time distributions for multiple tier, DIVERT, and SPLIT subscriptions compared with baseline (1 node/1 tier) controller.

.

with a variable numbers of servers in tier 2 was configured with servers that requested Divert directives in one test, and Split directives in another. For the second topology, each server was assigned a different priority to fix the dispatch order across multiple trials, with the last server in line hosting the PacketOut service and the rest, echo services that echo back a clone of a received PacketIn. For each of the three controller cases, ten trials of 10,000 PacketIns each were conducted for increasing tier height for the first topology, and increasing fanout for the second. Figure 5 shows the CDFs for the observed processing times of the three cases. The tests subject the control plane to the worst-case scenario where the higher tiers request escalation of every event.Discounting link delays, each additional tier adds an average of 0.32 ms of overhead. Similarly, each blocking and nonblocking server adds approximately 0.46 and 0.11 ms, respectively. The first two cases are similar, as the client must wait for the server to reply before taking action, with the differences in value due to each hop in the blocking server case handling the event with its modules.

VI. Conclusion

In this paper, we have presented a concept for hierarchical distributed control of SDN networks. The proposed architecture has the potential to improve scalability and increase service flexibility by distributing functionality between multiple controllers organized in a hierarchy. A proof-of-concept implementation was developed using a Floodlight SDN controller platform, and the results demonstrate feasibility for an intra-domain usage scenario. Future work will focus on extensions to the control plane necessary for interactions between SDN controllers across multiple administrative domains.

References

[1] *OpenFlow Downloads - OpenFlow Switching Reference System.* http://www.openflow.org/wp/downloads/

[2] *A. Tootoonchian and Y. Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. in INM/WREN'10 Proceedings of the 2010 internet network management conference on Research on enterprise networking, p. 3, Apr. 2010.*

[3] *T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. in OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation, pp. 1-6, Oct. 2010.*

[4] *H. Shimonishi, S. Ishii, Y. Chiba, T. Koide, M. Takahashi, Y. Takamiya, and L. Sun. Helios: Fully Distributed OpenFlow Controller Platform. 9th GENI Engineering Conference (GEC9) Demo, Nov. 2010.*

[5] *J. Stringer, Q. Fu, C. Lorier, R. Nelson, and C. Rothenberg. Cardigan: Deploying a Distributed Routing Fabric. HotSDN'13 Proceedings of the second workshop on Hot Topics in Software Defined Networking, Aug. 2013.*

[6] *U. Krishnaswamy, P. Berde, J. Hart, M. Kobayashi, P. Radoslavov, T. Lindberg, R. Sverdlov, S. Zhang, W. Snow, and G. Parulkar. Open Network Operating System: An Experimental Open-Source Distributed SDN OS.* http://www.slideshare.net/umeshkrishnaswamy/open-network-operating-system

[7] *R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, Oct. 2009.*

[8] *S. Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. in HotSDN'12 Proceedings of the first workshop on Hot topics in software defined networks, pp. 19-24, Aug. 2012.*

[9] *T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. in OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation, pp. 1-6, Oct. 2010.*

[10] C. Kozierok. *TCP/IP Guide - A Comprehensive, Illustrated Internet Protocols Reference*, Nostarch Press, Inc., San Francisco, Oct. 2005, Chap. 23.

[11] *Floodlight OpenFlow Controller* - http://floodlight.openflowhub.org/

[12] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh:*Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols*. Proceedings of the IEEE WCNC 2005, New Orleans, USA.

[13] *ORBIT Testbed* – http://www.orbit-lab.org

Selforganizing Cloud Platform

V. Kostenko, A. Plakunov
Moscow State University
Faculty of Computational Mathematics and Cybernetics
Moscow, Russia
kost@cs.msu.su, artacc@lvk.cs.msu.su

A. Nikolaev, V. Tabolin, R. Smeliansky, M. Shakhova
Applied Research Center For Computer Networks
Moscow, Russia
anikolaev@arccn.ru, vtabolin@arccn.ru, smel@cs.msu.su, mshakhova@arccn.ru

***Abstract*— Selforganizing cloud platform (SOC) to deploy virtual networks in DC is presented. The platform supports both IaaS mode and PaaS mode. The paper describes Platform architecture, virtual infrastructure description language, the comparison of the platform with known ones.**

Keywords—data center; virtualization; cloud computing; OpenStack; NFV

I. Introduction

In this paper we consider a selforganizing cloud platform (SOC) which allows one to deploy virtual networks to be used for a data center both in Infrastructure-as-a-Service (IaaS) mode and in Platform-as-a-service (PaaS) mode [1]. In order to provide efficient work of a data center in these modes and ensure guaranteed levels of service specified in a service level agreement (SLA) the platform must conform the following requirements:

1. Consistently distribute compute, storage and network resources; consider these resources as manageable. Scheduling should be performed consistently in terms of SLA compliance;
2. Allow virtual resource migration in order to eliminate segmentation of physical resources which occurs in the process of data center operation. Administer local network: for example, specify certain routing policy for data flows in virtual networks;
3. Allow the users to define and use virtual network functions (VNF) for purpose of virtual network organization, including VNFs which are managed by external providers as well as VNFs which are managed by one cloud user for another one. Allow to join networks managed by different users and organizations and efficiently transfer data between these networks.

Existing commercial cloud platforms [2-8] and OpenStack platform [9] do not possess all these properties in the aggregate. Also, none of the known algorithms [10-23] that map requests to physical data center resources meet the requirements 1-2.

In this paper we propose a cloud platform for a data center. This platform meets all of the above listed requirements and is compliant with and based on OpenStack.

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60714X0070.

II. SOC Platform Architecture

SOC platform uses some components of OpenStack (Nova, Cinder, Keystone, Rabbit Message Queue) in combination with the original specialized components: OpenFlow controller, orchestrator, unified scheduler for consistent resource allocation, graphical user interface (GUI) for network definition, an extensive “sensor” system for physical resources monitoring and management, and modified OpenStack component Neutron. SOC platform architecture is given on Figure 1.

Orchestrator is the central element of the platform. It controls all other components and coordinates their interaction. Orchestrator generates databases of physical resources which keep track of actual load of the resources, accepts requests for new virtual networks creation, for modification and deletion of existing virtual networks. As soon as all data about actual state of the data center are collected orchestrator launches a scheduler. Based on the results provided by the scheduler the orchestrator performs control function by means of OpenStack components API and OpenFlow controller API. Orchestrator uses Rabbit Message Queue to interact with the OpenStack components.

In our model Network, Compute and Storage nodes are considered as roles. These roles are assigned to physical servers in accordance with their hardware resources. It is possible to assign several roles to one single physical server.

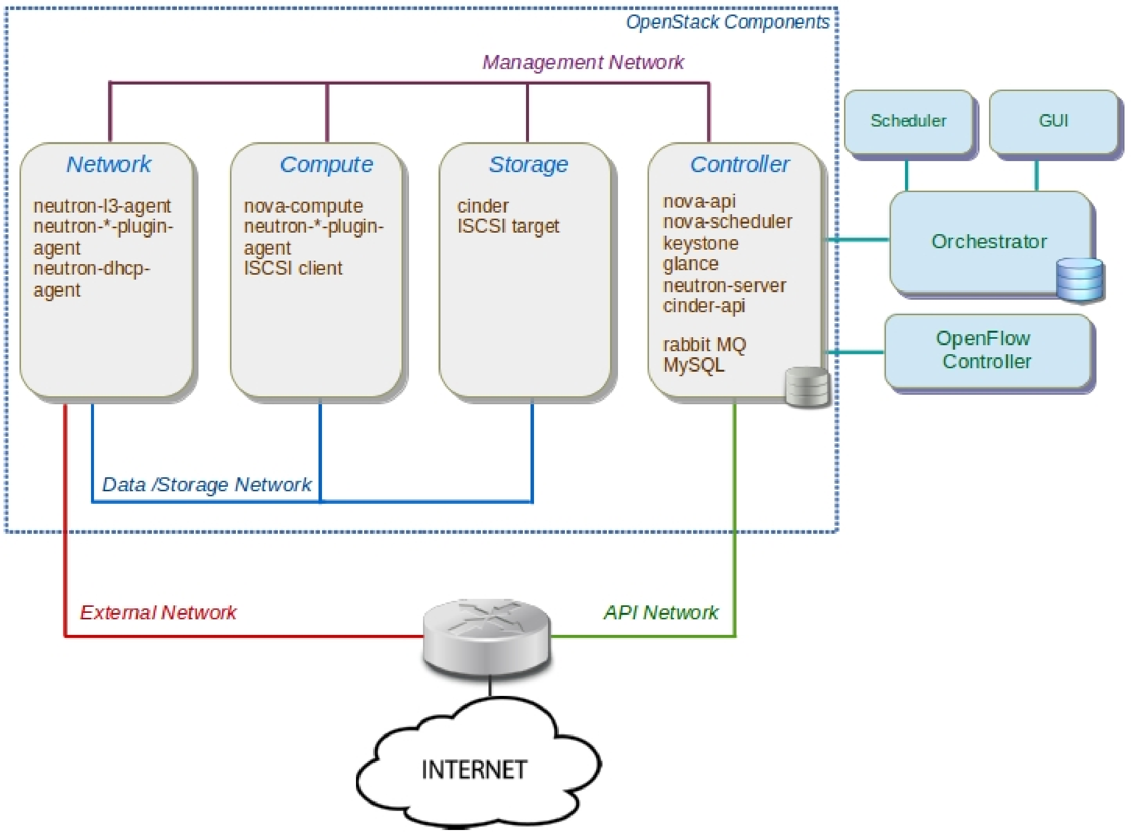
Network virtualization in the proposed platform has several distinguished characteristics:

- Network owner can arbitrarily define virtual network topology;
- It is possible to use VNF as a network element. In particular, it is possible to provide VNF which is designed and supported in one user network and is used in another user network;
- A user can specify SLA for virtual commutation elements, for example, virtual channel capacity, as well as for virtual servers, virtual storage elements, or VNF elements.

OpenFlow controller integrated into the SOC platform plays an important role both in virtual network organization and in sensor system by providing actual data about physical connections load and distribution of this load between virtual networks. Also controller provides centralized control of channel aggregation [24] for servers connectivity which ensures fault tolerance level not worse than for LACP protocol. This leads to the decreasing of the network infrastructure management costs.

Information about physical resources, virtual network description and mapping of virtual resources on physical resources is stored in SQL database. The database is used by the orchestrator and the scheduler. In order to describe virtual network we develop OVF-like data format language [25]. Virtual network description can be provided either by means of graphical tool (GUI) or by special file in the format described above.

FIGURE 1. CLOUD PLATFORM ARCHITECTURE



III. RESOURCE ALLOCATION SCHEDULER

Mathematical formalization of the resource mapping problem and solution algorithms which are used in the scheduler are described in details in [26,27,28]. Below we provide summary of these works.

A model of physical resources of the data center is represented by a graph

$$H=(P\cup M\cup K,L),$$

where P is a set of compute nodes, M – set of data storages, K – set of commutation elements of data center's network, L – set of physical data links. We define vector functions on sets P, M, K and L. Values of these functions are characteristics of corresponding compute node, data storage, commutation element or data link:

$$(ph_1, ph_2,\dots, ph_{n1}) = fph(p), p\in P,$$

$$(mh_1, mh_2,\dots, mh_{n2}) = fmh(m), m\in M,$$

$$(kh_1, kh_2,\dots, kh_{n3}) = fkh(k), k\in K,$$

$$(lh_1, lh_2,\dots, lh_{n4}) = flh(l), l\in L.$$

Detailed description of these characteristics is provided in Chapter IV.

A request for virtual network creation is defined by a graph

$$G=(W\cup S,E),$$

where W is set of virtual machines used by applications, S – set of virtual data storages (storage elements), E – set of virtual data links between virtual machines and storage elements of the request. Commutation elements of the virtual network are considered as virtual machines. We define vector functions on sets *W, S* and *E*. Values of these functions are characteristics (required SLA) of corresponding virtual machines, storage elements, or virtual links:

$$(wg_1, wg_2, \ldots, wg_{n1}) = fwg(w), w \in W ,$$

$$(sg_1, sg_2, \ldots, sg_{n2}) = fsg(s), s \in S ,$$

$$(eg_1, eg_2, \ldots, eg_{n4}) = feg(e), e \in E .$$

SLA characteristics of an element of request match characteristics of corresponding physical resource (i. e. the physical resource on which this element is assigned).

We define *a request assignment* as a mapping:

$$A: G \to H = \{W \to P, S \to M, E \to \{K, L\}\} .$$

Let us distinguish three types of relations between characteristics of request and corresponding characteristics of physical resource. Denote by x_i a request characteristic number *i* and y_j as corresponding characteristic of physical resource *j*. Then these constraints can be written as follows:

1. Impossibility of physical resource overload:

$$\sum_{i \in R_j} x_i \leq y_j ,$$

here R_j is set of requests, elements of which are assigned on physical resource *j*, x_i – corresponding characteristic of the scheduled request. As an example of such resources, we can consider number of cores, RAM, channel capacity.

2. Correspondence between the type of requested resource and the type of physical resource:

$x_i = y_j$.

In this case we consider qualitative characteristics, for example type of operating system or type of CPU.

3. Availability of requested characteristics of the physical resource:

$$x_i \leq y_j .$$

As an example, we can consider core frequency or cache memory (i. e. technical characteristics).

As a characteristics of the requests and physical resources, SLA criteria described in Chapter IV.

Let us call mapping

$$A: G \to H = \{W \to P, S \to M, E \to \{K, L\}\}$$

to be correct, if for all physical resources and their characteristics corresponding condition from 1-3 is fulfilled.

Residual graph of the available resources is the graph H_{res}, for which we redefine values of functions on the characteristics to fulfill relation 1:

$$fph_{res}(p) = fph(p) - \sum_{w \in W_p} fwg(w) ,$$

$$fmh_{res}(m) = fmh(m) - \sum_{s \in S_m} fsg(s) ,$$

$$fkh_{res}(k) = fkh(k) - \sum_{e \in E_K} feg(e) ,$$

$$flh_{res}(l) = flh(l) - \sum_{e \in E_l} feg(e)$$

Here W_p – set of virtual machines scheduled for compute node *p*, E_l – set of virtual links mapped to physical link *l*, E_k – set of virtual links passing through commutation element *k*, S_m – set of storage elements allocated on data storage *m*.

Let us define the *input data* for the request allocation problem:

1. Set of requests $Z = \{Gi\}$. Set $\{G_i\}$ is formed by the orchestrator. This set can contain both new requests and requests which are in progress and for which migration is allowed. Also orchestrator defines the time when scheduler starts.
2. Residual graph of resources available:

$$H_{res} = (P \cup M \cup K, L) .$$

Required: schedule maximal number of requests from set *Z* for which mapping *A* is correct.

In order to map requests to physical resources three algorithms were developed: two algorithms are based on combination of greedy strategies and limited search strategies [26,27], and the third algorithm is based on the use of ant colony schemes [28]. For algorithms based on combination of greedy strategies and limited search strategies we set search depth parameter for limited search procedure. This parameter allows to regulate computational complexity and accuracy of the algorithm. The first algorithm is the most effective when data exchange network is a critical resource, second – when compute capacity or size of data storage elements is a critical resource [26]. Ant colony optimization algorithm provides best solutions on the test examples for all classes of input data, but has greater computational complexity [28].

The distinguished features of the problem formulated and of algorithms proposed are:

1. Mapping of all types of request elements (compute resources, data storages and network resources) on physical resources occurs consistently in terms of SLA compliance.
2. In case when set of SLA characteristics is changed there is no need to modify algorithm.

In paper [26] it was shown that the algorithm used allows to provide mapping of requests to physical resources results of which substantially exceeds the results of algorithms provided in OpenStack platform. In some cases this difference can be up to 65%. In our experiments, we compared the results of the algorithms of the OpenStack platform and of the developed algorithm, which is used in the SOC platform. The most considerable difference between the results of the algorithms is shown for a set of queries consisting of virtual machines that require more than 90-95% of server resources, and virtual machines that require about 5-10% percent of server resources.

FIGURE 2. RESULTS OF THE ALGORITHMS

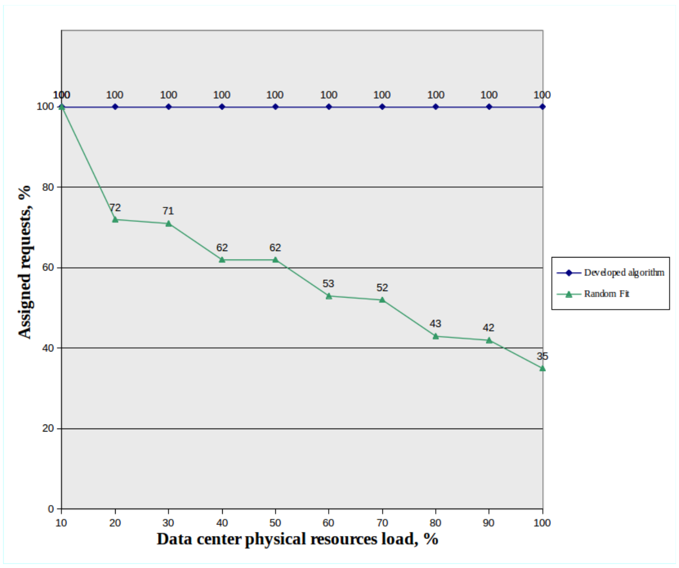


Figure 2 shows the dependence of the number of assigned requests from data center physical resources load for the algorithm of SOC platform and the algorithm of OpenStack platform “selection of random query and physical resources from a variety of appropriate resources”. Detailed description of input data and results of experiments is presented in [26]. You can compare results of algorithms used in SOC platform and in OpenStack platform on your own input data using experimental system of investigation of the properties of algorithms. In order to run algorithms and check their applicability to your data center you need:

- Send a request indicating your name to email address ev@arccn.ru;
- Install any VNC client;
- Read the manual.

Instrumental system allows one to describe data center resources, create set of requests and see how these requests will be distributed on physical resources by the algorithm selected.

IV. VIRTUAL NETWORKS DESCRIPTION LANGUAGE

There are many approaches to describe requirements to cloud system configuration. OVF (Open Virtualization Format, [25]) standard is an open universal standard for declarative description of virtual machines. This standard does not depend on virtualization system and hardware architecture. OVF standard allows to store the most complete information about virtual machine. This standard is extendable. First version of OVF standard did not allow to describe such virtual network elements as virtual data storages or virtual network switching equipment. Starting from 2.0 version (January, 2013) OVF standard supports network configuration specification and network data storages description. OVF standard is actively used by such companies as VmWare, Citrix, RedHat, Cisco, and others.

TOSCA (Topology and Orchestration Specification for Cloud Applications) [29] standard is designed to describe small networks taking into account their functionality, roles of network applications, means and characteristics of application deployment. This standard is relatively new: first version of TOSCA appeared in January, 2014. TOSCA standard is developed by such companies as IBM, SAP, HP, Rackspace. Templates of this standard are supported in OpenStack Heat project. The aim of TOSCA is to standardize interaction between cloud platforms and to provide cross-platform compatibility for applications and services.

CIM (Common Information Model, [30]) standard is designed in order to provide uniform data exchange between network nodes of different types having different sets of parameters. The aim of CIM standard is to describe control information in a standard way. CIM maps different control schemes, including SNMP management information bases (MIBs), to their data structure. CIM can be considered as a data dictionary used to manage systems and networks and to document how their features should correlate. To some extent CIM can be considered as an SNMP protocol extension. CIM is the basis for many other DMTF (Distributed Management Task Force) standards as well as for SMI-S (Storage Management Initiative — Specification, [31]) standard, which is designed to manage data storage systems.

One more language which is designed for network management, monitoring and modeling is Yang (RFC 6020) [32], the add-on for NETCONF protocol (RFC 6241) [33]. NETCONF protocol provides mechanisms for placement, management and removal of network devices configuration by using RPC (Remote Procedure Call) mechanisms. NETCONF uses XML for configuration provisioning and message generation. NETCONF protocol is used over transport protocol (e.g. SSH). NETCONF represents extended and improved model of network resources management and monitoring. Yang is a language of network infrastructure description which is rather procedural than declarative. It describes not only network nodes and their parameters, but also procedures of network nodes management. Network description language presented in this work realizes composite data types by means of dictionaries and therefore in our case there is no need for complex tools for new data types creation realized in Yang.

In order to manage resource allocation and provisioning, Global Environment for Network Innovations (GENI) project uses Resource Specification (RSpec) [34] standard. RSpec is a language for resources querying (request) and resources provisioning (manifest). Requests are provided as XML files of predefined format. GENI project tools allow to combine computational, network, sensor, and authorization schemes as well as configurations of preinstalled software. Request description language RSpec is represented as an XML Schema and is essentially similar to the network description language proposed in this work. At the same time, in GENI RSpec there is no description of virtual services (virtual network functions). Also, GENI data structure is not fully compliant with OpenStack data model.

In our case the goal is to provide a network description which is based on unified elements, such as virtual machine, data storage, virtual network. Structure of these elements is fully described in OpenStack data model to which we are trying to fit as much as possible. Moreover, our virtual network description language allows to describe such objects as virtual network functions (VNF) [35], as well as to describe interconnection between virtual networks and providing VNF services from one virtual network to another. In order to solve these problems we developed a domain specific language (DSL) [36], which represents a declarative description of virtual network. In order to describe the grammar of this language we use Backus-Naur Form (BNF) [37]. BNF notation provides simple tools to describe grammar constructs and does not need to use large amount of syntax rules. BNF can be interpreted using standard language interpreter generators (lexers and parsers) [38], which are based on DSL grammar definition provided in BNF notation.

The node of virtual network can be one of the following: virtual machine, VNF, commutation element, data storage element, network domain. Nodes are connected to each other by virtual links.

Every node is described by a list of parameters and by a set of SLA criteria specific for this node type. A user can select certain values of SLA requirements from allowed range or use default values which are predefined by the provider.

For virtual machine the following parameters are set: unique name; identifier of the deployment image; network port names; SLA criteria (number of virtual cores, frequency, core type, RAM, number of mounted disks).

For VNF the following parameters are set: unique name, identifier from a predefined set to describe deployment scenario, port names. Scenarios are provided in form of XML files. A scenario XML file contains description of SLA criteria which are needed for network function deployment.

For data storage element the following parameters are set: unique name, port names, SLA criteria (required storage size and data storage type).

SLA parameters for commutation element are: unique name, commutation element type (e.g. switch or router), port names. For a router commutation element ip address and subnet mask are defined.

Domain is a nested L2 subnet in the virtual network. For domain the following parameters are set: unique name, number of physical servers and data storage elements, network type, names of external ports. If network type is bus then at any moment of time only one connection between domain elements (virtual machine, data storage element, external port) is possible. If network type is switch then at any moment of time it is possible to connect any domain element to any other domain element. SLA criteria are equal for all virtual machines and data storage elements of the domain and are set similarly to appropriate virtual network nodes.

For virtual link unique name and SLA parameter (link capacity) are set. Link name is defined by names of nodes and ports which are connected by this link.

Also, for any element of the network the following parameters are set: creation time, update time, deletion time.

For virtual network we specify desirable SLA level:

- guaranteed SLA for all network nodes and virtual links – to meet SLA at any moment of time;
- non-guaranteed SLA for all network nodes and virtual links – preferred by non-mandatory SLA criteria which can be provided.

The language proposed allows to provide services by one network to another and get services either from other networks or from provider network in form of secure VNFs.

Our network description language allows one to set routing policies for virtual networks. Routing policy for non-managed virtual network is the list of descriptions of allowed and forbidden flows in the network (for example, set of nodes through which all routes of the flow should or should not pass). If the type of a virtual network is set to be manageable then the owner of the network manages routing policies himself in the process of network operation.

V. Conclusion

SOC cloud platform considered in this paper allows to deploy both manageable and non-manageable virtual networks in the data center. Possibility of virtual resources migration, consistent scheduling and management of computing resources allows one to ensure high load of physical resources and guaranteed SLA compliance for the network as a whole. Request for virtual network creation can be defined either by means of the network description language or by means of GUI. The SOC cloud platform is consistent with OpenStack.

Especially one should note that this platform is neither pure PaaS nor IaaS or SaaS. We intentionally escape from outdated “anatomical” approach to cloud design philosophy when, according to “purity of concept”, user was limited in obtaining particular services and in abilities to deploy and use complex functional blocks in virtual networks.

References

[1] Amies A, Sluiman H., Tong Q.G., et al, Developing and Hosting Applications on the Cloud // IBM Press, 2012.

[2] Venkata Josyula, Malcolm Orr, Greg Page, Cloud Computing: Automating the Virtualized Data Center // Cisco Press, 2012

[3] Pietro Iannucci, Manav Gupta, IBM SmartCloud: Building a Cloud Enabled Data Center // IBM Redbooks, 2013

[4] John Arrasjid, Vmware Vcloud Architecture Toolkit (Vcat) // Vmware Press, Pearson Education, Limited, 2013

[5] David Ziembicki, Microsoft System Center: Integrated Cloud Platform // Microsoft Press, 2013

[6] Harding Ozihel, HP Cloud Service Automation Software // Frac Press, 2012

[7] Hemant Kumar Mehta, Getting Started with Oracle Public Cloud // Packt Publishing Ltd, 2013

[8] Borko Furht, Armando Escalante, Handbook of Cloud // Springer Science & Business Media, 2010

[9] Pepple K. Deploying OpenStack // O'Reilly, 2011.

[10] Urgaonkar B., Rosenberg A.L., Shenoy P. Application placement on a cluster of servers // Intern. J. of Foundations of Computer Science, 2007, T. 18, №05, P.1023-1041.

[11] Bein D., Bein W., Venigella S. Cloud Storage and Online Bin Packing // Proc. of the 5th Intern. Symp. on Intelligent Distributed Computing, 2011, Delft: IDC, P. 63-68.

[12] Nagendram S., Lakshmi J.V., Rao D.V., et al Efficient Resource Scheduling in Data Centers using MRIS // Indian J. of Computer Science and Engineering, 2011, V. 2. Issue 5, P. 764-769.

[13] Arzuaga E., Kaeli D.R. Quantifying load imbalance on virtualized enterprise servers // Proc. of the first joint WOSP/SIPEW international conference on Performance engineering, 2010, San Josa, CA: ACM, P.235-242.

[14] Mishra M., Sahoo A. On theory of VM placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach // Cloud Computing (CLOUD), IEEE International Conference, 2011, Washington: IEEE Press, P.275-282.

[15] Botero J.F., Hesselbach X., Fischer A., et al Optimal mapping of virtual networks with hidden hops // Telecommunication Systems, 2012. V.51, №4, P.273-282.

[16] Yu M., Yi Y., Rexford J., et al Rethinking virtual network embedding: substrate support for path splitting and migration // ACM SIGCOMM Computer Communication Review, 2008, V.38, №2, P.17-29.

[17] Lischka J., Karl H. A virtual network mapping algorithm based on subgraph isomorphism detection // Proc. of the 1st ACM workshop on Virtualized infrastructure systems and architectures, 2009, Barcelona: ACM, P.81-88.

[18] Zhu Y., Ammar M.H. Algorithms for Assigning Substrate Network Resources to Virtual Network Components // 25th Intern. Conference on Computer Communications, 2006, Barcelona: INFOCOM, P.1-12.

[19] Chowdhury N.M.M.K., Rahman M.R., Boutaba R. Virtual network embedding with coordinated node and link mapping // 28th Intern. Conference on Computer Communications, 2009, Barcelona: INFOCOM, P.783-791.

[20] Cheng X., Sen S., Zhongbao Z., et al Virtual network embedding through topology-aware node ranking // ACM SIGCOMM Computer Communication Review, 2011, V.41, №2, P.38-47.

[21] Korupolu M., Singh A., Bamba B. Coupled placement in modern data centers // IEEE Intern. Symposium on Parallel & Distributed Processing, 2009, New York: IPDPS, P.1-12.

[22] Singh A., Korupolu M., Mohapatra D. Server-storage virtualization: integration and load balancing in data centers // Proc. of the 2008 ACM/IEEE conference on Supercomputing, 2008, Austin: IEEE Press, P.1-12.

[23] Jiang J.W., Tian L., Sangtae H., et al Joint VM placement and routing for data center traffic engineering // 31th Intern. Conference on Computer Communications, 2012, Orlando: INFOCOM, P.2876-2880.

[24] Sidnie Feit, Local Area High Speed Networks // Sams Publishing, 2000.

[25] Open Virtualization Format Specification. DMTF Standard. (http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf).

[26] Vdovin P.M., Zotov I.A., Kostenko V.A., Plakunov A.V., Smelyansky R.L. Comparing Various Approaches to Resource Allocating in Data Centers // J. of Computer and Systems Sciences Intern. 2014.V. 53. № 5.

[27] Vdovin P.M., Kostenko V.A. Algorithm for Resource Allocation in Data Centers with Independent Schedulers for Different Types of Resources // J. of Computer and Systems Sciences Intern. 2014. V. 53. № 6.

[28] A.V. Plakunov, V.A. Kostenko. Data Center Resource Mapping Algorithm Based on the Ant Colony Optimization// Proc. of the 2014 International Science and Technology Conference «(MoNeTec)»

[29] Topology and Orchestration Specification for Cloud Applications (http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html)

[30] Common Information Model. DMTF Standard. (http://www.dmtf.org/standards/cim)

[31] ISO/IEC 24775:2011. Information technology -- Storage management. (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=55234)

[32] YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)(https://datatracker.ietf.org/doc/rfc6020/)

[33] Network Configuration Protocol (NETCONF) (http://datatracker.ietf.org/doc/rfc6241/)

[34] Resource Specification (RSpec) Documents in GENI (http://groups.geni.net/geni/wiki/GeniRspec)

[35] Network Functions Visualization. ETSI Technology Specification. (http://www.etsi.org/technologies-clusters/technologies/nfv)

[36] Martin Fowler. Domain-Specific Languages // Pearson Education, 2010.

[37] Kent D. Lee. Programming Languages: An Active Learning Approach // Springer Science & Business Media, 2008.

[38] John R. Levine, Tony Mason, Doug Brown. Lex & Yacc // "O'Reilly Media, Inc.", 1992

SDI Defense Against DDoS Attacks Based on IP Fast Hopping Method

V. Krylov, K. Kravtsov, E. Sokolova, D. Lyakhmanov
Nizhny Novgorod State Technical University n.a. R.E. Alekseev,
Nizhny Novgorod, 603950, Russia
vkrylov@heterarchica.com, kirill@kravtsov.biz, essokolowa@gmail.com, dm.virger@gmail.com

***Abstract*—We introduce IP Fast Hopping, easily deployable network-layer software solution against DDoS attacks. Our approach enhances server's SDN environment by providing an easy way for SDN controllers to protect servers against DDoS attacks and traffic interception by hiding of these servers behind a set of physical network switches.**

Keywords—* *SDI security, DDoS attacks, traffic interception, switcher, IP hopping, IP Fast Hopping

I. INTRODUCTION

A Denial-of-Service attack is characterized by an explicit attempt to prevent the legitimate use of a service. A Distributed Denial-of-Service attack deploys multiple attacking entities to attain this goal [1]. In such attacks, a single bot or a group of bots are sending a large number of packets that lead to exhausting of victim's bandwidth capacity or software processing capabilities.

According to [1], methods of DDoS attacks can be divided by the two groups: semantic attacks and brute-force (flood) attacks. A semantic attack exploits a specific feature or implementation bug of some protocol or application installed at the victim in order to consume excess amounts of its resources. For example, an attacker can send a specific sequence of packets initiating CPU time consuming procedures on the server. In case of a large number of such requests, the victim is unable to handle requests from legitimate clients. Undesirable impact from such attack can be minimized by protocol or software modifying and by applying of special filter mechanisms. In our paper, we introduce a DDoS defense mechanism that aims to filter all TCP traffic issued by unauthorized clients on network level. Therefore unauthorized malefactor is unable to perform semantic attacks based on TCP protocol on the victim server.

A brute-force attack is an attack aimed to prevent legitimate service using by exhausting of bandwidth. E.g. it is a large number of short requests to the victim which initiates heavy responses sent by the victim. Together these streams overfills bandwidth of the victim server or it's ISP. In contrast to semantic attacks, brute-force attacks abuses legal services so installing of filtering mechanisms for such requests will impact traffic from legitimate client too. During brute-force DDoS attacks a number of malefactor terminals (botnet) and legitimate users are connected to the victim at the same time (see Fig. 1). Each bot sends a big number of requests to the victim which creates heavy malicious traffic targeted at the server. Since the increase in the flow of requests is created here increase the number of terminals, then whichever level of server performance has not been achieved starting from a certain number of bots, they create the flow of requests exceeds the permissible level for any server.

Size and frequency of DDoS attack is continuing to grow despite on the fact that a large number of defense mechanisms have been proposed. According to [2] application layer attacks rose approximately 42% in 2013 from 2012, infrastructure layer attacks increased approximately 30% at the same period.

The infrastructure layer DDoS attacks are still most popular: around 77% of DDoS attacks in Q4 2013 were infrastructure layer attacks. The average size of attacks increased by 19.5% from Q1 2012 to Q1 2013 (up to 1.77 GB/s) [3]. So, developing new DDoS prevention mechanisms is still a topical issue.

Each new DDoS defense method should satisfy the following main principles:

- Real world applicability. Now a number of different approaches have been suggested in literature which require a significant changes of the existing network architecture of ISPs or entire Internet architecture. But the main problem here that mostly DDoS attacks threaten organizations which provide services to end users [4], and so this problem is not very vital for transit ISPs because they actually suffer very little from such attacks. Thus, priority will be given to such systems, which filters malicious traffic without changing of global network architecture, into server's or edge ISP's networks.

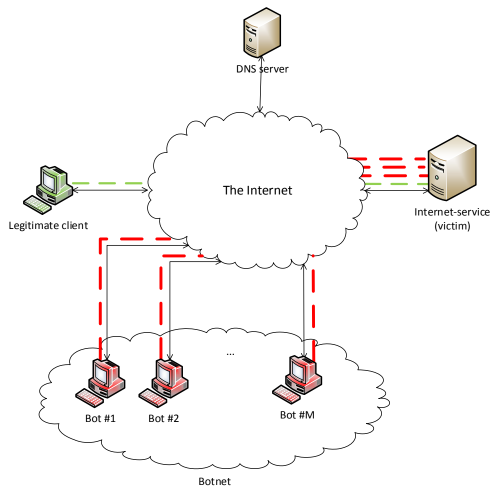


Fig. 1. Schema of brute-force DDoS attack.

- The solution must be designed to prevent misuse. So, it must be impossible to exploit the method to increase impact issues by an attack or to filter legitimate traffic.

Taking into account these principles, our work suggest DDoS prevention mechanism as software solution which does not require additional hardware equipment. Due to this fact, described approach can be implemented as part of Software Defined Infrastructure of a server.

II. RELATED WORK

The rise of DDoS attack frequency in recent years has resulted in many proposed defense approaches from the community. For example, patent [5] suggests solution on application level. According to the article, the server sends a special response on first connection attempt from a client. The client uses this response to identify new URL address, after that the client creates new requests and sends this new requests to this new URL. After receiving this second request, the server validates the new URL based on sent response. If the value does not exceed preset load threshold, the packet will be prioritized and processed by the server. Thus the solution is based on applying of special filter on server side; this filter controls prioritizing of client's requests depending on load on the server at the moment of receiving of the first request from the client.

Other high level approach was suggested under [6]. This method introduced special server responsible for creating and updating of cryptographically secured keys. Each client can access to the service only after successful legitimacy verification on this special server. Thus, client should be successfully authorized on the special server to get a secure key which should be used for processing of a special scenario. The aim of this scenario is identifying client's legitimacy. These approaches purpose defense methods on application layer and does not impact IP packets exchange. So, a malefactor can perform brute force attack on the server.

Also, the research community suggested a wide scope of different more low level approaches. For example, [7] purposes dividing of data stream transmitted between server and client into two consecutive segments on TCP level. This work suggested comparing of keys of two consecutive segments to detect possible segments from not legitimate source. In case of detection of such segments, data receiving will be blocked to prevent possible impact from attack.

Paper [8] introduces DDoS defense mechanism based on dynamic change of server's IP address. Server's IP address is changing according to pseudo-random law which is known only for authorized clients. At the first sight the work [8] purposes a similar DDoS prevention mechanism (dynamic changing of IP address), but this contains some significant differences. Among others, are:

- IP address of the victim is changing only during active DDoS attack on the server
- The new IP address is assigned for all client sessions simultaneously on a relatively long time (suggested period is around 5 minutes)
- Accurate time synchronization is required for calculation of each next IP address since external timestamp is using.

III. IP FAST HOPPING METHOD

In our paper, we introduce a DDoS prevention mechanism based on protocol level defense methods which was suggested under patent [9]. The main goal of this technique is counteraction to exhausting of server's resources initiated by attackers and prevention of legitimate traffic filtering. To achieve these aims, the method is using real-time changing of server's IP address according to a schedule which is available only for authorized clients. Attackers can't get access to this schedule, so they cannot send requests to the correct IP address. Due to this effect, bots are unable to create enough high load on the server to prevent normal system behavior.

In our work, we suggest to call this method as IP Fast Hopping.

The method suggested in this paper is similar to radio systems with frequency hopping. In such systems, receiver and transmitter are switching from one frequency to other frequency synchronously during an ongoing data transmission session. A malefactor's transmitter, which is going to introduce a noise into such session, has not an actual schedule of frequency hopping; therefore such attacker cannot create a noticeable harm for the legitimate transmitter defended by frequency hopping mechanism.

In our case, frequency can be treated like IP address. So, the legitimate client must know schedule of server's IP address changing. At the same time, the schedule should be unavailable for non-legitimate clients.

The method of IP masquerading for received packets is utilized in Network Address Translation technology. In contrast to the technique suggested in this paper, such IP masquerading is permanent during the entire session, i.e. mapping of the internal constant address to a temporary external address is not changing during a session [10]. This approach provides a way to share limited external network resources between a large number of devices. In our paper, we propose to make such mapping dynamic.

According to DDoS prevention mechanism based on IP hopping approach, DNS entries are equal to IP address of the authorization server instead of IP address of the protected server. To access the protected server, each client must be tested on legitimacy on this authorization server. Authorization process can cover validation of user's login/password, user's subscription on a service and so on. In case of successful client's authorization, the client is redirecting to special server, IP Hopper Manager which can be a part of SDN controller, instead of to the protected server. This server is controller of enhanced secured sessions. In this paper, enhanced secured session is a communication session between client and server which is protected by IP Fast Hopping method. The legitimate client must establish secured connection to this controller. The IP Hopper Manager will use this connection to transmit a pool of IP addresses and unique identifier of the session to the client's terminal. The IP Hopper Manager sends the same information to a set of edge switches randomly located in the Internet. These switches must support IP Fast Hopping method and the server must be signed on this service. In this paper, the edge switcher is high performance switcher which is edge relatively to the suggested protection mechanism, because in the network sector between this switcher and the protected server the data stream has not any difference in comparison with the case when IP Fast Hopping was not deployed. The IP pool is unordered and each IP address must be related to this switches set. Also, this pool should not contain the real server's IP and the "initial" IP. The initial IP is public virtual address of the protected server. All client's applications use the initial IP address instead of real IP address of the server.

After such handshake the client starts communication session with the server. During communication session between the client and the protected server, IP address of the server is hopping between addresses from this pool in real-time. The client's terminal is changing initial IP address in the destination address field of each outgoing packet on an address from the pool of IP addresses according to a special hash function. This hash function is mapping timestamps field of TCP header [11] and unique identifier of the session to an entry of the IP pool. This UID can be obtained for the private key of a certificate installed on the client's equipment or can be received from IP Hopper Manager as was mentioned above. After such replacement of the initial address to a virtual address from the IP pool, the packet is transmitted over the Internet to one of edge switches according to common switching protocols.

When the edge switcher received the packet from the client, the switcher calculates the same pseudo-random function with the same arguments as was done on the client side. If the result of this calculation is the same to the destination address field of IP header, the packet is forwarded to the real IP address of the server as legitimate packet. Otherwise, this packet will be dropped as malicious packet.

The same procedures (but in reverse mode) will be applied for each server's responses to the client. After receiving of such packets, client's terminal changes server's virtual IP into source field on initial IP address, after that the packet can be processed by client's application by a common way.

As the result, from the point of view of an external observer of the client-server communication session, the IP address of the server is changing regularly to a random address with each increment of timestamps field into TCP header of the packet (usually every millisecond).

Prediction of destination IP of the next packet is very difficult for an external observer due to the fact that destination IP is changing according to pseudo-random function and this observer has not information about parameters of this function (UID or real server's IP address).

If the IP pool is not large enough, a botnet can start an attack on each IP address using masquerading of malicious traffic as legitimate data stream by IP spoofing technique [12]. In this case, edge routers redirects part of hateful traffic together with legal traffic to the protected server. In this paper, we suggest the following options to mitigate such risks:

1. The IP pool which is used for IP Fast Hopping should be large enough to make such excessive attack very resource consuming and non-efficient for possible attackers. Obviously, the method will be more efficient in IPv6 systems. In this case, IP pool can contain a thousands of addresses related to a number of different routers in the Internet.
2. IP providers should apply IP spoofing filtering mechanisms, e.g. [13]

The described particular qualities of introduced DDoS protection mechanism allow to use this method not only for DDoS prevention but also for defense of communication session between server and client. In this paper, all TCP packets in each client-server communication session transmitted via network according to IP Fast Hopping rules without depending on existing of active attack on the server. This fact causes the following effect: for an external observer close to the client, the communication session between the client and the Internet service does not look like packet stream between terminal of the client and a server on which this Internet server is hosted. This session is visible for an external observer as different communication sessions between the client and a large scope of different servers in the Internet and data stream is randomly mixed between these streams. From an external observer point of view, interpretation of these data stream into one logical data stream are difficult process. Also, due to the fact that one pool of virtual IP addresses is shared between different Internet services at the same time, such external observer close to the client is unable to identify server which established communication session. So, IP Fast Hopping could be used in cases when clients want to hide content of data stream and destination of this stream.

IV. SYSTEM ARCHITECTURE

Our paper introduces DDoS protection mechanism aims to prevent access to the server from a botnet by dynamic changing of IP address of the server. Such system can have a scope of different implementations, but our work takes into account the following requirement: the suggested approach should be easy deployable in real world conditions and should not require a significant changes of the existing network architecture or network equipment. Easy deployability means that switching to the suggested defense method does not require a considerable preparation or workflow changes for an Internet server or its clients.

To achieve these goals, our work uses only existing commonly used technologies and protocols and also the logical core of the system is deployable into external (for the server and its clients) networks (e.g. into ISP networks).

We can say that our work is a new point of view on using of existing abilities of TCP/IP protocols. Our paper introduced re-use of already used technologies for DDoS attacks prevention. An example of such alternative utilization, the timestamps field of TCP packet header, are suggested to be used not only to identify the correct packets order [11], but at the same time this field can be used to identify the correct destination address of the packet as was described above.

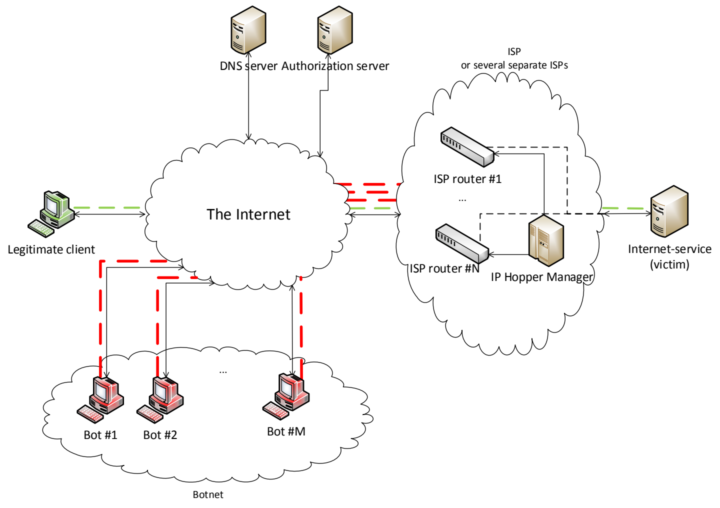


Fig. 2. IP Fast Hopping architecture.

Also, our work introduced distributed DDoS defense mechanism: an Internet server is being hidden behind a large pool of virtual IP addresses which belong to a big number of routers in different sectors of the Internet. Since this IP address pool is public, botnets can initiate DDoS attacks on one or several of these IP addresses. But the pool is divided into groups of addresses which belong to various routers in various Internet sectors. So the stream of malicious packets initiated by a botnet is divided into several sub-streams directed to several Internet sectors by commonly used switching protocols. And, according to our work, this stream will be filtered into this different networks. This approach defends the victim server and also our method decreases load on network infrastructure of victim and it's ISP during active DDoS attack.

In case of deploying of the introduced defense mechanism, the original client-server architecture (see Fig. 1) contains some new blocks (see updated schema on Fig. 2). The suggested architecture has the following difference:

1. As was mentioned above, the DNS server contains link to IP address of Authorization server instead of IP address of Internet-service
2. Introduced Authorization server which validates legitimation of the client. If client successfully authorized and the client's terminal has a special SSL certificate and supports IP Fast Hopping algorithm (i.e. installed special software – IP Hopper Core), Authorization server initiates handshake between the client's IP Hopper Core and IP Hopper Manager
3. IP Hopper Core is special system utility installed on client's terminal and ISP routers #1 - #N (entire IP pool belongs to these routers). This utility is performing establishing of enhanced secured connection and real-time changing of initial IP address of the Internet-service on one address from IP pool according to rules of IP Fast Hopping.
4. IP Hopper Manager is a server which is responsible for controlling the enhanced secure connections between Internet-services and clients. Can be implemented as part of SDN controller.

Time chart of introduced defense mechanism can be found on Fig. 3.

V. IMPLEMENTATION

As was noted above, one of requirements for our work is real world deployability. Therefore, we implement IP Fast Hopping mechanism as kernel module of OS GNU/Linux. In this case, installing this module on routers based on GNU/Linux is enough to deploy the suggested system. In our work we build such routers based on Debian OS.

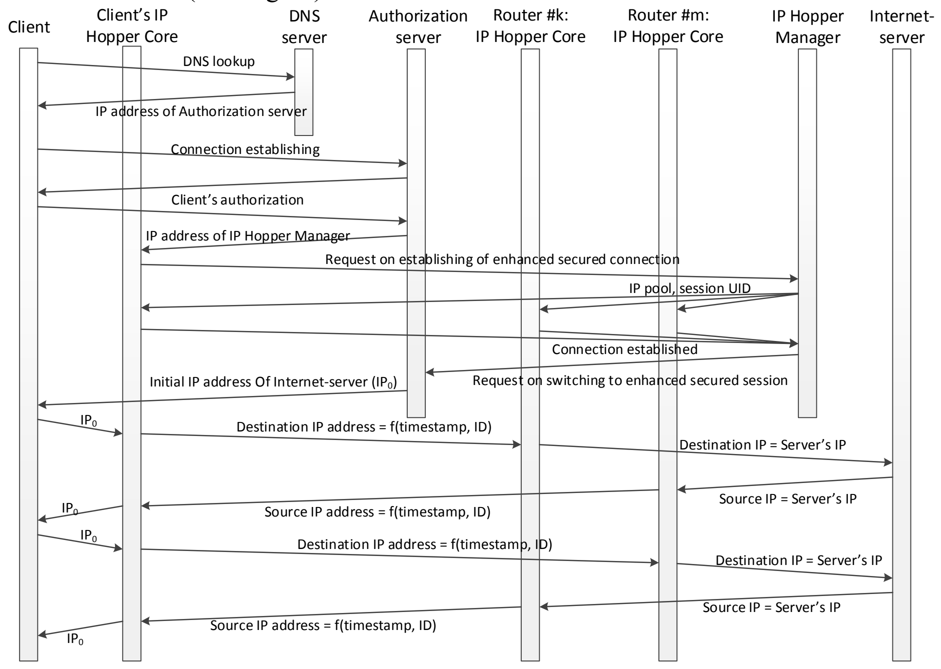

Client
Client's IP Hopper Core
DNS server
Authorization server
Router #k: IP Hopper Core
Router #m: IP Hopper Core
IP Hopper Manager
Internet-server
DNS lookup
IP address of Authorization server
Connection establishing
Client's authorization
IP address of IP Hopper Manager
Request on establishing of enhanced secured connection
IP pool, session UID
Connection established
Request on switching to enhanced secured session
Initial IP address Of Internet-server (IP$_0$)
IP$_0$
Destination IP address = f(timestamp, ID)
Destination IP = Server's IP
Source IP = Server's IP
Source IP address = f(timestamp, ID)
IP$_0$
IP$_0$
Destination IP address = f(timestamp, ID)
Destination IP = Server's IP
Source IP = Server's IP
Source IP address = f(timestamp, ID)
IP$_0$

Fig. 3. Time Chart of IP Fast Hopping

Linux kernel contains built-in firewall Netfilter [14], which is responsible on packet filtering and forwarding according to predefined rules by iptables utility. Netfilter architecture is scope of hooks of ordered rules. Netfilter performs a predefined action with a packet, which is passed to a hook, according to the corresponding rule.

Netfilter supports 5 hooks: PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING. When the packet comes to the system, the packet is processed by PREROUTING hook. If this packet is addressed to a local process, it is passed to INPUT hook, otherwise it is passed to FORWARD. All packets sent by local processes are processed by OUTPUT hook. The final processing of the packet outgoing from the system (forwarded under FORWARD hook or issued by a local process) is performing by POSTROUTING hook.

In our work, Netfilter contains new module which is responsible for changing of IP address into destination field of outgoing packets and into source field of ingoing packets. This module is calculating the new IP address according to IP Fast Hopping rules (by timestamp field and session UID). During handshake, IP Hopper Manager adds new set of rules into POSTROUTING hook on client's terminal and into PREROUTING each edge switcher. This rule activates the kernel module which implements the following algorithm:

- On the client side this module calculates hash-function using timestamps field and session UID for each outgoing packet addressed to the initial IP address. After that the module uses this result as index of correct address into IP pool which should be put into destination field of the packet. For each ingoing packet from the same communication session, the module performs the same actions for source field: checks the current value of the field (by calculation of the same hash-function) and changes it on the initial address.
- On switches side this module calculates hash-function using timestamps field and session UID for each ingoing packet addressed to IP addresses from IP pool. If the current destination address corresponds to the timestamps field and session UID, the real IP address of the server will be placed into the destination field. Otherwise, the packet will be dropped. For all ingoing packets issued by the server, the module will replace source field by one of virtual addresses according to current value of hash-function.

VI. CONCLUSIONS

We presented IP Fast Hopping, a new approach that can prevent exhausting of server's resources during brute-force DDoS attacks and can be used to hide content and destination of client's communication session. This method hides the real IP address of the server behind a big number of "virtual" IP addresses. The mapping of the real IP address on one of "virtual" is unique for each communication session and changes dynamically every millisecond. The introduced approach is distributed: it divides the traffic from legitimate users and botnets into a number of sub-streams. This leads to a decrease of load on network infrastructure during active DDoS attack. The method is easily deployable and can filter even the biggest malicious streams.

This work was supported by Ministry of Education of Russian Federation under government contract №14.574.21.0034 from 06/17/2014.

REFERENCES

[1] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review,* no. Volume 34 Issue 2, pp. 39 - 53, 2004.

[2] Prolexic Technologies, Inc, "Prolexic Quarterly Global DDoS Attack Report," Q4 4013.

[3] S. M. Kerner, "DDoS Attacks: Growing, but How Much?," 26 April 2013. http://www.esecurityplanet.com/network-security/ddos-attacks-growing-but-how-much.html.

[4] NSFOCUS, Inc., "NSFOCUS Mid-Year DDoS Threat Report," 2013.

[5] W.-C. Feng and E. Kaiser, "Systems and methods for protecting against denial of service attacks". USA Patent 20100031315 A1, 2010.

[6] Arun K. Iyengar, Mudhakar Srivatsa and Jian Yin, "Protecting against denial of service attacks using trust, quality of service, personalization, and hide port messages". USA Patent US20100235632 A1, 2010.

[7] D. R. Marquardt, P. A. Paranjape and P. S. Patil, "Securing a communication protocol against attacks". USA Patent 20110283367 A1, 13 May 2011.

[8] P. Mittal, D. Kim, Y.-C. Hu and M. Caesar, "Mirage: Towards Deployable DDoS Defense for Web Applications," arXiv, 2012.

[9] V. V. Krylov and D. M. Ponomarev, "Method of interaction of terminal client device with server over Internet with high level of security from DDoS attack and system for realising said method". Russia Patent 2496136C1, 14 May 2012.

[10] E. S. Guha, K. Biswas, B. Ford, S. Sivakumar and P. Srisuresh, "RFC 5382 NAT Behavioral Requirements for TCP," 2008. http://www.ietf.org/rfc/rfc5382.txt.

[11] V. Jacobson, R. Braden and D. Borman, "RFC 1323 TCP Extensions for High Performance," 1992. http://www.ietf.org/rfc/rfc1323.txt.

[12] A. Farha, "IP Spoofing," http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_10-4/104_ip-spoofing.html.

[13] P. Ferguson and D. Senie, "RFC 2827 Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," 2000. http://www.ietf.org/rfc/rfc2827.txt.

[14] "netfilter/iptables project," http://www.netfilter.org/.

Software-Defined Network Exchanges (SDXs) and Infrastructure (SDI): Emerging Innovations In SDN and SDI Interdomain Multi-Layer Services and Capabilities

J. Mambretti, J. Chen, F. Yeh
International Center for Advanced
Internet Research (iCAIR)
Northwestern University
750 North Lake Shore Drive, Ste 600
1-312-503-0735
j-mambretti@northwestern.edu, jim-chen@northwestern.edu, fyeh@northwestern.edu

Abstract—**Software-Defined-Networking (SDN) is quickly transforming the networking landscape. Programmable networking based on many types of virtualization techniques, including SDN, enable high levels of abstraction for network services, control and management functions, and underlying technology resources. These approaches enable network designers to create a much wider range of services and capability, including through Software Defined Networking Exchanges (SDXs) than can be provided with traditional networks and exchange facilities, enabling a) many more dynamic provisioning options, including in real time b) faster implementation of new and enhanced services c) enabling applications, edge processes and even individuals to directly control core resources; e) substantially improved options for creating customizable networks and e) enhanced operational efficiency and effectiveness. In addition, these capabilities are now being extended to other types of Software Defined Infrastructure (SDI), including clouds, compute grids, storage devices, instruments, and many other types of edge devices.**

Categories and Subject Descriptors

[**Sofware Defined Networking Exchanges (SDX)**]: Software Defined Networking (SDN), Software Defined Infrastructure (SDI) and Highly Distributed Environments

General Terms

Design, Experimentation,Theory,Verification.

Keywords—Software Defined Network Exchanges (SDXs), Programmable networking and distributed clouds, Software Defined Networking (SDN), Software Defined Infrastructure (SDI), highly distributed environments, multi-domain networking, multi-service networking, multi-layer networking, GENI, international GENI (iGENI), programmable clouds

I. INTRODUCTION

Software-Defined-Networking (SDN) has rapidly changed the how networks are designed, implemented, and operated. Traditionally, communication services and their underlying support infrastructure have been designed and deployed in anticipation of their remaining fairly static for long periods of production. However, increasingly, this model has been made obsolete by continually changing demands at all levels. Consequently, a new architectural approach is required to enable more dynamic services and infrastructure. To date much progress has been made in this area by using programmable networking based on SDN/OpenFlow and other virtualization techniques. This approach has enabled significantly higher levels of abstraction for network services, control and management functions, and across foundation resource technologies. These approaches are allowing network designers to create a much wider range of programmable services and capabilities than can be provided with traditional networks. Consequently, they are providing for a) many more dynamic provisioning options, including real time provisioning b) faster implementation of new and enhanced services c) enabling applications, edge processes and even individuals to directly control core resources; e) substantially improved options for creating customizable networks e) enhanced operational efficiency and effectiveness and f) many more options for traffic engineering.

These capabilities have been proven as especially important resources for services based on distributed clouds, particularly those that are distributed across multiple domains. In part, because SDN enables a more optimal dynamic networking and matching of communication service requirements and network resources. The demonstrated success of SDN techniques with distributed clouds has given rise to considerations of developing other types of Software Defined Infrastructure (SDI), including clouds, compute grids, storage devices, instruments, and many other types of edge devices.

By now, the many benefits of SDN are fairly well known, particularly with regard to data center networks and private Wide Area Networks (WANs) interconnecting data centers. However, SDN architecture is single domain oriented, and, consequently, to date almost all of its implementations have been within single domains. Therefore, increasingly SDN deployments have created many isolated SDN islands. Also, currently, SDN implementations have also been somewhat isolated from non-SDN environments. Both of these issues are challenges that require new capabilities for multi-domain, multi-service SDN provisioning that can be integrated with existing network services. One approach to addressing both these issues is a Software Defined Networking Exchange (SDX). Although the need for SDXs is recognized, no consensus exists about how their services, architecture, capabilities, and underlying technologies should be designed, implemented, and operated. Currently, much debate and discussion is taking place on all of these issues. Nonetheless, a number of research communities are proceeding to develop models of SDXs. Therefore, today, although no production SDX exists, a number of SDXs are being planned, and several prototypes have already been implemented.

One of these prototypes was designed and deployed by the International Center for Advanced Internet Research (iCAIR) and its research partners at the StarLight International/National Communications Exchange Facility, a major exchange facility for world-wide international, national, and regional research and education networks, data intensive science networks, federal agency networks, and large scale national and international network research testbeds. This prototype SDX is being used for research experimentation to explore various approaches to SDX services, protocols, and technologies. It is also being used to demonstration these approaches at a national and global scale. Three core issues being addressed are a) international and national multi-domain SDX interoperability enabling federated controllers in different domains to manage network resources across WANs using an integrated control planes b) multi-service SDX provisioning across and among network layers, including hybrid services and c) enabling interoperability among SDN environments and non-SDN environments.

II. Software Defined Networking (SDN)

Programmable networking using SDN is generally based on the OpenFlow protocol, an architectural approach that separates the control plane from the data plane, abstracts the forwarding path, and enables a controller, connected by a secure channel to network devices to address network functions. [1] An OpenFlow switch has a flow table that stores cached information on traffic streams. This information can be interrogated and analyzedat a highly granulated level so that the results can initiate required responses to control the behaviors of specific individual flows supported by the switch. The controller can monitor the cached information, detect flow attributes and patterns and then react dynamically to the resulting analysis. This technique was initially developed and deployed for primarily for L2 services, and then was extended to both L2 and L3 services. It has also been used for L1 and L0 services.

The SDN technique enables a detailed centralized overview of network services, configurations and resources. However, to date this view has been possible only within and across a single domain. However, there is a need to extend SDN capabilities across and among multiple domains. This requirement is a primary motivation for creating Software Defined Networking Exchanges (SDXs). However, SDXs also are being developed to provide bridges between SDN domains and non-SDN domains, and to enable multi-service networks, based on integrating traffic among all traditional network layers.

As noted, the design, capabilities, and technologies of SDXs are under active discussion. Certainly, one objective is to address these requirements. However, there are many other requirement considerations, for example, providing for control and network resource APIs, precise techniques for multi-domain integrated and federated controller interoperability, controller signaling, including edge signaling, SDN/OF multi layer traffic integration, multi domain resource advertisement and discovery, topology exchange services, highly granulated policy based resource access including through edge processes signaling, rapid configuration and reconfiguration of resources, gateways to non-SDN/OF environments, integration of OF and Non-OF paths, including 3rd party integration, programmability for core resources including large scale large capacity transport streams, etc. To address these requirements, specialized facilities are required.

III. STARLIGHT NATIONAL INTERNATIONAL COMMUNICATIONS EXCHANGE FACILITY

A prototype SDX described has been implemented at the StarLight International/National Communications Exchange Facility in Chicago, which has direct access to over 130 private networks, including many large scale nation and international networks and twenty major experimental network research testbeds, including international testbeds. [2] StarLight was designed to allow for traffic exchange at all layers, and across all layers. The facility created innovative techniques for dynamic L2 and lightpath provisioning. Because StarLight supports multiple data intensive science communities, it interconnects almost 30 individual 100 Gbps paths as well as many 40 Gbps paths and several hundred 10 Gbps paths, all channels on optical fiber based lightpaths. StarLight supports connections among multiple communication exchanges and networks around the world. StarLight is a core component of a larger world-wide facility, the Global Lambda Integrated Facility (GLIF). (Ref Fig. 1) [3] This facility provides optical fiber based lightpaths that can be used to create customized production, prototype, and testbed networks among multiple GLIF Open Lambda Exchanges (GOLEs) around the world. StarLight is one of these GOLEs, all which provide multi-layer interconnection services for communities around the world.

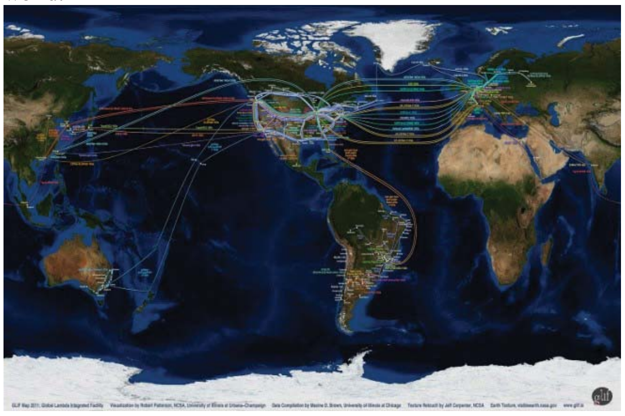
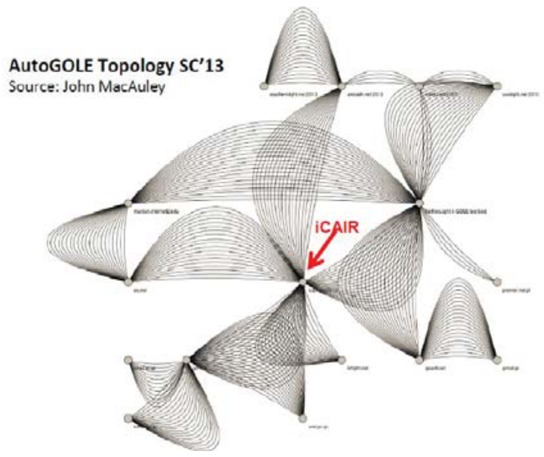


Fig. 1. Global Lambda Integrated Facility (GLIF).

The design of the prototype SDX at StarLight was informed by several wider development contexts. One is the overall IT transition to virtualization at all levels, Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), etc. Another has been an interdomain architecture development project initially undertaken by the GLIF community. Almost all of these exchanges have adopted a different control framework architecture for the resources within those exchanges. Consequently, GLIF community, including iCAIR and the StarLight consortium has undertaken a project Open Grid Forum, a standards organization to develop an interface, the Network Services Interface Connection Service, or NSI CS (currently published as NSI CS 2.0) as an API for the various control frameworks that are used by GOLEs to manage services and resources. [4] A recent major current initiative is a project that is integrating NSI CS 2.0 with OpenFlow/SDN techniques and instantiating these services at exchange points within the GLIF. Another context has been the many years of development of programmable networks for highly distributed Grid environments. [5] This capability has been showcase through multiple demonstrations at national and international conferences through the AutoGOLE initiative. The figure below shows the word wide individual VLANs that were directly addressable through the NSI SC API.



Two other reference contexts have been the National Science Foundation's (NSF's) Global Environment for Network innovations (GENI) and the International GENI (iGENI), large scale, highly distributed infrastructure environments.

IV. SOFTWARE DEFINED NETWORKING EXCHANGES (SDXs) DESIGN AND SERVICE CONSIDERATIONS

Conceptually, the StarLight SDX can be considered an ultra large scale virtual switch comprised of a collection of resources that can be partitioned and integrated for use by external controllers within other domains. The real foundation consists of actual physical SDN/OpenFlow Switches. (Ref: Fig 2). The SDX resources can appear as components that are extensions of external domains. The architectural design is intended to remove middle processes among domains. Of course, this "removal" process is policy driven. Federation policies and processes are required to providing services based on this architecture.

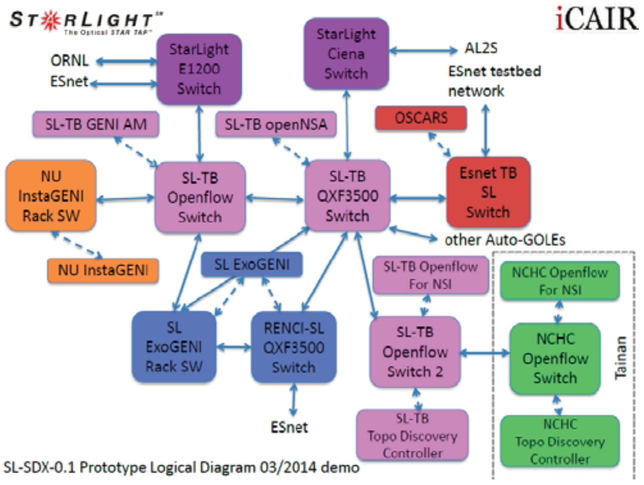


Fig. 2. SDX Prototype

A. SDX Services

As programmable facilities, SDXs can be used to provide an almost unlimited range of services, including specialized new services, such as application specific peering exchanges and large scale encrypted stream exchanges. However, as noted a primary motivation is simply to provide a mechanism for interconnecting the growing number of single domain SDN islands. This capability is especially important for large scale services based on highly distributed clouds, which provide services from a small number of data centers located around the world and are connected through high performance WANs. Today, SDN capabilities have been extremely beneficial for dynamic provisioning to quickly respond to changing traffic flow attributes, and to optimize matching service requirements and network within and among such data centers, and to perform granulated traffic engineering. However, currently, there are no services to allow for interconnections outside those single isolated domains. Because SDNs have not been implemented in current exchanges, they prevent such extensions.

Because of its virtualized resources, options for segmentation and partitioning, and resource programmability, an SDX provides an opportunity to address multi-domain and multi-services interoperability. For example, an SDX supports techniques that enable L2 resources to be discovered, acquired, and integrated by edge controllers in multiple different domains.

Currently, L2 and OF implementations have almost all been deployed as separate environments. In contrast, the StarLight SDX provides support for L2 services, OpenFlow services, and integrated and hybrid L2/OF services.

For SDX multi-services provisioning and integration, these techniques can be extended to any service layer, including L3, L0, L4-L8 and to hybrid services composed of multiple layers.

B. Path Controller

To accomplish the design objectives described in the previous section, the StarLight SDX has implemented an SDN/OF/L2 integrated path controller. This controller is essentially a link controller, which provides options for SDX resource management by other controllers, including federated controllers residing in other domains. This architecture relies on abstracted capabilities to support services and functions, including path optimization, resource discovery, dynamic network resource provisioning, precise explicit data flow provisioning, static resource provisioning, resource monitoring operations and administrative management to be obtained by using signaling supported by a control framework to manipulate lower layer function and physical and virtual resources.

The basic resource used is an OpenFlow controller supporting the control framework as a programmable platform that has been extended to L2 paths. Consequently, the network programmability of SDN/OpenFlow is extended to L2 paths. Also, this technique enables L2 features to be integrated into SDN/OpenFlow environments, which generally are limited without access to those features. This approach also enables edge processes (as well as applications and external services) to customize network services and resources to meet highly specific and defined requirements.

The StarLight SDX was developed with support from the NSF's GENI program, which has developed a nationwide distributed environment for experimental network research. [6] Therefore, the StarLight SDX is based in part on GENI software.

One GENI component, implemented within the StarLight SDX is the Flowvisor OpenFlow Aggregate Manager (FOAM), which interacts with other GENI domains as a resource allocation interface. [7] The StarLight SDX has also implemented Floodlight and other Open SDN controllers. The control functions for these controllers have been extended to L2 services.

C. *Multi-Domain 3rd Part Integration Within the SDX SDN/OF/L2 Environment*

The StarLight SDX, in part, is an extension of an initiative established by an international network research consortium, which designed and implemented a world-wide OpenFlow/SDN testbed, extended the programmable environment termed the "International GENI" (iGENI). [8] For almost 5 years, this international community of network research organizations been developing a large scale global advanced network research testbed based on OpenFlow/SDN, using the GLIF optical networking infrastructure. This programmable distributed environment has a large collection of network resources that can be discovered and integrated, as partitioned resources isolated from others within the environment. Different research groups have used it to conduct experiments, trials, prototypes, and demonstrations. For several years, this programmable testbed was showcased at international SC supercomputing conferences. (Ref: Figure 3)

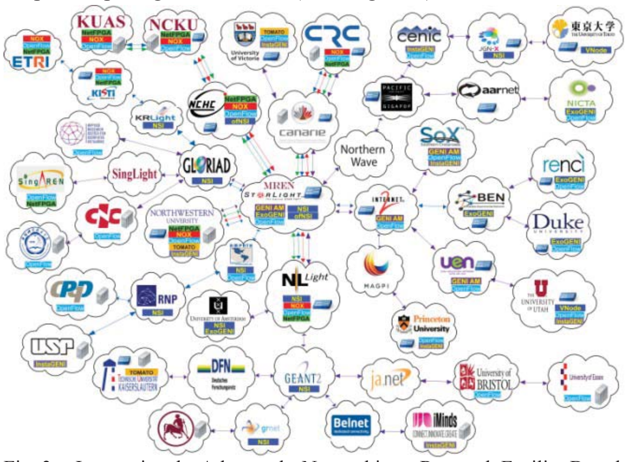


Fig. 3. International Advanced Networking Research Facility Based On OpenFlow/SDN

The different colors in these schematics designate different experiments and demonstrations, undertaken by separate groups of researchers . (Ref: Figures 4, 5, and 6). Topics included an array of L2 functions including a POX based VLAN translation service, a NOX based multi-domain LLDP (Link Layer Discovery Protocol service), a NOX based OAM Continuity Check Message service (CCM), a Multipath TCP (MPTCP) integrated with Floodlight, and many other capabilities. [9, 10, 11, 12, 13]

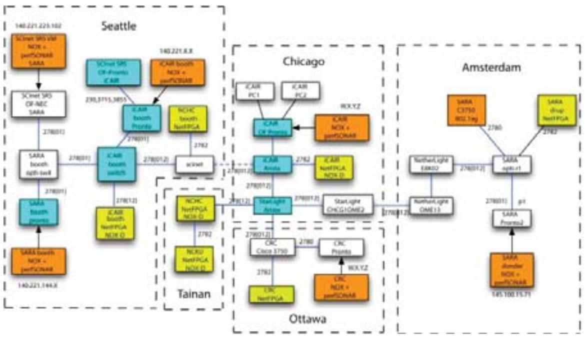


Fig. 4. SC11 SRS/ International Openflow Testbeds

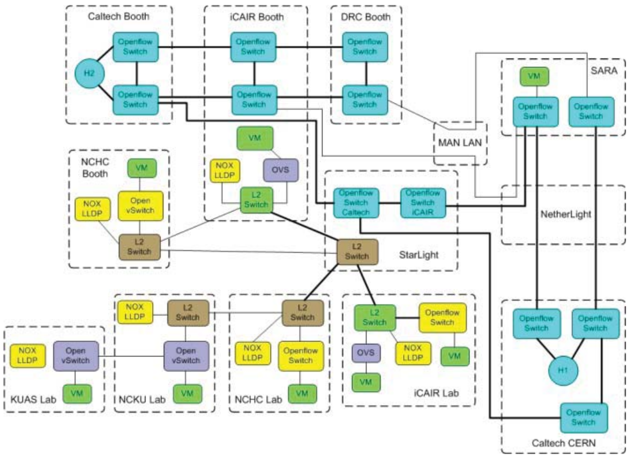


Fig. 5. SC12 SRS/ International Openflow Testbeds

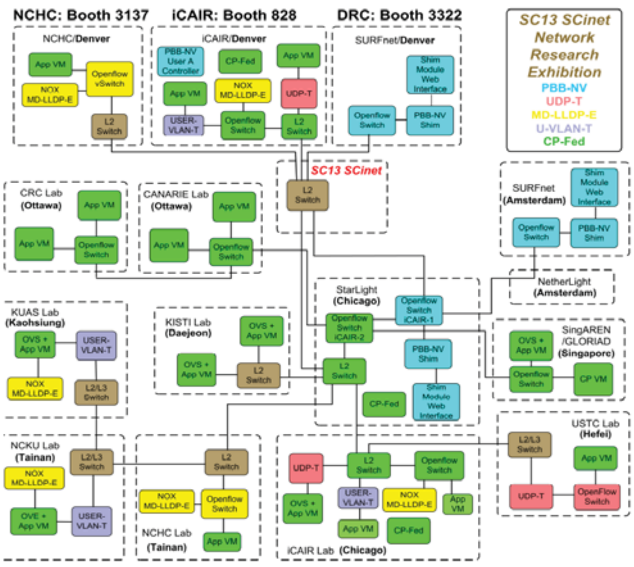


Fig. 6. SC13 SRS/ International Openflow Testbeds

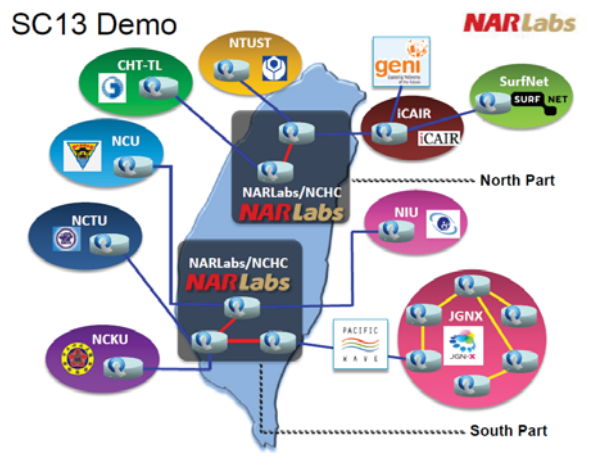


Fig. 7. SC13 NRE/International Openflow/SDN Testbeds

One set of experiments and demonstrations was staged by the National Center for High Performance Computing (NCHC), which operates and maintains the Taiwan Advanced Research and Education Network (TWAREN), and which has developed the Future Internet Testbed in Taiwan based on OpenFlow and extended it to the StarLight/iCAIR facilities. The Future Internet Testbed actively supports many projects by multiple universities, including the integrated project "Research of NetFPGA-based testbed for Future Inter-Cloud Computing Systems" with participants of NCHC, NCKU, KUAS, NTUST (National Taiwan University of Science and Technology) and NCU (National Central University), "Computing and Communications on the Clouds: Applications and Platforms (C^3AP)" initiated by NCTU, and the Cloud Test Center operated by Telecommunication Lab of ChungHwa Telcom.

A particular challenge for intercontroller communications and interoperability is federated controller signaling. To address this issue, one of the research projects is using this testbed to explore new techniques for multiple domain services, specifically methods that allow for large scale, international, multi-domain automatic network topology discovery (MDANTD) and interactivity. [14] This technique is key for adding East⇔West capabilities to the standard North⇔South SDN/OpenFlow protocols. Other methods are being designed to anticipate ongoing changing information related to the availability and location of highly distributed network resources. This approach has been used with other innovative OpenFlow and NOX controller techniques, including multipathing with MPTCP. [15].

D. SDX Demonstrations

Recently, the StarLight prototype SDX has been used to stage a number of demonstrations. In March of 2014, this SDX was used as one of the facilities for a showcase demonstration at the GENI Engineering Conference in Atlanta (GEC 19), illustrating distributed capabilities over five domains. For the demonstration, the StarLight SDX was interconnected to a prototype SDX being developed by Georgia Tech and the Southern Crossroads Exchange (SOX). The two SDXs were interconnected over three separate network domains provisioned on private optical fiber between Chicago and Atlanta. To demonstrate the utility of these two interoperable SDXs, a severe weather prediction application being developed by the University of Massachusetts at Amherst was demonstrated (Ref Image below)

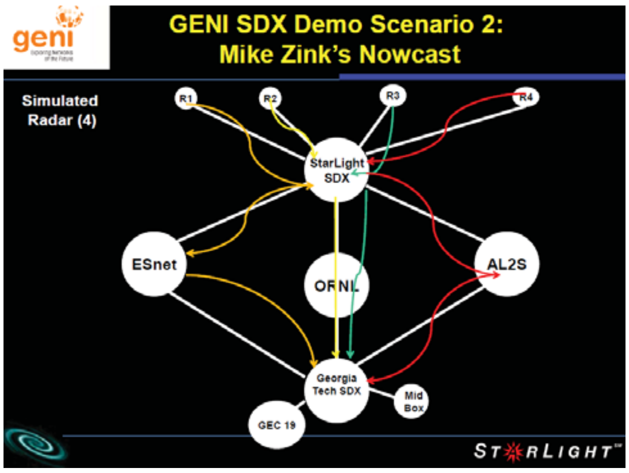


Fig. 8. GEC 19 Dynamic Provisioning via Interoperable SDXs

This application is based on small form factor Doppler radar, which generates extremely large volumes of data that cannot be stored, computed, analyzed, or visualized locally. It must be sent to remote facilities for processing and then results must be returned in real time. At the same time, the topology of the network must change continually. For the TERENA Networking Conference in May 2014, an international interoperability demonstration was staged interconnecting the StarLight SDX with a prototype SDX at NetherLight in Amsterdam (Ref Figure 9)

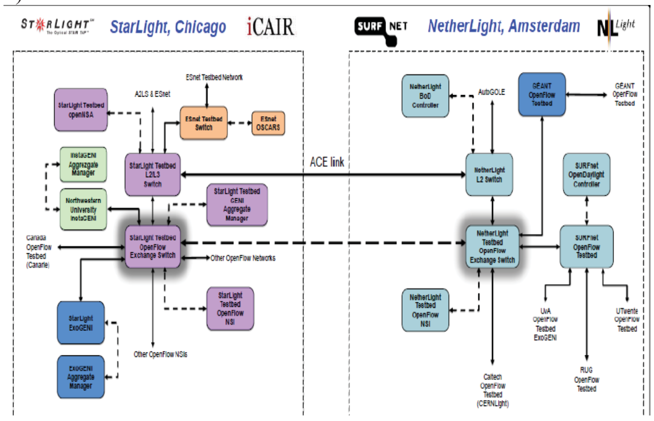

STARLIGHT
StarLight, Chicago
iCAIR
SURF NET
NetherLight, Amsterdam
NL Light
ESnet Testbed Network
AL2S & ESnet
StarLight Testbed openNSA
ESnet Testbed Switch
ESnet OSCARS
InstaGENI Aggregate Manager
StarLight Testbed L2/L3 Switch
StarLight Testbed GENI Aggregate Manager
Northwestern University InstaGENI
Canada OpenFlow Testbed (Canarie)
StarLight Testbed OpenFlow Exchange Switch
Other OpenFlow Networks
StarLight ExoGENI
StarLight Testbed OpenFlow NSI
Other OpenFlow NSIs
ExoGENI Aggregate Manager
ACE link
NetherLight BoD Controller
AutoGOLE
GEANT OpenFlow Testbed
GEANT OpenFlow Testbed
NetherLight L2 Switch
SURFnet OpenDaylight Controller
NetherLight Testbed OpenFlow Exchange Switch
SURFnet OpenFlow Testbed
NetherLight Testbed OpenFlow NSI
UvA OpenFlow Testbed ExoGENI
UTwente OpenFlow Testbed
Caltech OpenFlow Testbed (CERNLight)
RUG OpenFlow Testbed

Fig. 9. International SDX Interoperability Staged for the TERENA Networking Conference

V. FUTURE INITIATIVES

The design, architecture, and technology for SDXs are currently being vigorously debated, because there are many potential opportunities for these types of facilities. Many SDX and SDI research topics are being explored, including semantic network descriptions (e.g., Network Description Language (NDL) initiative at the University of Amsterdam [16, 17], enhanced APIs, resource signaling, resource integration, mechanisms for topology exchanges, and closer integration with edge computers, clouds, storage devices, instruments, "Internet of Things" devices, etc. Several research communities are investigating Software Defined Internet Exchanges, with a focus on L3 traffic management, control, and optimization. [18] Also, the concept of designing a completely virtualized SDX is being discussed as is highly distributed SDXs

VI. SUMMARY

Software-Defined-Networking (SDN) is transforming all aspects of how networks are designed, implemented and operated. Programmable networking based on innovative virtualization techniques, including SDN, are enabling high levels of abstraction for network services, control and management functions, and underlying technology resources. Consequently, networks can be designed to provide many more services and capabilities than traditional networks. SDNs and SDN Exchanges (SDXs) enable a) many more dynamic provisioning options, including in real time b) faster implementation of new and enhanced services c) enabling applications, edge processes and even individuals to directly control core resources; e) substantially improved options for creating customizable networks and e) enhanced operational efficiency and effectiveness. Also, these capabilities are being extended to other types of Software Defined Infrastructure (SDI), including clouds, compute grids, storage devices, instruments, and many other edge devices.

ACKNOWLEDGMENTS

For their excellent support, the authors would like to thank the US National Science Foundation, in particular, the Computer and Information Science and Engineering (CISE) Directorate, and its Global Environment for Network Innovations (GENI) program, including the International GENI (iGENI) program and the GENI Program Office the Office of Cyberinfrastructure and its International Research Network Connections (IRNC) program, the National Center for High-Performance Computing (NCHC), Taiwan, the National Kaohsiung University of Applied Science (KUAS), the National Cheng Kung University (NCKU) KUAS/NCKU, the National Taiwan University of Science and Technology (NTUST), the National Central University (NCU), SURFSara, SURFnet, the GigaPort program of the Dutch Government, the Geant3 project of the European Commission. Research at the Communications Research Centre of Canada. Canada's Advanced Research and Innovation Network (CANARIE), the US Department of Energy's Office of Science, and members of the iCAIR research partnership consortiums, including members of the Global Lambda Integrated Facility (GLIF), the StarLight consortium, the Metropolitan Research and Education Network, the Energy Science Network (ESnet), GENI, International GENI, and the Open Cloud Consortium.

REFERENCES

[1] N. McKeown, et al., OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communication Review, 2008, 2, 69-74.

[2] J. Mambretti, T. DeFanti, M. Brown, StarLight: Next-Generation Communication Services, Exchanges, and Global Facilities, Advances in Computers. 01/2010; 80:191-207.

[3] Global Lambda Integrated Facility (GLIF) www.glif.is

[4] G. Roberts, T. Kudoh, I. Monga, J. Sobieski, J. MacAuley, C. Guok, NSI Connection Service V2.0, Open Grid Forum, GFD-R-P.212, NSI-WG June 2014.

[5] F. Travostino, J. Mambretti, G. Karmous-Edwards (editors), Grid Networks: Enabling Grids with Advanced Communication Technology, John Wiley & Sons, July 2006.

[6] www.geni.net

[7] R. Sherwood, et al., "FlowVisor: A Network Virtualization Layer," OpenFlow Technical Report TR-2009-1.

[8] http://groups.geni.net/geni/wiki/IGENI

[9] M. Luo, S. Lin, J. Chen, "From Monolithic Systems to a Federated E-Learning Cloud System", IEEE International Conference on Cloud Engineering, March 25-28, San Francisco, California, USA, 2013.

[10] M. Luo, J. Chen, J. Mambretti, S. Lin, P. Tsai, F. Yeh, and C. Yang, "Network Virtualization Implementation over Global Research Production Networks", Journal of Internet Technology, Vol. 14, No.7, pp. 1061-1072, 2013.

[11] J. Mambretti, J. Chen, F. Yeh, T.-L. Liu, M.-Y. Luo, C.-S. Yang, R. v. d. Pol, A. Barczyk, F. Dijkstra, and G. v. Malensteinz, "OpenFlow Services for Science: An International Experimental Research Network Demonstrating Multi-Domain Automatic Network Topology Discovery, Direct Dynamic Path Provisioning Using Edge Signaling and Control, Integration With Multipathing Using MPTCP," in SCinet Research Sandbox at International Conference for High Performance Computing, Networking, Storage, and Analysis, White Paper and Presentation, SC12, November, 2012.

[12] M. Luo, J. Chen, "Towards Network Virtualization Management for Federated Cloud Systems", IEEE 6th International Conference on Cloud Computing, June 27-July 2, 2013, Santa Clara, CA, USA.

[13] M. Luo and J. Chen, "Software Defined Networking Across Distributed Datacenters over Cloud," 5th IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom), Bristol, UK, December 2-5, 2013.

[14] W-Y. Huang, J-W. Hu, S-C. Lin, T-L. Liu, P-W. Tsai, C-S. Yang, F. Yeh, J. Hao Chen, J. Mambretti, "Design and Implementation of An Automatic Network Topology Discovery System for the Future Internet Across Different Domains," Proceedings of IEEE 26th International Conference on Advanced Information Networking and Applications Workshops (AINAW'12), Singapore, March 2012.

[15] R. van der Pol, Sander Boele, Freek Dijkstra, Artur Barczyky, Gerben van Malensteinz, Jim Hao Chen, and Joe Mambretti, Multipathing with MPTCP and OpenFlow, forthcoming, Proceedings Companion SC12, November 2012

[16] M. Ghijsen, J. van der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, C. de Laat, "A Semantic-Web Approach for Modeling Computing Infrastructures", Journal of Computers and Electrical Engineering, Elsevier, Volume 39, Issue 8, November 2013, Pages 2553–2565, doi: 10.1016/j.compeleceng.2013.08.011.

[17] D. Schwerdel, D. Hock, D. Günther, B. Reuther, P. Müller, P. Tran-Gia. ToMaTo - A Network Experimentation Tool. 7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2011), Shanghai, China, April 2011.

[18] N. Feamster, J. Rexford, S. Shenkerz, D. Levin, R. Clark, J. Bailey, SDX: A Software Defined Internet Exchange, White Paper, University of Maryland.

Program Tools and Language for Network Simulation and Analysis

A. Mikov
Department of Computing Technologies,
CubSU
Krasnodar, Russia
Alexander_ Mikov@mail.ru

E. Zamiatina
The Department of Information Technologies in Business,
HSE
Perm, Russia
e_zamyatina@mail.ru

Abstract— **This paper considers software tools and linguistic constructions of the network simulator TRIADNS. Nowadays network applications – especially in the area of wireless networks – are becoming more and more complex which makes the design and the testing almost impossible without appropriate software. This software available to aid the user in simulating previously designed scenarios, scalable algorithms and changing structure of computer network. So it is necessary to have effective and flexible program tools for computer network design and simulation. Network simulator must design and investigate not only hardware, but software too, explore computer networks, considering in particular the specific characteristics of a variety of computer networks. Besides computer networks may include a lot of nodes. This paper discusses approaches allowing to decide the problems mentioned above: hierarchical model, using ontologies and Data Mining methods for the analyses of simulation results, using several computing nodes for computer network simulation (distributed and parallel simulation).**

Keywords— simulation, computer networks, ontologies, routing algorithms, Data Mining, distributed and parallel simulation

I. Introduction

Computer networks are very wide spread now. Indeed computer networks are used in information systems, Grid computing, cloud computing and so on.

Widespread computer networks impose requirements to the speed and reliability of information transfer, to its effective treatment. For this reason, it becomes necessary to study traffic, to investigate new protocols, to design and develop new devices and new algorithms.

It is not always possible to apply analytical methods to investigate computer network because of the complexity of modeling object and, moreover, natural experiments can't investigate all aspects of this object too.

So the designers prefer to use simulation methods and appropriate program tools (network simulators). A lot of network simulators were developed recently [1]. We consider some of them below.

Because of complexity of modeling object (computer networks) simulators should have the following properties:

This work was supported in part by Russian Foundation for Basic Research (grants 12-07-00302, 13-07-96506).

- *Simulation experiment should be optimized in respect to time.* Indeed very often it is necessary to investigate large-scale networks with a tremendous amount of computing nodes. It is clear that the simulation of large-scale networks must be terminated within a reasonable time [2, 3]. But it is possible if one can perform simulation experiment on a supercomputer (cluster and so on). Besides, the investigators need the special software tools implementing special synchronization algorithm (conservative or optimistic), managing time advancement [4, 5, 6]. Moreover it is necessary to solve a problem of the equal workload on the computing nodes [7, 8, 9]. And nowadays new class of computer network simulators appears – there are simulators using graphical processors (GPU) [10].
- *A joint study of hardware and software of computer networks.* The computer network designers usually consider separately the hardware and software. However, the most appropriate solution would be to have software tools for design and analysis hardware, design and analysis of algorithms that control hardware, and for the co-design of hardware and software [11]. For example, it is very important to analyze the behavior of routing algorithm after the moment when the topology of computer network is changed (new computing node appears or some nodes become not accessible). In this case, the designer is interested in the topological characteristics of the network. These characteristics may affect the communication complexity of the algorithm. The structure of network may be represented as a graph. So it is important to investigate the structure of network using known graph algorithms (the shortest distance, for example). Nowadays the adaptable routing algorithms are applied in networks. These algorithms change their behavior depending on the values of certain characteristics of the network (overload of communication lines, for example). So it is advisable to simulate routing algorithm. Moreover it is important to simulate the behavior of various devices of computer networks and algorithms which control the behavior of these devices.
- *Adaptability of software simulators to incorporate into a simulation model new devices and new algorithms*

that govern their work. There are various software tools to design the computer networks nowadays. The most popular are: NS-2 [12] (the design of the local and global networks, multiprocessor and distributed computing systems, the ability to assess the performance of the designed system , etc.); OpNet [13] (a discrete event simulator that allows investigators to explore all levels of computer networks and to include customer modules into simulation model), OMNeT ++ [14], etc. Each of these simulators has specific characteristics. Some tools are designed to manage local networks, while others permit the design and analyses of global networks. Some of these software tools allow network designing, but have limited modeling capabilities, others are able to perform complex analysis of specific networks (may be only global networks or local or sensor ones). Network simulators have to be able to design, simulate and analyze new types of computer networks, new devices, new algorithms and technologies because of rapid development of network technologies.

The designers and developers of computer networks simulator TRIADNS tried to consider the experience of various software tools of this kind. This simulator is based on CAD Triad [15]. The ideas embodied in CAD system Triad allow it to adapt to rapid change of computer networks, new algorithms and technologies due to special linguistic and program tools:

- Linguistic and program tools for the description of the structure of computer networks and the behavior of the devices and computing nodes;
- Advanced analysis subsystem, which includes a library of standard information procedures (information procedures are obtained to collect the information about simulation model during simulation experiment and to process it) and linguistic tools to create new procedures and, therefore, new algorithms of analysis.

Furthermore, the effectiveness of the simulator is provided by distributed (parallel) simulation experiment (using the resources of several nodes of computer network, cluster or multiprocessor (the advantages of a distributed (parallel) simulation experiment are listed in [5, 16]). Optimistic synchronization algorithm (based on knowledge)(subsystem TriadRule) and load balancing subsystem (TriadBalance) are implemented in simulator TRIADNS. This software permits to reduce the time needed for simulation experiment.

Moreover the effectiveness of simulation system may be achieved by the subsystem of collecting and processing of the simulation model characteristics (the processing of data may be partly carried out during simulation experiments) and intelligent analysis of simulation results (based on the methods of Data Mining).

The flexibility of simulation software is achieved through the use of ontologies and the mechanism of redefining models, interoperability (including in the model components developed in the other modeling systems).

First of all, we should talk about how the simulation model is presented in the simulator TRIADNS, the architecture of simulator and the description of each it's subsystem.

II. Simulation model representation in TRIADNS

A. *Simulation Model and Three Layers*

Simulation model in Triad.Net is represented by several objects functioning according to some scenario and interacting with one another by sending messages. So simulation model is $\mu=\{STR, ROUT, MES\}$ and it consists of three layers, where STR is a layer of structures, ROUT – a layer of routines and MES – a layer of messages appropriately.

The layer of structure is dedicated to describe objects and their interconnections, but the layer of routines presents their behavior. Each object can send a message to another object. So, each object has the input and output poles (P_{in} – input poles are used to send the messages, P_{out} – output poles serve to receive the messages).

One level of the structure is presented by graph P = {U, V, W}. P-graph is named as graph with poles. A set of nodes V presents a set of programming objects, W – a set of connections between them, U – a set of external poles. The internal poles are used for information exchange within the same structure level; in contrast, the set of external poles serves to send messages to the objects situated on higher or underlying levels of description. Special statement <message> ***through*** <name of pole> is used to send the messages.

B. *The Layer of Structure*

One can describe the structure of a system to be simulated using such a linguistic construction:
structure <name of structure> ***def*** (<a list of generic parameters>)
(<a list of input and output parameters>)
<a list of variables description> <statements>)
endstr

The investigator may not describe all the layers. So if it is necessary to study structural characteristics of the model, only the layer of structures can be described. The example of computer network (the layer of structure) is given below. This computer network consists of a server and several clients.

Note, please, that the *layer of structure* is a procedure with parameters.

Triad-model is considered as a variable. Initially it may be void and further may be constructed with the special statements of Triad-language (operations within the layer of structures).

The structure of some computer network will include a different number of nodes and edges connecting them obtained as a result of operations on graphs. This number depends on the values of parameters of procedure ***structure*** or procedure ***routine***. These parameters can indicate the range of the transceivers, current time, and so on in the representation of wireless ad hoc networks. Thus, the description of networks in TRIADNS is varying in time and space. So it is corresponds to the idea of ad hoc computer networks, for example. Fig.1. gives the structure of network “Client_Server”. It consists of the node “Server” and the attached array of nodes “Client”.

The links between nodes are set within the cycle <*for*> with the help of arcs. Input and output poles have to be specified: (arc (Server.Send -- Client[i].Receive)). The number of nodes Client may be changed by formal parameter Number_of_Client.

```
Structure Client_Server[ integer  Number_of_Clients]
      def
 Client_Server := node Server<Receive, Send>
 + node Клиент[ 0 : Number_of_Clients - 1 ]
 < Receive, Send >;
       integer i;
         for i := 0 by 1 to Number_of_Clients  - 1 do
         Number_of_Clients := Number_of_Clients  +
       arc ( Client[ i ].Send -- Сервер.Receive ) +
          arc ( Сервер.Send -- Клиент[ i ].Receive );
         endf;
endstr
```

Fig.1. The Structure of layer for Client-Server description

C. Graphical Interface

There are two ways to describe model in Triad: via text editor or via graphical editor. The description of a layer of structure being built with the help of graphical editor is given below (fig.2.).

This description is a fragment of computer network. It consists of several workstations sending messages between them. Besides, the computer network includes the routers responsible for the searching of the route.

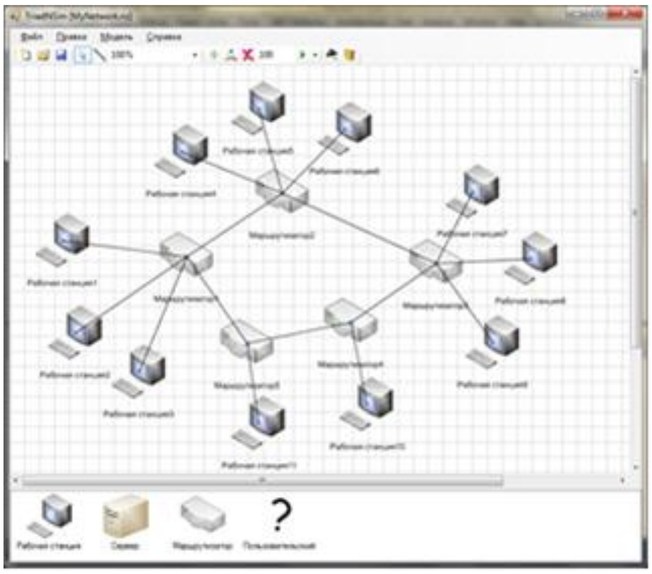


Fig.2. The fragment of computer network. Graphical editor

The description of this fragment of computer network being built with the help of text editor is given on fig.3.

```
Type Router,Host; integer i;
M:=dcycle(Rout[5]<Pol>[5]);
M:=M+node (Hst[11]<Pol>);
for i:=1 by 1 to 5 do
      M.Rout[i]=>Router;
      M:=M+edge(Rout[i].Pol[1] — Hst[i]);
endf
for i:=1 by 1 to 3 do
     M:=M+edge(Rout[i].Pol[2] — Hst[2*i-1]);
endf;
for i:=0 by 1 to 11 do M.Hst[i]=>Host; endf;
```

Fig.3. The fragment of computer network in Triad language.

D. Graph Constants

Simulation model (see fig.3.) is built using graph constants. A set of special linguistic units - graph constants - presents the basic types of topologies of computer network. In the text given above the graph constant “directed cycle” (Dcycle) was used.

E. Semantic Type

Besides, in above example the semantic types (Type Router,Host) were used. Namely they are “router” and “host”. The semantic types are used for simulation model redefining. More details will be given later.

F. Standard Procedures

There are the several standard procedures in the structure layer. The investigator is able to take out from the structure of model a lot of characteristics: a set of nodes, a set of arcs, a set of edges and etc. Moreover one can find the shortest distance between two nodes or cfn find connected components (procedure GetStronglyConnectedComponents(G)) or fulfill the selection of the structure layer (procedure GetGraphWithoutRoutines(M)) and so on.

Besides, the investigator obtains the linguistic and programming tools enabling him to write the absent procedure by himself. The investigation of the structure layer only is static process. The simulation process may take place only after the definition of the behavior of all nodes.

The behavior is determined by the statement ***Put***. The example will be given later. The investigator may take the description of the node’s behavior in repository (or via Internet) or may describe using special statements and linguistic construction of Triad-language.

G. The Layer of Routines

Special algorithms (named “routine”) define the behavior of an object. It is associated with particular node of graph P = {U, V, W}. Each routine is specified by a set of events (E-set), the linearly ordered set of time moments (T-set), and a set of states {Q-set}. State is specified by the local variable values. Local variables are defined in routine. The state is changed if an event occurs only. One event schedules another event. Routine (as an object) has input and output poles (Prin and Prout). An input pole serves to receive messages, output – to send them. One can pick out input event ein. All the input poles are processed by an input event, an output poles – by the other (usual) event.

So the formal rules of routine one can see here:

routine<name>(<a list of generic parameters>)(<a list of input and output formal parameters>)
initial <a sequence of a statements> *endi* event <a sequence of a statements> ***ende***
event <a name of an event> <a sequence of statements> ***ende*** …
event<a name of an event><a sequence of a statements> ***ende***
endrout

Let us return to the description of Client-Server model. Client behavior scenario is described with special linguistic unit which is named as “routine”. The syntax of routine is given above. One can see that the routine consists of initialization part, input event (without name) and several events (these events have names) scheduling one another. The description of the “Client” behavior is given below:

```
routine Client ( input Receive; output Send )[ real deltaT ]
initial boolean Quiery_is_Send;
   Quiery_is_Send := false; schedule Quiery in 0;
  Print "Client Initialization";
Endi
event Quiry; (* it is an event *) out "I send a quiry" through
Send; Print "A Client sends a quiry to Server";
        schedule ЗАПРОС in deltaT;
ende
endrout
```

Fig.4. The Routine “Client”.

The routine is a procedure with parameters too, it includes not only the interface parameters (input and output interface parameters “Receive” and “Send”, but the parameter deltaT- the time interval between the queries of Clients to Server). So the parameter deltaT may be changed during simulation experiment (in accordance with the behavior of real process, object or system of objects being investigated).

The instances of routine are formed by the statement ***let Client*** (clientDeltaT) ***be*** Client. An instance of routine may be “put” on an appropriate node with the help of statement: ***put*** Client ***on*** Model.Client[i]<Receive=Receive,Send=Send>. The input and output poles of routine are matched to the poles of node here. Consequently, the program tools of simulator become more flexible because of that fact that the investigator can change the behavior of some node during simulation experiment (statement ***simulate,*** it will be described below).

The simulation model is complete if all of nodes have appropriate routines and only complete model can take part in simulation experiment.

The behavior of routines may describe the algorithm functioning in some computational environment. The computational environment is described with the help of parameterized procedure ***structure***.

It is possible to select only the layer of structure (the layer of structure usually describes hardware of computer network), the layer of routines.

So TRIADNS permits to carry out the design and analyses of hardware (the layer of structure), the design and analyses of software (the layer of routines) and co-design of hardware and software (the complete model).

The linguistic constructions of parameterized procedures ***structure*** and ***routine*** allow to incorporate new devices and algorithms in simulation model.

III. Simulation experiment

The objects of simulation model are managed by the special algorithm during the simulation run. Let us name it as “simulation algorithm” (CAD system Triad has distributed version and corresponding algorithm for distributed objects of simulation model too) [15]. CAD system Triad includes analyses subsystem implementing the algorithm of investigation - special algorithm for data (the results of simulation run) collection and processing.

The analysis subsystem includes special objects of two types**:** *information procedures* and *conditions of simulation*. Information procedures are “connected” to nodes or, more precisely, to routines, which describe the behavior of particular nodes during simulation experiment. Information procedures inspect the execution process and play a role of monitors of test desk. *Conditions of simulation* are special linguistic constructions defining the algorithm of investigation because the corresponding linguistic construction includes a list of information procedures which are necessary for investigator.

The algorithm of investigation is detached from the simulation model. Hence it is possible to change the algorithm of investigation if investigator would be interested in the other specifications of simulation model. For this one need to change the conditions of simulation. But the simulation model remains invariant. We may remind that it is not possible in some simulation systems.

One can describe the information procedure as so:

information procedure<name>
(<a list of generic parameters>)
(<input and output formal parameters>)
initial <a sequence of statements> ***endi***
<a sequence of statements>processing <a sequence of statements>
endinf

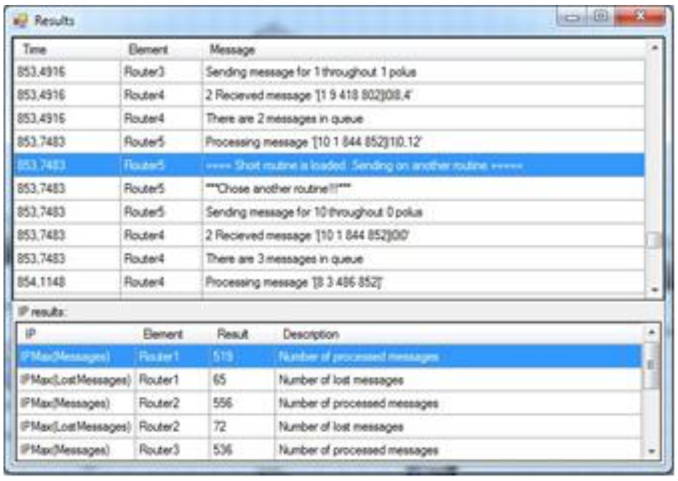
It is possible to examine the value of local variables, the event occurrence and the value of messages which were sent or received. A part of linguistic construction ‘processing’ defines the final processing of data being collected during simulation run (mean, variance and so on).

Let us present the linguistic construction conditions of simulation:

Conditions of simulation<*name*>
(<a list of generic parameters>)

(<input and output formal parameters>)
initial <a *sequence of statements*> ***endi***
<*a list of information* procedures>
<a sequence of statements>
processing <a sequence of statements>…***endcond***

The linguistic construction conditions of simulation describes the algorithm of investigation which defines not only the list of information procedures but the final processing of some information procedure and checks if conditions of simulation correspond to the end of simulation. The subsystem of visualization represents the results of simulation. One can see the representation of the results of simulation run at fig.5.


Results

Time	Element	Message
853,4916	Router3	Sending message for 1 throughout 1 polus
853,4916	Router4	2 Recieved message '[1 9 418 802](08.4'
853,4916	Router4	There are 2 messages in queue
853,7483	Router5	Processing message '[10 1 844 852](10.12'
853,7483	Router5	***** Short routine is loaded. Sending on another routine *****
853,7483	Router5	***Chose another routine!!!***
853,7483	Router5	Sending message for 10 throughout 0 polus
853,7483	Router4	2 Recieved message '[10 1 844 852](0.0'
853,7483	Router4	There are 3 messages in queue
854,1148	Router4	Processing message '[8 3 486 852]'

IP results:

IP	Element	Result	Description
IPMax(Messages)	Router1	519	Number of processed messages
IPMax(LostMessages)	Router1	65	Number of lost messages
IPMax(Messages)	Router2	556	Number of processed messages
IPMax(LostMessages)	Router2	72	Number of lost messages
IPMax(Messages)	Router3	536	Number of processed messages

Fig.5. The results of simulation.

Simulation run is initialized after simulation statement processing. One can pay an attention to the fact that the several models may be simulated under the same conditions of simulation simultaneously.

simulate <a list of an elements of models, being inspected>
on conditions of simulation <name>
(a list of actual generic parameters>)
[<a list of input and output actual parameters>]
(<a list of information procedures>
<a list of statements>…)
endsim

IV. THE COMPONENTS OF SIMULATION SYSTEM TRIADNS

Let us consider simulation modeling system TriadNS, its appointment, its components and functions of each component. TriadNS – it is simulation system dedicated for computer networks analysis. It is based on object-oriented simulation system Triad.Net. Simulation system Triad.Net is a modern version of previous simulation modeling system Triad [6] dedicated to computer aided design and simulation of computer systems. Triad.Net is designed as distributed simulation system (it may be consider as one of the class of PDES systems – parallel discrete event simulation), so various objects of simulation model may be distributed on the different compute nodes of a computer system. One more specific characteristic of Triad.Net – remote access, so several investigators may fulfill a certain project from different computers situating in different geographical points.

Distributed simulation system Triad.Net consists of some subsystems: compiler (TriadCompile), core of simulation system (TriadCore), graphical and text editors, subsystem of testing and debugging (TriadDebugger), subsystem of distributed simulation (synchronization of simulation model objects which are situated on different compute nodes of computer system, conservative and optimistic algorithms realization)(TriadRule), subsystem for equal workload of compute nodes (TriadBalance), subsystem of remote and local access (TriadEditor), subsystem of automatic and semiautomatic simulation model completeness (TriadBuilder), the subsystem for remote access and a security subsystem from external and internal threats TriadSecurity), the subsystem of automatically extending the definition of the model (TriadBuilder), the subsystem of intellectual processing of the results of simulation experiment (TriadMining). Initially we address to the specific characteristics of simulation model in TriadNS.

V. THE FLEXIBILITY OF THE SIMULATION TOOLKIT

A. Using ontologies in TRIADNS

It is important to involve into the simulation process not only the specialists in simulation but the specialist in specific domains and specialists in the other spheres of knowledge. That is why it is necessary to adjust a simulation system to specific domain. Indeed the investigator of computer network may use a graph theory while studying the structure of network, or a queue network theory, or the theory of Petri Nets. Ontologies are used in TriadNS to adjust the simulation system to specific domain.

Ontologies can be applied on the different stages of simulation [17, 18]. Very often ontologies are applied for the simulation model assembly. So the simulation model may consist of separately designed and reusable components. These components may be kept in repositories or may be found via Internet. The ontologies keep the information about interconnections of simulation model components and other characteristics of these components. Ontologies enable investigators to use one and the same terminology. Ontologies allow to make the repositories of components to store not only an information about their characteristics, interfaces, but the information about their interconnections.

The base ontology is designed in TriadNS. Its basic classes are: TriadEntity (any named logic entity), Model (simulation model), ModelElement (a part of simulation model and all the specific characteristics of a node of structure layer), Routine (node behavior), Message (note, please, that structure layer nodes of simulation model can interchange with messages) and so on.

The basic properties of base ontology are: (a) *the property of ownership:* model has a structure, a structure has a node, a node has a pole and so on; (b) *the property to belong to somethin0g* -– inverse properties to previous one. The structure belongs to the model, the node belong to strucrure, the pole belong to the node and so on; (c) *the properties of a pole and an arc connection* – connectsWithArc(Pole,Arc), connectsWithPole (Arc, Pole); (d) *the property of a node and* *an appropriate routine binding*-putsOn (Routine, Node); (e) *The properties of a node and an appropriate structure binding:* explicatesNode (Structure, Node), explicatedByStructure (Node, Structure); (f) *The property of the model and conditions of simulation binding* (Model, ModelingCondition).

The simulator TriadNS has some additional special subclasses of the base classes (specific domain – computer networks): (a) *ComputerNetworkModel* (a model of a computer network); (b) ComputerNetworkStructure (a structure of a computer network model); (c) *ComputerNetworkNode* (a computer network element, it contain several subclasses: Workstation, Server, Router); (d) *ComputerNetworkRoutine* (a routine of a computer network) и т.д. This ontology includes two special properties of a pole. These properties are used to *check* the conditions of matching routine to a node, for example a property check if it is necessary to connect a pole with another pole or a property checking the semantic type of an element of a structure being connected.

B. Redefining of Simulation Model

An ordinary simulation system is able to perform a simulation run for a completely described model only. At the initial stage of designing process an investigator may describe a model only partly omitting description of behavior of a model element μ_{r*} = {STR, ROUT*, MES}). Simulation model may be described without any indication on the information flows effectin0g the model (μ_{s*} = {STR*, ROUT*, MES}) or without the rules of signal transformation in the layer of messages (μ_{m*} = {STR, ROUT*, MES}). However for the simulation run and the following analysis of the model all these elements have to be described may be approximately.

For example, in a completely described model each terminal node $v_i \in V$ has an elementary routine $r_i \in$ ROUT. An elementary routine is represented by a procedure. This procedure has to be called if one of poles of node v_i receives a message. But some of the terminal nodes v_i of partly described model do not have any routines. Therefore the task of an automatic completion of a simulation model consists either in “calculation” of appropriate elementary routines for these nodes, i.e. in defining $r_i = f(v_i)$, either in “calculation” of a structure graph $s_i = h(v_i)$ to open it with (in order to receive more detailed description of object being designed). It was mentioned above that the routine specifies behavioral function assigned to the node, but the structure graph specifies additional structure level of the model description. And at the same time, all structures s_i must be completely described as the submodels.

These actions have to be fulfilled by the subsystem TriadBuilder. Subsystem TriadBuilder [19] attempts to search the appropriate routine by the help of base ontology (it was described earlier). It may be found thanks to special semantic type (semantic type “Router” and “Host”, for example). Model completion subsystem starts when the internal form of simulation model is built according to a Triad code.

First, *model analyzer* searches the model for incomplete nodes, and marks them. Thus, the model analyzer will mark all *Rout* nodes. After the inference module starts looking for an appropriate routine instance for each of marked nodes according to specification condition (the semantic type of node and routine must coincide).Then the condition of configuration must be checked (the number of input and output poles of node and the number of poles of routine must coincide). After the appropriate instance has been found, it may be put on the node.

C. Intellectual Analysis of the Simulation Experiment Results

It is well known that the goal of a simulation experiment is to obtain the most accurate and adequate characteristic of the studied object. This stage of simulation deals with data collection and processing. The special syntax units such as information procedures and conditions of simulation are designed in TriadNS. Information procedures and conditions of simulation are described above. Note, please, that data collection and data processing with the help of information procedures permit to obtain more adequacy results. Information procedures monitor only these characteristics of simulation model which are interested for investigator. In contrary some other simulators able to monitor and to collect a set of predefined characteristics.

But we can note another problem: the results of simulation experiment are not ordered and not structured. The processing of a simulation experiment results requires highly skilled analysts. So we can state the appearance of several papers with the suggestion to make the additional processing of the results of simulation experiments [20] and to apply the methods of Data Mining for these purposes [21]. Usually investigators obtain standard report with the results of simulation. The additional processing allow to find dependences between characteristics of the modelling objects.

The analyses of these dependences allow to reduce the overall data capacity, dimension of problem and eventually to optimize the simulation experiment.

The additional processing may be done with the special software tools of TriadNS (component TriadMining). TriadMining use the results of the information procedures, the results are processed with the help of regression analyses, time serious, Bayesian networks and so on. We mentioned above that an information procedure monitors the implementation of the sequence of events, the variables changing and so on. It is well known that the sequence of the predefined events allow to find crashes in nodes of telecommunication systems. Here is an example of information procedure.

```
information procedure event_sequence (in ref event
E1,E2,E3;out Boolean arrived)
initial interlock (E2,E3); Arrived := false;
 case of e1:available(e2);
   e2:available( E3):
          e3:ARRIVED:=true;
 endc
endinf
```

Fig. 6. The information procedure to detect the proper sequence of events.

So investigator may detect the arrival of the sequence of events E1→E2→E3. The statement ***interlock*** provides input parameter blocking (event E1 in this case). It means that information procedure doesn’t watch parameters being marked in interlock statement. The statement ***available*** allows beginning the marked parameter monitoring again.

Information procedure monitors the changing of variables and the moments of appropriate time. So the time series may be formed. It is necessary to analyze the similarity of two or more time series. So it is possible to find dependences between the elements of simulation model and reduce the data capacity.

VI. The Effectiveness of the Simulation Toolkit

A. Distributed simulation model representation

It is necessary to use several nodes of cluster, network or mainframe in order to design effective simulation toolkit. A distributed simulation model is presented as several logical processes carried out on different compute nodes in this case. Logical processes are functioning and interacting with one another sending and receiving messages. These messages have the time stamps – the local time of the event being carried out.

B. Optimistic and conservative algorithms

There are two main approaches to provide the causality of the events in parallel/distributed simulation: conservative and optimistic. Conservative algorithm defines the time of the "safety" event from the list of scheduled and not processed events. One may name the event as "safety" if a logical process does not receive message with lower time stamp than the time stamp of event from the list of scheduled events. Conservative algorithm does not process event if it is not safety. More details are given in papers [4, 5]. Optimistic algorithms allow carrying out the logical process without local causality restrictions. One of the famous optimistic algorithms is a Time Warp [22]. When a logical process receives an event with the lower time stamp than the time stamp of processed event the process performs a rollback and processes this event again in the chronological order. Time Warp algorithm uses the mechanism of anti-messages.

The analysis of improved conservative and optimistic algorithms shows that their efficiency becomes higher due to increase of knowledge about the model (lookahead, lookback, time stamp of the next event and so on). So it is necessary to use the information about the model more precisely, the knowledge of a researcher about the behavior of specific model for increase of the simulation experiment efficiency.

C. Knowledge Based Sinchronization Algorithm

Usually a researcher has some knowledge about the specific behavior of the model. We propose to present this knowledge as production rules in the knowledge base. Rules may be presented as: IF e1 AND e2 AND e3 AND … AND en THEN ek CF <0..100>. These rules show that the event ek depends on e_1, e_2, ,e_n. CF is a trust coefficient. 0 - no trust, 100 - maximum trust. The rules reflect the causality between events, but the events are not exact, that is why each rule is assigned a trust coefficient.

However it is not enough to relay upon the knowledge of a researcher. Some knowledge has to be received within the simulation experiment in order to replenish and to improve rules in the knowledge base. The authors have developed the specific program tools (TriadRule) to collect and to process the specific knowledge required to replenish and to improve the production rules in the knowledge base. The application of TriadRule shows that efficiency of the optimistic algorithm is actually increased.

D. Load balancing subsystem

Load balancing subsystem is dedicated to optimal distribution of program model among compute nodes (in multiprocessor computer or in network) and consequently to enhance the performance of these computers.

Load balancing it is a problem of non-isomorphic vertex-connected graphs mapping B: PM $\rightarrow$ NG, where PM – a set of graphs of program models, NG – a set of graphs – computer network configurations. Graph G $\in$ NG, G = {C, Ed}, can be defined by a set of calculating nodes C and a set of edges Ed (edges Ed are associated with communication lines). One can consider NG as a super graph, containing all eventual (admissible) graphs G_i as subgraphs. Graph M $\in$ PM represents program model.

It is possible to use three kinds of load balancing: static Bs, dynamic (automatic) Ba and dynamic (controlled) Bc. Preliminary allocation of program objects (static Bs) is not effective. This is explicable from the following facts: (a) a program model can be changed due to new processes appearance, terminating some processes; (b) a compute environment can be changed because one or the several processors (or computers) are failed. In any case, the benefit of distributing the logical processes between compute nodes before the program execution is very often not seen.

In regards to dynamic balancing Ba the graphs G and M are considered to be loaded. The nodes of the first graph have a parameter – performance, edges – data rate. The characteristics of nodes in the second graph – time complexity, the characteristics of edges – the intensity of a traffic flow. The weights of nodes and edges in graph NG are considered to be known. The corresponding graph M parameters must be defined during the program execution. The "bottle neck" of the program model and computer system is determined in accordance with some algorithm, and migration of the program objects without interruption until the program is executed.

There is a new approach of implementation of controlled dynamic load balancing based on knowledge in Triad.Net (B_c). Controlled dynamic load balancing subsystem includes expert component and information procedures developed by a model designer (nonstandard information procedures in other words). Expert component consists of optimization rules defined by the author of the given model (or of class of models). Nonstandard information procedures are intended to estimate the events (or conditions) of rule applications.

Restoring the balance of the workload is a well-known problem. There are several solutions of this problem and a lot of algorithms were developed. However, very often, these algorithms are applicable only for a specific simulation model. Researchers attempted to develop adaptable algorithms (SPEEDES[7] and Charm ++[8], for example). The experiments have shown that the effectiveness of these algorithms maybe achieved only in special cases. Indeed, the development of some universal algorithm is almost impossible. The authors wish to solve at least partially this problem by applying a controlled balance. Controlled load balancing uses the knowledge of concrete simulation model. For example, the researcher knows that the intensity of data exchange between two compute nodes would be much higher after one hour from the beginning of computer network functioning. So a researcher may formulate the appropriate rule which can be used by load balancing subsystem.

There are two ways to implement controlled dynamic load balancing subsystem: in centralized manner or in distributed one.

The knowledge-based load balancing subsystem in Triad.Net includes: (a) *Expert system* with knowledge base, rules editor, inference engine and module of explanations. Knowledge base consists of rules for optimal distribution of program model objects among the calculating nodes. (b) *Simulation model and computing environment analysis subsystem*. Analysis subsystem consists of information procedures to collect data: a frequency of interchanges among the objects, a frequency of event occurrence and etc; to collect data on computational environment (flow capacity of communication lines, workload of computers). (c) *Subsystem for simulation model and calculating environment visualization*. (d) *Migration subsystem* which carries out program object migration from one compute node to another.

Expert component carries out some operations on graph G (this graph represents the structure of program model) mapped on graph M – graph of computing environment.

Rules imply operations on graph G. Rules are productions such as «if then else…» and could be described by Triad language.

But this controlled load balancing uses centralized algorithm, all rules are in single knowledge base, simulation model and computing environment analyses subsystem is situated on a selected compute node too and interacts with other compute nodes. Authors of this paper propose multi-agent approach in order to reduce the time needed to exchange the data.

E. Multi-Agent approach

Dynamic multi-agent load balancing subsystem TriadBalance consists of different agents: (a) Agent-sensor of compute node; (b) Agent-sensor of simulation model; (c) Agent of distribution; (d) Agent of migration; (e) Agent of analyses. Each agent works in accordance to its scenario, but together they carry out the load balancing algorithm.

More precisely: (a) *The agent-sensor of compute node* permanently collects data about the state of a compute node (computational load on the node and link capacity). (b) *The agent-sensor of simulation model* permanently monitors a simulation model during the simulation run, recording the intensity of the exchange between the objects, the frequency of certain events, the rate of change of variables, etc. Agent-sensor of the simulation model uses *information procedures*. *The agent of analysis* interacts with agents-sensors (these agents are reactive) and decide if it is necessary to distribute the load or not. It is a cognitive object and it uses the rules of expert system in order to make a decision.

The agent of the distribution receives information from the agent of analysis. The purpose of the agent of the distribution is to define a portion of the load (it is needed to select some objects of simulation model located on compute node) which should be referred to other nodes in order to avoid imbalances, and to identify the target compute node for transfer a portion of load.

In order to identify the target node it is necessary to check the load on the neighbor nodes. If the node with the lowest load is not found, the distribution agent tries to find the address of the node from its neighbors. If the load of the compute node is less than the limit then the agent of distribution informs its neighbors that compute node may place the additional load. Agent of distribution is a cognitive one and it acts in accordance to rules from the knowledge base. These rules are defined by a modeler and are corrected during the simulation run. The knowledge base includes the information about all neighbors of a concrete compute node and this information must be updated during the interaction with neighbors.

The *agent of migration* must transfer the selected portion of load to a target node and perform it in optimal way.

Cognitive agents have to be adapted to the conditions which could be changed during simulation run. For this purpose the meta rules were designed.

The experiments show that multi-agent load balancing subsystem reduces the time of simulation run. One can see it in the figures 7 and 8.

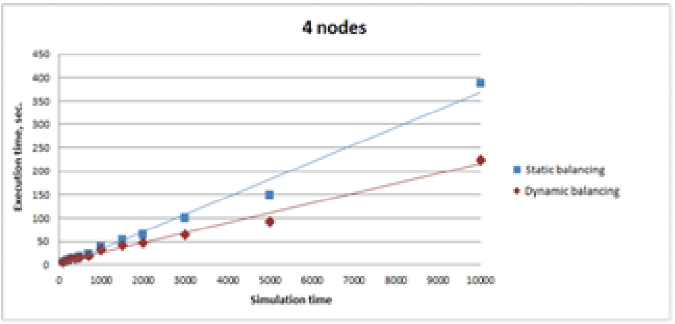


Fig.7. Static and dynamic multi-agent load balancing with 4 compute nodes.

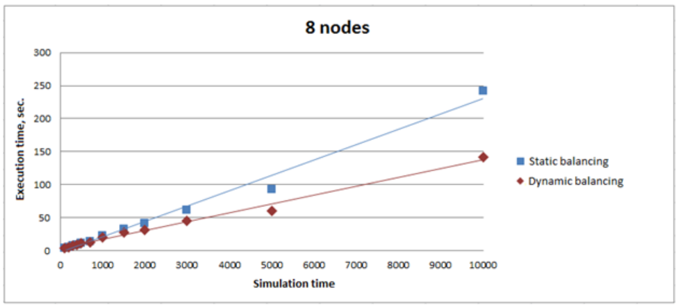


Fig.8. Static and dynamic multi-agent load balancing with 8 compute nodes.

VII. Conclusion

The paper discusses the problems of flexible and effective software for computer network simulation. Authors consider ontology approach application to automatic redefining of simulation model and to adjusting the simulation system to the specific domain.

Simulator TRIADNS is provided with a convenient graphical interface. Simulator permits separate and joint hardware and software modelling. Another distinguished characteristic of the simulator is the ability to make a distributed simulation experiment.

The Data Mining methods allow to simplify the analyses of the simulation experiment results. Ontologies enable to automate the simulation model construction and to achieve the interoperability of the software tools (to use components designed in the other simulation systems).

Authors suggest special optimistic algorithm and load balancing based on knowledge in order to reduce the overall time of simulation experiment. So software being under consideration is effective and flexible.

References

[1] S. Salmon, H.Elarag. Simulation Based Experiments Using Ednas: The Event-Driven Network Architecture Simulator. In Proceedings of the 2011 Winter Simulation Conference S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds. The 2011 Winter Simulation Conference 11-14 December 2011. Grand Arizona Resort Phoenix, AZ, pp. 3266-3277.

[2] A.I.Mikov, E.B.Zamy0atina The simulation model technologies for big systems investigation // In Proceedings of the Scientific Conference "Scientific service on the Internet" – M.: MSU, 2008. C.199-204.[in Russian]

[3] Y.Liu, Y.He. A Large-Scale Real-Time Network Simulation Study Using Prime. M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, eds. The 2009 Winter Simulation Conference 13-16 December 2009. Hilton Austin Hotel, Austin, TX, pp. 797-806.

[4] Riley, R.M. Fujimoto, M. Ammar. A Generic Framework for Parallelization of Network Simulations", in Proc. 7th Int.Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1999, p. 128-135.

[5] R.M. Fujimoto Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. The 2003 Winter Simulation Conference 7-10 December 2003. The Fairmont New Orleans, New Orleans, LA, pp. 124-134

[6] E. Zamyatina, S. Ermakov. The Synchronization Algorithm of Distributed Simulation Model in TRIAD.Net. Applicable Information Models. ITHEA, Sofia, Bulgaria, 2011, ISBN: 978-954-16-0050-4, pp.211-220.[in Russian]

[7] L. F. Wilson, W. Shen Experiments in load migration and dynamic load balancing in Speedes // Proc. of the Winter simulation conf. / Ed. by D. J. Medeiros, E. F.Watson, J. S. Carson, M. S. Manivannan. Piscataway (New Jersey): Inst. of Electric. and Electron. Engrs, 1998. P. 487‑490.

[8] G. Zheng Achieving high performance on extremely large parallel machines: Performance prediction and load balancing: Ph.D. Thesis. Department Comput. Sci., Univ. of Illinois at Urbana-Champaign, 2005. 165 p. [Electron. resource]. http://charm.cs.uiuc.edu/.

[9] A.I.Mikov, E.B.Zamyatina, A.A.Kozlov The Multiagent Approach to the Equel Distribution of the Workload. Natural and Artificial Intelligence, ITHEA, Sofia, Bulgaria, 2010, pp.173-180.

[10] L. Djinevski., S. Filiposka, D.Trajanov Network Simulator Tools and GPU Parallel Systems. In Proceedings of Small Systems Simulation Symposium 2012, Niš, Serbia, 12th-14th February 2012, pp.111-114

[11] W.Hu, H.S. Sarjoughian A Co-Design Modeling Approach For Computer Network Systems. . In Proceedings of the 2007 Winter Simulation Conference S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds. The 2007 Winter Simulation Conference 9-12 December 2007 J.W. Marriott Hotel, Washington, D.C., pp. 124-134

[12] [NS-2. 2004] The Network Simulator - NS-2. Доступно на сайте: http://www.isi.edu/nsnam/ns [Проверено 21 марта 2012]

[13] [OPNET, 2004] OPNET Modeler. Доступно на сайте: <http://www.opnet.com> [Проверено: 21 марта 2012]

[14] [OMNeT++, 2005] OMNeT++ Community Site. Доступно на сайте: http://www.omnetpp.org. [Проверено: 21 марта 2012]

[15] A.I. Mikov Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages, Las Vegas, USA, 1995. pp. 15-20.

[16] R.E. Nance Distributed Simulation With Federated Models: Expectations, Realizations And Limitations. In Proceedings of the 1999 Winter Simulation Conference. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds., The 1999 Winter Simulation Conference 5 – 8 December 1999 Squaw Peak, Phoenix, AZ, pp. 1026-1031.

[17] P Benjamin., K.V Akella., K Malek., R Fernandes. An Ontology-Driven Framework for Process-Oriented Applications // Proceedings of the 2005 Winter Simulation Conference / M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.,– pp 2355-2363

[18] P. Benjamin.,M. Patki, R. J Mayer. Using Ontologies For Simulation Modeling // Proceedings of the 2006 Winter Simulation Conference/ L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds. –pp.1161-1167

[19] A.Mikov A., E.Zamyatina, E. Kubrak. Implementation of simulation process under incomplete knowledge using domain ontology. In proceedings of 6-th EUROSIM Congress on modeling and Simulation. 9-14, September, 2007, Ljubljana, Slovenia, Vol.2. Full papers, 7 pp.

[20] G. Neumann, J.Tolujew , From Tracefile Analysis to Understanding the Message of Simulation Results, proceeding of the 7th EUROSIM Congress on Modeling and Simulation, Prague, Czechia, 2010, 7 pp.100-117

[21] T. Brady, E.Yellig, Simulation Data Mining: a new form of simulation output, 37th Winter Simulation Conference, Orlando, USA, 2005, pp 285-289.

[22] D.Jefferson, H.Sowizral, Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control // Rand Note N-1906AF, Rand Corp., Santa Monica, Cal., 1982

Application and Device Specific Slicing for MVNO

A. Nakao, P. Du
The University of Tokyo
nakao@nakao-lab.org, ping@nakao-lab.org

***Abstract*—In this paper, we apply the concept of software-defined data plane to defining new services for Mobile Virtual Network Operators (MVNOs). Although there are a large number of MVNOs proliferating all over the world and most of them provide low bandwidth at low price, we propose a new busi- ness model for MVNOs and empower them with capability of tailoring fine-grained subscription plans that can meet users' demands, for example allocate abundant bandwidth for some specific applications, but the rest of the applica- tions are limited to low bandwidth. For this purpose, we propose *application and/or device specific slicing* that clas- sify application and/or device specific traffic into slices and apply fine-grained quality of services (QoS). We also intro- duce various applications of our proposed system.**

Keywords—Software-Defined Networking (SDN), Network Functions Vir- tualisation (NFV), Network Virtualization

1. INTRODUCTION

Software-Defined Networking (SDN) and Network Functions Virtualization (NFV) have recently caught attentions from industries as technologies for reducing capital expense (CAPEX) and operational expense (OPEX), where software-defined programmable network equipment dispenses with high maintenance cost of hardware appliances and enables rapid revisions of functionalities and the automation of operation and management (OAM) of network. While SDN primarily focuses on the programmability on the control of networking, NFV aims at implementing data processing functions in software on top of virtual machines (VMs) especially that exist today as hardware network appliances. Data packets can be *programmatically redirected* by SDN and can be *programmatically processed* by NFV.

We have recently posited that *software-defined data plane*, i.e., arbitrarily defining data plane by software programming, significantly enhance the synergy between SDN and NFV [7]. In carefully designed sandboxes such as virtual machines inside network equipment, we should be able to enhance the data plane functionalities, e.g., those related to OAM, and publish the SBI for controllers to use them. Such enhancement is only recently discussed in a few research projects [1, 4]. Also, NFV is so far limited to implementing network appliances in software, and deals neither with crafting new protocols nor with OAM functionalities. Since the current SDN's data plane is not so much flexibly programmable because it is still often implemented in hardware, enhancing SDN with software-defined data plane would fill the gap in the current NFV.

In this paper, we apply the concept of software-defined data plane to defining new services for Mobile Virtual Network Operators (MVNOs) that obtain network services from mobile network operators and resell network services to customers at their own prices without owning the wireless network infrastructure on their own. There are a large number of MVNOs proliferating all over the world and most of them provide low bandwidth at low price. We propose a new business model for MVNOs and empower them with capability of tailoring fine-grained subscription plans that can meet users' demands, for example, abundant bandwidth is allocated for some specific applications, but the rest of the applications are limited to low bandwidth. To this end, we propose *application and/or device specific slicing* that classify application and/or device specific traffic into slices and apply fine-grained quality of services (QoS).

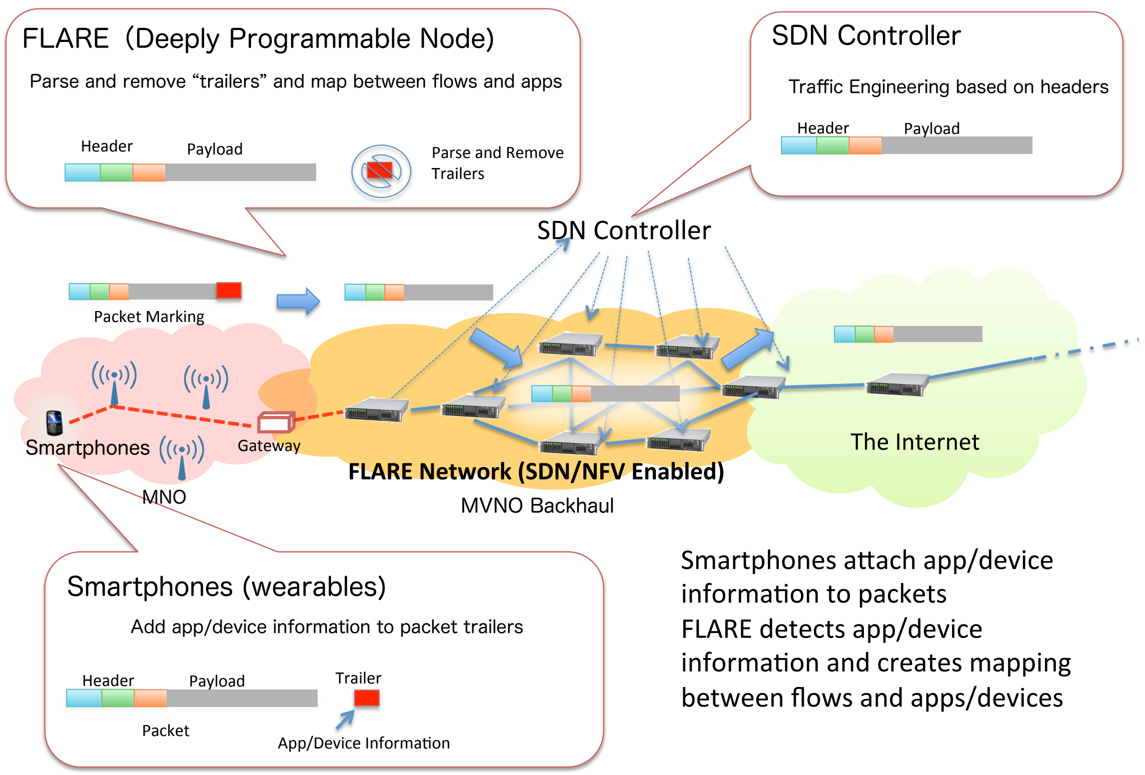
The rest of the paper is organized as follows. Section 2 introduces our design decisions for enabling application and/or device specific slicing utilizing programmable software-defined data plane. Section 3 discusses various applications of our proposed system. Section 4 introduces our programmable network node architecture called FLARE and shows our preliminary prototype implementation and experiments. Finally, Section 5 briefly concludes.

2. DESIGN

This section introduces our preliminary design for application and device specific slicing to enable Software-Defined Networking and Network-Functions Virtualization for MVNO.

2.1 Overview

In order to realize application and/or device specific slicing, we have designed *trailer slicing* [1] where *meta information*



FLARE (Deeply Programmable Node)
Parse and remove "trailers" and map between flows and apps
Header
Payload
Parse and Remove Trailers
SDN Controller
Traffic Engineering based on headers
Header
Payload
SDN Controller
Packet Marking
Smartphones
Gateway
MNO
FLARE Network (SDN/NFV Enabled)
MVNO Backhaul
The Internet
Smartphones (wearables)
Add app/device information to packet trailers
Header
Payload
Trailer
Packet
App/Device Information
Smartphones attach app/device information to packets
FLARE detects app/device information and creates mapping between flows and apps/devices

Figure 1: Application/Device Specific Slicing

on applications and devices at the end of packets (as discussed in more detail in Section 2.3.1.) Note that in our design, the meta-information may include many other kinds of information, but for the sake of brevity, we limit its scope to applications and devices within this paper.

In generic trailer slicing, each packet may carry trailer bits containing meta information of the packet, such as from which application process and/or from which device a packet has been transmitted. For example, we install our software on smartphones for capturing the very first packet an application transmits, i.e., a TCP SYN packet when the application establishes a TCP session, and then for attaching trailer. In more detail, we capture the header information of a TCP SYN packet and examine the process table and the socket table of the operating system to look for a corresponding application process that uses the flow space (such as IP addresses and port numbers) and attach the information regarding the application process such as a name and status as a trailer. Note that this approach can be easily applied also any other transport protocol such as UDP. Also we can attach device information as well as that of application.

A programmable node, e.g., FLARE (explained in 4.1) at the gateway to the MVNO backhaul network detects the trailer attached to the unusual TCP SYN (with non-zero payload size) or the packet that uses a flow space for the first time, and decodes/removes the information in the trailer. It also observes the flow space information of the packet at the same time and maps the information on the application processes and that of the flow. When the SDN controller can receive this mapping information, the subsequent packets can be controlled by the SDN switches along the route to the destination according to the flow space information associated with application/device information. For example, we can perform QoS traffic control such as bandwidth throttling for particular applications/devices using the traditional flow-based traffic control.

2.2 Filling a Gap between Application/Device and Network Programming

Although SDN and NFV are considered useful tools for programmable networking, we observe a gap between development of applications, services and devices, and that of programmable networking, mainly caused by the gap between abstractions defined in two worlds.

In the current Internet, applications and services implemented on end systems use socket interface to utilize services provided by the communication infrastructures. Since socket interface provides clear separation between end-systems and networking, the context of applications, services and devices are dismissed when packets are transmitted into the network. In other words, unless performing deep packet inspection (DPI) on packets or inferring from various characteristics such as packet length and timing, it is difficult to tell which application context sends/receives those packets.

Operating systems on top of end-systems use processes and threads as abstraction for programming applications and services. The current SDN networking equipment uses flow information as abstraction for programming network. In a sense, our proposal for application/device specific slicing bridges the abstractions used in operating systems and programmable networking.

2.3 Slicing Mechanism

2.3.1 Trailer-Slicing

The idea of trailer slicing is to attach a slice identifier at the end of the packets under the agreement of the existence of such bits among the users of the infrastructure, for example in mobile backhaul networks, at cloud data centers, and in any other administrative domains where the agreement may be established. As briefly explained in 2.1, a slice identifier may not just be an explicit number, but can be the meta information to identify a slice such as application name or device type under the agreement.

In SDN, we have been using the header information to define a slice, specifically, so-called flow information, which is a combination of MAC addresses, IP addresses and port numbers. However, when we consider cooperation between operating system entities and networking, we conclude that we should use a more straightforward identifier, such as a process name, an application name and a device type, etc. We can define a name space so that within the name space a slice can be identified uniquely, e.g., `com.android.google.youtube` in case of the name space for process names in the Android operating system.

The idea of trailer slicing is similar to MPLS in that we use bits (that can be view as label) for switching, but the difference is that the position of bits in layers and in packets and the length of the bits. We intentionally put a slice identifier at the end of packets. With adjustment of header fields in L3 and/or L4, we can get packets through with trailers through the existing network equipment, since they treat trailers as L7 data bits. Of course, we need to remove trailers before packets reach the destination, but that should be taken care of by the agreement of trailer slicing among administrative domain.

One may argue that we could use header option fields instead of a trailer for storing a slice identifier. However, there is a risk that non-standard header options may be removed or may cause network equipment to malfunction. Also option fields may be in short of bits, flexibility and extensibility. To avoid pressing header handling on the part of legacy network equipment, we decide to use a trailer since all the network equipment along the route of a packet treats a trailer as a part of payload data, so it keeps preserved till it gets parsed and removed. However, our scheme could be easily implemented in header options of course, when the concerns above are not an issue.

Note that not all packets need to carry trailers, although such design is certainly possible. As long as we agree on which packet in a flow carries a slice identifier, we can establish mapping between the traditional flow information and the slice identifier in network equipment. After the mapping is created, from then on, flow information could be used for the slice identifier.

As an aside, there is an interesting use case of trailer slicing called TagFlow [5], where we push expensive complex classification to the edge of the network and use one field trailer to simplify the classification at the core of network. In TagFlow, every packet is expected to carry a trailer.

2.3.2 TCP-SYN Piggy-backing

Some may argue that piggy-backing data in TCP SYN may render incompatibility and security issues. However, such unusual piggy-backing is not uncommon today. For example, Google does this in TCP Fast Open (TFO) [8] for the different purpose than ours, where they attempt to reduce the number of packets and the delay in three-way handshaking, storing “cookies” in newly emitted TCP SYN’s payloads for already authenticated end systems via the past three-way handshakes.

2.4 Signaling between End-Systems and Network

2.4.1 Out-of-band Signaling

Even if applications keep track of their flow information, they need to let the SDN controller know the flow information out of band, that is, besides the application data traffic, they must open control channels to convey such flow information to the SDN controller so that they may be able to control their flows. This approach is prohibitive for a large number of small devices such as smartphones and sensors since it may become significant overhead for them.

2.4.2 In-band Signaling

We propose a method to modify operating systems of the end systems such as smartphones, so that we can find application process information and convey such information through an in-band communication. Our prototype system attaches the application process information as a trailer on the part of the end systems, and decodes the information in it then removes it on the part of the programmable node located in the backhaul, ideally at the first hop from the gateway from a mobile network operator (MNO). In this way, we learn the mapping of the information on application processes and flows and inform the SDN controller so that subsequent nodes can just perform the conventional flow-based traffic control.

2.5 Deep Data Plane Programmability

We should note that in order to enable application/device specific slicing for MVNO, data-plane functionality must be extended from the current SDN model where data-plane elements have limited pattern match capabilities and too few actions. Especially note that the manipulation of the packet trailer at Layer 7 (L7) is largely missing from the current SDN data-plane elements and the extension to support such manipulation is useful to enable new applications.

3. APPLICATIONS

3.1 Traffic Engineering

Traffic engineering such as Quality of Service (QoS) and route/switch control for specific applications and devices is the immediate application of our proposal in this paper. We can create slices according to (1) *application names*, (2) *application processes*, (3) *device types*, and (4) *device status/location*, etc. However, it is obviously possible to extend a trailer to include much more information about applications, devices, the context of usage of them, etc.

3.2 Value-Add Services

After classifying application and/or device specific traffic into slices, we can apply NFV virtual functions to perform useful data processing such as *compression and decompression* and *packet caching*, etc. This application helps differentiating competing applications such as web browsers. For example, a certain browser can benefit from installing transparent data cache near smartphones, while other browsers may not. We expect more and more applications on smartphones can be empowered by small, yet smart functionalities embedded in NFV for aiding the operations of the applications.

3.3 In-Network Security

Another interesting example application is in-network security. *Malware containment in a slice* is one example application. In our prototype, as long as malwares on the smartphones transmit packets, we can catch the traffic from those processes and contain the traffic into a slice, by examining the application process names associated with flows. Our prototype system even raise alerts to smartphones that they may have accidentally installed malwares on them once their traffic get detected.

Also, *in-network parental control* is another example application in the security area. Usually, parents would like to restrict the usage of applications on their childrens' smartphones by installing parental control software on them. However, in most of the cases, those applications may be removed easily by the children. In our system, since application and device specific traffic can be classified into a slice, we can easily set policy and control bandwidth such traffic. For example, the traffic from a specific application on a specific device can be controlled on the part of network, not on the device, for a determined period of time. The parental control enabled by this mechanism is not easily removed by the hands of the children.

3.4 Big-Data Analysis

Neither capturing nor deeply inspecting users' traffic are allowed in several countries such as Japan. However, MVNO operators are interested in collecting application specific bandwidth usage to provide more fine-grained subscription plan. We intentionally design our system so that the privacy of user data (L7 payload data) may not be infringed. If users are fine with their application usage being collected, we believe we can alleviate the dilemma between MVNOs' demands for bandwidth usage data and users' privacy. Most of the related work for identifying applications from the traffic trace relies on deep packet inspection (DPI) of the user data, which may not work if DPI is restricted by law or the packet payload data is encrypted.

There are lots of MVNOs proliferating in Japan, but most of them offer low bandwidth at cheap price, which causes fighting for selling ever-lower-cost subscription plans among those MVNOs. We believe an MVNO may be able to create a fine-grained and tailored subscription plan that can meet users' demands, for example, provisioning bandwidth for some specific applications, but the rest of the applications are limited to low bandwidth. In order to come up with viable subscription plans, application traffic analysis becomes a key.

4. PRELIMINARY EVALUATION

4.1 FLARE

FLARE is a deeply programmable network node architecture [1] utilizing a hybrid of computational resources, such as network processors, general purpose processors, (and optionally GPGPU) hierarchically to extend data plane processing functions easily by software program.

FLARE tackles three research challenges, (1) ease of programming, (2) reasonable and predictable performance, and (3) enabling multiple concurrent isolated logics. For (1), we introduce Toy-Block networking programming model [6] to facilitate drag and drop data plane programming. For (2), we combine of high-frequency small-number-core processors for control and management functionalities, and low-frequency many-core processors for massively parallel processing for a large number of flows. And finally, for (3), we employ a lightweight resource virtualization technique called resource container for isolation of multiple logics. For the best isolation, we decide to partition many cores into groups and deploy a resource container per group.

The goal of FLARE is similar in spirit to that of OpenDataPlane [3], especially in that the purpose is to flexibly and easily extend data plane. However, the key difference is that we consider isolation of resources to support multiple concurrent data plane logics via virtualization. For example, FLARE program multiple concurrent logics such as OpenFlow 1.0 and OpenFlow 1.3 data plane elements in isolated execution environments.

4.2 Preliminary Prototyping and Experiments

Utilizing FLARE prototypes, we have implemented our prototype system to enable application and/or device specific slicing for MVNO as shown in the overview of our design depicted in Section 2.1. We have developed Android smartphone software to enable trailer slicing, i.e., embedding a slice identifier at the trailer of TCP SYN packets and QoS traffic engineering per slice on our FLARE platform [1]. We have discovered that we can use TCP SYN trailers unless ISPs do not filter unusual TCP SYN in fear of SYN Flooding, which is not really performed in most MVNO services of today.

In implementing our prototype, we reconsider southbound API (SBI) for your application. As reviewed in Section 2.2, we believe application users and developers, process-based traffic control is more natural than flow-based one. Extending the Openflow model, the right abstraction for programming in this case may be one such as,

<Application/Device><Action><Stat>,

instead of

<Flow><Action><Stat>,

although we may not have to follow OpenFlow's convention for programming abstraction and also one could rather define one's own programming abstraction, as long as it is open and published as an API.

We also jointly operate our prototype system with an ISP in Japan with 40 Android phones and successfully demonstrate our prototype system works on top of an MVNO. We plan to extend our experiments to enable various application ideas shown in Section 3. Note that the same prototype but with WiFi network has been demonstrated successfully at various venues such as GEC20 [2]. We believe that empowering MVNOs with application/device specific traffic engineering would become the norm of the next generation MVNO business.

5. CONCLUSION

Our contributions in this paper are four-fold.

First, we propose *application and/or device specific slicing* applying the concept of software-defined data plane to defining new services for MVNOs. More specifically, we use software-defined deeply programmable data plane to handle trailer bits to attach a slice identifier, so that we can classify application and/or device specific traffic into slices and apply fine-grained quality of services (QoS). Most MVNOs of today provide low bandwidth at low price and thus, they are forced into fighting for the market with ever-lower-cost subscription plans. However, low flat-rate bandwidth services may not be attractive to users any more, since certain applications, such as a YouTube browser needs more bandwidth than low flat-rate bandwidth. Our solution can provide pay-as-you-go bandwidth services for a specific set of applications of customers' choice, and low flat-rate bandwidth services for the rest of applications. We expect our proposed system change business models of MVNOs and enhance the market of the MVNO business.

Second, we also introduce various applications of our proposed system, e.g., (1) traffic engineering based on application names, application processes, device types, and device status/location etc., (2) value-add services for specific applications and devices such as acceleration of content access and traffic reduction through compression/decompression and packet caching, (3) realizing in-network security such as malwares containment in a slice, and in-network parental control, and finally (4) big-data analysis to improve the bandwidth utilization according to the statistical usage of applications and devices.

Third, our contribution includes not only providing new services for MVNOs, but also pointing out a compelling use case of software defined data plane, which is extended from the current SDN and NFV for allowing one (1) to define useful data processing within data plane in SDN and (2) to publish the access method to them as a (sub)set of southbound interface (SBI). We strongly believe that there are more and more useful use cases of software-defined data plane.

At last, another contribution is, while most people are paying attention to OPEX/CAPEX reduction in SDN/NFV, we attempt to create new values out of applications of SDN/NFV. For this, we believe that it is important to think *application-driven programmable networking* where starting from the application that cannot be built without the help from the in-network functions, i.e., the network functions embedded inside the data plane of SDN solution. Developing generic infrastructure to accommodate all the applications, that is, bottom-up approach may not correctly define APIs. It is important to think top-down, from applications that do not exist today due to the limitation in the network, down to defining what are necessary inside the data plane of SDN.

We strongly believe that enabling deeper programmability in SDN data-plane with ease of programming and reasonable performance surely open the door to bringing more innovations.

6. REFERENCES

[1] Flare: Deeply programmable network node architecture. `http://netseminar.stanford.edu/10_18_12.html`.

[2] Geni engineering conference 20. `http://groups.geni.net/geni/wiki/GEC20Agenda/EveningDemoSession`.

[3] Opendataplane. `http://www.opendataplane.org`.

[4] Protocol oblivious forwarding. `http://www.poforwarding.org`.

[5] H. Farhady and A. Nakao. Tagflow: Efficient flow classification in sdn. *IEICE Transactions on Communications*, 97(11), 2014.

[6] M. Fukushima, Y. Yoshida, A. Tagami, S. Yamamoto, and A. Nakao. Toy block networking: Easily deploying diverse network functions in programmable networks. In *Proceedings of ADMNET Workshop, COMSAC*, 2014.

[7] A. Nakao. Software-defined data plane enhancing sdn and nfv. *Special Section on Quality of Diversifying Communication Networks and Services, IEICE Transactions on Communications*, to appear, 2014.

[8] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan. Tcp fast open. In *Proceedings of the 7th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2011.

A Stateless Transport Protocol in Software Defined Networks*

M. Nikitinskiy
System analyst, programmer
A-Real Group, Energia-Info Inc.
Yaroslavl, Russia
man@a-real.ru

I. Alekseev
Director of the Internet Center
P. G. Demidov Yaroslavl State University
Yaroslavl, Russia
aiv@yars.free.net

Abstract — **Network traffic balancing is one of the major factors of building scalable and robust service oriented communication networks. A plethora of algorithms for traffic balancing exists for classical network architecture and for software-defined networks. Having spotted certain deficiencies of existing methods, we are presently developing a stateless transport protocol, one of key features of which is the possibility to use anycast for concurrent connection with several servers. In this article, we consider various methods of a stateless protocol application for enhancing traffic balancing in software-defined networks.**

Keywords — software-defined networking; asymmetric transport protocol; balancing algorithm; Internet Control Server; NewTrickles; Internet gateway

I. Introduction

Every year more and more companies and developers join the arena of software defined network (SDN) technologies. This happens due need to achieve significant economic benefit to organization by applying the SDN approach construction corporate network infrastructure. Financial component of savings expressed in reducing the cost of network equipment, energy consumption, as well as reduction of the number of staff serving the enterprise network infrastructure. The leading organizations in the development of SDN include CISCO SYSTEMS, International Business Machines, Hewlett-Packard, Nippon Electronics Corporation and others.

It may seem in general, that vast area of research can described as development of a universal controller for SDN. Equally, important parts are switching technology and operating communication protocols. The latter play the most important role for operation of the network end nodes. Due to the increasing number of mobile devices, massive data processing systems, virtualization and cloud systems, as well as sensor networks arises the problem of efficient protocols directly operating at the network end nodes and the creation of new algorithms for balancing network traffic. One approach to solving this problem is to use the transport protocol we developed, which controls the connection settings and stores its state at only one end node. Thus, it is possible to use new ways of balancing network traffic.

II. Load-balancing algorithms for SDN

Aster * x [2]. Developed at Stanford University specifically for NoX SDN controller. The algorithm based on the idea of balancing the type of traffic (for example, all http requests on a specific port are routed in pre-defined way). Aster * x also has the ability to disable balancing for certain types of queries.

In [3] authors proposed and evaluated a novel load balancing mechanism leveraged by flow admission control. Seamless connectivity enabled with SDN is the bottom-line of their 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. Their model reveals a 237% of improvement in terms of per-flow resource allocation. Furthermore, they 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, their 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.

Mentioned above are just some balancing algorithms. If we consider all of the algorithms, it is necessary to classify their intended use: energy saving, support of Quality of Service (QoS), for mobile solutions, for heavily loaded systems, trivial. Moreover, balancing algorithms can be divided into proactive (when balancing rules are set in advance) and reactive (rules are set for each newcomer flow). In this article, we consider the case of proactive balancing algorithm at the level of L4 for heavily loaded servers.

III. Asymmetric Transport Protocol

The main task of transmission control protocol (TCP) is a reliable and efficient data transfer between end systems through unreliable transmission medium - the

*The work was fulfilled at financial support of the Ministry of Education and Science of the Russian Federation, unique ID RFMEFI57614X0105.

communications network that can lose, reorder, and distort the transmitted data. For simplicity, we assume that the transmission is in one direction, and call the server side, transmit data, and the client - the host. In addition, each byte of data transmitted uniquely numbered increasing sequence numbers. Data reliability is ensured through a mechanism of cumulative acknowledgement - the successful receipt of each piece of data (which is also called a segment), transmitted to the server needs to be validated by the client. In that case, if a confirmation for a certain time has not come, the server retransmits the data [4, 5]. The transmission strategy is to utilize all available resources of the communication network, not allowing its unnecessary downtime and possible overload. On the other hand, to ensure reliable and efficient data transfer server and client are required to maintain the connection information before it is completed. For this purpose, the transmission control block (TCB) containing numbers of local and foreign socket, flags and priority of security for this connection, pointers send and receive buffers, pointers and the current segment retransmission queue used both at server and at client. Such a distributed organization of interaction between server and client leads to the following problems:

- For storage and processing of connection, status resources are required on both server and client side.
- Number of simultaneously connected clients to a server, is limited due to limited resources.
- There is possibility of Distributed Denial of Service (DDoS)-attacks on a server, which may cause undesired operation of the server.

Thus, the problem arises of modifying the transport protocol for more efficient operation directly in the final nodes of the network as possible without significantly reducing the performance of data transmission. One approach to solving this problem is a protocol Trickles.

The Trickles protocol was proposed in 2005 at Cornell University, USA [6].The main difference of Trickles protocol from TCP protocols is keeping of all of control parameters on the client side, while the backend does not store information about the transport connection. Transport protocol operating on this principle from server to client will be called *asymmetric* protocol or *unallocated* connection state.

Trickles protocol server completely mimics the server TCP New Reno [7]. To control the connection, the server must have Trickles control information, but since it does not store its information on the connection status, client sends it in each new segment (Fig. 1). After processing the request, the server closes the connection.

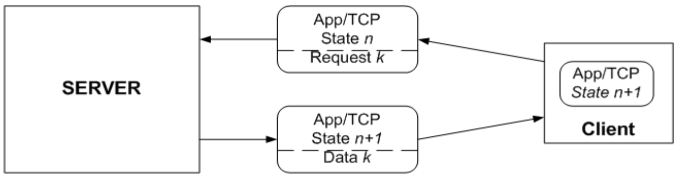


Fig. 1. Trickles segment.

Trickles connection consists of client requests and server responses to the client and at the same time there may be several such requests. We will call the continuation of flow (continuation) segment, the segment travelling from server to client and back. Consistent continuation flow forms an elementary stream (trickle). Thus, we can say that during the data transfer parallel operation of several elementary streams (Fig. 2) occur. During transit in the network, each elementary stream is independent of the other elementary streams, their synchronization only occurs at the client side.

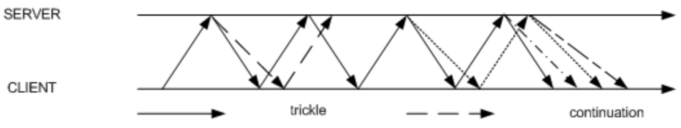


Fig. 2. Continuation and trickle

We describe flow control algorithm of Trickles protocol, since this part of the protocol has greatest influence on its performance. This algorithm runs for each elementary stream in the sense that when a segment belonging to a certain flow is in transit, it stores all the necessary parameters. At the same time server part of the protocol converts these parameters and takes further actions to control elementary streams. When a segment arrives at the server, there are three possible scenarios: the server sends a segment with the data in response to a request by continuing stream; server increases the number of elementary streams; server destroys the flow, not sending anything in response to the request. Since one elementary stream in the network is represented by one segment, the number of running elementary streams is a current assessment of network bandwidth by the protocol.

Mechanism of elementary streams management has three modes; this mechanism seeks to comply with a similar algorithm in TCP New Reno: slow start / congestion avoidance, fast recovery and retransmit timeout.

In the slow start / congestion avoidance, Trickles server [6] associates with each packet its request number k, and decides whether to continue the flow or separate it using TCPCwnd (k) function.

Transition to fast recovery mode is done by the client because the server does not store information about the state of the connection. Client received packets with numbers k+1 and k+2 believes that the package number k was lost and sends the server requests where SACK-blocks[9] not contain the number of k. Server receives such a request, halves the amount of trickle in the network. If the request is numbered k +1 and SACK-block does not contain a number k, then the server sends back the lost packet with the number k. After receiving confirmation that the loss has been restored, the server recalculates the control connection parameters and sends them to the client. Client received new connection settings, believes that fast recovery mode is finished, set the new connection settings and enters the slow start / congestion avoidance. Thus, fast recovery mode duration is equal to two Round Trip Time (RTT).

If there are two or more losses, the client enters Trickles retransmit timeout. Thus, the server, receiving a package wherein the SACK-unit has two or more loss terminates inbound trickle. Once triggered by retransmission timer, the client sends a request to restore any lost segments. Having recovered all the losses the client changes the control parameters and connections enters slow start / congestion.

This change in the pattern of the transport protocol has several advantages:

- The server part can be replicated across multiple physical devices, as transmitted segments contain all necessary information about the connection (i.e., there is an opportunity to work with several copies of the same server).
- Increased number of clients because there is no more need to keep detailed information about the connections.
- In mobile networks, it is possible to transfer transport connection from one network to another, which is a challenge when using the classical TCP.
- Increased resistance to DDoS-attacks.

It is evident that the class of asymmetric protocols can facilitate the operation of heavily loaded server that was shown in [6]. However, this work is not a comprehensive analysis of Trickles protocol performance, which is an important task in the development of any transport protocol.

Most methodologies of performance analysis of transport protocols are a simulation-based or an implementation as a part of the operation system. We have chosen a simulation-based analysis, since there exists the ns-2 [10] system which contains a fair number of built-in models of various TCP protocol versions. The ns-2 system is a de-facto standard for the performance analysis of protocols.

The ns-2 [10] system uses an object-oriented approach. The kernel of the system is implemented in C++ and network models must be implemented in OTcl. This approach, which involves the usage of two programming languages, is based on two reasons: first, models must be executed fast and C++ usage helps us to achieve it; second, models must be developed quickly, so the scripting nature of OTcl is helpful to us in doing it.

The central concept of the ns-2 system is the agent which is an entity executing on the endpoint. The Trickles protocol model is an example of a such an agent in the experiments. Typical workflow of the transport protocol model construction is the following procedure:

- Implement a packet model for the protocol introduced.
- Implement an agent C++ class, which models the protocol. For the sake of simplicity, most transport protocols are modeled with two kinds of agent: first one acts as a server, and the second one as a client.
- The C++ language is used for implementing packet and agent models. The final step is to provide OTcl bindings, so the model is available for experiments.

Our implementation of the Trickles protocol packet is based on the variant proposed in [6].

In the ns-2 system the parent class of all agents, the *Agent* class, encapsulates virtual methods which model the packet processing and triggering time-outs. The Trickles server model is implemented by deriving from the parent class *Agent* and redefining only the virtual *Agent :: recv()* method. The method accepts an incoming packet which contains the client request and immediately sends a response. The workflow for the client implementation is the same except for the fact that now we have to redefine the virtual *Agent :: timeout()* method for modeling time-outs.

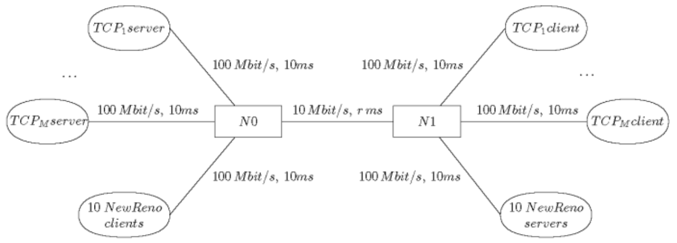


Fig. 3. Network configuration which is used for experiments.

IV. EXPERIMENTAL RESULTS

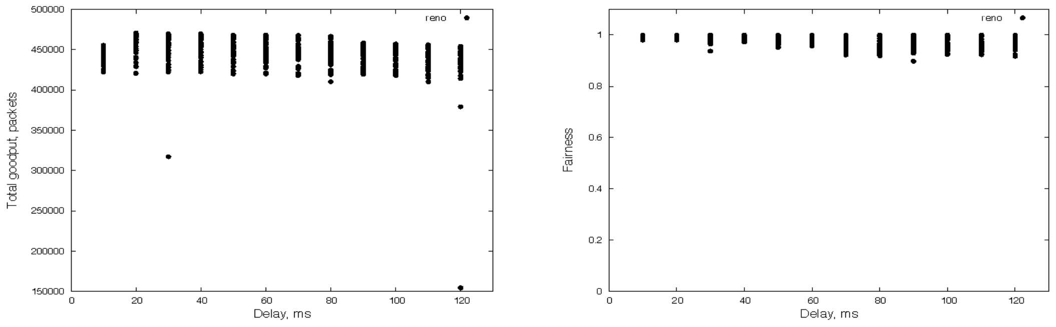
After a thorough analysis of the paper [6] we have implemented the original model of the Trickles protocol. This model was used for constructing the network model, which is shown in Fig. 3. This network contains two intermediate nodes *N0* and *N1*, each of which has a queue with the maximum size of 50 segments. The segment size is 1540 bytes. Traffic is transmitted from nodes, which are labelled as *servers* to nodes, which are labelled as *clients*. Acknowledgements are transmitted in the opposite direction.

We are interested in the performance of the protocol instances which are shown in Fig. 3 as *TCP_1 Server,..., TCP_M Server*. These instances are variants of transport protocols under consideration including built-in ns-2 models: TCP Reno, TCP SACK, TCP Vegas, TCP NewReno which were compared with the original Trickles model. During every experiment, we simulate 600 seconds long data transfer. In addition to competition of *TCP_1 Server,..., TCP_M Server* protocol instances for network resources there exists a traffic which discomforts acknowledgement transmission process. This traffic is generated by ten instances of TCP NewReno protocol. They start at the same time and transmit data from 150 to 300 seconds and from 450 to 600 seconds of the model time. The experiment has two parameters. First, one is *M*, the number of working protocol instances; *M* is changed from 10 to 50 with step 1. The second parameter *r* is the delay for data link between nodes *N0* and *N1*. The delay is changed from 10 to 120 ms with the step 10 ms.

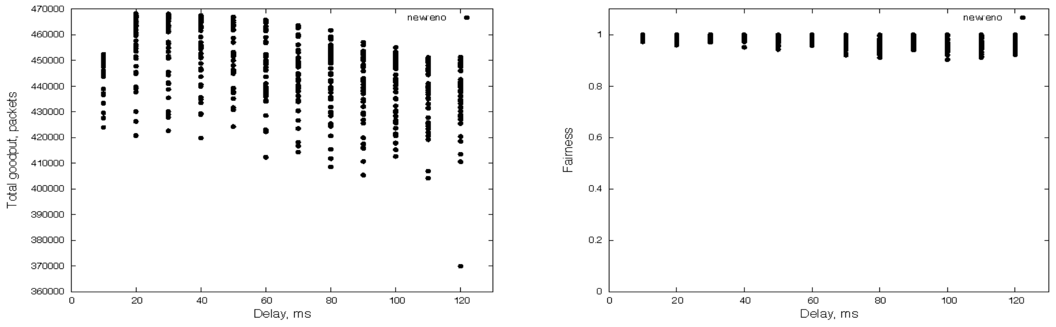
We calculate two performance metrics. The *goodput* metric shows us how much data were transferred from a server to a client, and can be calculated for a single connection as follows:

goodput = (send_data - retransmitted_data) / t,

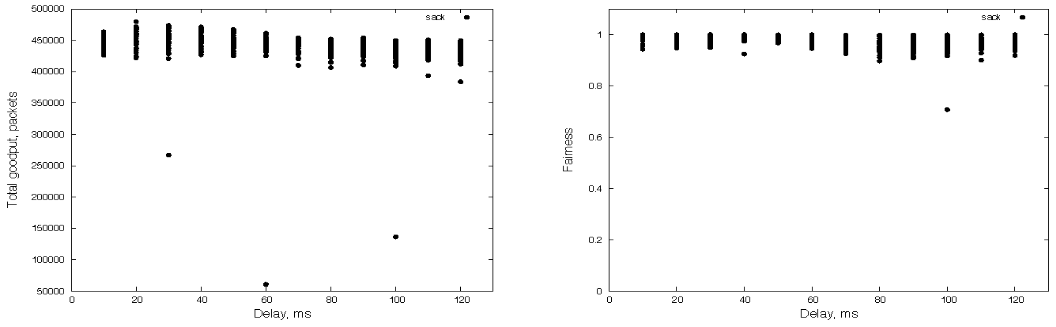
where *send_data* is the total amount of data segments which were transferred by the protocol, *retransmitted_data* is the number of retransmitted segments, and *t* is the time of the



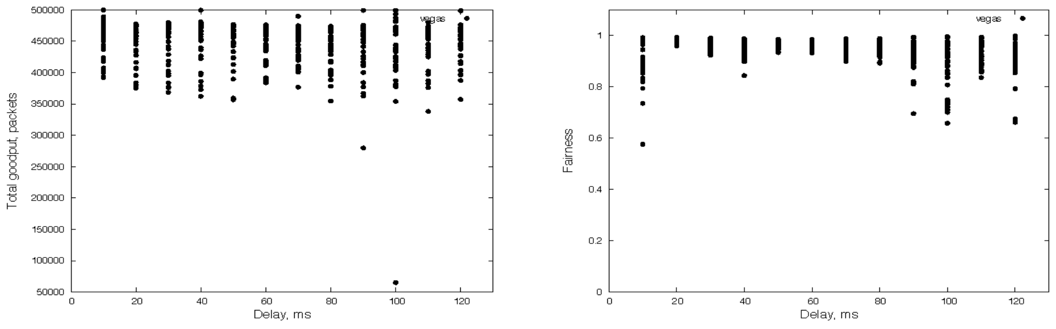
TCP Reno



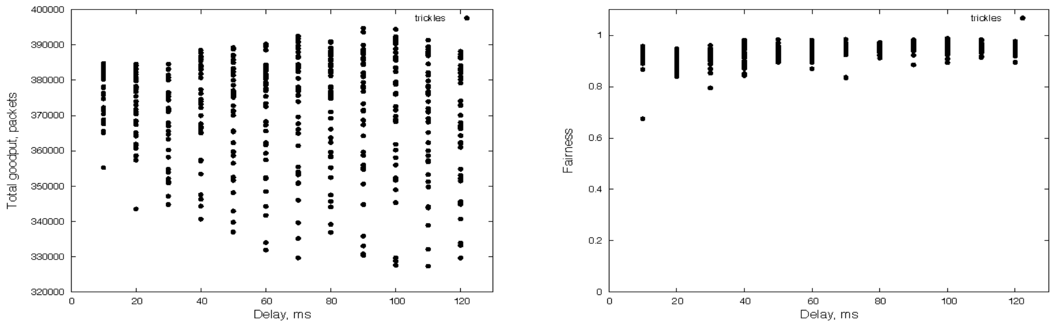
TCP NewReno



TCP SACK



TCP Vegas



Trickles

Fig. 4. Results of experiments. Goodput values are on the left side, Jain fairness index values are on the right side.

experiment. The second performance metric is the Jain fairness index [11], which can be calculated as follows:

$$J_{FI} = \frac{\left(\sum_{i=1}^{M} b_i\right)^2}{M \times \sum_{i=1}^{M} b_i^2}$$

where b_i is a goodput of i-th connection and M is the total number of connections which share the same networking resource.

The experimental results are shown in Fig. 4. The y axis shows the total goodput of all connections in the experiment and the x axis is the value of the delay r between nodes *N0* and *N1*. As we can see, the Trickles protocol transmitted the comparable to other protocols amount of data. The Jain fairness index shows that Trickless share network resources quite fairly. Thus, experimental results show us that the Trickles protocol is a quite efficient transport protocol in comparison with common variants of the TCP protocol. However, the experiments indicated some shortcomings of Trickles protocol, mainly its instability.

V. NEWTRICKLES AND SDN

Analysis of data obtained during the experiments showed that the fast recovery mode works for time equal to 2 * RTT in Trickles protocol, whereas in most versions of TCP, this mode works for time equal to RTT [12]. Therefore, for large values of RTT, or a large number of trickles, the probability of packet loss and frequent retransmit timeout transition becomes critical, which in turn affects the performance of the transport protocol in general.

We proposed a new algorithm of the fast recovery mode for asymmetric protocols. Transition to fast recovery mode is done by the client. Client received packets with numbers k+1 and k+2 believes that the package number k was lost. It regenerates the lost data request and sends it. This ensures compatibility of our algorithm with the Trickles protocol, in contrast to which the client decides to reduce the amount of trickle's network. When resent segment arrives, the client calculates the new connection settings and enters the slow start / congestion avoidance. Thus, we get recovery time equal to RTT. Other modes work as in the Trickles protocol.

We have proposed and developed a model of asymmetric transport protocol that uses this recovery algorithm, which we called - NewTrickles. This model has all the advantages of Trickles protocol, and also reduces the load on the communication network, and reduces the number of calculations run on the server and has recovery time equal to the RTT. Currently NewTrickles is being modeled in the network simulator.

Explosive growth in the number of network applications and devices in recent years has led us to the fact that it is necessary not only store information, but also have a copy of it on the network. Since asymmetric protocol does not require a rigid connection to the server, the method using anycast (delivery to any nearest) can simultaneously work with duplicates of a single server. Such interaction of server with client gives:

- More balanced load on the network that will be able to reduce the number of dropped segments in the network.
- Reducing the workload of servers and increase the number of clients due to the uniform redistribution of requests.
- Immunity to DDoS-attacks.

Now, we see two variants of the application of asymmetric protocols with SDN technology. The first involves the introduction of new rules on OpenFlow switches and in OpenFlow protocol. It is assumed that if the controller, inspecting the first packet consisting new flow, sees the reference to asymmetric protocol or application of anycast, the packet will be assigned to asymmetric protocol class. Then the controller will need to invoke anycast method on the switch / switches for the current flow. To implement this a specific option should be added in the operation of the OpenFlow protocol, as well as network equipment to support the change. We do not have the possibility of changing hardware or protocol specifications.

Second application, presumes existence of network edge device, which is able to assess the network bandwidth, network latency and dynamically adapt to the changing situation in the network, and the ability to split incoming stream into two. An SDN border gateway can be used in this role. In this case, the gateway must use the SDN controller as a tool for managing communication channels between clients and servers. Applying asymmetric transport protocol for clients and servers in the network SDN, we obtain an additional tool for balancing SDN, as well as improve its resiliency.

VI. CONCLUSION

In this paper we describe the principles of asymmetric transport protocol called Trickles. Considered it an advantage over using TCP. Learned the basic principles of creating a protocol model in the ns-2. Shown results of one of the experiments, and identified the main drawback of the protocol Trickles. On the basis of the identified deficiencies have developed a new mechanism of fast recovery. At the moment, we have created a module asymmetric transport protocol NewTrickles for network simulator.

At present, we are cooperating closely with the developers of the universal Internet gateway “Internet Control Server” produced by “A-Real Consulting” Inc. This gateway provides the following features: firewall, proxy server, mail server, built-in virus scanner, supports VoIP, multi-level traffic filtering, DNS, DHCP and various traffic counters. We expect to develop functions linking this universal Internet gateway with SDN controller, providing efficient solution for small and medium enterprises. This solution will provide all the necessary functionality to the enterprise, and the use of an asymmetric protocol for the relationship of clients and servers, will further increase efficiency and stability of an SDN network managed by this solution.

REFERENCES

[1] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 17-17, Berkeley, CA, USA, 2010. USENIX Association.

[2] N. Handigol, S. Seetharaman, M. Flajslik, R. Johari, and N. McKeown. Aster*x: Load-balancing as a network primitive. 9th GENI Engineering Conference (Plenary), November 2010.

[3] Suneth Namal, Ijaz Ahmad, Andrei Gurtov, Mika Ylianttila. SDN Based Inter-Technology Load Balancing Leveraged by Flow Admission Control. Future Network and Services (SDN4FNS), 2013 IEEE , pages 1-5.

[4] Transmission Control Protocol. DARPA Internet Program. Protocol Specification // RFC793,September, 1981. Web site: www.rfc-editor.org

[5] Paxson V., Allman M. Computing TCP's Retransmission Timer // RFC2988,November, 2000. Web site: www.rfc-editor.org

[6] Shieh, A., Myers, A. C., Sirer, E. G. A stateless approach to connection-oriented protocols // ACM Trans. Comput. Syst. 26, 3, Article 8 (September 2008), 50 pages.

[7] Floyd S., Henderson T., Gurtov A. The NewReno Modification to TCP's Fast Recovery Algorithm // RFC3782, April, 2004. Web site: www.rfc-editor.org

[8] M.A. Nikitinskiy, D.Ju. Chalyy. Performance analysis of trickles and TCP transport protocols under high-load network conditions // Automatic Control and Computer Sciences. December 2013, volume 47, issue 7, pp 359-365.

[9] Floyd S., Mahdavi J., Mathis M., Podolsky M. An Extension to the Selective Acknowledgement (SACK) Option for TCP // RFC2883, July, 2000. Web site: www.rfc-editor.org

[10] S. McCanne and S. Floyd. ns Network Simulator // http://www.isi.edu/nsnam/ns/

[11] Jain R. The art of computer systems performance analysis. // Jon Wiley and Sons, 1991.

[12] Nikitinskiy M.A., Novyy analogovyy algoritm vosstanovleniya dlya assimetrichnykh transportnykh protokolov // Tendentsii razvitiya prikladnoy informatiki: Sbornik statey po itogam mezhdunarodnoy nauchno-prakticheskoy konferentsii. Yaroslavl', 2013. S. 234-238 [in Russian]

Controller Failover for SDN Enterprise Networks

V. Pashkov, A. Shalimov, R. Smeliansky
Lomonosov Moscow State University,
Applied Research Center for
Computer Networks
Moscow, Russia
pashkov@lvk.cs.msu.su, ashalimov@lvk.cs.msu.su, smel@cs.msu.su

Abstract—**In SDN network based on OpenFlow a controller performs logically centralized control of enterprise network infrastructure, network policies, and data flows. At the same time the controller is a single point of failure which can cause a very serious problem (e.g. network outage) for network reliability and production use cases. To address this problem, we consider different active/standby strategies to provide a controller failover in case of controller failure. We propose a high-available controller (HAC) architecture, which allows to deploy a high-availability control plane for enterprise networks. We develop a HAC prototype to demonstrate the efficiency of our solution and also describe initial experimental results.**

Keywords—Software-Defined Networking; Control Plane Design; Controller Architecture; Fault-Tolerance; Redundancy.

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003

I. Introduction

SDN is a new approach in networking, which significantly improves the programmability and flexibility of network management, simplifies the logic of network devices and reduces the cost of the network infrastructure deployment and the cost of its maintenance in comparison with traditional approaches [1, 2]. SDN separates the control plane and the data plane, which enables their independent deployment, scalability and maintenance. SDN involves centralized management of network infrastructure and data flows, but this approach can lead to network resilience and scalability problems.

The control plane can be deployed on one or several SDN controllers, which is running on dedicated servers [3]. The set of hardware and software components for providing of centralized network management in SDN is a control platform. The controller supports an actual global network view (GNV), which is stored in its network information base (NIB). Using network view controller applications control network devices states and data flows. That is why SDN network performance, reliability and scalability is defined by control platform characteristics.

In spite of the SDN advantages, one of the serious problems of SDN is that the controller is a critical point of failure and, therefore, the controller decreases overall network availability. A Controller failure can be caused by various reasons: failure of the server where a controller is running, the server operating system failure, power outage, abnormal termination of the controller process, network application failure, network attacks on the controller and many others.

In this paper we address to control plane for OpenFlow networks, as the one of the most promising implementations of SDN approach [4]. OpenFlow protocol is the open interface between the control plane and the data plane. The control plane in OpenFlow includes a controller (or NOS — network operating system) to monitor and control the state of OpenFlow switches, a set of network applications for network traffic and policy management, OpenFlow communication channels between the controller and switches and OpenFlow protocol for their interaction. OpenFlow controller can install rules in OpenFlow switches for data flows supporting predictive, reactive, and proactive or hybrid flow installation modes.

At the present time there are about 30 different OpenFlow controller implementations [5, 6]: NOX, POX, Beacon, MUL, Floodlight and the others. However most of them do not support the control plane restoration mechanisms in the case of controller failure. Only distributed control platforms Onix [7], Kandoo [8] and some proprietary controllers with OpenFlow 1.0 [4] support restoration procedure in case of a controller failure. Thus, a controller failure in the control plane of SDN/OpenFlow is a pressing issue.

An approach for improving the SDN control plane availability in case of a controller failure in the enterprise software-defined networks is presented in the paper.

In summary, in this paper the following points are presented:

- comparative analysis of the different active/standby strategies to provide a controller failover;
- a fault-tolerant control plane architecture for enterprise software-defined networks;
- a High-Available Controller (HAC) architecture that provides the network ability to fast recovery of the control plane;
- the control recovery procedure and the procedure for network view synchronization between active and standby controller instances;
- the HAC prototype implementation with supporting OpenFlow version 1.3.

II. Background

A. Typical SDN controller architecture

A typical SDN/OpenFlow controller [6, 9, 10 and 11] includes:

- ***Controller core*** which handles and supports connectivity with switches and translates control protocol messages (e.g. OpenFlow) into internal controller events and vice versa.
- ***Controller network services*** which control, form and monitor network view, states of network devices, provide an interface (Northbound API) for controller applications. Usually network services include event dispatching, device managing, topology managing and the others.
- ***Controller network applications*** which configure network infrastructure and manage data flows to solve some business use cases.

The interaction between Network, controller services and applications is based on the publish-subscribe model.

B. Active/Standby strategies analysis for control platform

Let us consider the basic redundancy approaches to improve control platform availability for enterprise software-defined networks. The controllers can be active or standby mode in the control plane. An active controller directly receives and processes OpenFlow messages from network devices. A standby controller duplicates the functionality of the active controller, but receives and processes OpenFlow messages from network devices only in case of active controller failure. The number of standby controllers may be increased to tolerate more than one failure at a time. The primary controller for a network segment is a controller which configures network devices of its segment and installs the rules for data flows in this segment.

There are active/standby strategies and active/active strategies for controller redundancy. In case of active/standby strategies control platform has only one active primary controller. In case of active/active strategies control platform can have multiple active controllers. But in this paper we consider only active/standby strategies with one active primary controller in the control plane.

In case of primary controller failure the standby controller automatically takes over network infrastructure control and data flows management. This procedure is called ***controller failover***. ***Controller failback*** is the reverse procedure to failover. This procedure is used when the primary controller is restored.

The active/standby strategies based on the operational status of standby controllers (switch on/off and loading on/off before the start of work) and failover transparency are:

- no standby;
- cold standby;
- warm standby;
- hot standby.

«No standby» strategy. The control plane has a single active primary controller without standby controllers connectivity. In case of primary controller failure the network administrator manually resets the controller or replaces it. Thus, the control plane recovery time is significant and unpredictable and depends on the efficiency of support service.

«Cold standby» strategy. The control plane has an additional unloaded server connected to a server of the primary controller. The «cold standby» strategy uses automatically failover procedure. The standby controller is stateless. In case of primary controller failure a standby server runs the standby controller and its services and applications (including topology discovery service to form network view) from scratch. This strategy is preferable to use for stateless services and applications. Recovery time is determined by the controller start and time to restore the actual standby controller state.

In the case of cold standby strategy a redundancy hardware component is often unloaded, that is why it can be used for any optional extra work: for testing, debugging, maintenance and other services (e.g. testing of the new versions of controller network services and applications).

«Warm standby» strategy includes periodically primary controller state replication to standby controllers and automatically failover procedure. The «warm standby» strategy is usually provided by hardware and software redundancy. In case of primary controller failure the standby controller replaces a failed controller and continues to operate on the basis of its previous state. Control plane services for network devices are interrupted and some state is getting lost. The lost part of the control plane state is those state changes which were between the last state synchronization procedure and the primary controller failure.

«Hot standby» strategy includes full state synchronization of the primary and standby controllers and automatically failover procedure. No loss of the controller state provides the minimum recovery time. State of the primary controller is replicated to the standby controller for any change in it. In case of primary controller failure the standby controller replaces a failed controller and continues basing on the current state. The «hot standby» strategy is implemented by software and hardware redundancy.

TABLE I. COMPARATIVE ANALYSIS OF THE ACTIVE/STANDBY STRATEGIES

Criterion	**Active/Standby strategies**			
	No	***Cold***	***Warm***	***Hot***
Redundancy	hardware	hardware	hardware and software	hardware and software
Active controllers	1	1	1	1
Failover procedure	manually	auto-matically	auto-matically	auto-matically
State loss	complete loss of the state	complete loss of the state	partial loss of the state	without loss of the state
State and	no	no	regularly	up-to-date

Criterion	Active/Standby strategies			
	No	*Cold*	*Warm*	*Hot*
data syncronization				(any change)
Redundancy rate	1	1+N	1+N	1+N
Failover time	unpredictable	from minutes to seconds	seconds	from seconds to milleseconds
Cost	no cost to low cost	moderate	moderate to high	moderate to high
Network user impact	high	moderate	low	none

C. Key metrics

Key metrics which characterize a fault-tolerant control platform are the following:

Controller redundancy degree is a number of standby controller instances included in the control platform. It determines the cost of the control platform and the number of failures that can be avoided.

Controller delay in the worst case is the maximum delay for processing the network device flow installation request by the controller which is attained in the control recovery process.

Controller failover time is the time during which network device requests can be lost due to absence of the primary controller in the network, i.e. the time during which the network is not the primary controller.

Thus, the SDN control platform should have controller redundancy degree at least one, delay in the worst case no more than 150 milliseconds as recommended maximum time delay for services. Failover time should be as low and close to zero.

D. Fault-Tolerant control plane requirements

To support redundant controllers the control platform must meet the following requirements:

- there are to be at least two servers;
- identical hardware and software server configuration;
- internal network between servers to decouple control platform communications from OpenFlow communication channels and for accessing to data store;
- each server must have access to SDN network segment with independent links;
- identical controller instances (with identical versions of controller network services and applications).

These requirements are due to the following reasons:

- to avoid single point of failure;
- standby controller must have sufficient computing resources for network infrastructure and data flows management in case of primary controller failure;
- standby controller must provide the same set of functions as the primary controller.

III. Proposed Approach

A. Proposal

Since we have previously discussed the active/standby strategies it is very important to define controller state.

Controller state includes states of controller services and applications, event queue state, controller network view and controller data. The state of the controller service or application includes values of internal service/application significant variables.

Service/application snapshot is a service/application state at a particular time. Controller snapshot is controller services and applications snapshots and current network view.

For solving the controller failure issue using active/standby strategies we need to define the basic modes for control platform: an initial mode which describes the order of launch controller instances, an operational mode which describes controller instances synchronization procedure and a primary controller failure mode which describes failure detection and failover procedures.

Initial controller mode. Running the primary controller of the control plane:

- The controller starts in accordance with the configuration file.
- The controller launches a timer for connecting standby controllers.

Running the standby controller of control plane:

- The controller starts in accordance with the configuration file.
- Standby controller establishes a connection to the primary controller via the internal control network between controllers.
- Standby controller requests a list of network services and applications of the primary controller and launches a similar set of applications and services.
- Standby controller requests the current Network view and network interfaces list for control channels connections, current states of network services and applications.
- Standby controller launches primary controller state monitoring service.

Operational controller mode. In this mode primary controller processes OpenFlow messages from network devices and controls network data flows, the standby controllers monitor the primary controller state and synchronize with it.

Controllers state synchronization includes:

- network view synchronization;
- controller network services and applications states synchronization;
- controller data synchronization.

In this paper we use two strategies for controllers.

For network view redundancy and synchronization we use hot active/standby strategy. Primary controller pushes up each network view change to all standby controllers.

For controller network services and application redundancy and synchronization we use warm active/standby strategy. Primary controller periodically or conditionally pushes up snapshots of services and applications to all standby controllers.

For controller data synchronization we use reliable shared data storage between controllers.

Primary controller failure mode. The control plane recovery procedure consists of two stages:

- **Failure detection stage.** Primary controller failure detection mechanism is based on the heartbeat. The main parameters are: heartbeat interval — the time interval between heartbeat messages, and dead interval — the time interval through which standby controller fixes primary controller failure.
- **Recovery stage.** The recovery stage starts after primary controller failure detection. It includes the following steps:
    - Defining a new primary controller. The new primary controller is a standby controller with the highest ID (or IP).
    - The new primary controller informs the other controller about its status change.
    - Controller network services and application restoration.
    - Control network interfaces up.

B. *High-Available Controller Architecture*

High-available controller (HAC) architecture is based on adding of additional cluster middleware between the controller core and controller network services and applications (see Figure 1).

To provide fault-tolerance of the control platform the HAC cluster middleware includes the following managers and services:

- ***Controller Manager*** to coordinate start/restart/stop controller network services and applications and up and down control interface for network devices connections.
- ***Cluster Manager*** to control the operation of the controllers cluster and distribute responsibilities (primary or standby) in accordance with the cluster configuration file.
- ***Sync Manager*** to control controller network services and applications synchronization between controller instances in the cluster.
- ***Recovery Manager*** to coordinate the recovery process (failover and failback) in case of controller instance failure in the platform.
- ***Message Service*** to provide control message distribution to other controller instances in the controller cluster.
- ***Event Service*** to provide filtering, distribution and processing to or from other controller instances.
- ***Heartbeat Service*** to monitor the operational status of the controllers and detects controller failures in the controller cluster.

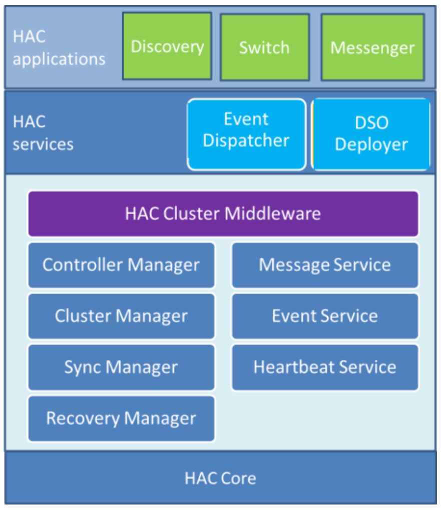


Fig. 1. High-Available Controller architecture

C. *Control Plane Design with HAC*

In order to avoid single point of failures in the control platform we propose the following design of the control platform (see Figure 2).

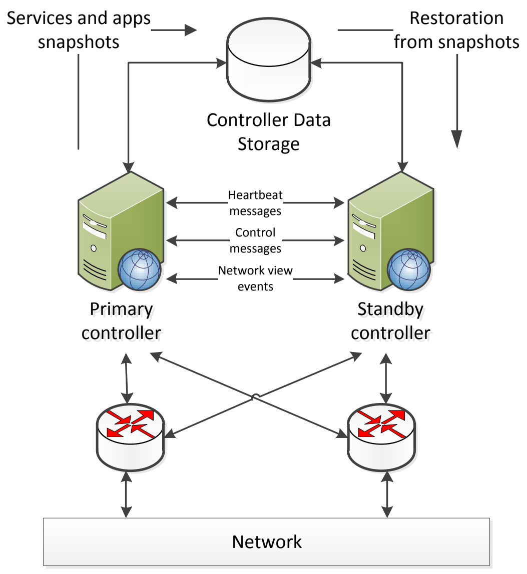


Fig. 2. Fault-tolerant control plane design with HAC controller

D. HAC Imlementation

Based on the review of modern open-source SDN control platforms [5] as a base controller for HAC controller we choose NOX13oflib from the laboratory CPqD [12]. This controller supports OpenFlow control protocol version 1.3.0 [13].

All HAC cluster middleware managers and services have been implemented in C++ with using Qt 4.8.1 and Boost libraries. Applications and services snapshots are formed using boost serialization mechanism. Interaction between the middleware services and managers provides through the Qt signal-slot mechanism to ensure the independence from the base controller.

IV. EVALUATION

The HAC controller prototype implementation has been deployed on a Linux virtual machine for functional and performance testing. Our experimental evaluation includes two parts: synchronization overhead evaluation and controller failover time evaluation. In the first part we evaluate performance overhead connected with primary and standby HAC controller synchronization. Using cbench we evaluate throughput of NOX13oflib and throughput of two-node fault-tolerant HAC cluster.

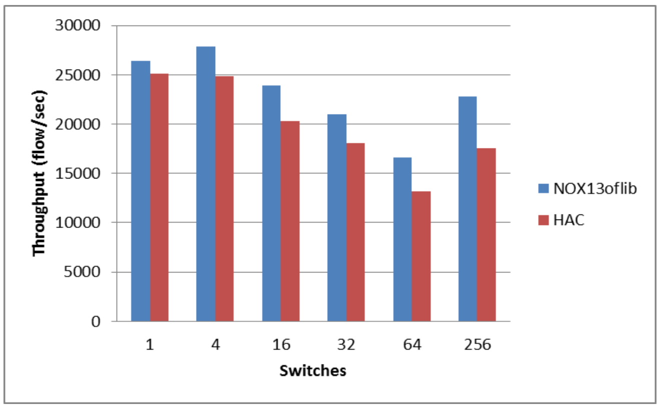


Fig. 3. HAC controller synchronization performance overhead

Synchronization overhead range is from 5 to 23 percent of the nox13oflib controller throughput (see Figure 3).

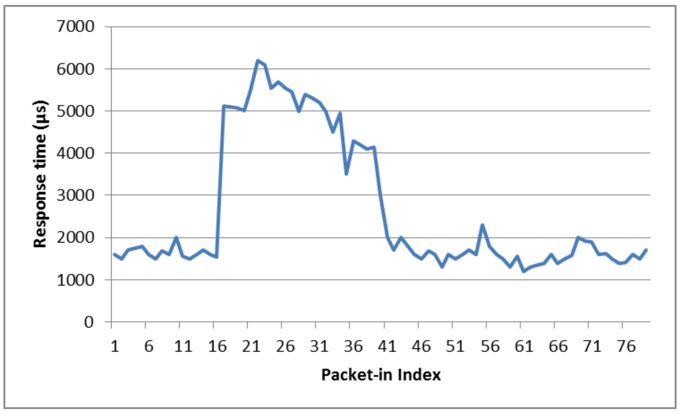


Fig. 4. Response time during HAC failover

Figure 4 shows the change of response time depending on the packet-in message index during primary controller failure and controller failover procedure. Initial experimental results showed that average failover time for two-node HAC cluster are from 40 to 50 ms, which is less than the maximum delay for services and that is why network services for end users will not be interrupted during controller failover.

V. CONCLUSION AND FUTURE WORK

Controller is a critical component of enterprise software-defined networks. In this paper, we showed the relevance and significance of the control plane availability problem for SDN in case of controller failure.

We considered and carried out a comparative analysis of active/standby strategies for their applicability to the control plane. We formulated a set of necessary requirements for controller redundancy.

Moreover, we presented control plane design, HAC controller architecture and tools for controller network applications and services synchronization and network view synchronization between controllers in the control plane. We implemented the HAC cluster middleware that can be easily adapted to other more productive basic controller implementation. We showed that our initial evaluation results are quite encouraging.

Thus, in this paper we proposed approach to solve controller failover problem for SDN control platform, we proposed middleware implementation which provides opportunities for active/active strategies studies and distributed controller development.

We are continuing the implementation of the HAC cluster middleware with focus on developing controller state synchronization algorithms and adding of load balancing mechanisms between controller instances. We plan to extend the list of failures that the control platform can prevent.

REFERENCES

[1] N. McKeown et al., “OpenFlow: Enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, i 2, April 2008, pp. 69-74.

[2] R.L. Smeliansky, “Software-Defined Networks,” Open Systems, N.9, 2012. [in Russian]

[3] Open Networking Foundation, “Software-Defined Networking: The New Norm for Networks,” ONF White Paper, 2012.

[4] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.0.0 (Wire Protocol 0x01),” 2009.

[5] A. Shalimov, D.Zuikov, D.Zimarina, V. Pashkov, R. Smeliansky, “Advanced Study of SDN/OpenFlow controllers,” Proceedings of the CEE-SECR13: Central and Eastern European Software Engineering Conference in Russia, ACM SIGCOFT, October 23-25, 2013, Moscow.

[6] A. Shalimov et al. “Analysis of SoftwareDefined Networks Performance and Functionality,” editor-in-chief R. Smelianskiy. – M.:MAKS Press, 2014. – 148 p. [in Russian]

[7] T. Koponen et al., “Onix: A Distributed Control Platform for Large-scale Production Networks,” in OSDI, 2010.

[8] S. H. Yeganeh and Y. Ganjali, “Kandoo: a framework for efficient and scalable offloading of control applications,” in HotSDN, 2012.

[9] D. Erickson, “The Beacon OpenFlow controller,” in Proc. HotSDN, Aug. 2013.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Mckeown, and S. Shenker, “NOX: Towards an Operating System for Networks,” in SIGCOMM CCR, 2008.

[11] Floodlight OpenFlow Controller. http://floodlight.openflowhub.org/.

[12] NOX 1.3 Oflib, https://github.com/CPqD/nox13oflib.

[13] Open Networking Foundation, “OpenFlow Switch Specification, Version 1.3.0 (Wire Protocol 0x04),” 2012.

Localizing Errors in Controller Applications

M. Perevedentsev, V. Antonenko
ARCCN,
Lomonosov Moscow State University
Moscow, Russian Federation
max.pereved@gmail.com, vantonenko@arccn.ru

***Abstract*—Modern networks are complex. The great number of network tasks require a complex structure of controller. A promising approach to this challenge is modularity. The controller runs many applications, each responsible for a specific task. Each application can disrupt the operation of another one thus causing errors. Such errors cannot be detected during development of a particular application. We discuss errors of competition for switch tables, i.e. disruption of the routing policy installed by one application due to operation of another one. In this work we propose an approach to detection of such errors including the integration with a specific controller.**

Modern network troubleshooting tools cannot help administrators correct errors in controller applications. In this paper we propose the Error Localization Tool (ELT) which helps in fixing errors. Given the faulty rules, it finds the place in code of applications responsible for each error. With these data the administrator can eliminate errors more quickly and make the complex system of controller applications work correctly.

We describe an implementation of these approaches. We carry out an experimental research to evaluate the performance of our prototype.

Keywords—Software-Defined Networks; OpenFlow; network troubleshooting

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and Russian Foundation for Basic Research, project 14-07-00743

I. Introduction

Errors in computer networks are constantly arising. If not detected and eliminated in time, the error can force the consequences various in impact on users from lost connectivity between specific hosts up to security vulnerability threatening loss of important data. Thus the methods of error detection and debugging are of great importance.

Administrators today are doomed to use ad hoc network troubleshooting tools such as tcpdump[19] and traceroute[6]. These tools give the vision of end-to-end connectivity and traffic routes. They help administrators much but do not completely eliminate the need for more complex and powerful tools.

Software-defined networking (SDN) is a paradigm of computer network construction that separates control layer responsible for routing from data layer transmitting traffic. SDNs provide the opportunity of more transparent and flexible network control when compared to legacy networks. A typical SDN consists of special software (can be distributed, e.g. Onix[10]) called controller (or network OS), and a set of switches. Controller interacts with switches being managed using a specific protocol (e.g. OpenFlow[16]). Due to centralization of network management on the controller network troubleshooting becomes simpler. The existence of a single device having all the information about the network (topology, routing rules, traffic) helps the developer to detect and localize an error.

The SDN approach can introduce new kinds of network errors as well. Here we consider an error as a deviation in network functioning (rules in switch tables, traffic routing etc.) from what is expected by administrator. A promising approach in controller construction is modularity[1][14]. According to this approach, several separate modules, called applications, can run simultaneously on controller. Controller provides high-level interface for applications to communicate with switches. Applications can specify a routing policy for a class of traffic (the actions to be applied to packets with specific header) by installing, modifying or deleting rules from switch tables. We assume that applications can share the flow space, i.e. there can be headers operated by multiple applications. Each of applications can cause errors in operation of other ones[13][14] [18]. In this work we introduce a class of errors called errors of competition for switch tables and propose a method for detecting and localizing them.

II. Related works

Existing SDN troubleshooting tools can be divided into two groups: static tools and dynamic ones.

We call static the tools which use different techniques of source code analysis. An example of such tool is NICE[2]. NICE uses symbolic execution of controller applications and model checking to explore the state space of an OpenFlow network. This approach is too slow to be used in complex cases.

Dynamic tools operate as a superstructure over a real network or use the data acquired from a real network. NetPlumber[7] uses control messages and network state updates to incrementally check for network policy compliance. VeriFlow[9] uses forwarding plane modifications to check for reachability invariants. Anteater[11] and HSA[8] use data plane snapshots to detect violations in key network invariants. OFRewind[20] records and reproduces sequences of OpenFlow commands. ATPG[21] generates and sends test packets against router configurations. ndb[5] uses packet postcards to reconstruct the path of a specific packet. It does not care about rules set by controller.

Current dynamic tools use only the functionality provided by OpenFlow protocol, and the set of installed rules. They are abstracted from network OS. This approach simplifies the transition from one network to another but reduces usability. As a result of operation of such tools a user will receive a conclusion about routing policy compliance; in case of a policy violation he may also receive the rules leading to this error. Afterwards he should select from a set of applications the ones responsible for this particular error, localize a snippet of source code where these rules were installed and correct this snippet. With the number of applications on a controller growing, this task becomes more difficult.

III. Errors of competition for switch tables

Let each switch have a single table. According to OpenFlow, when a packet arrives at a switch, the latter looks for rules with appropriate pattern in its table. If there are none of them, the switch sends the packet to the controller. Otherwise, the appropriate rule with highest priority is chosen, and the packet is processed according to it. If there are more than one rule with highest priority, the behavior of the switch is undefined.

A set of all rules in switch tables we call *network state*. A set of rules with maximum priority for each header on each switch we call *effective network state*. Each application routes packets with specific headers. These headers we call *application traffic class* (for simplicity, we consider the application traffic class to be the same on all switches). By adding, modifying and deleting rules the application can change the effective network state.

Each application expects the rules installed by it to remain unmodified and be executed for appropriate packets unless this application explicitly modifies them, adds other rules or these rules expire. A set of rules with maximum priority for each header on each switch from the rules installed and operated by a specific application is called *expected effective network state* for this application. In case of a single application, it will match the real one. In case of multiple applications running on top of a controller, each application can modify or delete a foreign rule or install a new rule preventing the old one from being executed. As a result, the effective network state will differ from the one expected by an application i.e. the expected effective network state will not be a subset of the real one. *Errors of competition for switch tables (CST errors)* are the errors occurring due to the deviation of the effective network state projected to application traffic class from the effective network state expected by this application. An example of this error can be two routing protocols which run in the same network and install contradictory rules.

More formally, CST error is defined as following. Assume we have a network specification: a predicate

$$Correct(tables, netgraph),$$

describing the requirements to packet routing in network, and the packets cannot be routed through controller (from switch to controller, then from controller to another switch). Since the packets cannot be processed according to rules with lower priority while there are rules with higher priority in switch table, this specification describes the requirements to the effective network state. A network *satisfies* a specification if for each network state which can be generated by a specified system of applications the specification is satisfied. Assume there is a system of applications and we want the network to satisfy specification while running these applications. Assume there is a subsystem of applications such that the network really satisfies the specification. Such a dead-end subsystem we call the *kernel* of the system in terms of specification. *Error of competition for switch tables* is a situation when the specification is not satisfied while running the full system of applications. As the subsystem satisfies the specification, its expected effective network state also satisfies the specification, thus such errors are caused by the deviation of the effective network state from the expectations of the applications from kernel due to operation of other applications.

In this case we can simply remove all the application but kernel ones. Practically important is the case where

$$Correct \sim Correct_1 \wedge Correct_2 \wedge \ldots \wedge Correct_n \qquad (1)$$

and there is no subsystem of applications that satisfies all $Correct_i$ although a specific subsystem exists for each $Correct_i$. The task is to modify applications so that the system satisfies the specification (we assume that specification is non-contradictory). From the user's point of view, each application is responsible for its own part of the specification and we need to remove the disruption of one application's operation by another one.

CST errors cannot be detected during the development of a particular application, since they depend on system of controller applications. Since such system is arbitrary, the probability of CST errors occurring is high in case of independent controller applications. In controller, applications can ask the controller to generate table modification messages or generate such messages themselves. In the first case, the controller should run CST error detection module prior to message generation to avoid conflicts. In the second case, such module is still necessary although it may not be a part of controller. In any case, methods of detecting and localizing such errors are important. An alternative approach to CST error troubleshooting is rejection of application independence and transition to centralized decision-making, as in [1], [13] and [14].

In this paper we focus on application conflicts within a single controller instance. More difficult cases, such as multiple controllers in a single network and distributed controllers, are left for future work.

IV. Error detection

Before formulating the CST error detection task, we must consider the following. Routing tables can be filled before network starts or during the operation. An empty table can satisfy the specification or not. Thus the specification can be violated at a specific moment or never be satisfied (e.g. an error before the table is filled while an empty table is non-satisfactory). To satisfy different conditions, we use the time T so that the network must satisfy the specification at any time after T. Moreover, CST errors are caused by the system of controller applications and the number of specification violations strongly depends on the traffic, so it is useless to count errors. We check whether the specification is violated after T or not.

For (1), CST error detection task is formulated as following:

For any non-contradictory specification, topology, traffic, time T after which the specification should be satisfied, and such system of applications so that for each $Correct_i$ exists a subsystem that satisfies the specification after T, if the specification is not satisfied at time t>T, the error message must be generated.

The task does not include the precision of detection but only the recall[22]. The reason is that the main topic of this paper is error debugging. The simplicity for user to fix errors and the detection precision seem to not go together, as it is shown below. These criteria we use to compare different CST error troubleshooting methods.

A. Existing methods

OpenFlow describes a flag that tells the switch to check for overlaps of new rule with existing ones. However, this checks only the rules with equal priority and does not have information about rule owners (the same application or different ones).

Static tools can detect CST errors. But their high complexity makes them practically inapplicable. Dynamic tools do not have access to controller applications so they have two possible approaches to detection of CST errors.

Firstly, dynamic tools can signal about the effect of all table modification messages to existing rules. This approach has a high rate of false positives because an application can affect its own rules, which is a correct behavior.

Secondly, they can check the specification. In general, such specification can be inaccessible. The user desiring to eliminate CST errors has to create the full specification of network routing, what is a difficult task. As a result of specification checking the user receives the answer "yes" or "no". Probably he will get the rules violating the specification or the specification may never be satisfied. A search for the cause of these violations is complex and is put on user. This approach does not help user to eliminate errors.

B. Facts of competition for switch tables

We propose signaling about the occurrence of *potentially dangerous situations*. From the definition of CST errors we know that there is a deviation of effective network state from the expectations of one application due to operation of another application. Such a deviation we call a *fact of competition for switch table (CST fact)*. In terms of OpenFlow protocol there are four types of such facts:

- *Foreign rule deleted* occurs while an application deletes a set of rules. This application can delete the rules installed by another application as well. Although the second application can acquire such information, this can be an error and cause additional overhead (sending a packet to the controller is expensive).
- *Foreign rule modified* occurs while an application modifies a set of rules or install a new rule with pattern and priority equal to an existing rule. By modification, the application can modify foreign rules as well. In that case, the owner of modified rules cannot acquire the information about such modifications. It will remain certain its rule is correct while the effective network state differs from the expected one.
- *Foreign rule masked* occurs when a switch table contains two rules with different owners and priorities but intersecting patterns (there are headers matching both). For the set of common headers, the rule with lower priority will not be executed for common headers. This type of CST facts occurs when a new rule is installed. If the new rule has higher priority, the old rule will not be executed. Otherwise, the new rule will not be executed. In both cases, the owner of the rule with lower priority cannot acquire such information and expects the effective network state to match expectations while that is not true.
- *Rule undefined* occurs when a switch table contains two rules with equal priorities and intersecting patterns. According to OpenFlow, switch behavior is unexpected in this case, so the effective network state will not match the expectations of the owners of both rules. Such CST fact occurs when a new rule is added if the new rule has equal priority and intersecting patterns with an existing one. This error can occur in case of a single application as well.

The definition of CST errors implies that for occurrence of such error, there must be a CST fact between a kernel and non-kernel application, at least for one of $Correct_i$(1). The construction of a kernel is impossible in the absence of specification. So we will not separate the application to kernel and non-kernel ones. By detection of all CST facts we detect all possible CST errors. Moreover, CST facts can occur before the network violates the specification. Thus for elimination of errors, CST facts are more important than errors themselves.

CST facts do not always lead to errors. Some applications, e.g. a traffic monitoring system, can modify foreign rules on purpose. This fact will lead to a considerable number of false positives. We expect the overhead on processing of them to be lower then the creation of a specification. The alternative approach is to explicitly specify, which application can choose whose rules.

The main advantage of detecting CST errors through CST facts is the list of participating rules. For every CST fact, there are two or more rule or table modification messages. This can be used to find the root cause of errors, as described below.

C. Error detection module

The described approach cannot be implemented without integration with a specific controller. This integration improves the precision of this approach in comparison with the first described in section IV.A. We detect only the table modification messages affecting foreign rules.

The detection of CST errors is performed by a controller module. This module must check each message being sent to a switch for the possibility of CST facts. If it is necessary, this module can prevent a message from being sent to a switch. Our prototype stores switch table models in controller module. Each rule is tagged by a set of applications which affected this rule. When it receives a table modification message it checks for overlaps in table using application tags and updates the table. When a CST fact is detected, this module stores type of CST fact and participating rules and messages to a report. To provide consistency of table model contents with switch tables our module should receive the messages about rule expiration from switches.

V. Error localization

CST errors are caused by conflicts in controller applications. Therefore these errors should be debugged in integration with controller. This is why dynamic tools operating as a proxy on OpenFlow channel are not useful in troubleshooting such errors. But there are different types of errors which can be debugged easily using the vision of controller operation. In [23] it is proposed to use traditional source code debuggers. We expect them to add unnecessary overhead and propose the creation of a controller module collecting necessary data. The integration of controller-independent error detection module operating as a control channel proxy with debugging module bound to a specific controller we call *complex debugging*.

In case of CST errors, the user wants each application to fulfill its function without disturbing other applications. This is not always possible so it is required to slightly modify the application to eliminate such disturbance. The search for the code snippet to be modified is wholly put on user.

While using a dynamic network troubleshooting tool which outputs faulty rules, the search for the responsible application and code snippet is user's task as well.

To simplify these tasks we propose Error Localization Tool (ELT). ELT cannot operate at the level of abstraction provided by OpenFlow and is bound to a specific controller (e.g. POX[17]). Due to this it receives the ability to trace the inner state of controller. In our prototype, we use the call stack leading to a particular message being sent. By collecting such traces ELT can find the application and its approximate execution path responsible for a faulty message or rule. This approach will work in the case where each processing step calls the next step as a function. In the case where the processing steps are called one after another, we can only find the applications using a call stack. To reproduce the logic of an application we need other information, e.g. the values of global variables.

In a simple case, an SDN consists of multiple switches and a single controller with several applications running on it. During the debugging process we put a proxy on the OpenFlow channel. This proxy should detect errors and send a report including error type and flow modification messages or rules involved to the network administrator. The main task of ELT is the creation of an extended error report. The final error report includes call stack for each message or rule (call stack for each message leading to this rule having specific fields). To provide such data ELT must record call stacks for flow modification messages and retrieve them by on request. Here we present our architecture of ELT consisting of three modules: controller proxy, database server and logger.

A. Controller proxy

The controller proxy is integrated with a specific controller. When a controller application sends flow modification message to switch, the proxy sends a copy of message to database server. Saving messages with call stacks cannot be implemented without integration with a specific controller. Modern programming languages used for writing controllers (Python, Java, Ruby) support accessing call stack from inside a program.

B. Database server

The database server is the central point of ELT architecture. It saves flow modification messages received from a client on the controller proxy. When a request for particular message is received, the database server looks through the database and responds with the call stack for the latest equal message. Here the messages are considered equal if they have the same patterns, action lists, priorities and target switches. When a request for a rule is received, the database server selects the sequences of messages which could shape this rule and returns call stacks for the latest possible sequence.

Database is put on a remote computer to provide controller independence. Proxies at different controllers can communicate with database using the common protocol. Thus we only need to write a new controller proxy to provide interaction with a new controller.

Each debugging module should be augmented by adding database clients and modified logging subsystem including stack traces.

C. Logger

Imagine we multiple error detection modules. There are two possible approaches to error logging: distributed and centralized.

In the distributed approach, each debugging module detects its errors, queries database to get call stack for each message and writes the output to its own log. The administrator then has to inspect all the logs, find different and duplicated errors and eliminate them using the call stacks saved before.

The centralized approach includes a single logging server responsible for logging all the errors. Debugging modules should connect as clients to logging server and send error reports using specific protocol. Then the logging server queries the database server for call stacks. The administrator is able to get all the error logs in a single place so this approach is easier for administrator. The database server can also benefit from communicating with a single client instead of a couple, by caching duplicate requests. Furthermore, it is possible to only modify debugging module logging for compatibility with logging server protocol without database client addition.

VI. Implementation

Here we present our implementation of ELT[4]. This prototype we use to evaluate the performance and applicability of our approach. We have chosen POX controller because of simplicity of python it is written in. Our whole system are written in python for easier compatibility with controller.

Database server uses a simple MySQL[15] database to store and retrieve table modification messages. The usage of database tables in 3NF[3] leads to complex queries and multiple indexes slow down message insertion but also speed up message retrieval. However, it may be faster to use custom trace files instead of general-purpose database.

We used a set of JSON-formatted set of messages as client-server protocol.

VII. Evaluation

We have tested our implementation on a laptop running Ubuntu Linux with 4GB of RAM and a Core i5 2.1 GHz processor.

To evaluate the performance of our prototype we created a testing environment using Mininet[12] network simulator. We used pox's *l2_learning* routing module as controller application because it is simple and fast enough to not affect our experiments much. We measured the maximum relative slowdown in network operation between the presence and absence of our prototype. If the complexity of controller applications increases and the prototype remains unchanged, the relative slowdown decreases. Thus it is correct to take the fastest application. To induce errors in our network we wrote the application called *Interrupter*. When a packet arrives at controller, it randomly selects a pattern *P*: one field more or less precise or equal to the pattern created by *l2_learning* (we have intentionally wildcarded the VLAN field in *l2_learning's* pattern). Then with a given probability it sends one of the following messages (using pattern *P*): delete rules, modify rules (with random output port) or add a rule (with random output port). These messages conflicted with the network state installed by *l2_learning*. CST facts were detected and localized by our prototype.

We used average end-to-end ICMP-echo delay as the measure of the influence of our prototype to network. This value includes Echo Request/Reply and ARP Request/Reply delays.

We used the following prototype operation modes:

- **No proxy.** Only *l2_learning* is working, *Interrupter* is idle. Information about messages is not stored. Switch table models do not work. This mode is a normal network functioning without our prototype.
- **Saving.** Only *l2_learning* is working, *Interrupter* is idle. Information about messages is stored. Switch table models do not work.
- **Table model no errors.** Only *l2_learning* is working, *Interrupter* is idle. Information about messages is stored. Table modification messages are checked using table models.
- **Errors X (X $\in$ {0.01, 0.1, 0.5}).** *l2_learning* is working, *Interrupter* takes the described actions with probability X. Information about messages is stored. Table modification messages are checked using table models. When a CST fact is detected, the information about the messages/rules taking part is extracted from database and written to log file.

The topology used in this experiment is shown in Fig. 1. After the network is started, each host sequentially sends ICMP requests to each host on the other side of the network. The hosts work in parallel. The experiment results are shown in Figure 2. From the results acquired, we can derive the following estimates of our prototype's influence on the end-to-end delay in the network (in % from the mode without our prototype):

- Saving the information about table modification messages increases the latency not more than by 200%.
- Checking table modification message correctness increases the latency not more than by 100%.
- When the errors occur rarely (1%) their impact on the latency is negligible.
- When the errors occur with the probability of 10%, the latency increases by 100% due to processing of them.

We chose a simple and fast routing application to evaluate maximum delay growth. In case of slower applications, the overhead (in %) is lower.

In comparison with the absence of our prototype in network, the fully functional version increases the end-to-end delay of the first packet for each new flow by 300%. The transmission time of the following packets remains unchanged. This overhead is less than the overhead of static debugging tools, but still requires controller load reduction when a network operates in debug mode with our prototype.

VIII. Conclusion

In this paper we described CST errors. We found out that there is no effective approach to troubleshooting this kind of errors. We proposed the method of detection of such errors that includes a controller module analyzing table modification messages.

The overview of existing methods of troubleshooting control layer errors showed that they do not help the developer to eliminate errors in applications. To help the developer, we need integration of error detection modules, which can operate using OpenFlow level of abstraction, with error localization modules, which must have an access to controller. As an example of such module we present ELT, which can find

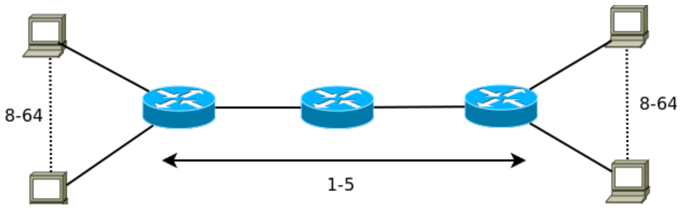


Fig. 1. Stress test topology.

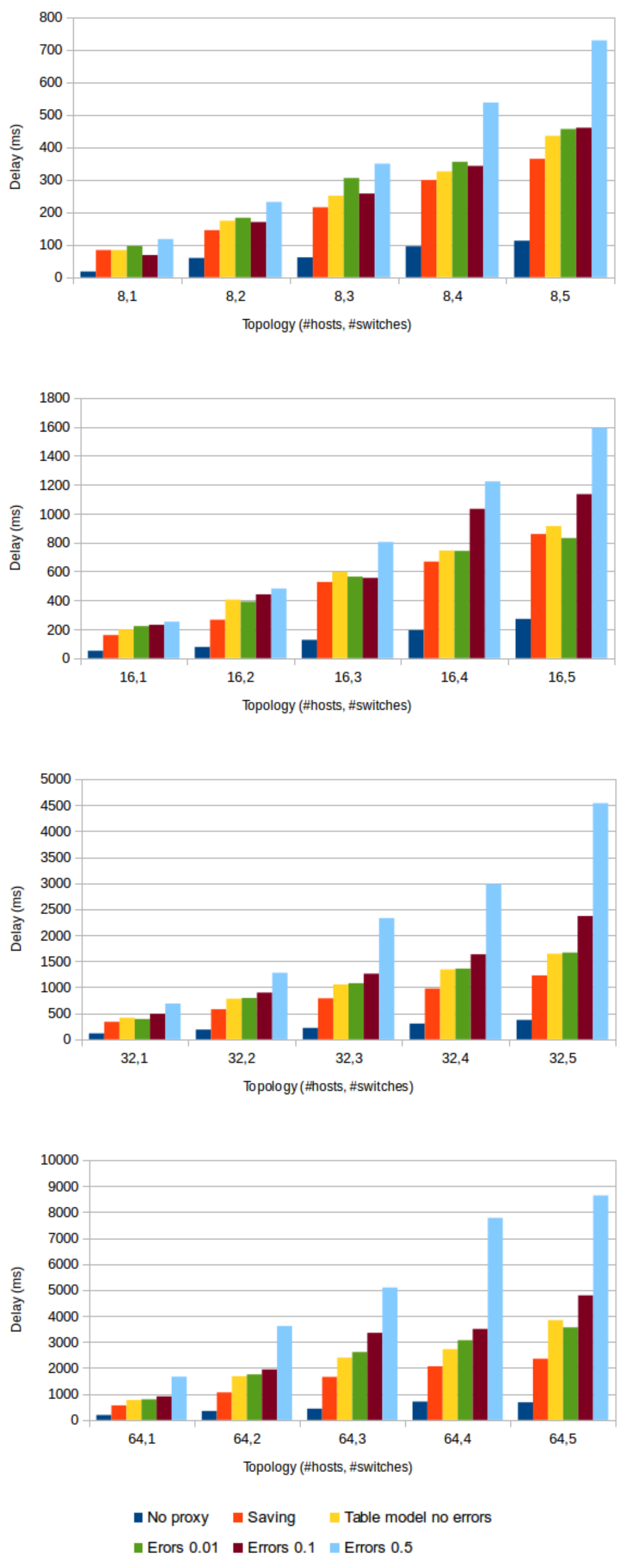


Fig. 2. Experiment results.

application and call stack responsible for an error using table modification messages and faulty rules.

We developed, implemented and evaluated a prototype of ELT. We found out that it successfully localizes errors in controller application execution. The tests showed that the addition of debugging modules increases the end-to-end delay of the first packet not more than by 300%, including 100% due to CST error detection module.

IX. Future work

On the results of this paper we identified three promising directions for further research.

Integration with network slice manager. Network slice manager (e.g. FlowVisor[18]) is a tool that helps many controllers work simultaneously. For each controller it handles a network slice including a subset of switches, ports and traffic classes. For each table modification message the network slice manager projects this message on the appropriate traffic class and sends the modified message to the switch. Modified messages can differ from the source ones and that makes ELT unable to find them in database. A module to slice manager will help in tracing such message transformations.

More accurate methods to troubleshoot CST errors. The proposed approach gives a certain number of false positives. Network administrator is the only one to know which application should affect which rules. Using the application interference specification provided by an administrator, it is possible to create a more accurate method to detect CST errors.

Debugging modules for controller. The complex debugging approach implemented in ELT involves integration of error detection modules and debugging modules for controller. Debugging modules for controller include a part of functionality of traditional source code debuggers. It is useful to research, which source code debugger's capabilities may be useful for network debugging.

References

[1] Marco Canini, Daniele De Cicco, Petr Kuznetsov, Dan Levin, Stefan Schmid, and Stefano Vissicchio. Stn: a robust and distributed sdn control plane. Open Networking Summit (ONS) Research track, March 2014.

[2] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A nice way to test openflow applications. In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[3] E. F. Codd. Further normalization of the data base relational model. IBM Research Report, San Jose, California, RJ909, August 1971.

[4] Error localization tool at github. http://github.com/ARCCN/elt.

[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazi ´res, and Nick McKeown. Where is the debugger for my software-defined network? In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[6] V. Jacobson. Unix traceroute man page, 1987.

[7] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, nsdi'13, pages 99–112, Berkeley, CA, USA, 2013. USENIX Association.

[8] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: static checking for networks. In Proceedings of the 9th USENIX Conference on Networked Systems Design and

Implementation, NSDI'12, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[9] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM.

[10] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a distributed control platform for large-scale production networks. In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[11] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, pages 290–301, New York, NY, USA, 2011. ACM.

[12] Mininet network simulator. http://mininet.org/.

[13] Jeffrey C. Mogul, Alvin AuYoung, Sujata Banerjee, Lucian Popa, Jeongkeun Lee, Jayaram Mudigonda, Puneet Sharma, and Yoshio Turner. Corybantic: towards the modular composition of sdn control programs. In Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, pages 1:1–1:7, New York, NY, USA, 2013. ACM.

[14] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, nsdi'13, pages 1–14, Berkeley, CA, USA, 2013. USENIX Association.

[15] Mysql open source database. http://www.mysql.com/.

[16] OpenFlow switch specification v1.1.0. http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[17] Pox controller. http://www.noxrepo.org/pox/about-pox/.

[18] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Can the production network be the testbed? In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[19] tcpdump official site. http://www.tcpdump.org/.

[20] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, USENIXATC'11, pages 29–29, Berkeley, CA, USA, 2011. USENIX Association.

[21] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In Proceedings of the 8th InternationalConference on Emerging Networking Experiments and Technologies, CoNEXT '12, pages 241–252, New York, NY, USA, 2012. ACM.

[22] David MW Powers. Evaluation: from precision, recall and f-factor to roc, informedness, markedness & correlation. SIE-07-001, 2007.

[23] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyi Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, Kyriakos Zarifis, and Peyman Kazemian. Leveraging sdn layering to systematically troubleshoot networks. HotSDN '13, pages 37-42, New York, NY, USA, 2013. ACM.

Data Center Resource Mapping Algorithm Based on the Ant Colony Optimization

A. Plakunov, V. Kostenko
Faculty of Computational Mathematics and Cybernetics
Moscow State University
Moscow, Russia
artacc@lvk.cs.msu.su, kost@cs.msu.su

***Abstract*—In this paper we present a data center resource mapping algorithm based on the ant colony optimization approach. The algorithm considers data centers to present IaaS model and can be used in a joint scheduler for all resource types. The algorithm uses ant colony optimization approach to map resource requests to physical computational nodes and data storages. The shortest path algorithm is used to map virtual channels to the data center's physical network channels and network switches. We then present a comparison of the developed algorithm with an algorithm based on greedy and limited exhaustive search strategies.**

***Keywords*—Ant Colony Optimization, Data Centers, Virtualization, Cloud Platforms**

I. Introduction

We consider a resource usage efficiency problem which is a crucial problem in Infrastructure-as-a-Service datacenters with physical resources load as en efficiency metrics. The problem consists of different resource placement subproblems, where each virtual resource has its own related physical resource. Each virtual resource requests certain SLA, and each physical resource has parameters related to this SLA. The problem of resource placement is to map virtual resources onto physical resources with all SLA to be guaranteed.

Existing algorithms either don't consider mapping of all three resource types [1-7] or can only be used for a fixed data center network topology [8,9]. In this paper we will show how we can implement multitype resource mapping for any network topology using an algorithm based on the ant colony optimization approach.

Ant colony optimization approach is used to map virtual machines on the computational nodes and to map virtual storages on the data storages. On mapping completion virtual channels are mapped on the physical ones using a Dijkstra shortest path algorithm. The advantage of this scheme is that ant colony optimization approach allows the algorithm to automatically adjust itself for a particular problem by additional input data marking up that is used to build a solution on each iteration. The solutions are improving as the number of iterations is growing. This mechanism can provide high quality solutions on a wide class of the input data [10].[1]

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and the Ministry of education and science of
the Russian Federation, Unique ID RFMEFI60714X0070

II. Problem Definition

Data center physical resource model is defined as a weighted graph [11]:

$$H = (P \cup M \cup K, L)$$

where P is a set of computational nodes, M is a set of data storages, K is a set of network switches, L is a set of network channels. The weights are defined as follows.

- Weights $vh(p)$ and $vr(p)$, defined on the set P, are the number of the CPU cores and the amount of the operational memory of the cpu node $p \in P$.
- Weight $uh(m)$, defined on the set M, is a capacity of the data storage $m \in M$ (bytes).
- Weight $bh(k)$, defined on the set K, is a bandwidth of the network switch $k \in K$ (bytes per second). The network switch bandwidth is defined as a maximum total bandwidth of the virtual channels coming through the switch. We consider all input and output switch ports to have an equal priority.
- Weight $rh(l)$, defined on the set L, is a bandwidth of the network channel $l \in L$ (bytes per second).

Resource request is defined as a weighted graph:

$$T = (W \cup S, E)$$

where W is a set of virtual machine requests, S is a set of virtual storage requests, E is a set of virtual channel requests. The weights are defined as follows.

- Weights $v(w)$ and $r(w)$, defined on the set W, are the requested number of CPU cores and the requested amount of operational memory of the virtual machine request $w \in W$;
- Weight $u(s)$, defined on the set S, is a required capacity for the virtual storage request $s \in S$ (bytes);
- Weight $r(e)$, defined on the set E, is a bandwidth for the virtual channel request $e \in E$ (bytes per second).

Resource request mapping is defined as follows:

$$A : T \to H = \{W \to P, S \to M, E \to \{K, L\}\}$$

Given:

- A set of requests Z = $\{T_i\}$ received by the data center scheduler.
- Data center physical resource model $H_{res} = (P \cup M \cup K, L)$.

The problem is to define resource request mappings $\{A_i\}$ for as much requests as possible (target function $F = |\{A_i\}|$). The mappings should meet the following constraints:

- $\sum_{w \in W_p} v(w) \le vh(p)$
- $\sum_{w \in W_p} r(w) \le rh(p)$
- $\sum_{e \in E_l} r(e) \le rh(l)$
- $\sum_{e \in E_k} r(e) \le \tau h(k)$
- $\sum_{s \in S_m} u(s) \le uh(m)$

Here W_p is a set of virtual machines mapped on the computational node *p*, E_l is a set of virtual channels mapped on the physical channel *l*, E_k is a set of virtual channels mapped on the switch *k*, S_m is a set of virtual storages mapped on the physical storage *m*. These constraints mean that capacity of physical resources cannot be exceeded. This guarantees the SLA to be satisfied.

The algorithm is supposed to be used inside a cloud platform [12]. There is a controller which gathers incoming requests for a certain time span and decides when to launch the scheduler. Round of scheduling should last at least 15 minutes due to the algorithm requiring around 15 minutes to complete on a set of 100 requests. There are also additional constraints defined by the platform:

1) CPU cores cannot be shared between virtual machines.

2) Virtual channel should be mapped onto a path in the physical network

III. PROPOSED ALGORITHM

A. Algorithm Common Scheme

The initial problem can be divided into three subproblems:

1) Mapping virtual machines to physical computational nodes.

2) Mapping virtual storages to data storages.

3) Mapping virtual channels to physical network channels.

We can solve the subproblems 1 and 2 by an algorithm based on the ant colony optimization approach. Since the ant colony optimization approach is intended for optimization problems represented in the form of the shortest path problem, subproblems 1 and 2 should be reduced to it (target reduction graph is denoted as *G*). The reduction process is described in section 3.2.1.

After steps 1 and 2 are complete we can solve the problem of mapping the virtual channels to physical ones by a greedy algorithm.

The algorithm's scheme is:

1) Build the graph *G*. The graph form is chosen so that path in the graph determines mapping of virtual machines and virtual storages

2) Build paths B_i in the graph *G*. The path is built according to the restrictions on maximum computational node performance *vh(p)* and maximum data storage memory volume *uh(m)*.

3) For each B_i map virtual channels to physical ones given that virtual machines and virtual storages are mapped according to path B_i.

4) Calculate the target function F_i for each path B_i.

5) Update the pheromone values on the arcs of the graph *A* depending on the target function values F_i.

6) If the stop condition isn't satisfied, go to stage 2.

B. Basic Algorithm Operations

1) Building Graph

Let *N* be the number of computational nodes in the data center and *S* be the number of data storages, and let *R* be the number of resource requests to be mapped. Each request consists of $n_i, i = 1..R$ virtual machine requests and $s_i, i = 1..R$ virtual storage requests. The vertices $V_1, ..., V_N$ and $S_1, ..., S_S$ are added where the vertex V_i corresponds to the computational node with the number *i*, and the vertex S_j corresponds to the data storage with the number *j*. For each $k = 1..n, n = \sum_{i=1}^{R} n_i$ let one graph vertex to correspond to *k*-th virtual machine request: this vertex V_0^k is connected to each of the vertices $V_1, ..., V_N$ by two differently directed arcs.

For each $k = 1..n$ vertex V_0^k is connected to each of the vertices $V_0^l, l = 1..n, l \ne k$ by two differently directed arcs (figure 1).

Similarly, for each $k = 1..s, s = \sum_{i=1}^{R} s_i$ let one graph vertices to correspond to *k*-th virtual storage request. For each $k = 1..s$ vertices S_0^k are connected to each of the $S_0^l, l = 1..s, l \ne k$ by two differently directed arcs (fig. 1).

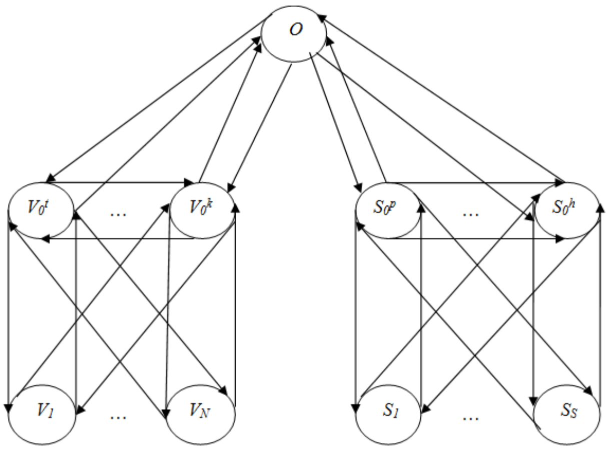


Fig. 1. Graph G structure

Let's also add vertex O connected to each of the $V_0^m, m = 1..n$ and $S_0^l, l = 1..s$ by two differently directed arcs. This vertex will be the starting vertex for each ant. Ants can only choose this vertex when they have no other vertices to choose.

The following two values correspond to each arc in the graph: τ_{ij} is the amount of pheromone on the arc (i,j) and η_{ij} is a heuristic function set for arc (i,j). The current value of i-th computational node load corresponds to each of the vertices $V_1,...,V_i^k,...,V_N$ and the current amount of free memory on the i-th data storage corresponds to each of the vertices $S_1,...,S_i,...,S_S$.

2) Building Paths in the Graph

Each ant starts its path in the vertex O. The same ant can't go on the same arc twice. The ant chooses the next vertex by a probabilistic rule. The probability for the k-th ant to travel from vertex i to vertex j on the iteration t depends on the list of visited arcs, amount on pheromone and heuristic values on the available arcs (1).

$$B_{ij,k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha} \cdot (\eta_{ij}(t))^{\beta}}{\sum_{l \in J_k} (\tau_{il}(t))^{\alpha} \cdot (\eta_{il}(t))^{\beta}}, j \notin L_k \\ 0, (i,j) \in L_k \end{cases} \tag{1}$$

Here $\tau_{ij}(t)$is the amount of pheromone of the arc (i,j), $\eta_{ij}(t)$ is a heuristic function on the arc (i,j), $\alpha \geq 0$ and $\beta \geq 0$ are algorithm's parameters determining the importance of the pheromone and heuristic in the process of choosing the arc, L_k is the list of visited arcs of the k-th ant.

When building a path from the vertex S_0^k to the vertex S_u the type of the virtual storage request that corresponds to the vertex S_0^k, is compared to the type of the u-th data storage that corresponds to the vertex S_u. If the types aren't match, the ant can't choose this arc.

When building a path from the vertex V_0^k to the vertex V_0^u the ant can't choose the arc if he has already visited the vertex V_0^u.

After the ant has chosen a vertex V_u, the virtual machine request that corresponds to the vertex V_0^k is added to the W_u set. The current value of u-th computational node load is increased by the value $v(k)$. Same actions are performed with the S_u set when a virtual storage request is chosen.

One of the possible ways to set the $\eta_{ij}(t)$ function is as follows:

- $\eta_{ij}(t) = \dfrac{v(w)}{\max\limits_{m \in W}(v(m))}$ for $j = V_0^k, \forall i, \forall k = 1..n$
- $\eta_{ij}(t) = \dfrac{u(s)}{\max\limits_{r \in S}(u(r))}$ for $j = S_0^k, \forall i, \forall k = 1..s$

Here $v(w)$ and $u(s)$ are, respectively, the requested performance of the virtual machine request and the capacity of the virtual storage request.

If $i = V_0^k, j = V_m, \forall m = 1..N, \forall k = 1..n$ the function $\eta_{ij}(t)$ is calculated as shown in (2).

$$\eta_{ij}(t) = \left(\sum_{w \in W_j} v(w) \right) - v(i) \tag{2}$$

If $i = S_0^k, j = S_m, \forall m = 1..S, \forall k = 1..s$ the function $\eta_{ij}(t)$ is calculated as shown in (3).

$$\eta_{ij}(t) = \left(\sum_{s \in S_j} u(s) \right) - u(i) \tag{3}$$

Here W_j (S_j) is the set of currently mapped virtual machines (the set of currently mapped virtual storages) to the computational node (to the data storage), corresponding to the vertex j. If, respectively, $\eta_{ij}(t) < 0$ then $\eta_{ij}(t)$ is set to zero.

If the corresponding virtual machine (virtual storage) can't be mapped to the computational node (data storage) due to performance (memory) restrictions violation the probability for the ant to choose the arc is zero. If the probability is zero for all the available arcs at the moment, ant skips the step and the entire resource request is considered as a non-mapped request, and all the virtual machine requests and virtual storage requests corresponding to the same resource request are removed from the mapping. After the function $\eta_{ij}(t)$ is calculated for all the arcs of one vertex, $\eta_{ij}(t)$ values for these arcs are normalized (4).

$$\eta_{ij}(t) = \frac{\eta_{ij}(t)}{\max_i \eta_{ij}(t)} \tag{4}$$

3) Virtual Channel Mapping Algorithm

Let vl be the virtual channel to map and let i and j be the vertices of the physical resource graph H where the virtual resources connected by vl are mapped to.

Physical resource graph H is temporarily modified before vl is mapped:

- All the arcs (p,q) where vertex q is a network switch and p isn't, and $p \neq i$ are deleted.
- All the arcs (p,q), where vertex p is a network switch and q isn't, and $q \neq j$ are deleted.
- Arcs connecting two network switches are duplicated and set to different directions.
- Arcs connecting a computational node or a data storage to a network switch are directed towards the network switch.

Let's consider a connected component C in the modified graph H containing the vertex i. This connected component also contains vertex j and contains no other vertices that aren't a network switch by construction. The connected component C is used to map vl as follows:

- A weight is assigned to all the remaining arcs (p,q) in the graph C. The weight equals to $(\tau h(q) - r(vl)) + (rh(l_{pq}) - r(vl))$ if vertex p is a network switch, and equals to $rh(l_{pq}) - r(vl)$ in other cases, where l_{pq} is a physical channel connecting the arcs p and q. If one of the weights or the summands in the formula is below zero, the corresponding arc is temporarily deleted from the graph. The weights are chosen so all the arcs which the channel can't be mapped to are deleted. The less capacity remains on the network channel after the virtual one is mapped, the less the weight of the corresponding arc.
- Dijkstra's algorithm is used to build the shortest path from the vertex i to the vertex j in the weighted graph.

4) Pheromone Update Rule

After the target functions are calculated the pheromone values for each arc in the graph are updated. The additional pheromone value for an arc depends on the target function value that corresponds to the path this arc is included in (5).

$$\Delta\tau_{ij,k}(t) = \begin{cases} F_k, (i,j) \in B_k(t) \\ 0, (i,j) \notin B_k(t) \end{cases} \tag{5}$$

Here $B_k(t)$ is the path built by the k-th ant and F_i is the target function equals to the number of the successfully mapped requests divided to the total number of the requests.

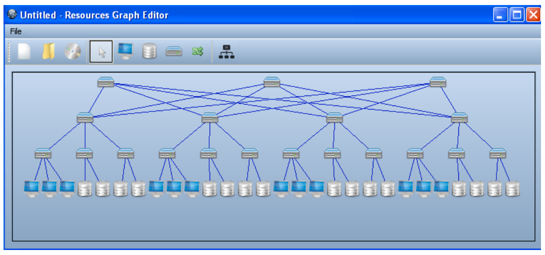


Fig. 2. Network topology used for research

A pheromone evaporation coefficient $p \in [0;1]$ defines how much pheromone will be left after previous iterations. So the total value of the pheromone on the arc (i, j) after iteration t is calculated as shown in (6).

$$\tau_{ij}(t+1) = (1-p)\cdot\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \tag{6}$$

IV. EXPERIMENTAL RESEARCH

The purpose of the research is to compare the results of the developed algorithm and an algorithm combining greedy and exhaustive search strategies (heuristic algorithm) [13] on one of the typical data center physical network topologies using modelled data. Papers [11,12] also show that the number of mapped requests by this algorithm greatly exceeds the number of mapped requests by algorithms currently used in OpenStack.

A. Research Metodology

The research was conducted using the following input data parameter values:

- Standart data center topology "fattree" (fig. 2) with 60 computational nodes (each with 16 CPU cores and 1000 arbitrary units of operational memory) and 60 data storages (each with the capacity of 1000 arbitrary units).

The following patterns were used to generate the set of requests:

- The pattern with a small number of virtual channels: seven virtual machines, five virtual storages, and eleven virtual channels (on the average, two virtual channels for one virtual storage).
- The pattern with a large number of virtual channels: five virtual machines, two virtual storages, and eight virtual channels (four virtual channels for each virtual storage);
- The pattern with a small number of virtual machines: three virtual machines and two virtual storages. One of the virtual storages was connected to all virtual machines and the other one was connected to one virtual machine (four virtual channels in total);

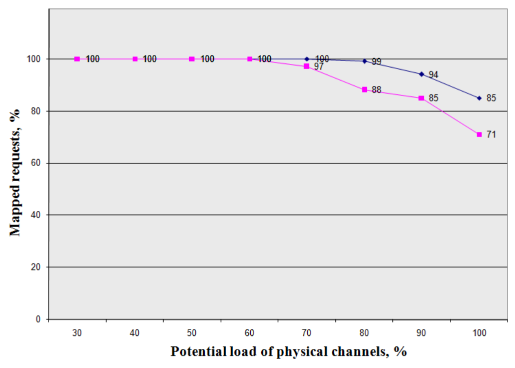


Fig. 3. Algorithm comparison on the first class of data, —◆— – ant colony algorithm, —■— – heuristic algorithm.

- The pattern with high requirements for the network bandwidth: two virtual machines, one virtual storage and two virtual channels between them. The requested virtual channel bandwidths were chosen so that their sum was 900 arbitrary units (with the bandwidth of 1000 arbitrary units for a channel connected to a data storage).

The sets of requests were divided into two basic classes and were generated using the following schemes.

- In the first class the potentially possible load of the channels (with the optimal mapping) connected to data storages varied from 0.3 to 1.0. The load of the computation nodes and data storages was fixed to 0.75. In this class only requests generated by the first and second patterns were used.
- In the second class only requests generated by the third and fourth patterns were used. In this class, the number of requests generated by the fourth pattern varied from 0 to 30; in this case, the potentially possible load of the network varied from 0.5 to 0.8. The load of the computation nodes and data storages was fixed to 0.75.

The number of requests in each set was 100.

B. Research Results

Fig. 3 demonstrates the percentage of mapped requests depending on the network load (the first class of data).

The graph shows that when the network load is greater than 0.6 the developed algorithm maps more requests and the difference reaches 14% as the network load grows.

Fig. 4 demonstrates the percentage of mapped requests depending on the number of requests generated by the fourth pattern (the second class of data).

Unlike the first class of data, the maximum difference between the algorithms is 6% and barely change as the number of the fourth-type requests grow.

Also note that the first class of data is mainly focused on proper mapping of the virtual channels: there are a large number of channels with a low requested bandwidth. The second class of data requires the algorithms to properly map the virtual machines and virtual storages: in case of ineffective mapping of these elements it becomes impossible to map the virtual channels with high requested bandwidth.

Since the heuristic algorithm maps virtual machines and virtual storages separately and uses exhaustive search to improve the mapping it does fairly well on the second class of data. This advantage does not apply to the first class of data though: even if virtual machines and virtual storages are mapped effectively, there might be no way to map virtual channels. The ant colony algorithm bypasses this problem as it considers more ways to map resources and improves the best mappings from iteration to iteration.

The developed algorithm shows about the same results on both classes of the input data which means that it is more universal than the heuristic algorithm.

V. Conclusions

The paper proposes a data center resource mapping algorithm based on the ant colony optimization approach. The algorithm allows to map virtual machines, virtual storages and virtual channels and is not bound to a certain data center network topology.

The experimental research showed that the developed algorithm is more universal than the heuristic algorithm. The algorithm maps 98-100% requests when the network load is less than 70% and 90-95% requests on higher network load on the considered classes of input data.

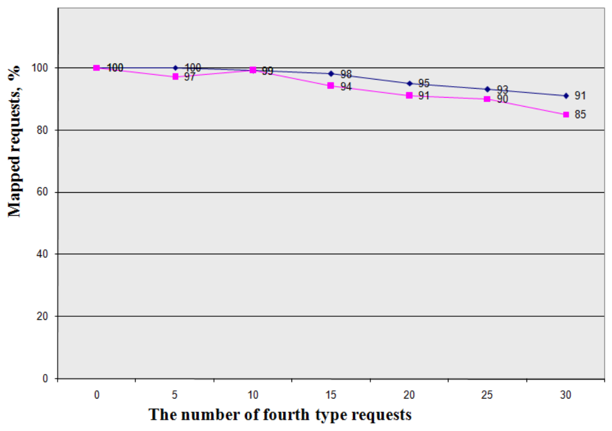


Fig. 4. Algorithm comparison on the second class of data, —◆— – ant colony algorithm, —■— – heuristic algorithm.

References

[1] Nagendram S., Lakshmi J.V., Rao D.V., Jyothi Ch.N. Efficient resource scheduling in Data Centers using MRIS // Indian J. Computer Science and Engineering. 2011. V. 2. Issue 5. P. 764-769.

[2] Arzuaga E., Kaeli D.R. Quantifying load imbalance on virtualized enterprise servers // Proc. of the First Joint WOSP/SIPEW Intern. Conf. on Performance Engineering. San Josa, CA: ACM, 2010. P.235-242.

[3] Mishra M., Sahoo A. On Theory of VM placement: anomalies in existing methodologies and their mitigation using a novel vector based approach // Cloud Computing (CLOUD), IEEE Intern. Conf. Washington: IEEE Press, 2011. P.275-282.

[4] Botero J.F., Hesselbach X., Fischer A., Hermann M. Optimal mapping of virtual networks with hidden hops // Telecommunication Systems. 2012. V.51. №4. P.273-282.

[5] Yu M., Yi Y., Rexford J., Chiang M. Rethinking virtual network embedding: substrate support for path splitting and migration // ACM SIGCOMM Computer Communication Review. 2008. V.38. №2. P.17-29.

[6] Lischka J., Karl H. A Virtual network mapping algorithm based on subgraph isomorphism detection // Proc. of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures. Barcelona: ACM, 2009. P.81-88.

[7] Cheng X., Sen S., Zhongbao Z., Hanchi W., Fangchun Y. Virtual network embedding through topology-aware node ranking // ACM SIGCOMM Computer Communication Review. 2011. V.41. №2. P.38-47.

[8] Korupolu M., Singh A., Bamba B. Coupled placement in modern Data Centers // IEEE Intern. Symp. on Parallel & Distributed Processing. N. Y.: IPDPS, 2009. P.1-12.

[9] Singh A., Korupolu M., Mohapatra D. Server-storage virtualization: integration and load balancing in Data Centers // Proc. of the 2008 ACM/IEEE Conf. on Supercomputing. Austin: IEEE Press, 2008. P.1-12.

[10] Kostenko V.A., Plakunov A.V. An Algorithm for constructing single machine schedules based on Ant Colony Approach // Journal of Computer and System Sciences International, 2013, Vol.52, No.6, pp. 928-937.

[11] Vdovin P.M., Zotov I.A., Kostenko V.A., Plakunov A.V., Smelyansky R.L. Comparing Various Approaches to Resource Allocating in Data Centers // J. of Computer and Systems Sciences Intern. 2014. V. 53. № 5.

[12] Kostenko V., Plakunov A., Nikolaev A., Tabolin A., Smeliansky R., Shakhova M. Selforganizing cloud platform // Proceedings of the 2014 International Science and Technology Conference «Modern Networking Technologies (MoNeTec)»

[13] Vdovin P.M., Kostenko V.A. Algorithm for Resource Allocation in Data Centers with Independent Schedulers for Different Types of Resources // J. of Computer and Systems Sciences Intern. 2014. V. 53. № 6.

Development of educational resource datacenters based on software defined networks

P. Polezhaev, A. Shukhman, A. Konnov
Mathematics Department
Orenburg State University
Orenburg, Russian Federation
peter.polezhaev@gmail.com

***Abstract*—This paper describes application features of software defined networks (SDN) to build educational resource datacenters (ERD), which are intended for shared remote access to paid software for students from different educational institutions. We proposed a virtual classroom scheduling algorithm based on simulated annealing heuristic, a genetic algorithm for traffic routing and providing QoS for data flows. We have implemented these algorithms using C++ and partially tested them on the ERD simulator, which is still under development. Preliminary experimental studies have demonstrated their efficiency. The main feature of the proposed solutions is an interrelation between the virtual classroom scheduling algorithm and the algorithm for proactive routing and providing QoS parameters. The first algorithm reports the information about an assignment of virtual classroom's machines to the physical servers and its communication pattern to the second algorithm.**

***Keywords*—SDN; educational resource datacenter; virtual classroom scheduling; traffic routing**

I. INTRODUCTION

At present the majority of educational institutions in the Russian Federation are not sufficiently funded for purchasing software required in the educational process. It results in lower quality of education and in illegal software use. This problem can be solved by creating educational resource datacenter (ERD) with the possibility of remote access to shared paid software for educational institutions [1]. The ERD can be built on the base of cloud datacenter (fig.1).

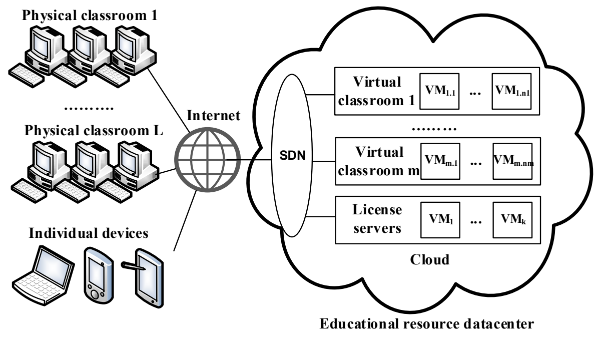


Fig. 1. Logical structure of educational resource datacenter

This work is supported by the Russian Foundation for Basic Research (grants 13-07-97046 and 14-07-97034) and by the Foundation for Assistance of Development of Small Businesses in the Field of Science and Technology (grant UMNIK 2628GU1/2014).

The cloud system provides the DaaS (Desktop as a Service) with the access to virtual machines (VM) including all essential free and paid software for each student. VMs are grouped into virtual computer classrooms created by a coordinator for each educational institution. Access is granted to students over Internet using the computers from actual classrooms or their own devices as notebooks, smartphones and tablets. Additionally the ERD includes several virtual machines containing the license servers for paid software limiting the number of active applications at same time.

Note that educational institutions can use outdated low-performance computers for access to software with browser. The ERD allows using expensive software in the learning process for study researches, distance learning, access to virtual laboratories.

Using the ERD can be organized as follows. An educational institution shall appoint a responsible coordinator for working with the ERD. The coordinator gathers information on planned lessons and for each of them determines the number of required VMs and the list of needed software. The educational institution and the ERD conclude a contract about services, the institution transfers money to its prepaid account. The coordinator creates the needed number of virtual classrooms with needed software through the ERD control system, determines the lessons for each classroom, and accepts one of the schedule variants offered by the ERD control system. Then the control system creates virtual machines for virtual classrooms with automatic installation of the selected software and updates schedule of the ERD. During the academic year, teachers use virtual classes at lessons according to the schedule. If it is necessary, the coordinator can change the time of lessons, install additional software, and create new virtual classrooms; if the schedule of the ERD allow it. Payment is done from prepaid account in accordance with the consumption of the ERD resources and the use of paid software. This concept implies the solution of a number of tasks.

A. Automatic creation and setting of virtual classrooms

The coordinator has to be able to create and set up virtual classrooms with the use of ERD web-site choosing the virtual machine characteristics, setting the number of their copies and determining the essential software, further settings performed automatically. To achieve this we propose to integrate cloud management tools for creating and controlling virtual machines and software configuration management system automatically installing and configurating software. For silent software installation the scripts have been developed.

B. Virtual classroom and machine scheduling

The ERD scheduling algorithms should be developed to consider: predetermined time intervals for sessions, weekly schedule cyclicity, actual server restrictions, and software license restrictions. The existing studies on cloud scheduling [2-8] and cloud resource control systems, such as OpenStack [9], OpenNebula [10], Eucalyptus [11], Amazon EC2 [12], VMWare vCloud Suit [13], Oracle Enterprise Manager Cloud Control [14], Moab Cloud Suite [15], Citrix Cloud Platform [16], don't consider the above factors. An ERD simulator is being developed for estimation of the algorithm efficiency for different hardware configurations of datacenter, software restrictions and administrator request flows. Further on the best algorithm variations are to be studied within an actual ERD at our University.

C. Efficient routing within ERD with required QoS

To provide acceptable response time for remote desktops the QoS characteristics have to be configurated as the minimum guaranteed bandwidth and the maximum guaranteed delay. The data flow routing should not violate the QoS requirements for the other flows and should provide dynamic route changes in case of virtual machine migration.

II. PROBLEM OF VIRTUAL CLASSROOM SCHEDULING

The ERD is a cloud-based system, A cloud system can be described by the triple

$$\mathrm{Cloud} = (\mathrm{Nodes}, \mathrm{Flavors}, \mathrm{Software}), \tag{1}$$

where $\mathrm{Nodes} = \{\mathrm{Node}_i\}_{i=\overline{1,m}}$ is a set of nodes (servers), $\mathrm{Flavors} = \{\mathrm{Flavor}_i\}_{i=\overline{1,q}}$ – set of typical virtual machine configurations, $\mathrm{Software} = \{\mathrm{Program}_i\}_{i=\overline{1,h}}$ – set of available software.

Each server Node_i is determined by the following parameters:

$$\mathrm{Node}_i = (C_i^{\mathrm{node}}, M_i^{\mathrm{node}}, D_i^{\mathrm{node}}), \tag{2}$$

where C_i^{node} is the number of its computational cores, M_i^{node} and D_i^{node} are accordingly sizes of its RAM and local HDD.

Each typical virtual machine configuration Flavor_i is characterized by the same parameters:

$$\mathrm{Flavor}_i = (C_i^{\mathrm{flavor}}, M_i^{\mathrm{flavor}}, D_i^{\mathrm{flavor}}). \tag{3}$$

Let the ERD serves r educational institutions. Each of them is represented by coordinator and is characterized by

$$K_i = (w_i, \mathrm{Classrooms}_i), \tag{4}$$

where $w_i \in [0;1]$ is his priority and $\mathrm{Classrooms}_i = \{\mathrm{Classroom}_{ij}\}_{j=\overline{1,p_i}}$ are his virtual classrooms. Each virtual classroom $\mathrm{Classroom}_{ij}$ is characterized by the following parameters:

$$\mathrm{Classroom}_{ij} = (n_{ij}, \mathrm{flavor}_{ij}, g_{ij}, s_{ij}), \tag{5}$$

where n_{ij} is the number of its virtual machines, $\mathrm{flavor}_{ij} \in \mathrm{Flavors}$ – their typical configuration, $g_{ij} = (g_{ij1}, ..., g_{ijR})$ – coordinator's requests representing a vector of preferred time slots for classes, $s_{ij} = (s_{ij1}, ..., s_{ijH})$ – a vector describing software installed on virtual machines ($s_{ijz} = 1$, if software $\mathrm{Program}_z$ is installed on virtual machines of $\mathrm{Classroom}_{ij}$, otherwise $s_{ijz} = 0$).

The ERD schedule is developed for the period of one or two weeks, the period is divided into equal intervals (time slots) for classes. All the time slots are numbered from 1 to R. $g_{ijk} = 1$, if i-th coordinator prefers to allocate k-th time slot for his j-th virtual classroom, otherwise $g_{ijk} = 0$. The sum $\sum_{k=1}^{R} g_{ijk}$ represents the number of classes for $\mathrm{Classroom}_{ij}$ planned by i-th coordinator.

The ERD schedule can be defined by the set of tuples:

$$S = \{(k, i, j, l, r)\}. \tag{6}$$

Each tuple (k, i, j, l, r) describes the assignment of one virtual machine from $\mathrm{Classroom}_{ij}$ to the r-th computational core of the l-th server for the k-th time slot.

Schedule S is feasible, if it satisfies the following constraints:

1) Physical constraints for the assignment of virtual machines to servers (for each node its RAM, HDD and computational cores are not overused in each time slot):

$$\forall k = \overline{1,R}, \forall l = \overline{1,m} \;\; M_l^{\mathrm{used}} = \sum_{\substack{i,j: \\ \exists r: (k,i,j,l,r) \in \mathrm{Schedule}}} M_{ij}^{\mathrm{flavor}} \le M_l^{\mathrm{node}}, \tag{7}$$

$$\forall k = \overline{1,R}, \forall l = \overline{1,m} \;\; D_l^{\mathrm{used}} = \sum_{\substack{i,j: \\ \exists r: (k,i,j,l,r) \in \mathrm{Schedule}}} D_{ij}^{\mathrm{flavor}} \le D_l^{\mathrm{node}}, \tag{8}$$

$$\forall k = \overline{1,R}, \forall l = \overline{1,m} \;\; C_l^{\mathrm{used}} = \sum_{\substack{i,j: \\ \exists r \, (k,i,j,l,r) \in \mathrm{Schedule}}} C_{ij}^{\mathrm{flavor}} \le C_l^{\mathrm{node}}. \tag{9}$$

2) Licensed software constraints (restriction on parallel execution of software instances in virtual machines for each time slot and each software):

$$\forall k=\overline{1,R}, \forall z=\overline{1,h}: \quad \sum_{\substack{i,j:\\ \exists l \exists r\\ (k,i,j,l,r)\in Schedule \& s_{ijz}=1}} n_{ij} \le I_z, \tag{10}$$

where I_z is the maximum allowed number of instances for software $Program_z$.

We formalize the virtual classroom scheduling problem as the optimization of the following function:

$$F(s)=\sum_{i=1}^{r} w_i\left[\alpha\sigma_i-\beta\mu_i-\gamma\xi_i\right]\to \max, \tag{11}$$

where α is an encouragement for assignment of one core of virtual machine to some server's core for a preferred time slot for corresponding coordinator, σ_i – the number of such cores for the i-th coordinator, β – the penalty for assignment of one core of virtual machine to some server's core for a not preferred time slot for corresponding coordinator, μ_i – the number of such cores of the i-th coordinator, γ – the penalty for not assigning any core of virtual machine to any available physical server of the ERD during one required time slot (considering that all the cores of all the virtual machines of each virtual classroom should be assigned simultaneously to the servers for any selected time slot), ξ_i – the number of such cores for the i-th coordinator, σ_i, μ_i, ξ_i are the characteristics of the ERD schedule s.

III. Proposed Simulated Annealing Algorithm for Virtual Classroom Scheduling

The optimization problem on constructing a schedule maximizing the function (11) and satisfying the constraints (7-10) can be solved by a simulated annealing heuristic algorithm. It is based on the physical process of substance crystallization involving controlled cooling (see detailed information in [17]).

The proposed algorithm has the following steps (see algorithm 1):

Algorithm 1 – Simulated annealing algorithm for virtual classroom scheduling in the ERD

Step 1. Create an initial $Schedule_1$. Let L_1 be a set of not assigned virtual classrooms for necessary classes due to the lack of free resources.

Step 2. Set the initial temperature $t_1 := t_{max}$, set the initial number of iteration $i := 1$.

Step 3. While $t_i > t_{min}$ and $i \le I_{max}$, do the following:

Step 3.1. Set $Schedule_c := Schedule_i$. Select a random value $r \in [0,1]$.

Step 3.2. If $r \le q$ then do the following:

Step 3.2.1. On the basis of $Schedule_i$ create M_i – the list of assigned virtual machines of virtual classrooms, which can be moved from one server to another one in the same time slot of the schedule considering resource constraints of the servers.

Step 3.2.2. If $|M_i| = 0$ then go to the step 3.3.1.

Step 3.2.3. Select a random number $k \in \overline{1,|M_i|}$.

Step 3.2.4. Select the best fit server $Node_j$ for virtual machine $VM_k \in M_i$ in its time slot. The following criteria should be used for selection (to minimize remaining resources):

$$j=\arg\min_{j=1,m}\{\omega_M\cdot\left|\frac{M_j^{node}-M_j^{used}-M_k^{flavor}}{M_j^{node}}\right|+ \\ +\omega_D\cdot\left|\frac{D_j^{node}-D_j^{used}-D_k^{flavor}}{D_j^{node}}\right|+ \\ +\omega_C\cdot\left|\frac{C_j^{node}-C_j^{used}-C_k^{flavor}}{C_j^{node}}\right|\}, \tag{12}$$

where $0\le \omega_M,\omega_D,\omega_C \le 1$ are weight coefficients describing the fitness degree of RAM, HDD and the number of computational cores.

Step 3.2.5. Update $Schedule_c$ to apply the move of VM_k to $Node_j$.

Step 3.3. If $r > q$ then do the following:

Step 3.3.1. On the basis of $Schedule_i$ create M'_i – the list of virtual classrooms, which can be moved from one time slot to another one in the schedule considering resource and licensed software constraints.

Step 3.3.2. If $|M'_i| = 0$ then go to the step 3.6.1

Step 3.3.3. Select a random number $k \in \overline{1,|M'_i|}$.

Step 3.3.4. Select $Classroom_k \in M'_i$.

Step 3.3.5. If the schedule has preferred and suitable time slots, where $Classroom_k$ can be moved, then move it to any random of them and assign to the random servers which satisfy resource constraints. Else, move $Classroom_k$ to any suitable random time slot.

Step 3.3.6. Update $Schedule_c$ to apply the move of $Classroom_k$ between time slots.

Step 3.4. Set $L_{i+1} := L_i$.

Step 3.5. Select a random value $r' \in [0,1]$.

Step 3.6. If $r' \le q'$ then do the following:

Step 3.6.1. On the basis of L_i and Schedule_c create $M''_i \subseteq L_i$ – the list of virtual classrooms, which has classes not assigned to any time slots in the schedule, but which can be assigned to the servers of at least one time slot.

Step 3.6.2. If $|M_i| = |M'_i| = |M''_i| = 0$ then do the following:

Step 3.6.2.1. Select a random number $k \in \overline{1,R}$.

Step 3.6.2.2. Remove from Schedule_c all the classes assignments to the k-th time slot and add them to L_{i+1}.

Step 3.6.2.3. Go to the step 3.7.

Step 3.6.3. Select a random number $k \in \overline{1,|M''_i|}$.

Step 3.6.4. Select $\text{Classroom}_k \in M''_i$.

Step 3.3.5. If the schedule has preferred and suitable time slots, where Classroom_k can be assigned to, then assign it to any random of them and to its random servers which satisfy resource constraints. Else, assign Classroom_k to any suitable random time slot.

Step 3.3.6. Update Schedule_c to apply the assignment of Classroom_k to the selected time slot and to remove it from L_{i+1}.

Step 3.7. If $F(\text{Schedule}_c) \geq F(\text{Schedule}_i)$ then set $\text{Schedule}_{i+1} := \text{Schedule}_c$, else:

Step 3.7.1. Select a random value $r'' \in [0,1]$.

Step 3.7.2. If $r'' \leq e^{-\frac{F(\text{Schedule}_i)-F(\text{Schedule}_c)}{t_i}}$ then set $\text{Schedule}_{i+1} := \text{Schedule}_c$.

Step 3.8. Decrease the temperature by the formula $t_{i+1} := \frac{t_1 \cdot \tau}{i}$.

Step 3.9. Increment number of iteration $i := i+1$ and go to the step 3.

Step 4. Return the resulting Schedule_i and set L_i.

The first step of the algorithm includes the creation of an initial schedule satisfying resource and licensed software constraints. It is created randomly by selecting coordinators' requests with the probabilities equal to the weight w_i. Then in the cycle, under condition of decreasing the temperature, the schedule is changed randomly by moving one virtual machine between servers in the same time slot or by moving one virtual classroom between different time slots. After that, random virtual classroom with classes have not yet been assigned to, probably can be assigned to free resources in the new schedule. A new schedule is accepted as the schedule for the next iteration, if it improves optimizing function F, otherwise, it can be accepted with the probability $e^{-\frac{F(\text{Schedule}_i)-F(\text{Schedule}_c)}{t_i}}$. Algorithm ends when the minimum temperature is reached, or when the maximum number of iterations is exceeded.

This algorithm has the following parameters having effect on its work:

- t_{max}, t_{min} are the maximum and the minimum temperature. They have influence on the number of iterations and on the decisions at step 3.7.2.
- I_{max} is the maximum number of iterations.
- q is a probability of new schedule to be constructed by moving virtual machine between servers in the same time slot. Accordingly, $1-q$ is a probability of new schedule to be constructed by moving virtual classroom between time slots.
- q' is a probability of attempt to assign virtual classroom with classes have not yet been assigned to.
- $\omega_M, \omega_D, \omega_C$ are weighting coefficients describing fitness degree of RAM, HDD and number of computational cores.

We plan to study the proposed algorithm by the ERD simulator for different values of these parameters and coefficients α, β and γ in the optimized function F.

IV. Problem of routing and providing QoS

Papers [18], [19], [20] solve the problem of routing network traffic using SDN. The proposed algorithms do not consider the need to provide QoS parameters for the current or previously installed data flow routes. The existing algorithms [21], [22] for providing QoS in SDN are not efficient enough. The approach for dynamic routing of multimedia data flows is described in [21]. It provides the maximum guaranteed delay by LARAC (Lagrangian Relaxation Based Aggregated) algorithm. However, the authors consider the only case of unit delays for each network link and do not take into account the minimum guaranteed bandwidth. Similar approach is described in [22], the authors formalize and solve the optimization problem for lossless multimedia traffic transmission using alternate routes and leaving short routes for the general data flows. However, they optimize the delays and do not consider the need to ensure guaranteed bandwidth.

In this paper, we propose the approach based on the combination of routing and providing two QoS parameters for data flows – minimum guaranteed bandwidth and maximum guaranteed delay.

Let $G_T = (V, E)$ is an oriented multigraph describing current network topology of the ERD at time t. The set of its vertices $V = \text{Nodes} \cup \text{NetDevices}$ is the union of the ERD nodes (servers) set and other network devices (switches, gateways, data storages, and so on).

Each directed edge $e \in E$ corresponds to a specific network link between the vertices $beg(e) \in V$ and $end(e) \in V$. e has opposite directed edge due to duplex connection. In addition, several parallel edges can connect two vertices, for example, parallel network links between switches. They provide many alternate routes for the data transmission and providing QoS parameters.

Two following function are defined on the edge set E:

- $b : E \to R^{+} \cup \{0\}$ – current bandwidth of the link at time t.
- $d : E \to R^{+} \cup \{0\}$ – current delay on the link's output port at time t.

Let us denote by $G_P = (V', C)$ an oriented graph representing the communication pattern of some virtual classroom. This pattern is reported by virtual classroom scheduling algorithm to algorithm for routing and providing QoS. It is possible due to their close integration. $V' \subseteq Nodes$ is a set of servers used for virtual classroom assignment (its VMs are assigned to them), C is a set of directed edges, which correspond to existing data flows between servers.

It should be noted, that, if necessary, the set V' may also include other network devices, for example, gateways in the case when the most of the traffic comes from remote users to the servers.

Three following functions are defined on the edge set C:

- $\underline{b} : C \to R^{+} \cup \{0\}$ – minimum guaranteed bandwidth of data flow.
- $\bar{d} : C \to R^{+} \cup \{0\}$ – maximum guaranteed delay of data flow.
- $\hat{d} : C \to R^{+} \cup \{0\}$ – estimation of average delay that arises as a result of data flow's packets processing on the ports of network devices.

The algorithm for routing and providing QoS must construct a function $\varphi : C \to P(G_t)$, which relates each data flow $c \in C$ to its route r leading from the vertex $beg(c)$ to $end(c)$. Here $P(G_t)$ denotes a set of routes between any two vertices in topology graph G_T.

The function φ can be represented by a vector $R = (r_1, ..., r_{|C|})$, where $r_i = \varphi(c_i)$ is the route for data flow $c_i \in C$.

Let $\psi : E \to 2^{C}$ is the function relating each network link $e \in E$ to the set of data flows $c \in C$, for which corresponding routes are pass through e:

$$\psi(e) = \{c \in C \mid r = \varphi(c) \,\&\, e \in r\}. \qquad (13)$$

The vector R must contain the routes satisfying the following constraints for providing QoS:

1) The bandwidth of each route r_i with the influence of other data flows should not be less than the guaranteed bandwidth for the flow c_i:

$$\forall r_i \ \min_{e \in r_i}\{b(e) - \sum_{c \in \psi(e) \backslash \{c_i\}} \underline{b}(c)\} \geq \underline{b}(c_i). \qquad (14)$$

2) The summary delay of each route r_i with the influence of other data flows should not be greater than the guaranteed delay for the flow c_i:

$$\forall r_i \sum_{e \in r_i} (d(e) + \sum_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c)) \leq \bar{d}(c_i). \qquad (16)$$

It should be noted, that these constraints are flexible. They can be violated for some data flows, for example, when the ERD is overloaded. Hence, the optimized function has the following equation:

$$H(R) =$$

$$\alpha_b \cdot \sum_{\substack{c_i \in C: \\ \min\limits_{e \in r_i}\{b(e) - \sum\limits_{c \in \psi(e) \backslash \{c_i\}} \underline{b}(c)\} \geq \underline{b}(c_i) \& \\ \sum\limits_{e \in r_i}(d(e) + \sum\limits_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c)) \leq \bar{d}(c_i)}} \left[\min_{e \in r_i}\{b(e) - \sum_{c \in \psi(e)/\{c_i\}} \underline{b}(c)\} - \underline{b}(c_i) \right] +$$

$$+ \alpha_d \cdot \sum_{\substack{c_i \in C: \\ \min\limits_{e \in r_i}\{b(e) - \sum\limits_{c \in \psi(e) \backslash \{c_i\}} \underline{b}(c)\} \geq \underline{b}(c_i) \& \\ \sum\limits_{e \in r_i}(d(e) + \sum\limits_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c)) \leq \bar{d}(c_i)}} \left[\bar{d}(c_i) - \sum_{e \in r_i} d(e) - \sum_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c) \right] +$$

$$+ \beta_b \cdot \sum_{\substack{c_i \in C: \\ \min\limits_{e \in r_i}\{b(e) - \sum\limits_{c \in \psi(e)/\{c_i\}} \underline{b}(c)\} < \underline{b}(c_i)}} \left[\min_{e \in r_i}\{b(e) - \sum_{c \in \psi(e) \backslash \{c_i\}} \underline{b}(c)\} - \underline{b}(c_i) \right] +$$

$$+ \beta_d \cdot \sum_{\substack{c_i \in C: \\ \sum\limits_{e \in r_i}(d(e) + \sum\limits_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c)) > \bar{d}(c_i)}} \left| \bar{d}(c_i) - \sum_{e \in r_i} d(e) - \sum_{c \in \psi(e) \backslash \{c_i\}} \hat{d}(c) \right| \to \max.$$

(17)

Here $\alpha_b \geq 0$ and $\alpha_d \geq 0$ are the encouragements for upholding of corresponding constraints on bandwidth and delays, $\beta_b \geq 0$ and $\beta_d \geq 0$ are the penalties for their noncompliance.

In addition, there are strict constraints for the routes $r_i = (e_{i1}, ..., e_{in_i})$:

1) r_i is indeed a route:

$$\forall r_i \ \forall j = \overline{1, n_i - 1} \ end(e_{ij}) = beg(e_{ij+1}). \qquad (18)$$

2) r_i should begin at the first vertex of the edge c_i and end at the second vertex of c_i:

$$\forall r_i \ beg(e_{i1}) = beg(c_i) \,\&\, end(e_{in_i}) = end(c_i). \qquad (19)$$

3) r_i shoud not pass several times through the same vertex:

$$\forall r_i \ \ \forall j,k=\overline{1,n_i} \ \ j \neq k \Rightarrow \mathrm{beg}(e_{ij}) \neq \mathrm{beg}(e_{ik}) . \quad (20)$$

V. Proposed genetic algorithm for routing and providing QoS

The described optimization problem can be solved by a genetic algorithm (see detailed information on this heuristic in [23]). The problem solution is encoded by the chromosome representing a route vector $R=(r_1,...,r_{|C|})$. Population has a fixed size N.

The crossover operation is a single point crossover for two parent chromosomes $R^A=(r_1^A,..,r_{k-1}^A,r_k^A,...,r_{|C|}^A)$ and $R^B=(r_1^B,..,r_{k-1}^B,r_k^B,...,r_{|C|}^B)$. It selects a random number $k \in \overline{2,|C|-1}$ and creates two daughter chromosomes by combining parent genes separated at selected point:

$$R_1^{AB}=(r_1^A,..,r_{k-1}^A,r_k^B,...,r_{|C|}^B),$$

$$R_2^{AB}=(r_1^B,..,r_{k-1}^B,r_k^A,...,r_{|C|}^A). \quad (21)$$

The mutation operation for the chromosome $R=(r_1,...,r_{|C|})$ represents the selection of a random number $k \in \overline{1,|C|}$ and a random transformation of the route r_k. For the transformation it selects two vertices $\mathrm{beg}(e_{kj})$ and $\mathrm{beg}(e_{ks})$ ($1 \le j < s \le n_k$) (if there is alternate route between them) and replaces the subsequence $e_{kj},...,e_{ks-1}$ in r_k with the alternate subsequence $e'_{kj},...,e'_{kp}$.

The selection operation for each generation combines the choice of P_{top} percent the best chromosomes (elite selection) with the roulette selection of remaining chromosomes. In the last case, chromosomes are selected proportionally to a fitness value of the optimized function.

Stopping criteria for the genetic algorithm are the exceeding of the maximum time and the lack of significant improvements in the average fitness value for several generations. The last criteria can be written by the following inequality:

$$\max_{j=i-G+1,i} \left| \overline{H}_{Gi} - \overline{H}(R_j) \right| < \varepsilon, \quad (22)$$

where G is a number of controlled generations, $\overline{H}_{Gi}$ – the average fitness value for the last G generations, $\overline{H}(R_j)$ – the average fitness value for the j-th previous generation, ε – the preferred tolerance degree.

The proposed genetic algorithm has the following steps (see algorithm 2):

Algorithm 2 – Genetic algorithm for routing and providing QoS in ERD

Step 1. Save the current time to variable T_{start}.

Step 2. Create an initial $\mathrm{Population}_1$ of a fixed size N including the chromosome R which has routes calculated by Dijkstra's algorithm launched from each vertex from V'. It should minimize summary delays of the routes. Generate randomly other chromosomes of $\mathrm{Population}_1$.

Step 3. Let $i:=1$ be the number of iteration.

Step 4. While $T_{current} - T_{start} < T_{max}$ and ($i < G$ or $\max_{j=i-G+1,i} \left| \overline{H}_{Gi} - \overline{H}(R_j) \right| \ge \varepsilon$) do the following steps:

Step 4.1. Combine the parent chromosomes from $\mathrm{Population}_i$ in random pairs and perform the crossover operation for them with probability P. Save the resulting child chromosomes to $\mathrm{Population}'_i$.

Step 4.2. Perform the mutation operation for the chromosomes from $\mathrm{Population}'_i$ with probability Q.

Step 4.3. Join parent and child populations:

$$\mathrm{Population}''_i := \mathrm{Population}_i \cup \mathrm{Population}'_i . \quad (23)$$

Step 4.4. Perform the selection operation for $\mathrm{Population}''_i$, save the selected chromosomes to $\mathrm{Population}_{i+1}$.

Step 4.5. Increment iteration number $i:=i+1$ and go to step 4.

Step 5. Install all the routes from $\mathrm{Population}_i$ as a rules to the tables of OpenFlow switches.

For this algorithm $T_{current}$ means the current time of system clock.

This algorithm is only run for proactive calculation of data flow routes implementing communication patterns of virtual classrooms. Patterns are reported by virtual classroom scheduler of the ERD. Other traffic (which is not critical to delays and bandwidth restrictions) is routed by standard shortest path algorithms.

In addition, it should be noted, that the proposed algorithm also works in particular situation, when G_P contains two vertices. It corresponds to a single route calculation.

It is planned to study the proposed algorithm by the ERD simulator for different values of parameters α_b, α_d, β_b, β_d, T_{max}, P_{top}, G, ε, P and Q.

VI. Conclusions

The virtual classroom scheduling problem was formalized as an optimization problem for the ERD. It considers resource and software constraints, schedule cyclicity and time division into classes (time slots). For the solution of this problem we have proposed the scheduling algorithm based on a simulated annealing heuristic.

In addition, the optimization problem of proactive routing and providing QoS in the ERD was formalized. It takes into account flexible constraints on providing QoS parameters for data flows as the minimum guaranteed bandwidth and the maximum guaranteed delay. We have proposed genetic algorithm for solving this problem. It is based on SDN technology, which is used for gathering information on current network state and installing rules in the tables of OpenFlow switches to implement calculated routes.

We have implemented these algorithms using C++ and partially tested them on the ERD simulator, which is still under development. Different configurations of ERD were chosen for preliminary studying. They differ by the hardware configuration and the size of the ERD. Coordinators' requests for virtual classrooms with different configurations of VMs were randomly generated for the virtual classroom scheduling algorithm. Randomly generated communication patterns (All-to-All, One-to-All, Grid and etc.) were used for routing and providing the QoS algorithm. The preliminary experimental studies have demonstrated the efficiency (reduction of free windows in the virtual classrooms schedule, decreasing of QoS parameters violations for data flows) of proposed algorithms for different test scenarios. In the future, we plan to conduct a series of experiments on the ERD simulator and to describe in detail the obtained results.

The main feature of the proposed solutions is an interrelation between the virtual classroom scheduling algorithm and the algorithm for proactive routing and providing QoS parameters. The first algorithm reports the information about an assignment of virtual classroom's machines to the physical servers and its communication pattern to the second algorithm.

REFERENCES

[1] A.L. Konnov, L.V. Legashev, P.N. Polezhaev and A.E. Shukhman, "Concept of Cloud Educational Resource Datacenters for Remote Access to Software", Proceedings of 11th International Conference on Remote Engineering and Virtual Instrumentation (REV), Polytechnic of Porto (ISEP) in Porto, 2014, pp. 246-247.

[2] J. Tordsson, R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers", Journal Future Generation Computer Systems, volume 28, issue 2, February, 2012, pp. 358-367.

[3] R. Fourer, D.M. Gay, B.W. Kernighan, "A modeling language for mathematical programming", Management Science, volume 36, issue 5, May 1990, pp. 519-554.

[4] "CPLEX Optimizer. High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming", IBM Corporation, 2010, URL: http:// www.ilog.com/products/cplex/.

[5] T. Cordeiro, D. Damalio, N. Pereira, P. Endo, A. Palhares, G. Gonçalves, D. Sadok, J. Kelner, B. Melander, V. Souza, J.-E. Mångs, "Open Source Cloud Computing Platforms", Proceedings of 9th International Conference on Grid and Cooperative Computing (GCC), 2010, pp. 366-371.

[6] V.P. Solovev, A.O. Uvdovichenko, "Virtual machine placement method with resource redistribution", Software Programs and Systems, volume 1, 2012, pp. 134-138.

[7] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines", NSDI'05 Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation, volume 2, pp. 273-286.

[8] D.I. Kogan, "Problems and methods of finite-dimensional optimization. Part 3. Dynamic programming and multicriteria optimization", Nizhniy Novgorod: Izdatelstvo NGU, 2004.

[9] "Components of OpenStack", OpenStack.Ru, 2014, URL: http://openstack.ru/about/components/.

[10] "OpenNebula | Flexible Enterprise Cloud Made Simple", OpenNebula Project, 2014, URL: http://opennebula.org/.

[11] D. Taft, "Eucalyptus 3.2 adds new functions for data control and storafe", PCWEEK, 2012, URL: http://www.pcweek.ru/its/article/detail.php?ID=144980.

[12] "Amazon EC2. Amazon Web Services", Amazon Web Services, Inc., 2014, URL: http://aws.amazon.com/ec2/.

[13] "VMware vCloud Suite Datasheet. Standatd, Advanced and Enterprise Editions", VMware, Inc., 2014, URL: http://www.vmware.com/files/ru/pdf/products/vCloud/VMware-vCloud-Suite-Datasheet.pdf.

[14] "Oracle Enterprise Manager 12c", Oracle, 2014, URL: http://www.oracle.com/technetwork/oem/enterprise-manager/overview/index.html?ssSourceSiteId=ocomit.

[15] "Moab Cloud Suite", Adaptive Computing, Inc., 2014, URL: http://www.adaptivecomputing.com/products/cloud-products/moab-cloud-suite/.

[16] "CloudPlatform - Cloud Orchestration to support Infrastructure-as-a-Service - Citrix", Citrix Systems, Inc., URL: http://www.citrix.com/products/cloudplatform/overview.html.

[17] P.J.M. van Laarhoven, E.H.L. Aarts, "Simulated annealing: theory and applications", D. Reidel Publishing Company, Dordrecht, 1987.

[18] G. Ibáñez, J. Naous, E. Rojas, D. Rivera, B.D. Schuymer, T. Dietz, "Small Data Center Network of ARP-Path Bridges made of Openflow Switches", The 36th IEEE Conference on Local Computer Networks (LCN),2011.

[19] H. Shimonishi, H. Ochiai, E. Enomoto, A. Iwata, "Building Hierarchical Switch Network Using OpenFlow", Proceedings of Intelligent Networking and Collaborative Systems, 2009, pp. 391–394.

[20] A. Tavakoli, M. Casado, T. Koponen, S. Shenker, "Applying NOX to the datacenter", Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets-VIII), New York, 2009.

[21] H. E. Egilmez, S. T. Dane, K. T. Bagci, A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-dened networks", 2012.

[22] W. Kim, P. Sharma, J. Lee, S. Banerjee, "Automated and Scalable QoS Control for Network Convergence", In Proc. INM/WREN , 2010.

[23] L. Davis, Ed., "Handbook of Genetic Algorithms", New York: Van Nostrand Reinhold, 1991.

OpenFlow SDN testbed for Storage Area Network

O. Sadov, V. Grudinin, A. Shevel, D. Vlasov, S. Khoruzhnikov, V. Titov, A. Shkrebets, A. Kairkanov
ITMO University,
Russia
http://sdn.ifmo.ru

Abstract—The paper describes the testbed to determine the effectiveness of an approach to build network storage using Software-Defined networks (SDN) OpenFlow. It is assumed that main protocol to SAN is iSCSI over local area network. Prototyping tools for managing network resources and data flows on the basis of SDN and testing environments based on Free and Open Source software. We describe experiments with various modifications of OpenFlow controller NOX and set out the specifics for the use of various software and hardware OpenFlow switches. The main tests goals are Data Center SAN specific: implementation of QoS methods accordingly switchspecifics, topology changing, measuring of transmission parameters, simulating of large amount of requesting hosts (up to 100 thousands hosts).

Keywords—OpenFlow; SDN; SAN; network; NOX; QoS

I. INTRODUCTION

The aim of this work was to study the design principles and performance of Software-Defined Networks, as well as to develop prototypes of tools for managing network resources and data flows in SDN, the evaluation of the applicability of the SDN for data centers and distributed storage. For experiments were selected OpenFlow SDN and evaluated the effectiveness of their use for the management of iSCSI storage systems.

The requirements were specified for network resources management tools and Quality of Service (QoS) assurance.

II. TESTBED

A. *Software:*

- OpenFlow software switch based on CPqD/of12softswitch [1] and Open vSwitch [2];
- OpenFlow controllers based on CPqD/nox12oflib [3]and NOX [4];
- OpenFlow network emulator Mininet [5];
- VirtualBOX and KVM Virtual Machines with NauLinux 6.3/6.4 [6] distributions and Ubuntu 11.10 pre-configured CPqD OpenFlow-1.2 Virtual Machine [7].

B. *Hardware*

- OpenFlow switches – Pica8 3290 and HP 3500-24G-PoE yl.
- HP P4300 G2 7.2TB SAS Starter SAN BK716A was used as the iSCSI SAN.

III. SPECIALIZED SOFTWARE MODULES

For testing purposes was created a number of specialized Python modules and programs which used for changing of topology, QoS policies, starting/stopping of traffic generators and measuring of transmission characteristics. Developed prototypes were tailored for the hardware OpenFlow switches.

Specialized “switchqos” module was developed based on NOX module “switch” to manage network resources and data flows and to ensure QoS.

This module calculates routes for all packets in the testbed and generates flow tables for every OpenFlow switch. These calculations and flow tables modifications are performed after every topology change or data flow interruption.

The traffic classification for QoS control is based on TCP/UDP port numbers. Depending on switch type and capabilities, the different QoS control methods were used: OpenFlow queues, IP ToS, and VLAN PCP modifications.

Special software tools for QoS policy configuration of hardware switches were used to prioritize SAN traffic. The different switches (for example, Open vSwitch and HP ProCurve) had different QoS control mechanisms, which made the creation of a unified interface is quite a difficult task.

As the most important configurable parameters of QoS assurance, the bandwidth and the priority of the packet queues were selected.

The software prototypes for QoS control on HP 3500 and Pica8 in OVS mode were placed in the repository [8]. They can be easily extended to use different QoS settings.

Because different switches and controllers support variety versions of OpenFlow, several different modules were developed for NOX classic [8], NOX [4] and nox12oflib [3].

IV. NETWORK RESOURCES AND DATA FLOWS MANAGEMENT

As a system for network resources and data flows management, a set of software modules was developed for attaching and detaching links between switches.

In the emulation mode, this was carried out by means of Mininets Python module.

For hardware switches this was done via CLI commands over SSH connection, automated by a Python script.

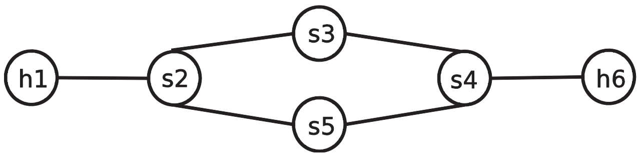


Fig. 1. Loop topology for experiments

A loop topology (Fig. 1) was selected for experiments, consisting of 4 switches (nodes s2, s3, s4 and s5), and two hosts for traffic generation and reception (nodes h1 and h6).

SDN routing modules based on standard regular MAC-learning NOX "switch" modules.

During the experiment, test traffic (ping) was sent from the host h1 to host h6. In an initial state all nodes were connected accordingly Fig. 1. The controller was in an undefined state, it had no routing scheme, and the packets have not passed. After detaching one link by test framework, the route was constructed by NOX "switchqos" module, and the pass of the packets was established. After that, the restoring of the detached link (and loop) did not break the traffic flow. Detaching the active link led to an automatic topology rediscovery and redirection of the traffic to a different route.

V. QOS ASSURANCE METHODS

Data flows prioritization was carried out with the Python modules. These modules set bandwidth for OpenFlow queues or ToS/PCP bandwidth. The dpctl utility was used for the software switch control. Hardware switches were managed by CLI commands sent over SSH.

For the evaluation of a possible use of SDN in data center, a data center model (Fig. 2) was created. This model consisted of iSCSI SAN and few VMs. The first VM acted as an OpenFlow 1.2 switch while the second one generated iSCSI traffic; the others performed in generating and receiving the load traffic.

During the experiment, the data were read from iSCSI SAN with simultaneous load traffic generation.

IP diagnostic utility Iperf and VoIP test tool SIPp were used to generate the load traffic.

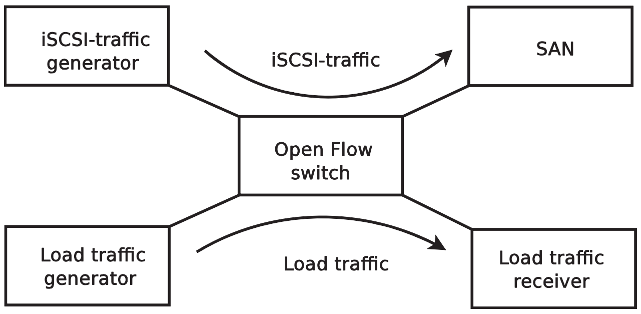


Fig. 2. SDN data center model

It was observed that under heavy load condition the iSCSI connectivity might be lost and later recovered. After iSCSI connectivity recovery the bandwidth is changing in arbitrary manner. To keep the same bandwidth after recovery we changed Linux Traffic Control dynamic bandwidth, which is defined by CpqD/of12softswitch, to static bandwidth setting. The modified module can be found in [1].

The utility dpctl sets the share of total bandwidth for selected QoS queues as percent of total bandwidth. The sum of shares is not necessary equal to 100.

Table I shows the influence of the presence of queuing on the resulting SAN I/O speed, but there is a little difference.

The experiment with the HP hardware switch has shown a correlation between the bandwidth share set and the resulting I/O speed (Fig. 2).

TABLE I. SAN I/O SPEED THROUGH SOFTWARE SWITCH DEPENDENCY ON QOS QUEUES BANDWIDTH SHARE

Bandwidth share, in % of the total		**Load traffic, Kb/s**
SAN I/O speed	***iSCSI traffic***	
100	0	35.1
100	0.1	31.6
100	100	8.3
0.1	100	5.4
0.1	0.1	9.2

TABLE II. SAN I/O SPEED THROUGH HARDWARE SWITCH DEPENDENCY ON QOS QUEUES BANDWIDTH SHARE

Bandwidth share, in % of total		**Load traffic, Mb/s**
SAN I/O speed	***iSCSI traffic***	
100	0	10.0
80	20	8.4
20	80	2.1
0	100	0

Not comparing the absolute transmission rate, it is possible, due to a priori restricted channel throughput, to specify the advantages of QoS control in hardware switches: a high degree of accuracy, an impossibility of setting a total bandwidth more than 100%.

VI. PROCESSING A LARGE NUMBER OF REQUESTS

In the test program (rd test) SCSI command "TEST UNIT READY" was sent to SAN in multithread mode via ioctl system call with SG IO code. The target characteristic was the number of completed requests for a selected period of time.

The developed "switchqos" module was optimized for speed of transmission of data passing through controlled switches. This optimization included a modification of the default NOX flow matching scheme. It was necessary because the used switches were unable to perform a flow match based on source and destination MAC addresses and VLAN PCP with the hardware acceleration. The software processing was limited to 10 000 packets per second.

Another setting was in increasing the idle timeout. It was found during the experiments that HP 3500 switch had not refreshed the flow packet statistics frequently enough for the hardware processed flows. Usually, after 5 seconds of idle time (default for NOX "switch" module), the switch erroneously removed the record from the flow table. After increasing the idle time parameter in "switchqos" module to 20 seconds, this behavior was corrected and the necessity for repeatedly creating records of flow matching was eliminated. At the same time, an excessively large idle timeout value could degrade the performance due to an increased flow table size.

After these optimizations, the performance of the system increased significantly, and the value of 100 000 requests to SAN per second through OpenFlow switch was surpassed. The example of test program output is shown below.

```
# ./rd_test /dev/sdb 2 100
```

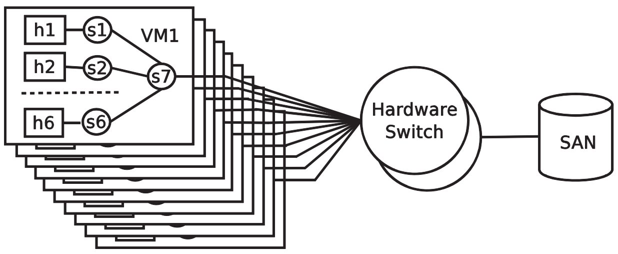


Fig. 3. Modeling the large number of requests to SAN in data center infrastructure

```
Result: 130124 requests/sec (260248/2)
```

VII. SAN RESPONSE TIME

Read operations were used to measure SAN response. SG IO ioctl was used to exclude the buffering influence, instead of the generic read.

The test program has measured the average latency and jitter performing SAN requests.

The results for 1000 packets and data block sizes 512 and 1024 bytes are as follows (the average latency and jitter are measured in seconds):

```
# ./rtt_iscsi_read /dev/raw/raw1 1000 \ 512 1024

Size=512 Packets=1000 Latency=0.000844 Jitter=0.000084

Size=1024 Packets=1000 Latency=0.000860 Jitter=0.000104
```

VIII. DATA CENTER MODELING

Our modeling of a data center involved a transmission of ICMP requests to SAN from different MAC addresses.

The test network consisted of SAN, 2 hardware OpenFlow switches from HP, VM with NauLinux 6.3 guest OS running NOX and 10 test nodes VMs running Ubuntu 11.10 and Mininet. Each test node launched 6 virtual hosts, 7 software switches Open vSwitch, and a local controller NOX (Fig. 3).

The test program, running on the main host, sent messages to the test nodes, starting local test programs, written as xinetd services. The local test programs on every virtual host pinged SAN from every MAC address in a specified range. Requests were forwarded to SAN through hardware switches, controllable by NOX launched in multithread mode (10 threads) on the main host. This controller instance has logged the number of different MAC addresses in the processed requests and the requests distribution in the running threads. After getting 100 000 different MAC addresses, test programs stopped.

IX. CONCLUSION

Described experiments have shown that developed OpenFlow testbed could be used for testing the dynamic (re)configurations of the network elements, (re)setting various data transfer parameters for different traffic types. It was shown the testbed is able to serve the requests from large number of hosts. Suggested inexpensive testbed might be used for detailed investigation of OpenFlow approach to the network architecture of data centers and distributed storage.

Software repositories [1], [3] and [4] contain developed software modules and tests. The controller applications are packaged in binary and source forms for NauLinux operating system distribution [6], binary compatible with RHEL/Oracle Linux/CentOS/Scientific Linux distributions.

REFERENCES

[1] ITMO OpenFlow 1.2 software switch repository, avail-able at https://github.com/itmo-infocom/of12softswitch

[2] Open vSwitch project, available at http://openvswitch.org

[3] ITMO OpenFlow 1.2 NOX repository, available at https://github.com/itmo-infocom/nox12oflib

[4] ITMO NOX repository, available at https://github.com/ itmo-infocom/nox

[5] Mininet, available at http://mininet.github.com

[6] NauLinux distribution, available at http://downloads. naulinux.ru/pub/NauLinux/

[7] CPqD OpenFlow-1.2-Tutorial, available at https:// github.com/CPqD/OpenFlow-1.2-Tutorial/wiki

[8] ITMO OpenFlow tests repository, available at https:// github.com/itmo-infocom/of-tests

In-kernel offloading of an SDN/OpenFlow Controller

A. Shalimov
Applied Research Center for
Computer Networks
Lomonosov Moscow State University
ashalimov@arccn.ru

P. Ivashchenko
Applied Research Center for
Computer Networks
pivashchenko@arccn.ru

***Abstract*—This paper presents the novel approach on offloading the most time consuming and frequently used functionality of the SDN/OpenFlow controller to the Linux kernel space. This speeds up network applications in 2.5 times together with possibility of using the all userspace libraries and programming tools.**

I. Introduction

SDN/Openflow is already a mainstream in the area of computer networks [1]. It allows us to automate and to simplify network management and administration: fine-grained flows control, observing the entire network, unified open API to write your own network management applications. All control decisions are done first in a centralized controller and then moves down to overseen network's switches. In other words, the controller is a heart of SDN/OpenFlow network and its characteristics determine the performance of the whole network. The controller throughput means how big and active our network can be in terms of switches, hosts, and flows. The response latency directly affects network's congestion time and end-user QoE. Moreover, as faster controller we have as more reactive network we can introduce: faster reaction on host migration and topology changes, more granular flow control, advanced network application like load balancing techniques, security features, and so on.

The recent SDN/OpenFlow controllers performance evaluations show that the throughput of the controllers are not enough for modern datacenters' networks and large scale networks [5], [6]. There are two complimentary ways to cover this performance gap. The first way is to use multiple instances of a controller collaboratively managing the network and forming a distributed control plane. But this way brings a lot of complexity and overheads on maintaining a consistent network view between all instances. The second way is to improve single controller itself by leveraging ability of contemporary multicore systems and by reducing existing bottlenecks and overheads in data communication path in operating systems. Note these two ways can and should be used together to create high efficient distributed control plane.

In this paper, we presents an extended approach on offloading of frequently used SDN/OpenFlow controller functions down to Linux Kernel to create high performance network applications. The paper is structured as follows. Section 2 describes related works and motivation. Section 3 contains the main idea of the proposed approach on in-kernel offloading of an SDN/OpenFlow controller. Section 4 explains implementations details of our in-kernel offload engine. Section 5 shows the result of performance evaluation of the proposed approach.

II. Background

At present, there are a more than 30 different SDN/OpenFlow controllers created by different vendors/universities/research groups, written in different languages (Python, Java, C/C++, Haskell, Erlang, Ruby), using different runtime multithreading techniques, showing different performance numbers [4]. These controllers are implemented as ordinary applications running in Linux userspace.

From the system point of view, implementation in Linux userspace have several performance drawbacks. Every system call (malloc, free, read and write packet(s) from the socket, etc) leads to context switching between userspace and kernel space that requires additional time. Approximately this time for FreeBSD Linux is 0.1ms and takes 10% time for whole system call [3]. Under the high load this leads to significantly time overhead. Moreover, the userspace programs work in virtual memory that also require additional memory translation and isolation mechanism: hierarchical vs linear address translation.

In our previous work [7], to avoid above mentioned overheads we have implemented the OpenFlow controller as a module inside the Linux kernel space. Our experiment evaluation shows that it has 5 times higher performance than all existing controllers. But, as we understood later in practice, it's very hard to write our own application for Linux kernel space. There are several programming challenges: low-level programming language (object C), limited number of libraries and tools, high risk to corrupt the whole system. Thus, we need to find out a way to simplify network applications programming for the in-kernel controller.

III. Proposed approach

As already mentioned, the Linux kernel allows us to significantly speed up the SDN/OpenFlow controllers and provides abilities to create high performance network management applications. The idea is to use kernel space to accelerate the most time consuming functionality of the controller. We call our approach as in-kernel offloading.

There are several important tasks: determine what functionality should be offloaded, what northbound programming API we should provide for a user, and how to implement this command and data passing interfaces between kernel and user space.

Usually a controller consists of three main layers:

- OpenFlow network layer is responsible for communication with OpenFlow switching devices. It implements TCP server listening new switches connections and OpenFlow library for parsing incoming OpenFlow messages from TCP streams.

- Service layer contains the most frequently used network functions like link discovery, topology, and routing.
- Application layer represents user-written network applications that might use services and subscribe on events from the network layer (for instance, L2 learning switch, firewall, DDOS).

Figure 1 shows the basic scheme of the proposed idea. Figure 1(a) and Figure 1(d) represent two opposite situations where the all layers of the controller reside fully either in the userspace or in the kernel space, correspondingly. In the userpace, the controller has wide range of applications and libraries but low performance. In kernel space, the controller has fastest performance but limited number of applications.

During offloading procedure the controller is gradually been dipping down to the kernel space. The offload scheme supports two operational modes: pass-through mode and driven mode. Both modes describe which functions run inside the kernel. In the pass-through mode, the in-kernel part receives new OpenFlow messages, parses them, and puts into shared queues (figure 1(b)). In the driven mode, the in-kernel part also runs services inside the kernel (figure 1(c)). In this case, it notifies the userspace applications about changes (e.g., topology) and provides an high level interface to manage the network.

IV. Offload engine

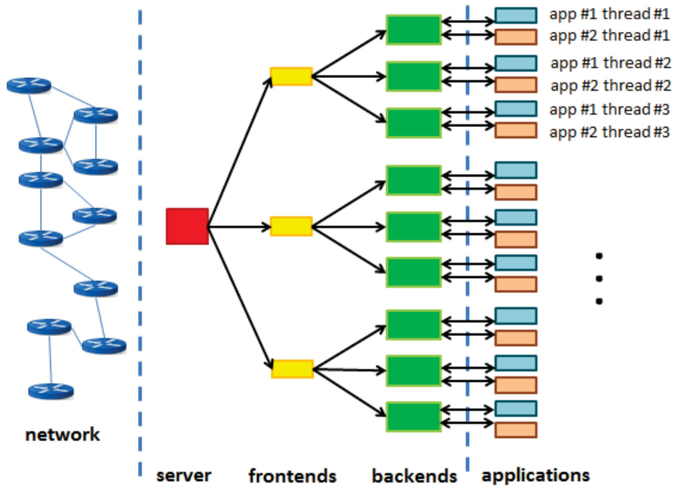


Fig. 2. The SDN/OpenFlow controller offloading architecture.

The figure 2 shows the offload engine architecture. Logically there are three main levels in the offload architecture: in-kernel controller that is responsible for communication with switches, shared data structures that are used to pass information to the userspace, and an userspace network application itself.

The in-kernel has three-tier architecture:

- **Server**. Server thread listens to a socket, accepts new connections from switches and distributes connections between frontend and backend threads.
- **Frontend**. Frontend threads initialize connections and check their correctness: openflow version, hello, features reply. The correctness of headers are checked for every messages in the input buffer until a features reply OpenFlow message will be sent. If all verification is done, connections move to backends.
- **Backend**. Backend threads work with switches and applications. They do the main job on sending and receiving OpenFlow messages. Inside the thread we use poll() to wait for changes in the sockets' descriptors.

Applications running in the userspace communicates with backends through shared data structures. Each backend thread has its own shared data structures. So, to get full speed the userspace application must be multithreaded with the number of threads equal to the number of backends threads in the kernel space, $N\_threads\_app = N\_threads\_kernel$. If $N\_threads\_app < N\_threads\_kernel$ an application will not able to show full power and to process all events coming from the network. If $N\_threads\_app < N\_threads\_kernel$ an application would need to have additional locking mechanism to access to shared backends data structures and thus don't get the full speed either.

Currently multiple applications have to subscribe to different type of OpenFlow messages because we don't store multiple copies of the messages.

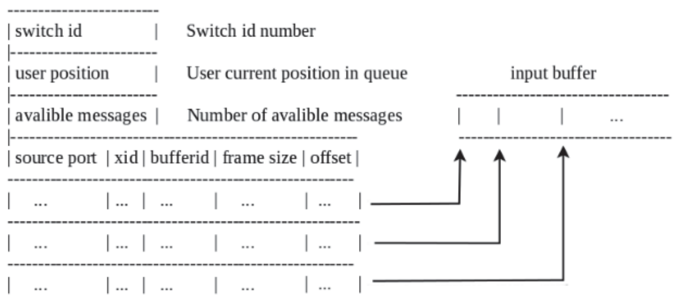

switch id
Switch id number
user position
User current position in queue
avalible messages
Number of avalible messages
source port | xid | bufferid | frame size | offset
input buffer

Fig. 3. Packet In queue organization scheme

There are two types of shared data structures in backend threads:

- **Buffer**. All incoming and outgoing raw OpenFlow messages are stored in input and output buffers correspondingly. All buffers are reachable from the userspace through memory mapped regions.
- **Queues**. The data queue is designed to hold pre-parsed OpenFlow messages (see figure 3). For instance, for PacketIn message it holds the following information: source port, xid, bufferid, dpid, ethernet frame (offset and size). An ethernet frame itself resides in input buffer. The control queue is used for communication between kernel and userspace part. In the driven mode, this queue is also used for passing information from the services.

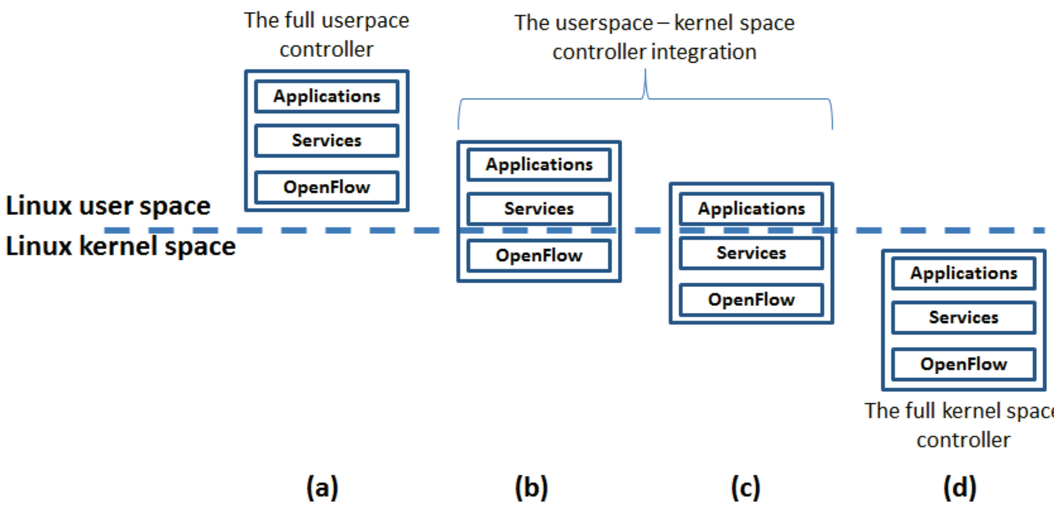

The full userpace controller
The userspace – kernel space controller integration
Linux user space
Linux kernel space
Applications
Services
OpenFlow
The full kernel space controller
(a)
(b)
(c)
(d)

Fig. 1. The basic offload procedure: (a) userspace mode, (b) pass-through mode, (c) driven mode, (d) kernel mode.

From the programming prospective an application's threads open */dev/ctrl* and issue an *ioctl()* to register in controller. Controller queues and OpenFlow packets are in an *mmap()* region with well defined ownership, so that lock free access is possible.

The *poll()* returns the following flags:

- **POLLIN** indicates new events in the data queue (e.g., new packet-in message)and the control queue.
- **POLLRDNORM** means there are events only in the data queue.
- **POLLRDBAND** means there are events only in the control queue.
- **POLLOUT** says all input events have been processed.

A kernel thread reads data from socket and fills buffers, while user application thread reads data from queues and fills output buffer. The kernel thread waits while user application processes all input messages. When the application is done, it calls *write()* function. After that the kernel thread wakes up and finally sends output messages to appropriate switches. Note to speed on application and to decrease network overheads the output message buffer are flushed either by timer in the kernel space or by the application itself. The last option is preferable for fast I/O throughput.

The example below shows the userspace L2 learning switch application that communicated with in-kernel controller through memory API.

```
fds.fd = open("/dev/ctrl", O_RDWR);
fds.events = POLLIN|POLLOUT;
// get memory mapped region size
mem_size = get_memory_size(fds.fd);
// mapping the memory
p = mmap(NULL, size, PROT_READ|PROT_WRITE,
    MAP_SHARED, fds.fd, 0);
// registering the application
app_thread_registration(p->thread_number);
// subscribing to packet-in messages
subscribe_packet_in();
// communicating with in-kernel controller
rx_q = &(p->rx_q);
while (1){
 // reading latest events from the kernel space
 ret = poll(&fds, 1, 2000);
 // nothing to do, wait
 if (ret == 0) continue;
 if (ret > 0){
  // new packet_in messages, process them as
      l2 learning
  if (fds.revents & POLLIN){
   for (; rx_q->avail > 0 ; rx_q->avail--){
    l2(rx_q->id, p->thread_number, rx_q->cur);
        rx_q->cur++;
   }
   continue;
  }
  // the output buffer is full, then send all
      data to switches
  if (fds.revents & POLLOUT){
   write(fds.fd, &p->thread_number,
       sizeof(int));
   continue;
  }
  // kernel space is off
  if (fds.revents & POLLERR)
       error("userpace-kernelspace
           communication failed")
}
```

The driven mode becomes possible when we have implemented pass-through mode and measured that while the userspace thread is 100% loaded, the dedicated kernel thread is only 25% loaded. This observation shows the kernel threads might perform some additional useful functions. This list includes topology discovery, endpoint tracking, dynamic routing, working with some dataplane control protocols like ARP. We add additional type of data and control messages in order tu push changes and information to applications' threads. Applications can send requests through control queues or read push in changes from data queues *[service, type, data]*.

V. Experimental Evaluation

Our experimental evaluation consists of two parts. The first part is performance evaluation of pass-through mode of the OpenFlow controller where the goal is to measure I/O overheads on offload engine and kernel/userspace communications based on L2 learning application. The second part is for driven mode based on L3 forwarding application in order to use a topology service.

A. *Pass-through mode evaluation*

For performance evaluation we use the methodology described in [4]. There we used only one 10Gb channel and on cbench because no one controller was able to process all messages from the channel. In our case, we need two 10Gb channels and two cbench'es. Finally, the test-bed consists of two servers connected with two 10Gb links and two cbench'es generating the packetin messages over these two links.

Figure 4 and Table 1 shows the renewed throughput and latency numbers for the existing controller against the pure in-kernel controller and the in-kernel controller in pass-through mode. The throughput of the pure in-kernel controller is almost 30M flow per second that is 5 times faster than all others. The throughput of the pass-through in-kernel controller is lower with 15M flow per second but it's still 2.5 times faster than others. The latency of the pure in-kernel controller and the pass-through controller are 45us and 50us, respectively.

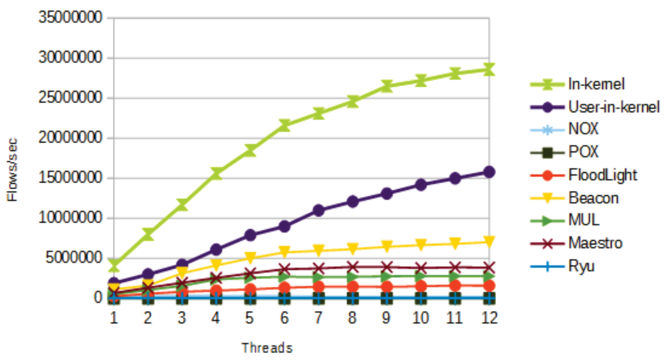


Fig. 4. The average throughput achieved with different number of threads (with 32 switches, 10^5 hosts per switch)(Intel(R) Xeon(R) CPU E5645 2.40GHz)

In-Kernel	**45**
Pass-through	**50**
NOX	91
POX	323
Floodlight	75
Beacon	57
MuL	50
Maestro	129
Ryu	105

TABLE I. The minimum response time (10^{-6} secs/ flow)

B. *Driven mode evaluation*

The controller run L3 forwarding application (calculating the path between two hosts using Dijkstra algorithm) in the userspace and topology discovery services in the kernel space.

We measured the time required for initial topology discovery in the driven mode and the userspace mode. We used mininet to create an OpenFlow network with a tree topology of depth 3 and fanout 3 (i.e 27 hosts, 13 switches, 39 links). It takes 24ms in the userspace mode and 5ms in the driven mode to find out the whole topology. The Beacon controller [8] requires almost 55ms to discover this topology.

We also tried to use the ten physical servers running two instances of Open vSwitch connected with different topologies. The times are slightly less but still different in 4 to 5 times.

Comparing path calculation procedure we measured the 3-4 times difference: 10ms in the userspace mode and 2ms in the driven mode.

VI. Conclusions

Such offloading mechanism accelerates the most time consuming and frequently used parts of the OpenFlow controller using the Linux kernelspace. This allow us to easily create high performance network application. The proposed architecture can be easily extended with other services like verification, link status monitoring, etc. Further work will include the development of new services and simplify API between the kernel space and the userspace.

Our in kernel offloading implementation shows high performance number comparing with existed controllers. The userpace application is still 2.5 times faster with 15M flows per second. Services might be speed on up to 5 times by moving them into the kernel side.

Our approach is the future of previous approaches to inkernel HTTP servers that were only able to return static data to user requests [9].

Acknowledgment

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003.

References

[1] M. Casado, T. Koponen, D. Moon, S. Shenker. *Rethinking Packet Forwarding Hardware*. In Proc. of HotNets, 2008

[2] T. Benson, A. Akella, D. Maltz, *Network traffic characteristics of data centers in the wild*, IMC, 2010

[3] *Netmap*, info.iet.unipi.it/∼luigi/netmap/

[4] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, *Advanced Study of SDN/OpenFlow controllers*, Proceedings of the CEE-SECR13: Central and Eastern European Software Engineering Conference in Russia, ACM SIGSOFT, October 23-25, 2013, Moscow, Russian Federation

[5] A. Shalimov, R. Smeliansky, *On Bringing Software Engineering to Computer Networks with Software Defined Networking*, Proceeding of the 7th Spring/Summer Young Researchers' Colloqium on Software Engineering (SYRCoSE 2013), May 30-31, 2013, Kazan, Russia

[6] Advait Dixit, *Towards an Elastic Distributed SDN Controller*, Proceeding of the ACM SIGCOMM HOTSDN 13, Hong Kong.

[7] P. Ivashchenko, A. Shalimov, R. Smeliansky, *High performance in-kernel SDN/OpenFlow controller*, Proceedings of the 2014 Open Networking Summit Research Track, USENIX, 2014, Santa Clara

[8] David Erickson, *The Beacon OpenFlow Controller*, Proceeding of the ACM SIGCOMM HOTSDN 13, Hong Kong.

[9] kHTTPd - Linux HTTP accelerator, http://www.fenrus.demon.nl/

Queuing Systems with Multiple Queues and Batch Arrivals for Cloud Computing System Performance Analysis

S. Shorgin, A. Pechinkin
Institute of Informatics Problems
Russian Academy of Sciences
Moscow, Russia
sshorgin@ipiran.ru, apechinkin@ipiran.ru

K. Samouylov, Y. Gaidamaka, E. Sopin, E. Mokrov
Telecommunication Systems Department
Peoples' Friendship University of Russia
Moscow, Russia
{ksam, ygaidamaka}@sci.pfu.edu.ru,
{sopin-eduard, melkor77}@yandex.ru

***Abstract*— Cloud computing became a popular computing technology, that provides efficient resource utilization to deliver IT services. Each user requests cloud computing system for use of resources. If the system is busy, then user needs to wait until current user finishes the job. This may result in waiting time increase and drop of request Thus, cloud computing service provider needs tools to evaluate and reduce waiting and processing times. In the paper, each request is assumed to consist of several independent sub-requests according to the number of virtual cloud servers in the system. All sub-requests of the same request arrive simultaneously and each server receives exactly one sub-request in its queue. One of the main performance measures of cloud computing system is a maximum waiting and processing time of all sub-requests, which is called response time of the request. In order to evaluate this characteristic, we develop a model in terms of queuing system with multiple queues and batch arrivals. We provide algorithm to obtain steady-state probabilities that allow evaluating various performance measures.**

Keywords—cloud computing, batch arrivals, queuing system.

I. Introduction

Cloud computing is a new approach to computing infrastructure formation. Investment to computing resources was one of the main items of expenses for majority of organizations so far. Using cloud computing services, these expenses may be considered as operational costs. Cloud computing system includes network devices, computing resources and data repositories that may be located faraway from each other. Operator of cloud computing system combines all these components to form unified computing infrastructure [1-4].

In the paper, we propose mathematical model of cloud computing system is terms of multiple-server multiple-queue queuing system with batch arrivals. Similar model with focus on optimal power balancing and ordinary arrival of requests was investigated in [5]. Under the assumptions of Poisson arrival process and exponentially distributed service times we derive two computing method for steady-state probabilities. First one is based on transition rate matrix of corresponding random process [6]. It is shown that transition rate matrix has block-diagonal structure that allows solving system of equilibrium equations using well-known numerical methods. Second algorithm is based on elimination method described in [7]. Finally, we provide evaluation of system response time, which is one of the most important performance measures of cloud computing system.

The rest of the paper is organized as follows. Section II gives brief description of mathematical model. Section III provides computing methods for steady-state probabilities. In Section IV numerical analysis results are presented and Section V concludes the paper.

II. Model Description

We study a cloud computing system with K vendors included. A request sent by user to the system, is split into K sub requests and each vendor serves one sub request. As soon as vendor finishes sub request processing, it reports user. It is considered that the system responded to a request when all vendors finish processing their sub requests.

In order to analyze cloud computing system behavior, we consider K-server queuing system with separate queue for each server. Customers arrive in batches with exactly K customers in a batch. Batch arrival process is assumed to have Poisson distribution with rate λ and customer service time to be exponentially distributed. Denote τ_k response time of subsystem *k*, and according to [5] the total response time of the whole system can be calculated by the following equation:

$$\tau = \max_{1 \le k \le K} \tau_k . \tag{1}$$

Let us denote r_k - buffer capacity of server *k* and μ_k - service rate of server *k*. Figure 1 illustrates the proposed model.

This work was partially supported by the RFBR, research project No. 14-07-00090.

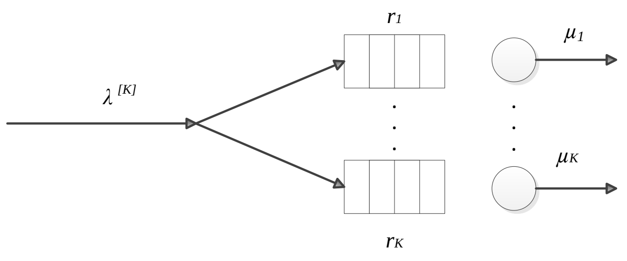


Fig. 1. Multiple-server queuing system with batch arrivals.

Let $n_k(t)$ be the number of customers in subsystem k at time $t>0$, $0 \le n_k(t) \le R_k$, where $R_k = r_k + 1$ is capacity of subsystem k, $k = \overline{1,K}$. System behavior is described by random Markov process $\mathbf{N}(t) = (n_1(t), \dots, n_K(t))$ with the following state space:

$$\mathcal{X} = \left\{ \mathbf{n} = (n_1, \dots, n_K) : 0 \le n_\bullet \le R_\bullet n_k \le R_k, k = \overline{1,K} \right\}, \quad (2)$$

where

$$R_\bullet = \sum_{k=1}^{K} R_k, \; n_\bullet = \sum_{k=1}^{K} n_k .$$

Denote $\mathcal{X}_m = \{ \mathbf{n} \in \mathcal{X} : n_\bullet = m \}$ - a subspace of states with exactly m customers $m = \overline{1, R_\bullet}$. It can be easily proved that the state space of Markov process $\mathbf{N}(t)$ can be expressed in the following form:

$$\mathcal{X} = \bigcup_{m=0}^{R_\bullet} \mathcal{X}_m, \; |\mathcal{X}_m| = C_{K+m-1}^{K-1} - \sum_{i=1}^{m-K} C_{K+i-1}^{K-1}, \quad (3)$$

where C_n^k is binomial coefficient.

III. Steady-State Probabilities

In this section, we provide computing method for steady-state probabilities of the considered system.

A. *Transition Rate Matrix Based Method*

Let $\mathbf{A}$ be a transition rate matrix for the Markov process $\mathbf{N}(t)$ and $\gamma = \max\limits_{k=1,K} R_k$ - maximum possible value of vector $\mathbf{n} \in \mathcal{X}$ elements. Denote $\mathbf{n}(\gamma+1) = (n_1, n_2, \dots, n_K)_{\gamma+1}$ - a number composed of vector $\mathbf{n}$ elements and expressed by numeration system with number base $\gamma+1$. It can be shown that value of $\mathbf{n}(\gamma+1)$ in decimal number system can be calculated as follows:

$$(\mathbf{n}(\gamma+1))_{10} = \sum_{k=1}^{K} n_k (\gamma+1)^{K\text{-}k} .$$

We introduce following lexicographic order on state space $\mathcal{X}$:

$$\mathbf{n}' > \mathbf{n}'' \Leftrightarrow \left(\mathbf{n}'_\bullet > \mathbf{n}''_\bullet \right) \cup \left(\left(\mathbf{n}'_\bullet = \mathbf{n}''_\bullet \right) \cap \left(\mathbf{n}'(\gamma+1)_{10} - \mathbf{n}''(\gamma+1)_{10} > 0 \right) \right). \quad (4)$$

$\mathbf{A}$	$\mathcal{X}_0$	$\mathcal{X}_1$	$\mathcal{X}_2$	…	$\mathcal{X}_m$	…	$\mathcal{X}_K$	$\mathcal{X}_{K+1}$	$\mathcal{X}_{K+2}$	…	$\mathcal{X}_{K+m+1}$	…	$\mathcal{X}_{R_\bullet-1}$	$\mathcal{X}_{R_\bullet}$
$\mathcal{X}_0$	$\mathbf{D}_0$	0	0	…	0	…	$\mathbf{U}_0$	0	0	…	0	…	0	0
$\mathcal{X}_1$	$\mathbf{L}_1$	$\mathbf{D}_1$	0	…	0	…	0	$\mathbf{U}_1$	0	…	0	…	0	0
$\mathcal{X}_2$	0	$\mathbf{L}_2$	$\mathbf{D}_2$	…	0	…	0	0	$\mathbf{U}_2$	…	0	…	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
$\mathcal{X}_m$	0	0	0	…	$\mathbf{D}_m$	…	0	0	0	…	$\mathbf{U}_m$	…	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
$\mathcal{X}_{R_\bullet-K-1}$	0	0	0	…	0	…	0	0	0	…	0	…	$\mathbf{U}_{R_\bullet\text{-}K\text{-}1}$	0
$\mathcal{X}_{R_\bullet-K}$	0	0	0	…	0	…	0	0	0	…	0	…	0	$\mathbf{U}_{R_\bullet\text{-}K}$
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
$\mathcal{X}_{K+m}$	0	0	0	…	0	…	0	0	0	…	0	…	0	0
$\mathcal{X}_{K+m+1}$	0	0	0	…	0	…	0	0	0	…	$\mathbf{D}_{K+m+1}$	…	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
$\mathcal{X}_{R_\bullet-1}$	0	0	0	…	0	…	0	0	0	…	0	…	$\mathbf{D}_{R_\bullet\text{-}1}$	0
$\mathcal{X}_{R_\bullet}$	0	0	0	…	0	…	0	0	0	…	0	…	$\mathbf{L}_{R_\bullet}$	$\mathbf{D}_{R_\bullet}$

Fig. 2. Transition rate matrix $\mathbf{A}$ with block-tridiagonal structure

Considering lexicographic order (4), transition rate matrix has block-tridiagonal structure (Fig. 2), where

$$\mathbf{D}_m\left(\mathbf{n}',\mathbf{n}''\right)=\begin{cases}-\left(\prod_{k=1}^{K}\bar{\delta}\left(n_k',R_k\right)\lambda+\sum_{k:n_k'>0}\mu_k\right), \text{if } \mathbf{n}'=\mathbf{n}'',\\ 0, \qquad\qquad\qquad\qquad\qquad , \text{if } \mathbf{n}'\neq\mathbf{n}'',\end{cases} \quad (5)$$

$$\delta(i,j)=\begin{cases}1, i=j,\\ 0, i\neq j,\end{cases} \qquad m\in\{0,\ldots,R_\bullet\},$$

$$\mathbf{L}_m\left(\mathbf{n}',\mathbf{n}''\right)=\begin{cases}\mu_k, \text{ if } \mathbf{n}''=\mathbf{n}'-\mathbf{e}_k,\\ 0, \quad \text{if } \mathbf{n}''\neq\mathbf{n}'-\mathbf{e}_k,\end{cases} \qquad m\in\{0,\ldots,R_\bullet\} \quad (6)$$

$$\mathbf{e}_k^T=(\underbrace{0,\ldots,0}_{k-1},1,\underbrace{0,\ldots,0}_{K-k}),$$

$$\mathbf{U}_m\left(\mathbf{n}',\mathbf{n}''\right)=\begin{cases}\lambda, \text{ if } \mathbf{n}''=\mathbf{n}'+\mathbf{1},\\ 0, \text{ if } \mathbf{n}''\neq\mathbf{n}'+\mathbf{1},\end{cases} \qquad m\in\{0,\ldots,R_\bullet-K\}. \quad (7)$$

Steady-state probabilities $\mathbf{p}^T=(\mathbf{p}_1,\mathbf{p}_2,\ldots,\mathbf{p}_{R_\bullet})$ can be obtained by solving system of equilibrium equations, which can be written in following form considering block-tridiagonal structure of matrix $\mathbf{A}$:

$$\begin{cases}\mathbf{p}_m^T\mathbf{D}_m+\mathbf{p}_{m+1}^T\mathbf{L}_{m+1}=0,\ m=\overline{1,K-1}\\ \mathbf{p}_{m\text{-}K}^T\mathbf{U}_{m\text{-}K}+\mathbf{p}_m^T\mathbf{D}_m+\mathbf{p}_{m+1}^T\mathbf{L}_{m+1}=0,\ m=\overline{K,R_\bullet-K-1},\\ \mathbf{p}_{R_\bullet\text{-}K}^T\mathbf{U}_{R_\bullet-K}+\mathbf{p}_{R_\bullet}^T\mathbf{D}_{R_\bullet}=0,\end{cases} \quad (8)$$

where

$$\mathbf{p}_m^T=(p_{x(m)},p_{x(m)+1},\ldots,p_{x(m+1)\text{-}1}),\quad x(m)=\left|\mathcal{X}_m\right|.$$

Numerical solution of equations (5)-(8) and normalizing condition give steady-state probabilities distribution for the considered model.

B. Elimination method

Let $t_n, n=1,2,\ldots,$ be instants, at which arrival or departure of a customer occurs. Then system states at $(t_n+0), n=1,2,\ldots,$ instants form an embedded Markov chain $\tilde{\mathbf{N}}(t)$ with same state space $\mathcal{X}, |\mathcal{X}|=M$. Let $\boldsymbol{\pi}$ be a vector of steady-state probabilities of $\tilde{\mathbf{N}}(t)$.

Transition probability matrix $\mathbf{Q}$ of the Markov chain $\tilde{\mathbf{N}}(t)$ can be obtained from the transition rate matrix $\mathbf{A}$ of process $\mathbf{N}(t)$ using following equations:

$$q_{i,j}=\begin{cases}a_{i,j}\left(\sum\limits_{\substack{1\le j\le|\mathcal{X}|,\\ i\neq j}}a_{i,j}\right)^{-1}, \text{ if } i\neq j,\\ 0, \qquad\qquad\qquad \text{if } i=j.\end{cases} \quad (9)$$

Then we start to eliminate spaces of the original Markov chain $\tilde{\mathbf{N}}(t)$ sequentially, beginning with the greatest number state. At the first step, we receive a new Markov chain $\tilde{\mathbf{N}}^{(1)}(t)$ with state space $\mathcal{X}^{(1)}, \left|\mathcal{X}^{(1)}\right|=M-1$ and a modified transition probability matrix $\mathbf{Q}^{(1)}$:

$$q_{i,j}^{(1)}=q_{i,j}+\frac{q_{i,M}\cdot q_{M,j}}{1-q_{M,M}},\ i,j=\overline{1,M-1}.$$

At the s-th step, we have a Markov chain $\tilde{\mathbf{N}}^{(s)}(t)$ with state space $\mathcal{X}^{(s)}, \left|\mathcal{X}^{(s)}\right|=M-s$ and following matrix $\mathbf{Q}^{(s)}$:

$$q_{i,j}^{(s)}=q_{i,j}^{(s-1)}+\frac{q_{i,M-s+1}^{(s-1)}\cdot q_{M-s+1,j}^{(s-1)}}{1-q_{M-s+1,M-s+1}^{(s-1)}},\ i,j=\overline{1,M-s}. \quad (10)$$

The transformation is repeated $M-2$ times, until there is only two states left. Stationary probabilities of the Markov chain $\tilde{\mathbf{N}}^{(M-2)}(t)$ satisfy the balance equation

$$\pi_1^{(M-2)}q_{1,2}^{(M-2)}+\pi_2^{(M-2)}q_{2,2}^{(M-2)}=\pi_2^{(M-2)},$$

hence,

$$\pi_2^{(M-2)}=\frac{\pi_1^{(M-2)}q_{1,2}^{(M-2)}}{1-q_{2,2}^{(M-2)}}.$$

It is shown in [7] that steady-state probabilities of $\tilde{\mathbf{N}}^{(s)}(t)$ and $\tilde{\mathbf{N}}^{(v)}(t)$ are same accurate within a constant. Therefore, assume $\tilde{\pi}_1=1$ and using relations between stationary probabilities of Markov chains $\tilde{\mathbf{N}}^{(s)}(t), s=\overline{1,M-2}$, we can calculate stationary probabilities of initial chain $\tilde{\mathbf{N}}(t)$ accurate within a constant:

$$\tilde{\pi}_j = \frac{\sum_{i=1}^{j-1} \tilde{\pi}_i q_{i,j}^{(M-j)}}{1 - q_{j,j}^{(M-j)}}, \quad j = \overline{2, M}. \qquad (11)$$

Then, we use normalizing condition to calculate stationary probabilities of initial Markov chain $\tilde{\mathbf{N}}(t)$:

$$\pi_j = C^{-1}\tilde{\pi}_j, \quad j = \overline{1, M}, \quad C = \sum_{i=1}^{M} \tilde{\pi}_i . \qquad (12)$$

Finally, stationary probabilities of original Markov process $\mathbf{N}(t)$ are obtained:

$$p_j = G^{-1} \frac{\pi_j}{\left|a_{j,j}\right|}, \quad j = \overline{1, M}, \quad G = \sum_{j=1}^{M} \frac{\pi_j}{\left|a_{j,j}\right|} . \qquad (14)$$

IV. Numerical Results

In this section, results of numerical analysis are presented. We set $K = 3, r_1 = r_2 = r_3 = 3, \mu_1 = 1{,}4, \mu_2 = 1{,}6, \mu_3 = 1{,}8$ and calculate mean value of cloud computing system response time $E\tau$ for various load intensities $\rho = \max_{1 \le k \le K} \rho_k$, where $\rho_k = \lambda / \mu_k, k = \overline{1, K}$. Mean response time $E\tau_k$ of subsystem k can be evaluated using Little's formula:

$$E\tau_k = \frac{N_k}{\lambda}, \qquad (15)$$

where N_k is mean number of customers in subsystem k, and system response $E\tau$ time is calculated by equation (1).

Computing results for our model are compared with results of simulation model of finite capacity queuing system with batch arrival. Figure 3 shows that computational algorithm described in Section 3 provides almost the same mean response time value as simulation model.

Figure 3 also provides mean response time for infinite capacity system ($r_k = \infty, k = \overline{1, K}$). Infinite capacity model gives lower bound of response time for low load intensities ($\rho < 0.5$), but at higher values of ρ response time sharply increases. The reason of this effect is absence of customer loss in infinite system.

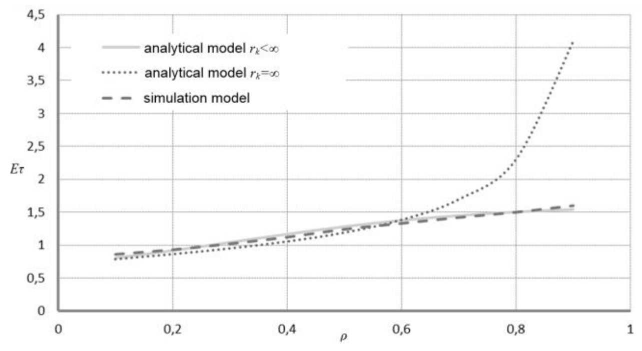


Fig. 3. Cloud computing system mean response time.

V. Conclusion

In our work, we developed analytical model of cloud computing system in terms of multi-server queue with batch arrivals. For the considered model, two computational algorithms for stationary probability distribution are provided. Numerical analysis showed that stationary probabilities and performance measures calculated according to these algorithms are close to simulation results.

Our further research will be devoted to development and analysis of cloud computing model with dynamic scaling and nondeterministic batch size.

References

[1] IEEE P2301 Guide for Cloud Portability and Interoperability Profiles (GCPIP) https://standards.ieee.org/develop/project/2301.html.

[2] IEEE P2302 Standard for Intercloud Interoperability and Federation (SIIF) http://standards.ieee.org/develop/project/2302.html.

[3] OASIS Identify in Cloud Technical Committee https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=id-cloud.

[4] Joint Coordination Activity on Cloud Computing (JCA-Cloud) http://www.itu.int/ITUT/.

[5] J. Cao, K. Li, I. Stojmenovic, "Optimal Power Allocation and Load Distribution for Multiple Heterogenous Multicore Server Processors across Cloud and Data Centers" IEEE Trans. on Computers, vol. 63, issue 1, pp. 45-58, 2014.

[6] M. Firdhous, O. Ghazali, S. Hassan, "Modeling of Cloud System using Erlang Formulas" Proc. of 17th Asia-Pacific Conference on Communications (APCC), pp. 411-416, Sabah, Malaysia, 2011.

[7] P.P. Bocharov, C. D'Apice, A.V. Pechinkin, S. Salerno, "Queueing Theory". Ultrecht Boston: VSP, 2004.

On the Benefits of 5G Wireless Technology for Future Mobile Cloud Computing

S. Shorgin
Institute of Informatics Problems
Russian Academy of Sciences
Moscow, Russia
sshorgin@ipiran.ru

K. Samouylov, I. Gudkova
Telecommunication Systems Department
Peoples' Friendship University of Russia
Moscow, Russia
{ksam, igudkova}@sci.pfu.edu.ru

O. Galinina, S. Andreev
Tampere University of Technology
Tampere, Finland
{olga.galinina, sergey.andreev}@tut.fi

***Abstract*—This paper comprehensively reviews the paradigm of mobile cloud computing, which comprises advantages of mobile computing, cloud computing, and networking. We systematically overview the major benefits offered to mobile cloud computing by the anticipated fifth-generation wireless technology, including the aspects of heterogeneous connectivity, device-to-device and machine-to-machine communications, as well as energy efficiency. Our work concludes by revealing open challenges as well as attractive directions for further research and may be useful for initial orientation in this field.**

Keywords*—mobile cloud computing; fifth generation (5G) wireless technology; heterogeneous networks (HetNets); device-to-device (D2D) communications; energy efficiency, Internet of Things (IoT), Internet of Services (IoS)

I. Modern Mobile Cloud Computing

Today, increasingly capable *mobile devices*, represented by advanced smartphones and tablets, are employed to aid people in their daily routines, from communication and social interaction to storing and processing their important private information. With handheld device industry now becoming a 150-billion-dollar business, we witness an unprecedented diversity of *mobile applications and services* across both consumer and enterprise markets. To this end, *mobile computing* has already developed into a crucial technology allowing us to access information and data anytime, anywhere. However, given limited bandwidth, battery life, and storage capacity of current user equipment, *cloud computing* has recently emerged as the aggregation of computing capability to augment the contemporary computing infrastructure.

A. The potential of cloud computing

Cloud computing generally offers on-demand provisioning of various applications, platforms, and heterogeneous computing infrastructures [1]. Given the scale of its use today, from entertainment, gaming, travel, and news to healthcare, business, and social networking, we expect cloud computing to eventually evolve into the *Internet of Services* (IoS) [2]. With IoS, everything that exists on the Internet today may be represented as a service and then delivered to the end user. Together with Internet by and for the people and the Internet of Things (IoT), the IoS is believed to pave the way for the future *networked society*, where "people, knowledge, devices, and information are networked for the growth of society, life, and business" [3].

The important structural components of the IoS are (*i*) Software as a Service (SaaS), enabling on-demand access to any application, (*ii*) Platform as a Service (PaaS), providing platform for construction and delivery of applications, and (*iii*) Infrastructure as a Service (IaaS) offering on-demand computing networking, and storage infrastructures. Ultimately, diverse applications will be delivered as services over the IoS infrastructure, whereas the hardware and systems software of data centers will be used to provide those services. Here, a vital ingredient for the cloud providers to maintain the scalability of their services as well as to improve the associated operational efficiency is the *virtualization* of cloud resources. Consequently, cloud operators increasingly rely on commodity hardware implementations by means of network function virtualization (NFV) and software defined networking (SDN).

B. Towards mobile cloud computing

Located at the intersection of mobile computing, cloud computing, and networking, *mobile cloud computing* (MCC) inherits the attractive benefits of mobility, communication, and portability [4]. It promises to significantly extend the battery lifetime of mobile user devices, improve their data storage capacity and processing power, as well as augment the reliability [5]. Therefore, it comes as no surprise that cloud-based mobile solutions have grown into a 10-billion-dollar market having applications in image and language processing, sharing Internet data, crowd computing, multimedia search, sensor data applications, and social networking.

Unfortunately, unpredictable user movement in mobile clouds may lead to frequent reconnections and hence brings along the major limitations of MCC, such as unstable connectivity, resource scarcity, and finite energy supply [6]. Therefore, considerable progress has to be made in

This work was partially supported by the RFBR, research project No. 14-07-00090.

communications technology before the MCC challenges could be met satisfactorily. Many, however, believe that recent advances in wireless connectivity hold a promise to mitigate the most pressing demands of MCC [7]. In what follows, **we review the latest developments in wireless communications technology and concentrate on its capabilities to unveil the full potential of future MCC**. Our ultimate goal is thus to provide a comprehensive overview suitable for initial orientation in this exciting area, as well as speculate on associated research challenges that lie ahead.

II. Fifth Generation Communications Technology

Current wireless systems are struggling to meet the anticipated acceleration in user traffic demand aggravated by the rapid proliferation in cloud-based services and applications. With the expected 13-fold growth of mobile data over the next five years, mobile network operators are challenged with the need to significantly *improve capacity and coverage* across their wireless deployments. To augment the existing cellular technology, mobile industry is taking decisive steps in many aspects of *fifth generation* (5G) wireless system design; some of them summarized in the course of our review.

A. Heterogeneous multi-radio multi-cell connectivity

It is a common belief that 5G wireless systems will not be a universal one-size-fits-all solution, but rather become a converged set of various radio access technologies (RATs), integrated under the control of the operator's cellular network. Along these lines, the paradigm of heterogeneous networks (HetNets) has been introduced as a next-generation networking architecture (see Figure 1) enabling aggressive capacity and coverage improvements towards future 5G networks [8]. Today, HetNets already comprise a hierarchical deployment of small cells, on various scales and by different RATs, for capacity together with macro cells for ubiquitous coverage, control coordination, and seamless mobility [9].

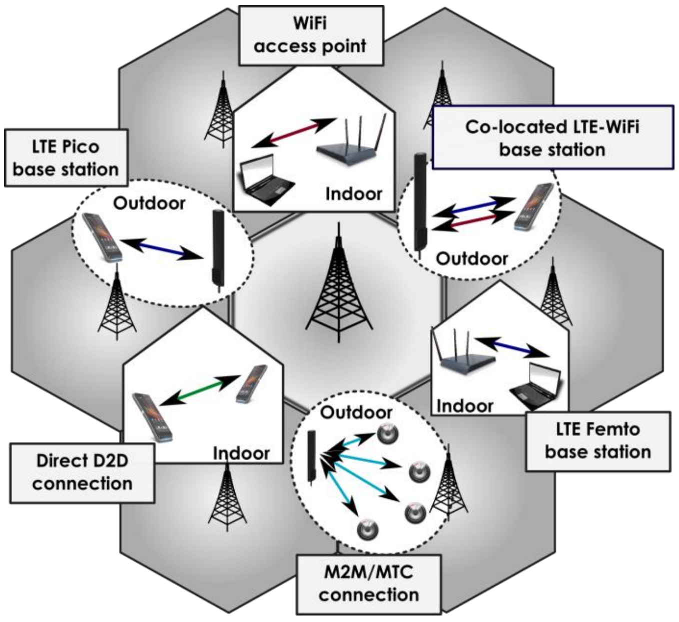


Fig. 1. Envisioned architecture of a 5G wireless system.

An important recent trend in HetNets is the increasing co-existence between cellular (e.g., 3GPP LTE) and local area networks (e.g., IEEE 802.11 a.k.a. WiFi) [10]. By contrast to cellular technology residing in expensive licensed spectrum, WiFi employs unlicensed frequency bands and thus may be preferred for opportunistic offloading of the cellular network traffic [11]. Additional benefits of WiFi stem from the fact that it exists in the multitude of forms (from conventional IEEE 802.11n and high-rate 802.11ac solutions, to mmWave 802.11ad systems, to low-power 802.11ah technology). Motivated originally by the operators' desire to relieve immediate congestion on their networks, the use of WiFi is, however, likely to remain in the mainstream of 5G development, with integrated small cells (employing co-located LTE and WiFi interfaces) flooding the market soon.

To facilitate integration of WiFi under the control of the cellular network, the 3GPP standards community is developing flexible lower-layer coordination mechanisms already for the Release 12 of LTE technology. Such control procedures reside on the radio access network (RAN) level and allow to, e.g., dynamically balance the loading of the associated RATs and even enable their simultaneous operation, when user equipment transmits on several radio interfaces [12]. The fine-grained control schemes employing RAN-level assistance are expected to deliver improved performance to future HetNets by enhancing them in many ways, from advanced RAT discovery and real-time network selection [13] to multi-RAT radio resource management, mobility, and session transfer functions.

B. Network-assisted device-to-device communications

Whereas deploying an increasing density of multi-radio small cells becomes the mainstream direction toward the 5G, network densification naturally implies considerable capital and operational expenditures to install and manage the extra base stations. Therefore, dense HetNets may sometimes require prohibitive investment from the network operators thus making them seek for alternative methods to offload cellular network traffic. Moreover, handling a network with multi-RAT small cells of different sizes may incur significant challenges in cross-cell interference coordination, as well as result in very complex control procedures for network assistance.

Fortunately, there is an alternative solution to offload some of the cellular traffic onto direct device-to-device (D2D) radio links as these are typically shorter and thus more spectrally efficient than the conventional small cell connections [14]. With much of the current mobile traffic growth coming from peer-to-peer applications and services, which typically involve people in close proximity, the benefits of D2D communications for data offloading are becoming increasingly attractive [15]. While D2D-based operation does not employ broadband infrastructure for transferring user data, cellular connectivity may still help by providing assistance with device discovery, D2D connection establishment, and service continuity. All in all, D2D technology can alleviate cellular congestion without the cost of additional networking infrastructure thus having the potential for new service revenues [16].

Direct connectivity may potentially exist in two different forms: as licensed-bands D2D (a.k.a LTE-Direct), when direct links between devices employ cellular spectrum, and unlicensed-bands D2D, utilizing other RATs than cellular for direct connections (e.g., over WiFi-Direct). The former solution is attractive as the cellular network has full control over the in-band D2D links [17], but it also requires significant intelligence to coordinate simultaneously running user transmissions and mitigate harmful interference between them, which does not exist in the standards today. Given the slow progress of respective study and work items in 3GPP (as the result of numerous technical challenges), we do not expect LTE-Direct technology on the market for several years to come. However, research on this topic becomes very timely and is steadily getting momentum worldwide.

An alternative to licensed-bands D2D communications is to connect proximate devices over the unlicensed frequencies, that is, by employing WiFi or Bluetooth technologies. Whereas there is a possibility to communicate over WiFi/Bluetooth also without centralized assistance, there are numerous ways in which cellular infrastructure may help improve otherwise uncoordinated connectivity [18]. Indeed, given that the lion's share of current user equipment is multi-radio devices capable of running simultaneous LTE and WiFi connections, the control coming from the cellular network may improve session continuity, reduce user contention, and facilitate security procedures. Therefore, investigation of unlicensed-bands D2D connectivity remains an attractive research area.

C. Convergence with Internet of Things

The complications of HetNets and D2D connectivity between people are aggravated today by the challenges coming from the integration with the IoT infrastructure [19]. As numerous unattended wireless devices (sensors, actuators, smart meters, etc.) connect to the 5G network, preventive measures are needed to ensure that their uncontrolled transmissions do not disrupt conventional communication [20]. Along these lines, wireless industry has been designing overload control mechanisms to protect priority human-centric communication. With respective procedures standardized previously for Release 11 of 3GPP LTE, the research community has now moved forward with the goal to enable efficient IoT operation [21].

Accordingly, it is widely known that the characteristics of machine-to-machine (M2M) or machine-type communications (MTC) are drastically different from those of human-generated traffic. With small and infrequent data patterns typical for MTC, the network needs additional mechanisms to carry such traffic with low overheads and high energy efficiency. This need is becoming especially pronounced in cellular systems, such as LTE, which have been historically optimized for streaming session-based traffic [22]. To make matters worse, the stringent delay and reliability requirements of industrial-grade MTC applications accentuate the need for further aggressive improvements, which are currently an extremely active discussion topic in the standards.

D. Energy-efficient and green networking

Both human- and machine-centric communication require efficient mechanisms to improve energy efficiency over the current levels due to the limited battery lifetime of mobile handheld devices. Whereas spectral efficiency has been the dominant topic in network optimization over the past decades, the focus of the recent research efforts has been shifting toward "bits-per-Joule" and "throughput-per-Joule" metrics, as demanded by small form-factor user equipment, where wireless power consumption contributes the most to the overall power budget [23]. Correspondingly, the emphasis of the latest investigations has been put onto accounting for the transmit power consumption, together with the associated circuit power expenditures, across a multi-radio multi-cell wireless environment to improve over existing power allocation mechanisms and approach green networking [24].

III. Summary and Open Challenges

In the course of this work, we have reviewed the essential improvements offered by the next-generation wireless communications technology to enable **ubiquitous MCC applications and services**. Our study reveals that despite the fact that significant progress has already been made along these lines, additional steps are required to improve heterogeneous connectivity, mindful of multi-radio access technology, before it may efficiently satisfy the **stringent MCC requirements**. Below we briefly conclude with important directions for future innovation in this area.

On the mobile communications side, further progress is necessary in enabling **higher-bandwidth MCC architectures** (including, but not limited to integrating mmWave access, massive MIMO, and ultra-dense networking technologies). **Service quality and availability** (connectivity, latency, mobility, energy-efficiency, etc.) need to be improved as well by offering more adequate mechanisms to handle heterogeneity in mobile devices, clouds, and wireless networks. On the computing side, additional challenges remain in enhancing the **efficiency of data access**, building **effective context-aware mobile cloud services**, offering more advanced **architectures for mobile computation offloading**, as well as upgrading **security, privacy, and trust**. To successfully pursue these challenges, the analytical modelling, by e.g., employing the mathematical teletraffic theory and queueing theory, is regarded as one of the important ways of understanding the effects of MCC [25-30].

References

[1] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges, IEEE Communications Surveys & Tutorials, vol. 16, pp. 369 – 392, 2014.

[2] R. Moreno-Vozmediano, R. Montero, and I. Llorente, Key Challenges in Cloud Computing: Enabling the Future Internet of Services, IEEE Internet Computing, vol. 14, pp. 18 – 25, 2013.

[3] D. Astely, E. Dahlman, G. Fodor, S. Parkvall, and J. Sachs, LTE release 12 and beyond, IEEE Communications Magazine, vol. 51, pp. 154 – 160, 2013.

[4] L. Lei, Z. Zhong, K. Zheng, J. Chen, and H. Meng, Challenges on wireless heterogeneous networks for mobile cloud computing, IEEE Wireless Communications, vol. 20, pp. 34 – 44, 2013.

[5] N. Fernando, S. Loke, and W. Rahayu, Mobile cloud computing: A survey, Future Generation Computer Systems, vol. 29, pp. 84 – 106, 2013.

[6] H. Dinh, C. Lee, D. Niyato, and P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless

Communications and Mobile Computing, vol. 13, pp. 1587 – 1611, 2013.

[7] P. Rost, C. Bernardos, A. De Domenico, M. Di Girolamo, M. Lalam, A. Maeder, D. Sabella, and D. Wubben, Cloud Technologies for Flexible 5G Radio Access Networks, IEEE Communications Magazine, vol. 52, pp. 68 – 76, 2014.

[8] J. Andrews, Seven ways that HetNets are a cellular paradigm shift, IEEE Communications Magazine, vol 51, pp. 136 – 144, 2013.

[9] B. Bangerter, S. Talwar, R. Arefi, and K. Stewart, Networks and devices for the 5G era, IEEE Communications Magazine, vol. 52, pp. 90 – 96, 2014.

[10] M. Bennis, M. Simsek, W. Saad, S. Valentin, and M. Debbah, When cellular meets WiFi in wireless small cell networks, IEEE Communications Magazine, vol. 51, pp. 44 – 50, 2013.

[11] O. Galinina, S. Andreev, M. Gerasimenko, Y. Koucheryavy, N. Himayat, S.-p. Yeh, and S. Talwar, Capturing spatial randomness of heterogeneous cellular/WLAN deployments with dynamic traffic, IEEE Journal on Selected Areas in Communications, 2014.

[12] S. Andreev, M. Gerasimenko, O. Galinina, Y. Koucheryavy, N. Himayat, S.-p. Yeh, and S. Talwar, Intelligent Access Network Selection in Converged Multi-Radio Heterogeneous Networks, IEEE Wireless Communications, 2014.

[13] L. Wang and G. Kuo, Mathematical Modeling for Network Selection in Heterogeneous Wireless Networks – A Tutorial, IEEE Communications Surveys & Tutorials, vol. 15, pp. 271 – 292, 2013.

[14] L. Al-Kanj, Z. Dawy, and E. Yaacoub, Energy-Aware Cooperative Content Distribution over Wireless Networks: Design Alternatives and Implementation Aspects, IEEE Communications Surveys & Tutorials, vol. 15, pp. 1736 – 1760, 2013.

[15] H. Bagheri, M. Katz, F. Fitzek, D. Lucani, and M. Pedersen, D2D-Based Mobile Clouds for Energy- and Spectral-Efficient Content Distribution, Smart Device to Smart Device Communication, pp. 237 – 280, 2014.

[16] N. Golrezaei, A. Molisch, A. Dimakis, and G. Caire, Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution, IEEE Communications Magazine, vol. 51, pp. 142 – 149, 2013.

[17] B. Kaufman, J. Lilleberg, and B. Aazhang, Spectrum sharing scheme between cellular users and ad-hoc device-to-device users, IEEE Transactions on Wireless Communications, vol. 12, pp. 1038 – 1049, 2013.

[18] S. Andreev, A. Pyattaev, K. Johnsson, O. Galinina, and Y. Koucheryavy, Cellular traffic offloading onto network-assisted device-to-device connections, IEEE Communications Magazine, vol. 52, pp. 20 – 31, 2014.

[19] K. Hwang, J. Dongarra, and G. Fox, Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Morgan Kaufmann, 672 p., 2011.

[20] M. Gerasimenko, V. Petrov, O. Galinina, S. Andreev, and Y. Koucheryavy, Impact of MTC on Energy and Delay Performance of Random-Access Channel in LTE-Advanced, Wiley Transactions on Emerging Telecommunications Technologies, vol. 24, pp. 366 – 377, 2013.

[21] M. Hasan, E. Hossain, and D. Niyato, Random access for machine-to-machine communication in LTE-advanced networks: issues and approaches, IEEE Communications Magazine, vol. 51, pp. 86 – 93, 2013.

[22] K. Zheng, F. Hu, W. Wang, W. Xiang, and M. Dohler, Radio resource allocation in LTE-advanced cellular networks with M2M communications, IEEE Communications Magazine, vol. 50, pp. 184 – 192, 2012.

[23] S. Andreev, P. Gonchukov, N. Himayat, Y. Koucheryavy, and A. Turlikov, Energy efficient communications for future broadband cellular networks, Computer Communications, vol. 35, pp. 1662 – 1671, 2012.

[24] O. Galinina, S. Andreev, A. Turlikov, and Y. Koucheryavy, Optimizing energy efficiency of a multi-radio mobile device in heterogeneous beyond-4G networks, Performance Evaluation, vol. 78, pp. 18 – 41, 2014.

[25] I. Gudkova and K. Samouylov, Analysis of an admission model in a fourth generation mobile network with triple play traffic, Automatic Control and Computer Sciences, vol. 47, pp. 202 – 210, 2013.

[26] V. Borodakiy, I. Buturlin, I. Gudkova, and K. Samouylov, Modelling and analysing a dynamic resource allocation scheme for M2M traffic in LTE networks, Lecture Notes in Computer Science, vol. 8121, pp. 420 – 426, 2013.

[27] I. Gudkova, K. Samouylov, I. Buturlin, V. Borodakiy, M. Gerasimenko, O. Galinina, and S. Andreev, Analyzing impacts of coexistence between M2M and H2H communication on 3GPP LTE system, Lecture Notes in Computer Science, vol. 8458, pp. 162 – 174, 2014.

[28] P. Abaev, Y. Gaidamaka, K. Samouylov, A. Pechinkin, R. Razumchik, and S. Shorgin, Hysteretic control technique for overload problem solution in network of SIP servers, Computing and Informatics, vol. 33, pp. 218 – 236, 2014.

[29] A. Cascone, R. Manzo, A. Pechinkin, and S. Shorgin, Geom/G/1/n system with LIFO discipline without interrupts and constrained total amount of customers, Automation and Remote Control, vol. 72, pp. 99 – 110, 2011.

[30] C. De Nicola, R. Manzo, A. Pechinkin, and S. Shorgin, A two-priority queueing system with trunk reservation, infinite capacity buffers for customers of both priorities, and different service intensities for high-priority and non-priority customers, Lecture Notes in Computer Science, vol. 6886, pp. 230 – 240, 2011.

SDN for network security

R. Smeliansky
Moscow State University,
Computer Systems Laboratory,
Moscow, Russia
smel@cs.msu.su

***Abstract*—Software Defined Networking, SDN, is the programmable separation of control and forwarding elements of networking that enables software control of network forwarding that can be logically and/or physically separated from physical switches and routers. The following important question is considered in this paper: to what extent can SDN–based networks address network security management problems? The new opportunities for enhancing network security brought by this separation are considered in the paper**

Keywords — SDN, NFV, VNF, control plane, data plane, network infrastructure, security, software, protocols.

This research is supported by the Skolkovo Foundation Grant N 79, July, 2012 and the Ministry of education and science of the Russian Federation, Unique ID RFMEFI60914X0003

I. Introduction

Traditionally, networks are defined by their physical topology i.e. how servers, switches and routers are cabled together. That means that once you have built out your network, changes are costly and complex. This type of networking is not compatible with the notion of a "lights-out" datacenter or a cloud environment that needs the flexibility to support varying workload demands.

Security in networks is usually treated as a consistent solution of three problems: confidentiality, integrity and availability of resources. The term "resources" is interpreted in the broadest sense. It could be physical resources, it may be logical resources (software) and it can be information resources (data).

Under the Software Defined Networking approach, the software can dynamically configure the network, allowing it to adapt to changing needs. An SDN-based solution can accomplish several tasks:

- Create virtual networks that run on top of the physical network. In a multi-tenant cloud virtual network might represent a tenant's network topology complete with the tenant's own IP addresses, subnets, and even routing topology. Through SDN virtual networks can be created dynamically, and can support VM mobility throughout the datacenter while preserving the logical network topology abstraction.
- Control the traffic flow within the network. Some classes of traffic may need forwarding to a particular appliance (or VM) for security analysis or monitoring (so-called Virtual Network Function (VNF)). One may need to provide bandwidth guarantees or enforce bandwidth caps on particular workloads. Through SDN, you can create these policies and dynamically change them according to the needs of your workloads.
- Create integrated policies that span the physical and virtual networks. Through SDN, you can ensure that your physical network and endpoints handle traffic similarly. For example, you may want to deploy common security profiles or you may want to share monitoring and metering infrastructure across both physical and virtual switches.

In summary, SDN is about being able to configure end hosts and physical network elements, dynamically adjust policies for how traffic flows through the network,

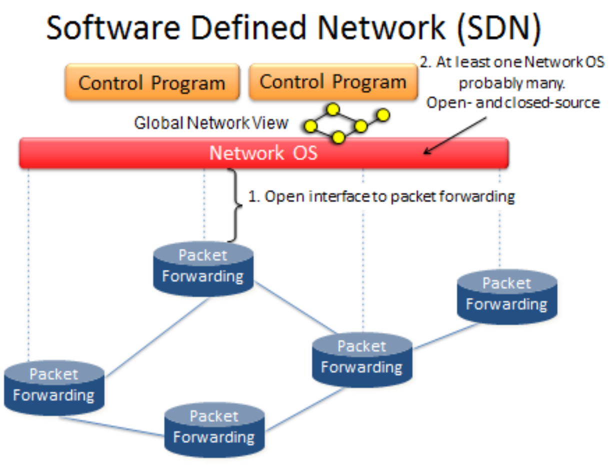


Fig. 1. Software Defined Network organization[1]

and create virtual network abstractions that support real-time VM instantiation and migration throughout the datacenter. SDN programmability includes not only the configuration of physical network elements. It is much broader and includes programmability of end hosts, enabling end-to-end software-based automation and ensuring the reliability in a network [1]. SDN allows to split the solid Data-Control Plane of a Traditional Architecture Network (TAN), make the network behavior control robust and fine grained. In the paper we will

[1]Nick McKeown Moscow talks 2012.

examine what influence this split has on a security of infrastructure, software and protocol of network.

In the following sections we consider the potential impact of applying SDN architecture. We begin by comparing the security of traditional networks (TANs) to those based on SDN, then go on to consider the security implications for the network infrastructure, software stack and network protocols.

II. SECURITY IN TRADITIONAL ARCHITECTURE NETWORKS

There are many areas that may be under different kinds of threats in TAN [14]: infrastructure, software, protocols. In those networks even a single compromised router can cause serious damage to the network and its customers.

In a TAN where Control Plane and Data Plane are inextricably intertwined an intruder can attack the control plane from the data plane by flows of faked requests or by address spoofing and many others ways. So, the control plane isolation enhances the robustness and provides for a reliable network behavior. At the same time this isolation will not help us avoid the typical threats in the data plane like malware, exploring software vulnerabilities, protocols vulnerabilities in data plane.

The number of geographically distributed locations with network equipment is growing in large heterogeneous networks. There are customers, who are engaged in a strong competition with each other. So, there is a need not just to protect the network from misbehavior of applications and customers, but to protect customers from each other, which implies the isolation of data plane of one customer from the data plane of the others.

The term “protect” is versatile. It implies the integrity and confidentiality of customer’s data, protection from the degradation of network services (due to e.g. DDoDs) etc. Spurred by the rapid evolution of networking technologies, we are witnessing the enormous growth of Internet throughput and a shift from the fixed client devices towards mobile devices (since 2003 the number networked devices, sensors etc. exceed the number of PCs. We had over 1 billion connected smartphones already in early 2013, and only about 200 million fixed devices). At the same time the efficiency of existing access control solutions is reduced [11, 12]. In terms of client device mobility, network configurations are changing rapidly and the information on network topology changes can no longer be used directly for access control. So the problem of network access control based on the information about the expected behavior of network applications (flows) is becoming more and more important.

III. INFRASTRUCTURE

One of the main threats in the area of infrastructure security is the physical access to the network devices. For example, in a large airport it’s impossible to guarantee the physical inaccessibility to the network devices. Once the intruder gains a physical access to the device, he/she can modify, replace the internal firmware of that device. An intruder can gain access to the network cabling as well. This is another example of the threat in this area. It is impossible to completely prevent such accesses. For example if a provider needs to exchange some network equipment in their network, no one can guarantee that the equipment on its way from a factory to the provider location was not modified.

In an SDN network, the situation is different. All intelligence resides away from the routers and switches, inside SDN network controllers (see Fig. 1). The server with the controller can be moved into a well protect environment. Programmable controllers can support a set of so-called applications (c-application or control program on Fig. 1) which will supply both ordinary, traditional network services like routing, load balancing for congestion avoidance or DDoS attacks mitigation, QoS management, filtering (as ordinary firewalls) etc. as well as new ones such as virtualization, resources provisioning, monitoring etc. In the security context, virtualization services mean separation, e.g. one virtualized entity which belongs to the one virtual space, should be strongly separated from the other virtualized entity even the same type but belong to another virtual space. We can treat “strongly” in two ways, depending on whether these virtualized entities share the same physical resources or not. This adds extra complexity to the routing or provisioning algorithms running as SDN controller applications. The source of this additional complexity is the dimension of the problem that introduces restrictions on mapping virtual entities onto physical resources.

IV. SOFTWARE

In order to evolve the Internet architecture forward, SDN should absorb and meet the basic principles of today’s Internet [7, 8]. One of the key questions here is what services should be placed in the control plane and which ones should be left in the data plane? There is no clear answer to this question yet. SDN would not be able to significantly enhance the security applications for the attacks exploiting application vulnerability, unauthorized code penetrations and changes i.e. typical attacks in data plane, which don’t involve the control plane.

In an SDN network the control software (c-applications) is concentrated in the controllers. So, another key question is where SDN controller should be placed [2]. As it can be seen today the likely deployment presents a hierarchy of controllers with different responsibilities. This hierarchy should have at least two levels: intra-domain and inter-domain [9, 10] (see Fig.2). On the intra-domain level it should be a controller for internal domain infrastructure management, resources virtualization and routing. Controllers on this level issue the approval for resource allocation under user requests and route the approved flows. They play the role of infrastructure resource managers. For example at an airport a new airline or a new company may issue a request for resources. In this request it may describe what kind of recourses are needed, amounts for each kind of resources and the desired QoS. The mapping of virtual resources onto physical ones is implemented by controllers of another kind. They issue a set of proper rules for the corresponding switches.

- when an incident happens the administrator must be able to determine what it was, recover, report the incident etc.;

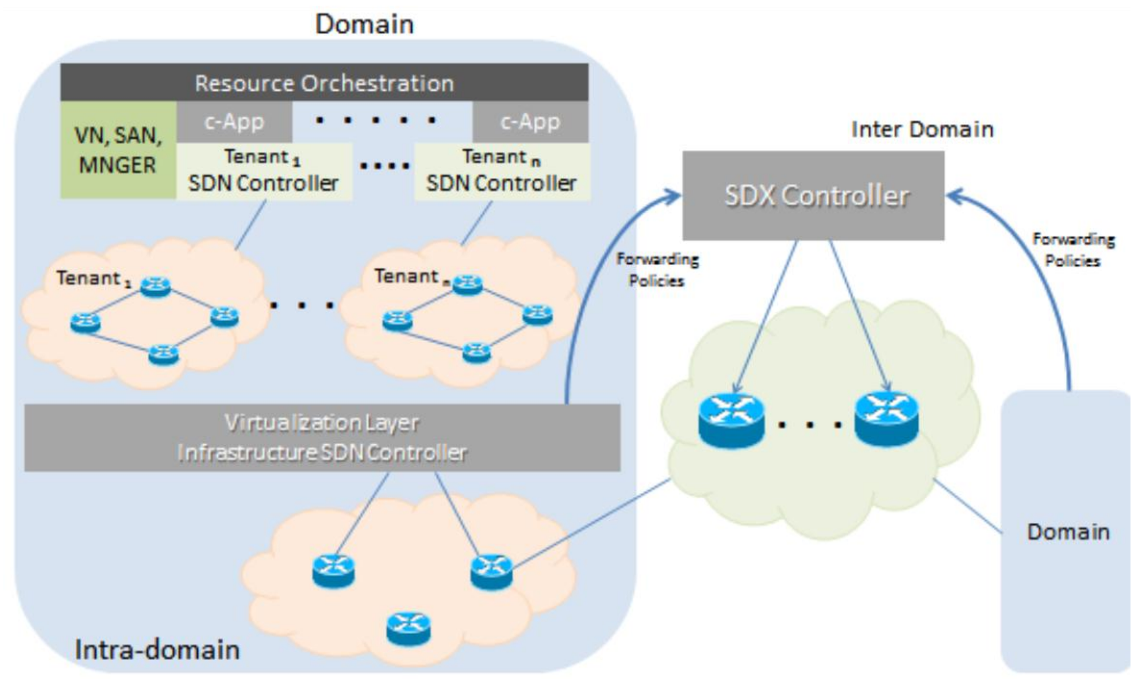


Fig.2. Multidomain SDN Architecture

At the inter-domain level the Controller and its c-applications should support services for specific peering, redirections to middleboxes, traffic offloading, inbound traffic engineering etc. [10]. From the security point of view the most important services would be the dropping of flows that do not correspond to inter domain routing policy even if there are some advertised routes, as well as the blocking of DDoS traffic etc.

A controller should meet the following:

1) Controllers at the same level should have a set of compatible c-applications;

2) C-applications should be reusable by different controllers placed near each other;

3) Different controller instances should be able to share the same instance of a c-application;

4) A controller should be placed in a trusted environment and be a trusted environment for c-applications. This means

- as the centralized decision making point the controller needs to be closely monitored;
- the c-applications and managed devices should be trusted entities;
- there should be a way to make sure the controllers are doing exactly what the administrator actually wants them to do;

5) A controller should be scalable; it means that if the workload is growing beyond the current computational power of the controller, it should be able to get more computational power, for example by splitting its activity with another controller instance, placed on another physical resource. If a controller goes down, e.g. because DDoS attacks, the network goes too therefore a controller should be high available.

6) If some controller instance shuts down, then some other controllers placed nearby should be able to take on those network switches, which were managed by the controller that was shut down.

A secure c-application is another problem. Here we are faced with the same problem, as developers of applications for iPhone, Android etc. A good solution for such problem could be a formal description of c-application behavior [3]. It seems this approach will be more effective, not resource consuming like formal verification e.g. Model checking [4].

Monitoring activities in control plane and in data plane is another crucial function for network security which can be improved by SDN. As SDN switch extracts headers of all layers from a packet at once, the c-applications on SDN controller can do cross-layer analysis and monitoring of the traffic. This opens a way to application aware load balancing, fine grain security policy, application-aware flow sampling, routing, congestion control etc. Monitoring can help in separation and isolation of the virtual data planes by sampling the data flows to ensure that virtualization has separated data flows properly, e.g. the data flows of competitors never share the same physical resource. It can help develop a feedback control loop like RTT monitoring is used in TCP congestion control mechanism. The close coupling and cross-layer awareness of the SDN-based monitoring allows to create novel network control applications that rapidly respond to new multi-vectored threats. Monitoring is an important function for the IPS within the enterprise network... Once an intruder is detected such a system should react properly up to restore the controller operation. It doesn't matter what form an intrusion takes, e.g. unauthorized software, misbehavior c/d-application[2] etc.

SDN has also strengthens security in the data plane by Network Function Virtualization (NFV) which is synergetic with SDN. Virtualized Network Function is controlled and placement managed by SDN controller from the control plane. SDN controller can route a flow through a VNF to filter the content of the data flow, for example to determine the level of malware propagation based on the monitoring information. To achieve this it is possible to build a chain from several properly selected VNFs.

V. Protocols

We will define protocol security as confidentiality and authentication. When discussing the protocol security in SDN environment, we will split the discussion into the following parts:

- Switch-controller protocol security;
- c-Application Protocol security;
- Controller-controller protocol security.

Switch-controller protocol security. According to Open Flow specification [15] in a typical SDN network segment between a switch and a controller an SSL secure connection is used. SSL provides the basic level of security and that may not be sufficient in a real world SDN deployment (requiring for example, more advanced cryptographic protocols: IPsec, Kerberos and etc.). These encryption techniques may be sufficient for Data Centers, but would not be appropriate for WAN networks or Autonomous Systems. We must note that Open Flow protocol does not meet the AAA requirements – Authentication, Authorization and Accounting. SDN controller has to be sure that it communicate with a proper SDN switch and vise verse. This is an example of the Authentication problem. SSL as a solution is bulky and awkward. It has all the problems related to key management, which are added to increased costs and delays associated with encryption [5, 6]. Because of that this mechanism of authentication in Open Flow protocol is turned off in many switches. What is needed is a more light-weight and easy-to-use mechanism. The possible candidate could be mpOTR protocol [16] which satisfies the following requirements: message encryption - no one else will be able to read the message; authentication buddies - confidence in the peer identity; perfect forward secrecy - if you lose the secret keys, the previous correspondence is not compromised; possibility of repudiation - a third person cannot claim that the messages to another recipient were written by someone else. The last point is important, as to not let a third party discover where the SDN controller is placed.

The authorization problem means the SDN controller is allowed to apply specific commands to the switch. There is currently no such mechanism in the OF specification.. The role of authorization mechanism should play a tool, which controls and validates the rules loaded into the OF switch for correspondence to forwarding policy of an appropriate tenant. Accounting can be described as a way of classifying, recording, and reporting events to facilitate effective monitoring activity. OF specification requires to support 40 counters for monitoring purposes. There are 32 counters treated as optional from them. It is impossible to change the semantics of these counters or add some new one if you need. For example, there is only one way to get application awareness through port number. But it is unlikely to be a reliable and secure way to do this.

c-Application protocol security. One of the problems is whether the existing solutions are sufficient and are they different for SDN c-applications? One such point is how encryption keys could be bootstrapped into a proper place in a secure way [13]? As another example, at first it may appear that a native place for traffic analysis is the c-application over the controller. But controller applications focus only on network packet header analysis. Since it requires forwarding the entire packet payload to controller over the typically low-bandwidth control channel, it is not recommended to put deep-packet inspection functions on the controller. The payload must never be inspected on a controller side because there is a risk that the controller will be the victim of an attack, for example, on the mechanisms of main memory manipulation like shell code. Instead, it should be done by a VNF in the data plane, to which a controller directs selected traffic. Another question here is, should the consideration of c-application protocol security include the communications with applications in the data plane (d-application)? On the one hand, the answer should be NO, because then we lose the separation of control plane from the data plane, which is one of the important points of SDN approach towards security. With the separation in place, there is no way to attack control plane from the data plane. However the communication between c-applications and d-applications can be desirable e.g. to let d-applications supply the requirements for QoS to c-application.

Controller-controller protocol security. In the near future SDN controllers will run in a local distributed computer environment like server clusters. In such cases SSL/TLS protocols could be used to protect the communication channel. The major component of a distributed controller is the protocol used for coordination among several controllers

[1] d-application means an application which operate in data plane.

instances in a local environment. Such a protocol can operate in one of two ways: out-of-band, and in-band. In the out-of-band case, a dedicated control network among controllers is used and there is no need for additional network protection. In the in-band case, it is important to provide a secure logical channel to forward data among controllers. In the case of WAN or MAN environment, the controller communications will have two cases mentioned above. If the protocol runs in the data plane it will require new sophisticated security techniques which come with performance and configuration overhead penalties. We must always keep in mind, that simplicity is power.

When considering SDN security in the case of a WAN, we have to understand that a controller would be an extremely desirable target for an intruder. This brings us to the fingerprinting problem, which asks if it is possible to know, whether you operating in TAN or in SDN environment based on e.g. delays, experienced by packets or other indirect measurements of the behavior of the flows traversing the network [16]. There is no the clear answer on this question so far. If the answer will be positive then the next question would be what way to identify the SDN controller location?

VI. Conclusion

Software Defined Networking (SDN) has developed rapidly and is now used by early adopters mostly in data-centers. It offers immediate capital cost savings by replacing proprietary routers with commodity switches and controllers; computer science abstractions in network management offer operational cost savings, which also bring performance and functionality improvements.

An SDN network has a lot of advantages for network security especially in physical security of network equipment. Splitting data plane and control plane allows for robust and fine-grained control of the applications. However a lot of additional research has to be done, especially in SDN software area. The example of one such area is the security of the protocols in switch-controller and controller-to-controller communications.

The major opportunity for the SDN approach is convenient and flexible configuration of packet forwarding policies. Using the functionality of OpenFlow protocol, we can configure forwarding of specific traffic types to go through special network points and also to verify that all the network packets come through these specific points. This feature promises a lot of opportunities for network security, but still requires additional research.

VII. Acknowledgements

The author of this paper is indebted to Ilya Baldin for his numerous discussions and helpful criticism of this paper and the author's English. The author has also considered as a pleasant duty to thank Vitaly Antonenko, Svetlana Gaivoronskay and Alexander Shalimov for the discussions of this paper.

References

[1] http://blogs.technet.com/b/windowsserver/archive/2012/08/22/software-efined-networking-enabled-in-windows-server-2012-and-system-center-2012-sp1-virtual-machine-manager.aspx

[2] B.Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in Proceedings of the first workshop on Hot topics in software defined networks, ser. HotSDN '12. ACM, 2012, pp. 7–12.

[3] R. L. Smeliansky, D.Gamaynov "The model of network applications behavior.", Programming and Computer Software, ISSN 0361-7688, 2007, Vol. 33, No. 6, pp. 308–316. ©Pleiades Publishing, Inc., 2007.

[4] Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled, Model Checking, MIT Press, 1999, ISBN 0-262-03270-8.

[5] M.Lepinski (Ed.) "BGPSEC Protocol Specification," Internet Engineering Task Force, Feb. 2013. [Online]. Available: http://www.ietf.org/id/draft-ietf-sidr-bgpsec-protocol-07.txt

[6] Domain Name System Security Extensions. RFC 2535

[7] Architectural Principles of the Internet. RFC 1958. http://www.ietf.org/rfc/rfc1958.txt

[8] David D. Clark, "The Design Philosophy of the DARPA Internet Protocols", Computer Communications Review 18:4, August 1988, pp. 106–114

[9] P.Lin, J.Hart, U.Krishnaswamy, K-C.Wang et all. Seamless Interworking of SDN and IP. SIGComm 2013, p.475-476.

[10] A. Gupta, M.Shahbaz, L.Vanbever, H.Kimy, R.Clarky, N.Feamster, J.Rexford, S.Shenker SDX: A Software Defined Internet Exchange http://vanbever.eu/pdfs/vanbever_sdx_tr_2013.pdf

[11] Yu-Fang Chung, Ming-Hsien Kao, Tzer-Long Chen, and Tzer-Shyong Chen Efficient Date-constraint Access Control and Key Management Scheme for Mobile Agents IMECS 2010, Hong Kong.

[12] Yang ZHANG and Jun-Liang CHEN Efficient Access Control of Sensitive Data Service in Outsourcing Scenarios. http://eprint.iacr.org/2010/242.pdf

[13] Trusted Computing Technologies. https://www.trustedcomputinggroup.org

[14] O. Santos End-to-End Network Security: Defence_in_Depth. Cisco Press, 2007

[15] Open Flow Switch Specification. Version 1.4.0 (Wire Protocol 0x05), Open Network Foundation, October 14, 2013

[16] Castro, Rui, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. "Network tomography: recent developments." *Statistical science* (2004): 499-517.

A network analytics system in the SDN

V. Sokolov, I. Alekseev, D. Mazilov
P. G. Demidov Yaroslavl State University
Yaroslavl, Russia
valery-sokolov@yandex.ru, aiv@yars.free.net,
denis.mazilov@gmail.com

M. Nikitinskiy
A-Real Group, Energia-Info Inc.
Yaroslavl, Russia
man@a-real.ru

Abstract—**The emergence of virtualization and security problems of the network services, their lack of scalability and flexibility force network operators to look for "smarter" tools for network design and management. With the continuous growth of the number of subscribers, the volume of traffic and competition at the telecommunication market, there is a stable interest in finding new ways to identify weak points of the existing architecture, preventing the collapse of the network as well as evaluating and predicting the risks of problems in the network. To solve the problems of increasing the fail-safety and the efficiency of the network infrastructure, we offer to use the analytical software in the SDN context.**

Keywords—software-defined networking; sdn; software system; application programming interface; smart tool; big data; analitics; fail-safety; openflow; protocol; flow; flow table; analysis; monitoring; network topology; heuristic; network statistics; weighted graph; dynamic network model; load balancing

I. Introduction

In recent years the explosive growth of the number of subscribers in computer and mobile networks, the emergence of new smart devices and applications have led to a significant increase of the network traffic. The growth of the mobile traffic proportion, network connection speed and many other factors result in qualitative changes in the Internet [1].

The existing architecture of the global network was not designed for the actual volumes of information. New technologies and network services complicate network structure and impose heavy demands on the communication channels organization and the network resource management. A list of these requirements is being expanded and updated, gradually creating a gap between the currently applied solutions and real needs of customers. The advent of virtualization and security problems of the network services, their lack of scalability and flexibility force operators to look for more "intelligent" tools for network design and management.

The traditional model of operator networking is static. It aims to solve common tasks and cannot provide a flexible approach to the implementation of individual schemes of the network service providing [2]. With the software-defined networks (SDN), a new approach to networking, providers received a powerful tool for the network infrastructure realization. The key idea of separating the control plane and the data transmission eliminates the traditional model in favor of efficient and flexible solutions. The use of the SDN will reduce the cost of deploying and maintaining the networks and intranet services, enhance the ability to provide unique services, making a technological basis for new business interactions.

With the continuous growth of the number of subscribers, the volume of traffic and competition at the telecommunication market it is extremely important for operators to promptly respond to various changes in the infrastructure and the network topology. It rouses interest for finding new ways to identify the weak sides of the existing architecture, to prevent the collapse of the network by correct load balancing, to evaluate and predict the risks of any problem situations in the network. In recent years, with the growth of computing power and technology in the field of processing big data, the development of software tools for analyzing various systems and processes is significantly accelerated.

The software system designed for the German national team to the World Cup 2014 can serve as an abstract example of success in this area. With the help of various means of monitoring this system collects all sorts of information about the players: the number of taps, the average time of ball possession, distance, running speed and changes of movement direction, and much more. The data collection results in a report on the effectiveness of the players, allowing to identify their strong and weak points. Thanks to the conclusions, the concrete and tangible result has been produced - the average time of possession has been reduced from 3.4 seconds in 2010 to 1.1 second in 2014.

To solve the problems of increasing the fail-safety and the efficiency of the network infrastructure, we offer to use an analytical software in the SDN context. The presence of a convenient tool for checking the current state of the topology, the existence of "bottlenecks", opportunities and conditions for the connection of new subscribers will reduce the cost of the maintenance of the network and increase its reliability.

II. Overall description

Based on the described problem, we came to the conclusion that there is a need to develop a software system that enables a comprehensive analysis of the state of a software-defined network. The key points that allow creating tools, that perform the network configuration and its monitoring from the outside, are an open application programming interface (API) and a centralized control plane in the SDN which contains information about all network resources and fully controls their distribution [3].

With the OpenFlow protocol, the application can obtain information about the flow tables from all switches and construct a graph of the network topology for further analysis of its properties and specifications. Various parameters of the network devices and connections can be objects for analyzing and monitoring:

- the number of processing flows,
- the number of packets in a given flow,
- the number of bytes in a given flow,
- the number of packets that pass through a given port,
- the lifetime of a given flow [4].

The result of this analysis should be a set of rules and heuristics dynamically created by the system that allow an administrator to receive early warning of any network problems, perform its safe reconfiguration and its optimization. Taking this into account, we can identify a number of possible applications of the tool for the analysis in the SDN area.

III. Collecting network statistics

One of important network characteristics is its numerical parameters: the traffic volume, the number and speed of connections. The ability to monitor these parameters allows to create a flexible system of reporting and collecting statistics both in the entire network and its individual segments in a given time period. In turn, the analysis of these parameters in real time enables the administrator to control the network.

The collection of the detailed statistics is provided by the OpenFlow protocol opportunities. When writing this article, the authors relied on version 1.4 of this protocol [4].

The controller can interrogate the controlled switches and to obtain the necessary information. For example, to obtain data on the work of a single flow, you can use the multipart request OFPMP_FLOW, which is offered by the OpenFlow specification. The number of bytes in the flow and the lifetime of the flow in seconds are among the return parameters. Through the regular request of this data we can form the full statistics of the flow rate and the amount of the transferred traffic. This will result in a clear detailing of the channel use by a particular user [4].

The use of this detailing can be different: from calculations of reporting charts of the load distribution of the channel in relation to a specified period prior to the formation of individual client proposals based on the frequency and intensity of the channel use.

The collected statistic is essential for the correct formation of network reconfiguration and routing policies in case of creating a channel for the new user. The OpenFlow allows collecting statistics not only in relation to the logical flow of data, but also in the context of specific switch ports. For example, through the multipart request OFPMP_PORT_STATS, it is possible to obtain information on the port lifetime, the number of transmitted packets and bytes, including dropped packets and errors [4]. Having collected statistics, in other words, the detailing of a specific port, we can check the possibility of its use for the transmission of a new data flow for the given connection conditions. An example of using this data will be presented below.

IV. A dynamic network model

As noted, an outer software system is able to collect and analyze information about the contents of flow tables, request detailed statistics on ports and data flows.

Based on these data, the system can build a dynamic network model, presented by a weighted graph. The calculation of graph weights depends on a number of parameters and changes dynamically on the base of the collected statistics and in accordance with the network changes.

Among the model parameters should be noted:

- the number of the flows for a specific port,
- the average value of the enabled bandwidth for a particular port,
- the peak statistics (if available - a rule) of the required bandwidth for a specific port.

The formation of such a network model allows automating a number of tasks of the network configuring, to inform the administrator, to carry out proactive support and verification of the changes in the configuration process. Below are a few examples of possible application of the intelligent system for the SDN analysis.

V. Setting up a new connection with specified parameters

For clarity, we give an example: an administrator received a request to create a new connection. The connection parameters (edge devices, speed) are fixed in the system which analyzes the existing topology and offers a network administrator to choose (approve) one of the channel options compiled by the system.

In this case, the program of the analysis has the following general workflow algorithm:

- Search all possible routes from point A to point B.
- Apply to the resulting options set of heuristics based on specified parameters (e.g. bandwidth and prioritization)
- Generate the final channel set for the network administrator.

The second step of the algorithm discards the routes that are unacceptable or ineffective. All parameters and threshold values the filter triggering can be changed in the application settings.

Thanks to the collected statistics the system will take into account the level of the data ports load while selecting a particular channel. If the requested bandwidth exceeds the difference between maximum allowable and average values for a specific port, it should be deleted from consideration. In the case of borderline situations when the bandwidth is wide enough, but the network can have a bottleneck in high load cases, it will be made one of the following steps, depending on the settings of the system heuristics: the route will be either dropped, or it comes to the final set with an appropriate warning.

VI. THE LOAD BALANCING IN THE EXISTING NETWORK

Checking an existing network or its individual segments for the presence of bottlenecks is also possible after the collection of the relevant statistics. Similarly to the previous example, heuristics will be applied to possible routes and then optimization variants will be generated.

VII. THE USE OF THE FALLBACK CHANNEL

The use of the collected statistics can be very useful from the view point of the fail-safety task in the network.

Dynamic changes in the model allow to track the problematic situation in which the partial or complete failure in customer service are possible. For example, it can be the breaking of the link between two nodes in the graph topology. To prevent such failures, the system calculates a fallback route for the connection. In case of the current route failure, an automatic switch reconfiguration will be made by using the fallback routing policies.

VIII. CONCLUSION

In today's world of technology we can clearly see the tendency to simplify the external side of the software, to lower the threshold of entering. This is due to the increase of the internal complexity of software systems, when an application takes more and more responsibility for the processes control.

The emergence of expert and analytical systems corresponds to this trend in various areas of the software systems support and maintenance.

By summarizing all the preceding, we can conclude that the development of systems for collecting network statistics and further analysis is an important step in the SDN evolution. It will reduce the risks of network failures, increase the administrator efficiency, reduce the cost of the network maintenance.

Our current task of creating the system of the network analytics is of interest together with tools for formal verification of the SDN. The consideration of formal models of the SDN, their verification, reviewing and comparing the existing solutions is beyond the scope of this article and is a theme for the future research.

The work was supported by the grant of the Ministry of Education and Science of the Russian Federation, unique ID RFMEFI57614X0105.

REFERENCES

[1] Cisco, “Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018”, February 2014

[2] S. Darbha, M. Shevenell, J. Normandin, “Impact of software-defined networking on infrastructure management”, CA Technology exchange: Disruptive technologies, vol. 4, issue 3, November 2013, pp 33-43.

[3] Active Broadband Networks, “Software-defined networking (SDN) transforms broadband service management”, September 2013.

[4] Open Networking Foundation, “OpenFlow Switch Specification Version 1.4.0”, October 2013

Network utilization optimizer for SD-WAN

L. Vdovin, P. Likin, A. Vilchinskii
MERA
Nizhny Novgorod, Russia
{lvdovin, plikin, avilchin}@mera.ru

Abstract— **the question how to use the maximum of network possibilities is still open. ISPs and large distributed companies use only 40-50% of total network bandwidth. Technology that helps to increase network bandwidth utilization and redundancy is crucial and there is still no generic and simple solution for this problem. The SDN architecture advocates the separation of data and control plane, and helps to simplify the network management and maintenance due to logically centralized software. Basing on this approach, our team has implemented a simple solution solving network utilization issue. Remote SDN controller runs on high performance server and this enables to apply relatively complex per-flow global routing algorithms. An application tracks network state. In case of link fault, the flows affected by outage are re-routed over alternative path. A whole network acts as the single distributed L2 switch from external connections perspective, but solution architecture allows to change a whole network representation from L2 switch to distributed L3 router. Application was developed by using OpenFlow technologies at data plane devices. The application uses modified Dijkstra algorithm. The algorithm searches for the route with the best spare capacity based on actual network utilization. Also the algorithm allows to control route length over per-hop penalty. So the developed application allows to apply per-flow policing in terms of bandwidth and latency. Nowadays OpenFlow controllers don't have a standardized API and it makes it impossible to change a controller for your application. To avoid this issue an OpenFlow independent Controller-Application specific interface has been developed. Interface uses application specific proprietary message format optimized to increase configuration performance. So our application is flexible in choosing OpenFlow controller. Characteristics for our prototype have been defined based on performance characteristics of Yarnet ISP located in Yaroslavl and it should work with 30 nodes (each node has at least 3 connections per switch) and establish 5000 flows per second and has traffic outage less than 1 second. The characteristics were measured using simulated and target test environment. Developed application will be used as the framework to implement traffic policing features, QoS, bandwidth and latency reservation.**

Keywords—Software-defined networking; Software-defined-WAN; OpenFlow; Floodlight

I. INTRODUCTION

Majority of Internet users travel between city areas during the day, so that they log in from different areas, but keep producing large amount of network traffic (Figure 1).

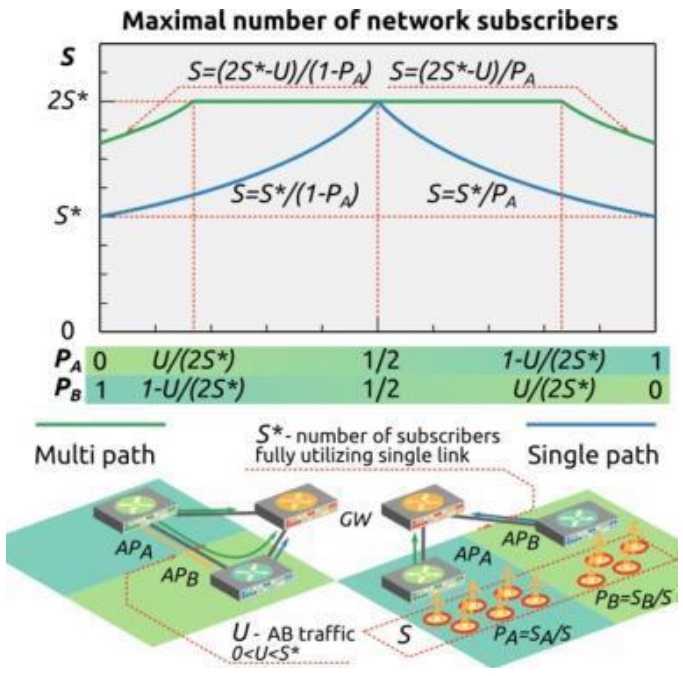

Maximal number of network subscribers
S
2S*
S*
0
S=(2S*-U)/(1-P_A)
S=(2S*-U)/P_A
S=S*/(1-P_A)
S=S*/P_A
P_A 0 U/(2S*) 1/2 1-U/(2S*) 1
P_B 1 1-U/(2S*) 1/2 U/(2S*) 0
Multi path
S* - number of subscribers fully utilizing single link
Single path
AP_A
AP_B
GW
AP_A
AP_B
P_B=S_B/S
U - AB traffic 0<U<S*
S
P_A=S_A/S

Figure 1: Benefits of using multi path instead of single path.

Traditional star and ring ISP routing network topology is not so flexible from subscribers' distribution perspective (Figure 2). In case majority of users are located in specific areas, network connection to this area may be oversubscribed, but other connections won't be used.

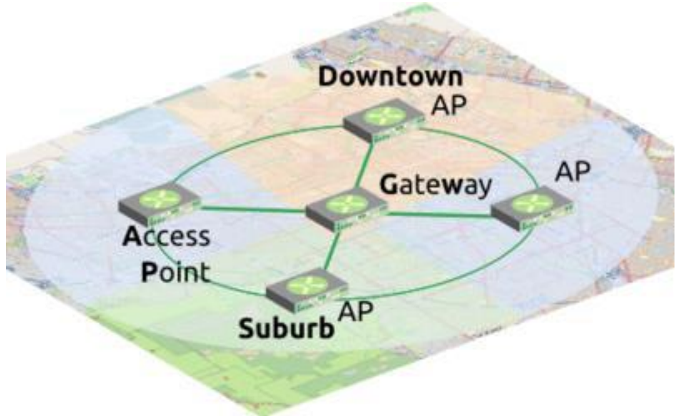


Figure 2: Start network topology.

Enterprise clouds, video calls and other peer-to-peer application apply new requirements for ISP network capacity. High bandwidth is required to connect subscribers' networks directly to each other. Mesh routers topology together with multipath routing technic may increase tolerance of the network capacity towards subscribers' geographical distribution. SDN approach may simplify deployment for these solutions sufficiently [1].

Our team set the goal to design a SDN application managing city scale ISP network with mesh connectivity as single forwarding media. Traffic flows should be equally distributed along the network based on actual links utilization and should develop the application prototype which meets 10% of system characteristics requirements. The following system requirements have been defined:

- managed network must act as single virtual forwarding equipment from external connections perspective (L2 learning switch, extendable to L3 router)
- packets from the same flow should be forwarded over the same route, flows are matched by defined user criteria (source/destination IP)
- in case of link failure, affected flows should be re-routed over alternative path.
- system should be prepared for QoS, DiffServ and bandwidth reservation features [2].
- it should be possible to integrate feature with different types of OpenFlow controllers [3].
- system architecture should be scalable to support big networks.

The system characteristics were based on characteristics of the one of Yaroslavl`s ISP [4]:

- network size: 300 nodes (30 for prototype)
- connectivity topology: each node has direct connection to 20% of nodes in the topology (at least 3 connections per switch for small topologies)
- flow establishing rate: 50k flows/s (5k flows/s for prototype)
- Traffic outage due to link fault: < 1s.
- Maximal number of flows per edge forwarding node: 300k flows (30k flows for prototype)

II. IMPLEMENTATION DETAILS

Network Utilization Optimizer (NUO) is standalone application communicating with one or more OpenFlow controllers over proprietary interface (App-Control interface).

We used to extend NOX OpenFlow controller for the very first prototypes. Unfortunately it was observed that NOX development had been stopped on GitHub. Hence it was decided to migrate to Floodlight controller whose contributors' community is more active. We should state that it is quite hard to forecast which of OpenFlow controllers will be the most suitable for our purposes next day. Also OpenFlow controllers don't have a standardized API nowadays, so that it is impossible to change a controller for your application. To avoid this issue we developed an OpenFlow independent Controller-Application specific interface. Interface uses application specific proprietary message format optimized to increase configuration performance.

So our application is flexible in choosing OpenFlow controller. Currently the solution with independent Controller-Application interface allows us to migrate to another OpenFlow controller again. It will be rapidly extending and we have chosen perspective OpenDaylight controller.

"Switch state control" system functions collect and track network topology information (network graph), switch ports operating modes (link bandwidth), switch states and bandwidth utilized by every flow. Meta Forwarding system functions process packets on external port as L2 learn switch (may be extended to support L3 routed functions).

As soon as a new flow is detected by the application at network external port, it is processed by Meta Forwarder logic according to L2 learning switch functions. Meta Forwarder logic stores the source address and then examines the destination address. If it is not found in forwarding database, the frame is to be flooded on all external ports from virtual equipment point of view. In case destination host for the flow is known – new internal route is calculated towards destination switch using modified Dijkstra to secure maximal network utilization (Figure 3). Packet flows for internal routing are classified based on Source IP address – Destination IP addresses pair. Flows are expired due to packet inactivity. In case port or switch goes down OF, controllers and application are notified about outage.

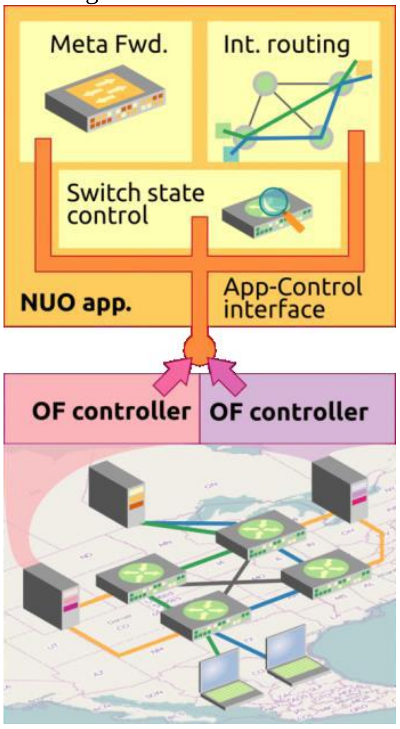


Figure 3: The NUO high level structure. Two new traffic flows have been routed through network with different paths.

All flows routed over faulty link are re-routed over alternative path. During link outage packets are forwarded over controller until new route is created.

OpenFlow controller polls PM counters for every created internal flow with fixed polling rate (Figure 4). In case packet rate crosses discreet threshold, “utilization change” indication is sent towards NUO application. NUO application uses the latest received values to calculate spare bandwidth for link in case new flow is being created.

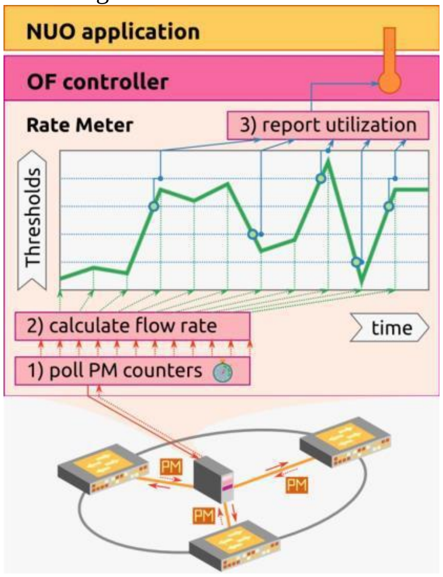


Figure 4: The NUO application makes it possible to poll PM counters and to change the discreet threshold for the flow.

It is reasonable to consider OpenFlow specification extension to support similar features at switch side. OpenFlow 1.3 meter functions may be extended with new meter type sending notification towards controller as soon as threshold is crossed [5]. The NUO application uses modified Dijkstra algorithm (Figure 5):

- Next hop is selected in order to secure maximal residual capacity along all links in the path.
- Each hop adds additional penalty in order to take into account the route length by using route “aspect ratio”
- Route “aspect ratio” controls balance between route length and capacity $A_R=0$ – 1 hop. $A_R=1$ - highest bandwidth.
- Route score directly depends on route residual capacity available at the route and hop count.

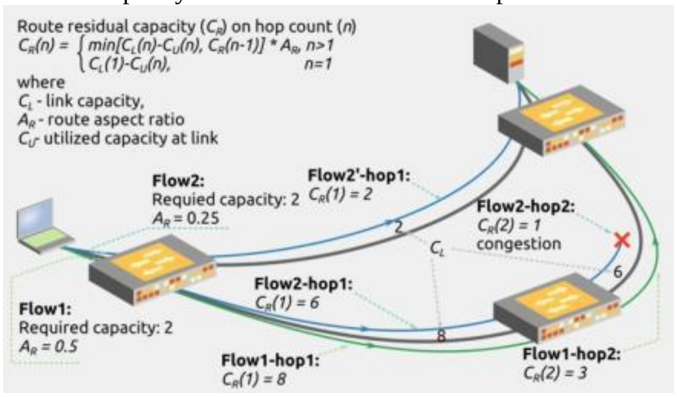


Figure 5: The NUO application modified Dijkstra algorithm.

The main parts of NUO application are (Figure 6):

- **Application-controller interface** (A-C). It is TCP based interfaces towards OF controller and control app. Proprietary message format is a subset of OpenFlow protocol however it has reduced message size.
- **The application** implements the main logic and includes A-C interface server with multi-client feature support. The application does not depend on the specific OpenFlow controller and may be split into parts for better scalability.
- **Floodlight controller** with stateless adaptations. Includes A-C interface client to convert A-C message into OpenFlow and vice-versa. Also polling logic of PM counters was implemented on the controller side [6].
- **OAM tools Nuoctrl and FlowTrack**. The tools provide CLI to control and supervise NUO application and GUI for network and flow visualization.

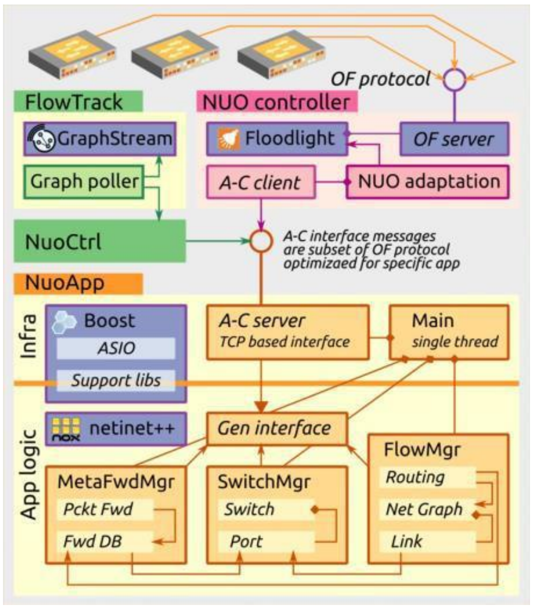


Figure 6: The NUO application main parts and its interaction.

III. The NUO Application Characteristics

An application performance was verified on all levels by unit, component and system tests. System testing had very limited capabilities due to low performance characteristics of mininet [7] and Tp-Link 1043ND switches with OpenFlow enabled OpenWRT software [8] used in the system. 50 flows with 50 pps was originated during the test, control and data path packet latency were measured based on OF PDU and A-C interface PDU timestamps (Figure 7).

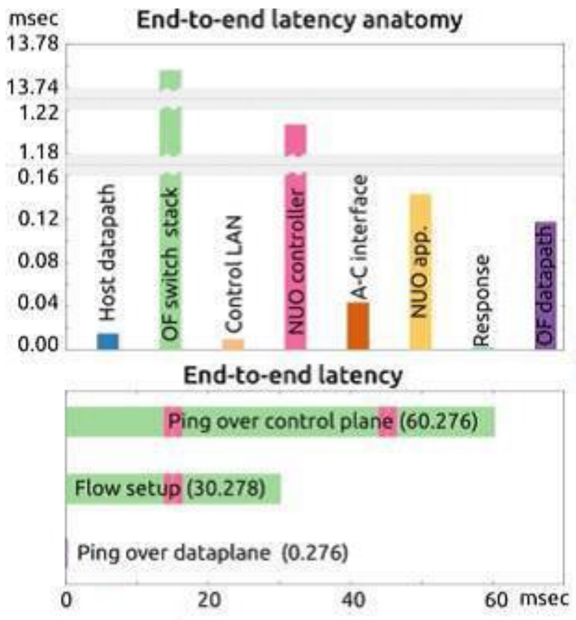


Figure 7: NUO application end-to-end latency.

Two main tests to compare the characteristics with classic networks have been performed. In the first test scenario test host originates 35 ICMP traffic streams with constant 60 pps rate, test server responds to ICMP requests [9, 10]. All streams run over the same link. Timestamps for ICMP requests and response are collected at host originating traffic (Figure 8). Ostinato network packet crafter was used to generate and analyze ICMP traffic streams.

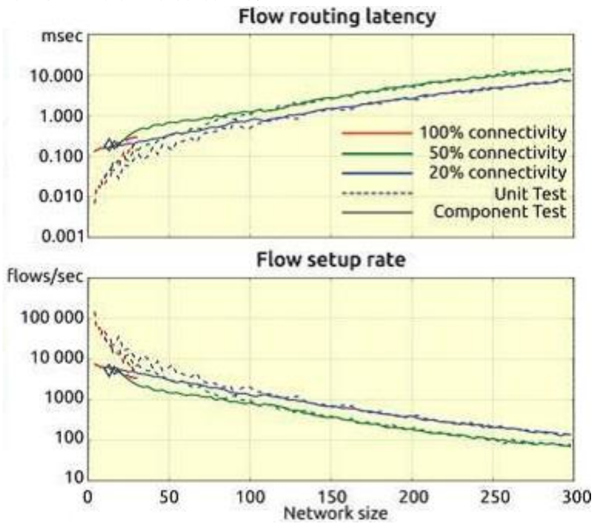


Figure 9: The NUO application forwarding characteristics.

Timestamps for OF control messages are collected at OF controller. Link is disabled, flows are re-routed by controller. During failover some packets are forwarded over OF controller. We got the following results after the test:

- **Latency increment** 30 msec for 3 flows.
- **Traffic disturbance time** 45 msec.
- **Failover time** 53 msec.

The second test case was developed to make sure NUO application is able to increase network throughput in case subscribers relocate between access points. Test covers the case when all subscribers are located near single access point or equally distributed between access points. Two Iperf sessions running simultaneously were used in order to measure traffic bandwidth improvement.

CONCLUSIONS

In conclusion we would like to say that SDN approach together with OpenFlow technology allows simple deployment of solutions providing global network control. Big companies, for example Google, are interested in effective resource management and already have production ready applications to manage scale networks [11].

Proposed approach for OpenFlow applications architecture allows to concentrate on functions development without dependency on specific controller. OpenFlow based approach delivers high level of redundancy and good failover characteristics in comparison with traditional L2 redundancy schemas. SDN technology allows to move complicated algorithms (e.g. routing) from network devices to high performance data centers, to reduce network deployment costs and to increase system performance. OpenFlow hardware performance characteristics are crucial to leverage SDN based solution into the market.

REFERENCES

[1] T. D. Nadeau, K. Gray. SDN: Software Defined Networks. O`Reilly, 2013. – 384p.

[2] K. K. Yap, T. Y. Huang, B. Dodson, M. S. Lam, N. McKeown. Towards Software-Friendly Networks. APSys, 2010.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner. OpenFlow: enabling innovaton in campus networks. SIGGCOMM Comput. Commun. Rev. 38, 2 , March, 2008.

[4] Yarnet, "The summary of core network load". 27 Jun. 2014; http://yarnet.ru/company/yarnet/asr/.

[5] Open Networking Foundation. OpenFlow Switch Specification version 1.3.3

[6] Open Networking Foundation. OpenFlow Switch Specification version 1.0

[7] B. Lantz, B. Heller, N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. SIGCOMM, November, 2010.

[8] OpenWRT, "TP-Link TL-WR1043ND". 27 Jun. 2014; http://wiki.openwrt.org/toh/tp-link/tl-wr1043nd#downloads.for.tl-wr1043nd.v1.x.

[9] B. Heller, C. Scott, N. McKeown, S. Shenkler, A. WunDsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarisx, P. Kazemian. Leveraging SDN Layering to Systematically Troubleshoot Networks. SIGCOMM HotSDN, 2013.

[10] H. Zeng, P. Kazemian, G. Varghese, N. McKeown. Automatic Test Packet Generation. IEEE/ACM Transactions on Networking, 2013.

[11] "OpenFlow at Google", 18 Aug. 2014; http://www.opennetsummit.org/archives/apr12/hoelzle-tue-openflow.pdf.

From Federated Software Defined Infrastructure to Future Internet Architecture

K.-C. Wang
Clemson University
Clemson, SC, USA

M. Brinn
Raytheon BBN Technologies
Cambridge, MA, USA

J. Mambretti
Northwestern University
Evanston, IL, USA

***Abstract*—Significant efforts have been devoted to creating large scale compute and network testbeds for studying future Internet challenges. Besides large geographic span, the common emphasis is programmability, allowing researchers to reserve or create, via software, flexible sets of compute and network resources over specified topologies to execute research prototypes of new protocols, processes, and applications. Also emphasized are virtualization, instrumentation, and software defined networking (SDN) capabilities of the infrastructure. SDN in particular stimulated significant interests in academia, industry, and public sectors to re-imagine the future computing and networking infrastructure landscape and roadmap while it becomes increasingly utilized in production environments. Amidst these interests, one can start to capture desirable characteristics to glimpse the potential architecture of the future Internet. In this paper, we discuss the significance of compute-network interaction across complex, highly customized federated architecture in the future Internet.**

Infrastructure federation has been happening across multiple dimensions. Federation expands the scope of infrastructure, geographically and administratively, for use by members of different organizations. For example, federation initiatives are underway among: 1) US Global Environment for Network Innovations (GENI), Europe Future Internet Research and Experimentation (FIRE), and future Internet testbeds in Asia, South America, and Canada, 2) university production infrastructure, 3) US cities, 4) US public research institutes, and 5) commercial infrastructure. While requirements and objectives differ, they must all address a common set of issues. Such federation suggests the fundamental needs of applications to interact with compute and network resources across a generic, federated, future Internet environment.

***Index Terms*— applications, software, hardware, network architecture, network federation, network testbeds, SDN, SDI, virtualization, future Internet.**

I. Introduction

Around the globe, research communities have commonly acknowledged the need of persistent, large-scale physical experimental facilities to study significant challenges facing Internet and to evolve the Internet to meet future requirements. One of the primary challenges is the wide range of diversity in the network quality of service expected by applications. The challenge poses itself in several aspects. First, the reduced core-to-edge bandwidth ratio due to the introduction of gigabit wireless edge networks and the increasing dominance of wireless and mobile traffic across Internet **require** *on-demand, topology-, mobility-, and application-specific quality of service control across Internet*. Second, the growing applications requiring access, processing, and exchange of large and often real-time data **require** *persistent connectivity among data centers and devices potentially at a global scale*. Third, the increasing use of Internet for critical applications with specialized security requirements **requires** *customized processing and monitoring of application-specific traffic*. With these requirements in mind, future Internet testbeds developed across the globe have emphasized deep programmability via software in the infrastructure to create a "software defined infrastructure" (SDI), encompassing elements from SDN to compute nodes, sensors, instruments, storage, and others. At the same time, researchers are looking into the federation of such SDIs to allow research at a truly global scale close to that for the future Internet.

In fact, federation is an inevitability considering the numerous Internet services, processes and other activities that are global in scope and must utilize resources in and across multiple domains. Furthermore, as most SDI testbeds are realized as part of production networks, decisions of the federation approaches among these testbeds have direct implications to the federation of the production infrastructure of the future Internet. This paper overviews a number of ongoing and proposed community efforts in SDI federation and explores open questions to be discussed and investigated.

II. Federation Defined

As defined in the Merriam-Webster dictionary, "federation is "…an organization that is made by loosely joining together smaller organizations". Put into the specific context of SDI, the US Global Environment for Network Innovations (GENI) project [1] defines: "A *federation* is a set of agreements among people or organizations, representing the policies and terms under which they will trust, collaborate, share resources or engage in other common activities" [2]. Indeed, federation defines the policies and terms of usage of resources *based on the preferences and needs of the people and organizations involved*. The primary stakeholders and their needs can then be identified to be its *entities*, *actions*, and *interfaces*.

Entities: In a federation, three types of logical entities are essential – resource users, resource providers, and the trust providers. In the GENI example, it defines itself as a research testbed; therefore, its users are research *experimenters*, its resource providers are operators of GENI compute and network resources on participating universities and research institutes. As far as the trust provider, GENI adopts a simple single-clearinghouse approach, based on the umbrella permission granted by participating universities to offer access to all users that can be formally identified as a member of a GENI-participating institution via InCommon [3]. Figure 1 illustrates the three roles. Mapping to our terminology, "experimenter tools" are users, "aggregates" are resource providers, "identify providers and slice authority" approximately map to the trust providers.

Actions: Actions are applied to resources to fulfill users' reservation requests. Each resource provider must, based on resource type and policy, define what available actions for the resource are exposed for user requests. For example, network resources may expose isolated or shared access at different scopes (e.g., optical circuit, layer 2, or layer 3), standard or customized packet switching, and different levels of quality of service; compute resources may expose physical or virtual machines with different operating system and storage; other resources may include special instruments, sensors, software services (e.g., firewalls, load balancers, encryption services) with respective customizable actions.

Interfaces: Per our definition, federations are created based on "human" interests. Nevertheless, their intentions and actions are manifested in policies expressed as software constructs that must be communicated among the federated entities via agreed interfaces.

Agreeing on the choice of interfaces across resources is just as challenging as agreeing on the choice of common human languages in international settings. The eventual selection will have to be the result of natural selection, i.e., the ones that become most popularly used would be the ones that persist. This is beyond the scope of this paper. Just as an example, we introduce the GENI Resource Specification (Rspec) language [4,5] and the Network Service Interface [6] as two currently widely used interfaces. The former is used by the GENI infrastructure to communicate compute and network (both SDN and non-SDN) resources. The latter is used by global

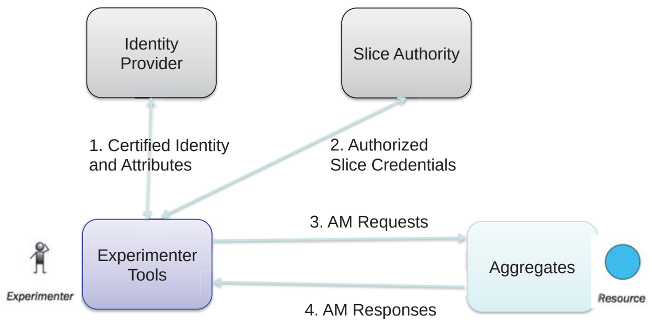


Figure 1. Key entities and actions in GENI federation [2].

optical research networks as a standard for provisioning cross-domain optical network lightpaths and and L2 circuits.

For SDN, the interfaces have so far been diverse and specific to respective implementations. Just to name a few, open source SDN implementations such as the Floodlight controller project, the OpenDayLight controller project, and the OpenVswitch project have all defined interfaces for external applications to request the setup of static network paths based on specific attributes. In order to federate among SDNs of different types, obviously additional "translation" among them must be done. Various communities have made attempts to suggest such a translating interface. In the OpenStack [7] and Cloudstack [8] open source cloud computing projects, networking plugin architectures are defined to provide a common interface for different SDN providers to offer the same sets of logical service for the cloud. Similarly, the Open Networking Foundation (ONF) has created a Northbound API working group [9], and the Object Management Group (OMG) has created a SDN working group [10] to attempt the creation of a standard interface for SDN. Recent papers also presented example interfacing of SDN with the legacy BGP-based IP network based on a HTTP RESTful interface of the Floodlight controller and a BGP router [11].

Another recent community effort has begun to focus on the Software Defined Exchange (SDX) concept as the means of interfacing different SDN networks, and to also incorporate other resources in SDIs such as compute, storage, and so on. The design of SDX necessarily must address the "interface" as well as the "body", such that it has the interfaces as well as the resources needed to orchestrate the interfacing action of different SDIs of different authorities. Later in the paper we describe a suggested SDI federation approach based on a proposed SDX framework that incrementally incorporates interfaces such as GENI Rspec, NSI, and other interfaces for established resources (such as Internet2 AL2S service [11]2]) and new resources (such as emerging policy-based SDN services [13]) based on participating organizations' needs and policies.

III. What Resources To Federate

In an SDI, the typical resources are physical and virtual compute servers, storage, and network connectivity. When cyber-physical systems are involved, sensors and actuators are also considered. Increasing emphasis is placed on software services as resources; e.g., software firewalls, encryption services, transcoding agents, and data transfer services. These types of software services are usually deployed on compute servers in the applications' data paths, often conveniently at the network gateway. Emerging cloud-computing systems such as OpenStack are looking into insertion of such services in arbitrary points in the network based on SDN [14].

The majority of resources in a federated SDI environment will be virtualized/shared. Full access to physical resources by a single user would be the rare use case and can be considered a special case equivalent to a virtualized resource request at full capacity. As an example, consider an organization operating an SDI with various resources shared by different application providers and users: 1) a point-to-point data request by a remote user to the organization's big data archive: *composed of* a data retrieval service, an application-specific data processing service, a firewall service, and high throughput data transfer agents on both ends – all over a user requested cross-domain SDN path; 2) a point-to-multi-point multicast live video streaming service: *composed of* a streaming server, quality of service monitors, and transcoder agents as needed at distributed points in a multicast tree spanning multiple federated domains; 3) a novel "connected vehicle" mobile connectivity proxy: *composed of* a cluster of data proxies distributed along the trajectory of moving users to assist reliable communication with remote data centers for real-time services, and 4) a monitored highly available smart grid connection: composed of sensor data handler service at a smart grid control center, distributed monitoring service, and physical sensors across vast geographic areas spanning more than one federated domains.

Analyzing the example above, a number of observations can be made about the resources involved:

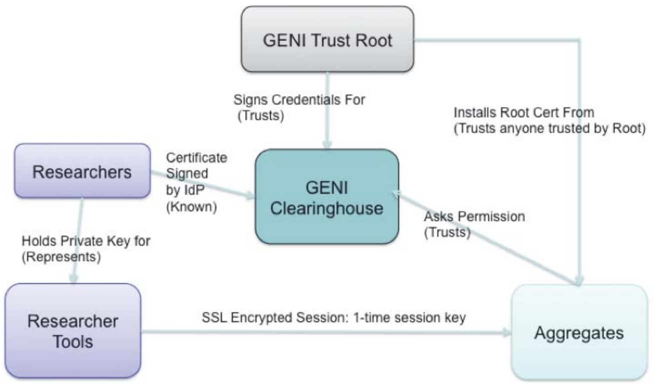
1) Software services are instantiated as part of the application end-to-end data path.
2) Software services run on virtualized compute host.
3) Resources can be persistent or launched each time a service request is initiated.
4) Resources can be in different federated domains.
5) All federated domains need not provide the same range of services.
6) An application can operate as long as any traversed federated domain meets its required resources.

The example above is in fact quite representative of the majority of Internet applications today. As a result, a sensible approach towards identifying a federation framework would be starting with a simple framework that can reserve compute and network resources effectively, and incrementally incorporate interfaces and reservation methods for new types of resources as they emerge. Recognizing that the primary focus of federation should be the policy and preferences of people and organization, the underlying framework, i.e., the resource API, should be kept as simple as possible, and as expressive as possible. This allows SDI operators to easily register new resources as well as integrate existing identity and resource management systems into the framework by adding simple API wrappers. In fact, in the GENI architecture, "aggregates" serve as such a common API wrapper that can be used to contain and allocate ANY resource (network connectivity, bandwidth, VLAN, computation, storage, etc.) using a common interface and common foundations for trust and identity.

Below we propose such a framework based on the GENI Rspec framework for compute and basic/SDN network reservation, integrated with the NSI API for wide area light path reservation, and the Internet2 AL2S API for L2 circuit reservation.

IV. Federation Framework

Figure 2 is an illustration of the current GENI federation architecture [2]. The "clearinghouse" represents the set of authentication and authorization services that provides users with the necessary credentials to request service from individual resource aggregates. Given the clearinghouse-approved credentials, the user can send resource requests with resource specific attributes to each resource aggregate. While each resource aggregate can subsequently approve/deny requests given the specific attributes, resource availability and user types, the clearinghouse can be seen as the anchor of all resources within a federation. Based on this concept, we propose a federation framework as illustrated in Figure 3. By allowing clearinghouses to communicate with each other, mutually share information and grant access of the available resources (based on customizable pairwise agreements, of course), federations can be infinitely extensible. This model is, in fact, not far from today's Internet architecture, where such clearinghouses are synonymous with the Internet Service Providers (ISP) and the mutual agreements are based on ISP peering relationships. In a federated SDI, a richer range of resources is being "peered". The resource instantiation, however, is not limited to a single federation. Instead, once a request is certified, applications have the flexibility to communicate directly with resource aggregates, enabling a more scalable and flexible implementation.


GENI Trust Root
Signs Credentials For (Trusts)
Installs Root Cert From (Trusts anyone trusted by Root)
Researchers
Certificate Signed by IdP (Known)
GENI Clearinghouse
Asks Permission (Trusts)
Holds Private Key for (Represents)
Researcher Tools
SSL Encrypted Session: 1-time session key
Aggregates
Figure 2. Current GENI Federation Architecture [2].

A. The Framework

The very factors that enable the creation of a single federation have parallels to those required in developing cross-federation

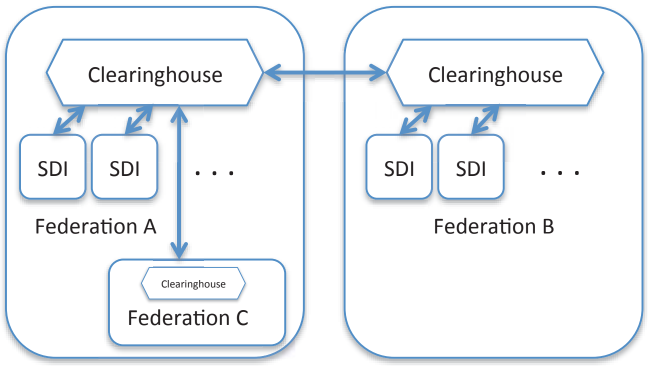


Figure 3. An extensible federation framework based on clearinghouse peering.

cooperation, though perhaps at a larger scale with additional parameters.

Each federation has its own set of *trust roots*, which represent a set of identities (typically certificate authorities or delegated authorities) whose signed statements of identity and attribute are accepted by members of the federation. Such trust roots allow for both federation-wide *authentication* (using SSL certificate verification, e.g.) and *authorization* by matching federation-authorized credentials against federation and aggregate-local policies. The representation of credentials have been standardized in these major categories:

- *Identity Credentials*: Typically X.509 certificates signed by a Member Authority
- *Role-based access-control (RBAC) credentials*: typically in SFA format [15]
- *Attribute-based access-control (ABAC) credentials*: typically in ABAC format [16]

The APIs are the basis for communicating with authorities for signed credentials. The GENI Federation API v2 [2] and the GENI Aggregate Manager API v2/v3 [3,4] are becoming de facto standard for negotiations between tools and aggregates for the allocation and management of resources.

A federation, be it individual or a federation-of-federations, typically requires a set of policies that control the issuing of credentials, the limits on resource allocations based on different kinds of requests and the attributes of the requesting user. These policies must be agreed upon, encoded and disseminated to all federate authorities and aggregates.

Finally, accountability procedures (e.g. monitoring, alerting, shutdown, forensics, credential revocation) must be extended and agreed to by all federating domains in order to assure that a cross-domain topology will be safe to use (from the user's perspective) and to contribute resources to (from the resource provider's perspective). These mechanisms may range from integration of monitoring and response automated processes to human processes to coordinate across operations centers.

B. Federating Wide Area Networks – Network Service Interface (NSI)

The Network Services Interface (NSI) initiative developed initially within a worldwide community of international research and education network providers. Over ten years ago, this community decided to design, implement and operate a Global Lambda Integrated Facility (GLIF), based on light paths over trans-oceanic and terrestrial optical fiber and DWDM edge switches hosted at GLIF Open Lambda Exchanges (GOLEs) [17] (Figure 4). The GLIF infrastructure is not a network, but rather a distributed facility within which it is possible to design and implement international customized networks, including those required by data intensive science. The GOLEs are exchange facility for GLIF network participants. These GOLEs not only exchange international network traffic but also traffic from national and regional research and education networks. Although initially the GLIF project was focused on dynamic light path switching, it was extended to include L2 switching and dynamic control of L3 based paths.

As GOLEs were implemented, almost all adopted a different control framework for resource management and control, e.g., DRAC, Autobahn, Argia, OSCARS, G-Lambda, and many others. The development of the GLIF model required a mechanism to enable a common interface to interact with these control frameworks. Consequently, the GLIF community formed a partnership with the Open Grid Forum (OGF), a standards organization, to develop a defined standard for such interfaces.

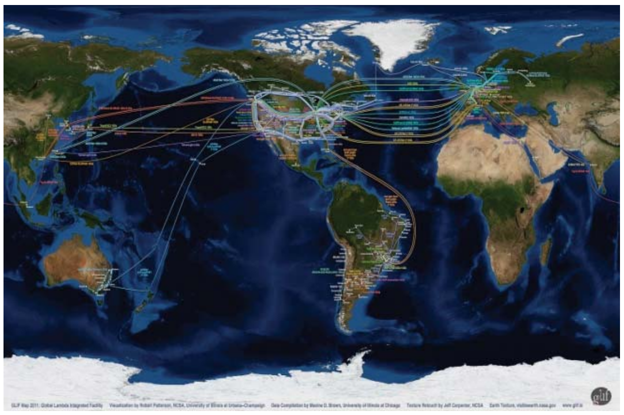


Figure 4: Global Lambda Integrated Facility (GLIF)

This initiative created a working group that addresses a range of architectural issues under the Network Services Framework (NSF) as described in OGF GWD-R-P "Network Service Framework v2.0" [6]. Within this framework a suite of protocols are being defined. The NSF approach assumes that resources and capabilities are presented externally through a set of defined Network Services. The NSF presents a unified model for how various processes interact with these services. These network services include those for creating connections (Connection Service), sharing topologies (Topology Service) and performing other services required by a federation of software agents (Discovery Service).

The discussion here focuses on NSI, which is an intermediate process between a software agent that requests a network service and the software agent that fulfills that network service. Specifically, it describes the NSI Connection Service (CS) 2.0 protocol, which enables the reservation, creation, management and removal of connections, under appropriate policy based authentication and authorization processes.

NSI has been designed to allow for the creation of network paths (termed "connections" in the NSI architecture) that can cross multiple network domains operated by separate network providers [18], effectively creating a federation. This approach is a major departure from common practice, which closely specifies service plane attributes within individual data plane implementations, and statically maps attributes to control plane protocols.

The old model works for single domain networks with minimal service interactions with other domains (e.g., limited to L3 peering). In contrast, NSI was designed as an API to support multi-domain services over many provider network and facility boundaries and many different implementations of data plane technology. This API can be used by organizations,

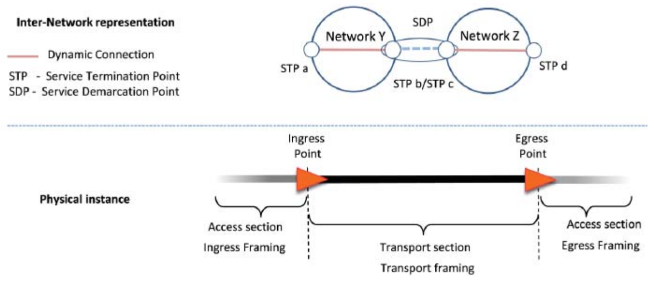


Figure 5. STPs interconnecting at a SDP.

applications, edge processes, and even individual users to invoke network services. NSI is agnostic to any specific technology. Consequently, NSI provides for the concept of a "connection," as an abstracted entity decoupled from any specific physical technology or configuration. The NSI architecture provides for a messaging function that includes a customizable schema that can be used to describe service-specific attributes, including constraints. NSI processes determine how these attributes can be mapped to resources in the various domains across which the paths are implemented. The NSI architecture incorporates the concept of a pathfinder function to determine the most optimal path that matches the requirement attributes of the request, across all reachable domains, regardless of technologies used in those domains. The architecture incorporates a two-phase commit function and options for advanced reservations.

The NSI architecture describes Service Termination Point (STP) objects, which are core components that are used by a connection request to determine key attributes of the connection, namely, its source, destination and intermediate points. An STP is identified with a network ID, a local ID and a label. Two or more STPs owned by different network domains interconnect at a Service Demarcation Point (SDP), which is a virtual point, not a physical component. Connections are implemented across domains by concatenating connection segments by selecting STPs so that the egress STP of one interlinks with the ingress STP of another as shown in Figure 5.

The NSI architecture defines an explicit set of messages and describes the state machines that are the foundation of the service. For over five years NSI has been showcased through successful demonstrations at multiple major national and international conferences, including the Supercomputing (SC) conferences. Almost all of these recent exhibitions have demonstrated the NSI utility for implementing L2 paths across multiple domains. However, dynamic L1 provisioning has also been demonstrated at multiple conferences. Below is an image of a dynamic international L2 path demonstration with participating sites around the world at SC13 in Denver in November 2013. Each of the points depicted, almost all are exchange facilities in different countries, is interconnected with individually controlled L2 paths using NSI.

Currently, NSI is being placed into production within a number of national research and education networks and at GLIF GOLEs. Several data intensive science research communities are also considering or experimenting with NSI implementations [19].

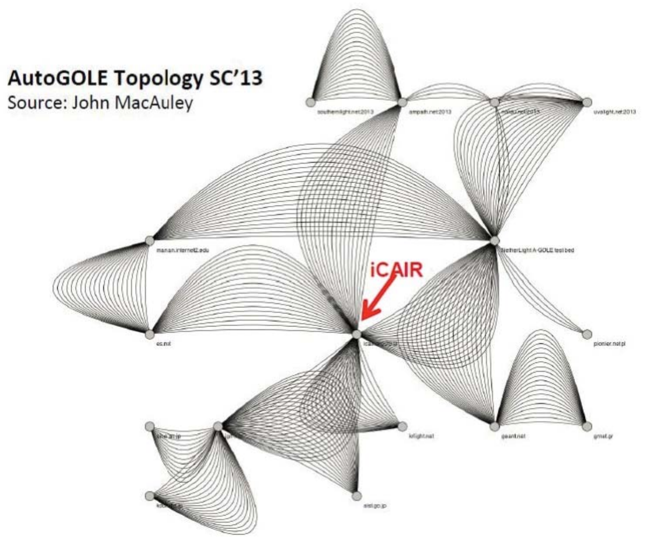


Figure 6. Demonstrated global AutoGOLE topology at SC'13.

C. Federating SDNs

To date the majority of SDN deployments are research demonstrations, of which most have been based on a single network domain by a single controller. On the one hand, the prime motivation of SDN is to explore the extent of controllability achievable by a single SDN controller. On the other hand, it is an inevitable reality that the Internet will not be made of a single SDN but a myriad of SDNs operated by different authorities. Before then, even interfacing a SDN with today's non-SDN Internet poses substantial design challenges. In [20], researchers demonstrated one such possibility of interfacing an OpenFlow SDN with a BGP router to demonstrate SDN-IP interfacing.

Interfacing two networks can be viewed as a basic form of federation. In today's Internet, autonomous systems (ASs) interface with each other based on exposed topology info exchanged in the standard BGP routing protocol. With SDN, there is currently no agreed standard to exchange such information. However, once such a standard takes shape, it will obviously allow much richer information to be exchanged among different SDN domains. GENI offers a mechanism for researchers to control traffic switching in different network domains via FlowVisor [21]. A FlowVisor at each network aggregate delegates control of the local Ethernet switches' switching to one designated OpenFlow controller, thereby allowing the controller to have visibility to the topology and control of the traffic switching at all aggregates. This demonstrates a basic form of federation at the switch-to-controller interface via the OpenFlow protocol. Going forward, the industry is also looking at network description languages at a higher layer of abstraction – namely the policy layer (see the Cisco OpFlex project [13]) – to prescribe desired network composition via logical policies that can be implemented by different underlying technologies.

D. Compute and Storage

Open source cloud computing software stack projects have brought together vibrant communities looking into ways to automate, customize, and scale data center implementations in open source software. OpenStack [7] and CloudStack [8] are two such projects among the largest ones. In both projects, plugin architectures are defined to allow instantiation and management of network, storage, and security resources to be utilized by virtual compute hosts in a data center. These stack management software provides APIs for users to instantiate, monitor, migrate, and terminate resources within, making them a form of federated infrastructure that can be readily integrated into a federated SDI environment.

E. Software Defined Exchange (SDX) for Federation

The notion of software defined exchange (SDX) was initiated in the recent work of [22] discussing how SDN can be used to implement the Internet Exchange Point (IXP) facilities to provide existing and new services. Since then, the discussion has expanded its scope as the community brainstorms about a wider range of possible scope and realization of SDXs. In this paper, we see SDX as a means of interfacing SDIs of multiple distinct authorities. Instead of just sharing network connectivity, SDXs facilitate sharing of any willingly exposed resources of each participating domains. We believe an SDX should be a logical entity that can be realized in one or across multiple facilities:

1) In one facility: In the same model of today's IXP, the SDX operator facilitates pass through of resource sharing information among interconnecting organizations;
2) Multiple facilities: As an end-to-end path in a federated SDI environment spans multiple authority domains, a single IXP is insufficient in orchestrating all resources; moreover, there can be resources that need to be instantiated in the source, intermediate, or destination domains rather than a single IXP facility.

Regardless of either option, the SDX logical abstraction will be the same, closely mirroring the proposed federation architecture.

In the 3rd quarter of 2013, iCAIR established a prototype Software Defined Network Exchange (SDX) at the StarLight International/National Communications Facility, as part of the GENI program [23]. This SDX was successfully demonstrated interoperating with the Georgia Tech prototype SDX in Atlanta at the GENI Engineering Conference (GEC19) and with a prototype SDX in Amsterdam for the TERENA 2014 conference in Dublin in May 2014. iCAIR currently is extending NSI capabilities by integrating its functionality, including federation options, with SDN/OpenFlow techniques.

It is our belief that the most important challenge is to engage operators of the production infrastructure today to guide the incremental integration of existing identity management and infrastructure operation tasks into a federated SDI API as a simple lightweight wrapper. As of right now, NSI has demonstrated success in integrating NSI with GENI, while Internet2 has also been integrating its advanced layer 2 service (AL2S) to the GENI Rspec API. We are actively working in this direction with our ongoing research projects.

V. Summary

Software defined infrastructure (SDI) offers a new way of customizing deployment of applications across Internet. Federated SDI addresses the need of most Internet applications to be conducted across end-to-end paths crossing multiple authority domains. The key challenges reside in the policy and needs perceived by "human" stakeholders, while they can be translated into simple software constructs to be handled by different resources. A simple, extensible framework based on GENI Rspec is proposed as a first step to tackle the federated SDI, or SDX, problem.

VI. Summary

The authors are supported, respectively, by NSF grants OCI-1245936, OCI-0962997 and the GENI project. GENI is funded by the US National Science Foundation under cooperative agreement CNS-0737890. Any opinions, findings, conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect the views of the National Science Foundation.

References

[1] Global Environment for Network Innovations, http://www.geni.net.
[2] GENI Architecture Team, "GENI Software Architecture", http://groups.geni.net/geni/raw-attachment/wiki/GeniArchitectTeam/GENI%20Software%20Architecture%20v1.0.pdf.
[3] InCommon Federation, http://www.incommonfederation.org/.
[4] GENI AM API v2, groups.geni.net/geni/wiki/GAPI_AM_API_V2.
[5] GENI AM API v3, groups.geni.net/geni/wiki/GAPI_AM_API_V3.
[6] Open Grid Forum, "Network Service Framework v2.0", http://redmine.ogf.org/projects/nsi-wg .
[7] OpenStack, http://www.openstack.org.
[8] CloudStack, http://cloudstack.apache.org.
[9] Open Networking Foundation, "The Northbound Interfaces", https://www.opennetworking.org/working-groups/northbound-interfaces.
[10] OMG Software Defined Networking Group, http://sdn.omg.org/.
[11] P. Lin, et al., "Seamless Interworking of SDN and IP", SIGCOMM 2013.
[12] Internet2 Advanced Layer 2 Service, www.internet2.edu/products-services/advanced-networking/.
[13] Cisco, "OpFlex – An Open Source Approach", http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731304.html.
[14] OpenStack Sevice Insertion, https://wiki.openstack.org/wiki/Neutron/ServiceInsertion.
[15] L. Peterson, et al., "Slice-based Facility Architecture", http://www.cs.princeton.edu/ ~llp/geniwrapper.pdf.
[16] NIST, Attribute Based Access Control, csrc.nist.gov/projects/abac/.
[17] Global Lambda Integrated Facility (GLIF), http://www.glif.is.
[18] G. Roberts, et al., NSI Connection Service V2.0, Open Grid Forum, GWD-R-P, NSI-WG 2013.
[19] J. Mambretti, et al., "OpenFlow Services for Science: An International Experimental Research Network Demonstrating Multi-Domain Automatic Network Topology Discovery, Direct Dynamic Path Provisioning Using Edge Signaling and Control, Integration With Multipathing Using MPTCP," in SC12, November, 2012.
[20] J. Mambretti, T. DeFanti, M. Brown, StarLight: Next-Generation Communication Services, Exchanges, and Global Facilities, Advances in Computers. 01/2010; 80:191-207.
[21] R. Sherwood, et al., "Can the Production Network Be the Testbed," in Proceedings of USENIX OSDI'10, pp. 1-6, 2010.
[22] A. Gupta, et al., "SDX: A Software Defined Internet Exchange", in Proceedings of ONS 2013, 2013.
[23] IGENI, http://groups.geni.net/geni/wiki/IGENI.

A Measurement Architecture for Software Defined Exchanges

M. Zink
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
zink@ecs.umass.edu

***Abstract*—Prototype deployments of Software Defined Exchanges (SDX) have recently come into existence as a platform for Future Internet architecture to eliminate the need for core routing technology used in today's Internet. In this paper, we motivate the need for an adequate measurement architecture for such SDXes to be able to evaluate their performance and inform further development. We present the major requirements for this architecture, introduce the idea of SDX and its first prototypes, and give an overview on a SDX measurement experiment we recently conducted.**

I. Introduction

Software Defined Networking (SDN) which, in contrast to traditional IP-based routing, decouples the data and the control plane is seen as a promising approach to enable new functionalities in the future Internet. While single-domain SDNs have been around for a few years (e.g., data centers, research networks [7] [22], and WANs of organizations [11]), inter-domain deployments that involve SDN implementations for inter-AS routing using BGP and MPLS have only come into existence in the recent past [13].

Recently, the research community is proposing the introduction of so-called Software Defined Exchanges (SDXes) [10] [21], which can be seen as the SDN equivalent to an Internet Exchange Point (IXP). The basic idea of an SDX is to connect several domains, allow traffic exchange and provide a platform for implementation of new policies through third-party control in future Internet architectures. Since SDN is radically different from today's Internet technology it has to be further investigated to understand what functionalities an SDX must provide. Some of the aspects that have to be further investigated are flow management between autonomous systems (similar to BGF, OSPF, etc. in today's Internet) and higher-level ones like peering policies, peak-usage scheduling and route-based prioritization [6]. An SDX should also incorporate SDN advantages such as complete resource virtualization, real-time traffic analysis, centralized control, plug-and-play hardware integration, security and third-party control services.

In addition to new functionalities that have to be provided by SDXes, it is also important to observe the performance of these new exchanges. As with any new technology, SDXes will be thoroughly studied by the research community before commercial versions might be deployed in the future Internet. In this paper, we present a measurement and monitoring architecture that is tailored for the performance analysis of SDXes.

The measurement architecture for SDXes has to fulfill the following requirements:

- **Scalability.** A measurement approach for SDXes has to be scalable since there is the potential of many flows crossing an SDX. Not only the shear amount of flows but also the detail of information that should be measured or monitored can put significant load on such a system. We have already seen such challenges in measurement and monitoring approaches for today's Internet.
- **Non-intrusive and Non-interfering.** The measurement and monitoring architecture has to be designed such that ongoing measurement activities do not impact the performance of the actual SDX processes (e.g., data forwarding).
- **Ease of Use.** The architecture should result in tools that will make it straight forward for researchers and developers to observe and analyze the performance of an SDX. Researchers should also be able to share their results in an easy and straight forward manner.
- **Calibration.** The quality of a measurement very much depends on the accuracy and performance of the measurement mechanisms and tools applied. To allow users of the architecture to determine the accuracy of a measurement the architecture will provide calibration mechanisms. E.g., the architecture will provide mechanisms that will allow the injection of specific traffic into the SDX.
- **From SDX to SDI.** While the main focus on SDXes is currently on the interconnection of different, independent SDX domains, we believe that the second generations of SDXes will also offer access to virtualized computation and storage. Therefore, it is our goal to develop an architecture that is not only capable of monitoring the network performance of an SDX but also the performance of its compute and storage capabilities.
- **Legacy.** Measurement and monitoring have been an essential component of the Internet and distributed systems since their inception. Our architecture will be based on proven mechanisms and approaches. We will build this architecture based on the GENI Measurement and Instrumentation Infrastructure (GIMI), which we built for the GENI project.
- **Monitoring and Measuring Control versus Data Plane.** Compared to the traditional Internet which only

has one transport plane there exists a control and a data plane in SDNs. Therefore, it is important that a measurement and monitoring infrastructure for SDXes offers the ability to monitor both planes and is able to identify potential interdependencies between the two. For example, a bottleneck in the control plane can easily lead to performance impairments in the data plane.

While a production deployment of SDXes in New Zealand has proven that SDX is a pragmatic approach for traffic exchange, our proposed architecture is complementary since it will allow researchers and developers to analyze the performance of this infrastructure. We will use SDXes that have been recently set up as part of the GENI project [8] to evaluate our measurement and monitoring architecture. In the long-term, it is our goal to make this architecture available to the research community.

The remainder of this paper is outlined as follows. In Section II, related work in the area of Software Defined Exchanges is presented. Section IV gives an overview on the measurement architecture for SDXes, and Section V introduces a prototype of such a measurement architecture. The paper finishes with conclusions and a look at future work in Section VI.

II. Related Work

OpenFlow [15] has been successfully deployed in several production data centers today. B4 [11] by Google is one of the first large scale deployments of a Wide Area Network that uses OpenFlow. Following this, there is some work that discusses network virtualization using Open APIs defined for Software Defined Networking [19]. In [2], Podleski et al. have discussed the feasibility of Software Defined Networking with multi-domain architectures using both slice-based and connection oriented approaches. The connection-oriented approach places emphasis on using namespace to enable the use of connection-related applications such as load-balancing, traffic monitoring and packet inspection while the slice-oriented approach talks about how slices maybe provisioned for different service providers. Another work talks about implementing a backward compatible algorithm to outsource control logic for BGP routing using SDNs. The authors run real traffic data on a simulated network to evaluate their approach [12].

During the last couple of years there has been a substantial amount of work that discusses the feasibility of deploying Software Defined Networks within the Internet. In [16], Nunes et al. discuss programmable networks in detail. They present a survey of several Software Defined Networks beginning with Ethane [9] to OpenFlow and present-day SDN applications. The authors of OSHI [18] describe the use of OpenFlow for SDN based IP forwarding and routing and emulate such a system on OFELIA [22], which is also an SDN-enabled research testbed. The authors here use Mininet and examine the effectiveness of different monitoring methods.

A large scale deployment for Software Defined Exchange was setup by Gupta et al. [10]. Their paper contains details of their SDX deployment at Southern Crossroads (SoX), which is a part of the network we use to develop and test our measurement architecture on. Here, they explain how their SDX can be used to implement different peering policies, efficient DNS-based load balancing and middle box traffic steering. Cardigan is one of the first OpenFlow SDX networking environments that has been deployed in a production setting [21]. The paper contains details about a real deployment of a "distributed routing fabric" between Research and Education Advanced Network of New Zealand (REANNZ) to the Wellington Internet Exchange (WIX). RouteFlow [20], an extension of Cardigan is an SDX deployment that carries real Internet traffic. However, their production deployment limits their ability to conduct performance characterisation.

While all of these papers have presented the advantages of Software Defined Networking in several ways and some also present production deployments of Software Defined Exchange, we have not seen any work that presents the performance analysis of actual applications. In this paper, we present a measurement architecture which is targeted to support experimenters in their research on SDXes. We demonstrate a prototype of this architecture by using an SDX-enabled GENI testbed and evaluate the performance of a short-term weather prediction application to give some insight into the capability of such a testbed for Future Internet.

III. SDX Prototypes

In this section, we will describe an SDX in more detail and present a prototype implementation.

A. Definition

Before we present the implementation of a prototype SDX, we first give our definition of its functionality and purpose. SDX is a relatively new topic and there are several versions of it currently discussed among the research community. Therefore, we believe that it is important that we first describe our understanding of an SDX by describing its functionality. The motivation for SDX is mainly driven by the fact that more and more SDN domains have been instantiated in recent past. Examples for such domains are Internet 2's Advanced Layer 2 Service (AL2S) and Google's B4. While SDNs have caused significant traction in academia, the latter example shows that this is a topic that is also of high interest to industry.

With the advent of several SDN domains the need for *exchange points* between such domains becomes imminent. Thus, first and foremost and SDX is an interconnect between two or more SDN domains. One example that emphasizes that need is NFS' Global Environment for Network Innovation (GENI) initiative, which is currently deploying a multi-domain SDN federation at $\sim$50 campuses in the US. Since GENI is build on the basis of a federation, there is a need for SDN infrastructure to span multiple operating domains. This is driven by the fact the GENI infrastructure is owned and operate by the host (campus) institutions and that experiments and services need to exert control across institutional borders in a consistent and controlled way. This need also exists at a larger scale where GENI federates with other national and international peer infrastructures. To address this need there are currently three active multi-domain SDN GENI projects [4], [1] that are in the process of implementing SDX prototypes. In Section III-B, we will present one of those prototypes in more detail.

It is important to mention that our vision of an SDX goes beyond the idea of it being an interconnect between two or more SDN domains. Inspired by the GENI architecture, we believe that software defined computation (i.e., cloud computing) and software defined storage will significantly improve the capabilities of such SDXes[1]. In Section V, we will give an example that illustrated the benefits of such an SDX.

B. SDX Prototype

Figure 1 gives a schematic overview of one of the three SDX prototypes that are currently developed within the GENI initiative. This SDX prototype is instantiated at the StarLight facility in Chicago. In this specific example, the SDX prototype connects three independent domains, Internet2 AL2S, ESNet, and ORNL, respectively. Both AL2S and ESNet are SDN domains, while ORNL is a plain layer 2 connection to another SDX prototype at SoX in Atlanta. In addition to the networking equipment shown in Figure 1 StartLight also houses a GENI rack and offers direct layer 2 connectivity to another one at Northwestern University. Thus, this SDX prototype does also offer storage and computation resources. In Section V, we will demonstrate how the latter resources can support data intensive applications.

The StartLight OpenFlow switch as well as the Northwestern and StarLight GENI racks interact with the GENI Aggregate Manager, which allows GENI researchers to reserve networking, computation, and storage resources. In essence, this SDX prototype allows researchers to obtain a slice of this SDX to perform their own multi-domain SDN experiments by reserving resources using standard GENI tools (e.g, with Omni as part of the GENI control framework or the GENI portal).

While SDXes are currently still in the very early development stages we believe that it is important to develop a measurement and monitoring infrastructure for these new exchanges in parallel. This will allow researchers to characterize the performance of SDXes which might lead to the early on identification of potential design issues. This, in turn, will support the rapid development of more sophisticated and advances second generation SDXes.

IV. Measurement Architecture for Software Defined Exchanges

In the following, we describe our proposed measurement architecture for SDX in more detail. We start by identifying, what we believe are, the most important requirements for such an architecture.

[1]In some cases this has been also described as Software Defined Infrastructure (SDI) but we will stick with the term SDX throughout the paper.

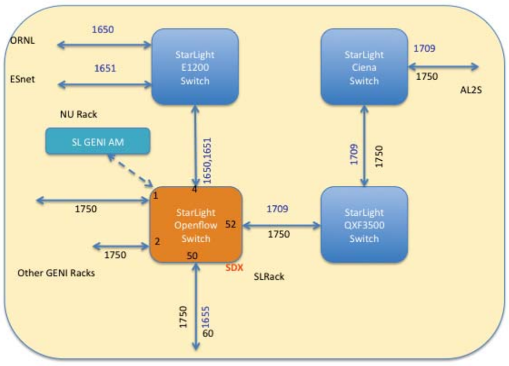


Fig. 1: SDX prototype implementation at StarLight

A. Scalability

Scalability is a major concern for an SDX measurement infrastructure. One can imagine that several 10,000s of flows will be handled by a single SDX. It is yet unclear if it is even possible to monitor each individual flow. Even the simple task of obtaining flow statistics might overwhelm either the switch or the controller of both.

While not all the flows traversing an SDX can potentially be measured an experimenter should have the option to measure a subset of flows. The measurement architecture has to give the experimenter the ability to select a sub-group of the flows traversing an SDX.

An open issue is the question if an experimenter can actually observe flows that are not part of his or her slice. The approach we would like to propose is that the measurement architecture should definitively support the functionality to observes flows that do not belong to an experimenters slice. An additional feature is required that will enable or disable this functionality based on policy. Each individual SDX can then have its own set of policies to decide if and for which individual experimenter this feature should be enabled.

B. Non-intrusive and Non-Interfering

One of the major challenges of measurement is to avoid that the actual measurement activity impacts the outcome of an experiment. For example, a measurement process on a local computer could consume a significant amount of processing power which could impact the performance of the application that is analyzed in the experiment. Similarly, measurement data that is transmitted from measurement points back to a central aggregation point can impact the network performance that is analyzed in an experiment. Since most SDX related measurements will most likely focus on network performance analysis it is important to design the architecture with the premiss to avoid interference of the measurement data transport as much as possible. The benefit of the current GENI testbed is the existence of a "global" control plane, which in the GENI case is the regular Internet. In this case, measurement data will only be transmitted on the control plane and not the data plane and, thus, not interfere with the actual measurement. The experiment described in Section V makes use of this infrastructure.

But it cannot always be assumed that such an infrastructure exists. Especially, if one envisions that SDXes might move from prototype, testbed environments into actual production networks. Therefore, alternative approaches are needed to prevent interference. Fortunately, SDN allows the separation of measurement and experiment data. For example, in the case of OpenFlow the measurement data can be transported via a different flow. With the introduction of Quality-of-Service in OpenFlow 1.3 [3] it will be possible to isolate flows and, thus, avoid interference completely – even if only one plane is available.

C. *Ease of Use*

To increase the likelihood that such an architecture and the tools that are based on this architecture are adopted by the research and experimenter community it is important that using measurement tools is as easy as possible. Through our experience with the development of a measurement and instrumentation infrastructure for GENI (GIMI) we have developed tools that allow the execution and observation of measurements in a straightforward manner, that also allows experimenters to easily repeat experiments and to have third parties also execute such experiments. We will re-use the tools developed within the GIMI project where appropriate and also apply our experience gained during the GIMI project for the development of new tools.

D. *Calibration*

Calibration is an essential component of each measurement infrastructure since it ensures the quality of the measurement data and the analysis that is based on these data. Thus, experimenters must be provided with means that allows them to perform calibration. To support the ease of use approach mentioned above, the architecture will provide calibration tools that can be used by experiments. In addition, the architecture will be designed such that experimenters can be easily create their custom-designed calibration methods. E.g., the architecture will provide mechanisms that will allow the injection of specific traffic into the SDX. Since the characteristics of this traffic are well known an experimenter can verify if a measurement produces accurate data.

E. *From SDX to SDI*

As already mentioned in Section III our vision of an SDX goes beyond the basic exchange point for SDN domains. We, much rather, believe that SDXes will also offer a significant amount of compute and storage resources. Thus, a measurement architecture must also include means to measure compute and storage performance. Fortunately, most of these tools already exist and have been extensively used for measurements in the GENI testbed. In Section V, we demonstrate how these tools can be applied in and SDX measurement.

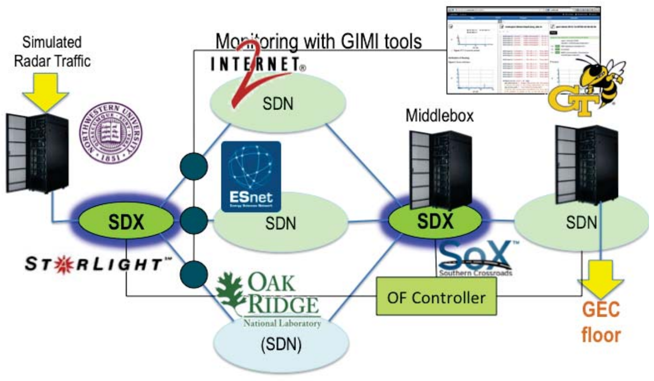


Fig. 2: SDX prototype implementation at StarLight

F. *Legacy*

We will base the SDX measurement architecture on existing architectures and tools that have been in use for network and distributed systems measurement in the past. A more detailed example of the use of these legacy tools is given in Section V.

G. *Control versus Data Plane*

SDNs consist of a control and data plane which is therefore inherent in an SDX. This requires a measurement architecture that allows the observation of data and control plane. Only a holistic analysis of an experiment can reveal certain artifacts that cannot be revealed by measuring either the data or the control plane. E.g., congestion on the link that connects the SDN switch with the controller can influence the performance of one or several flows in the data plane. Thus, an experiment must be able to measure both and in Section V we will describe one possible realization of this requirement based on our existing measurement infrastructure.

V. Measurement Prototype

In this section, we give a description of our measurement prototype by illustrating how it was used in an actual experiment that included two SDX prototypes. We first describe the resources used for this experiment and then give a brief overview on the experiment itself. This section concludes with a description of our measurement infrastructure and how it meets or will address some of the requirements listed in Section IV.

A. *Infrastructure*

For this experiment we used the SDX prototype at StarLight and the AL2S and ESNet SDN domains and the ORNL layer 2 connection as described in Section III-B. In addition, we also used the SDX prototype at SoX. This SDX is similar to the one at StarLight and also offers compute and storage resources through a GENI rack. A detailed overview of the resources used for this experiment is shown in Figure 2.

1) Local Rack VMs: Nowcast application - It consists of a network of 4 radars at each location where each radar sends measured atmospheric data to a central processing location. The application we run, called Nowcasting, is highly time-sensitive and is used for short-term weather prediction. The VMs that run on the Northwestern rack have previously collected radar data stored on them, which is replayed just as if it would come from a real radar.

2) SDXs: The SDX OpenFlow switches are at: (a) Southern Crossroads (SoX) - This is located in Atlanta, Georgia and is configured with a programmable FOAM/Flowvisor [10]. (b) StarLight - This Exchange network is located in Chicago and is the first provider to implement both national and international communications exchange to provide better management and control over provisioning resources within an Exchange network [14].

3) Domains: The network domains are: (a) Oakridge National Lab (ORNL) - This is a non-SDN domains that connects the Northwestern rack to the SoX rack. (b) Energy Sciences network (ESnet) - A large scale national network that offers an alternative path between the two racks. (c) Internet 2 (I2) - We use the Advanced Layer 2 Service link (AL2S) offered by Internet 2, an SDN Domain, which provides us with a VLAN between the Starlight Pronto switch in Chicago and the SoX NEC Switch in Atlanta (see Figure 1). This offers a third alternative path between the two SDXes.

All three domains provide 100Gbps bandwidth.

B. Experiment

1) Nowcasting: For the Nowcasting application, we do not use actual radars but use virtual machines in the Northwestern GENI rack to emulate radars. We load actual radar data that we previously collected from four real radars in Oklahoma on four virtual machines located in the Northwestern University GENI rack (see Figure 2). The Nowcasting algorithm runs on a bare metal server in the SoX rack. An Apache server is installed in a Xen virtual machine that runs on the Georgia Tech GENI rack. The Nowcasting algorithm processes the received radar data to generate Nowcast images of weather data which are transferred to the Georgia Tech rack to be displayed on a web page hosted at the web server.

2) Trema Controller: We programmed our OpenFlow controllers using Trema [5], a Ruby-based tool. Both controllers are based on a general "Learning Switch". This is a simple controller that is a part of the Trema package which floods the interfaces of all switches connected to the controller and stores a mapping of the interface and MAC address in a database that is local to the controller. This controller runs in the Georgia Tech rack also used to host the web server that makes the final nowcast results available to the users. As shown in Figure 2, the switches that connect to it are the SoX rack switch, GaTech rack switch and the NEC OpenFlow in the SoXSDX Domain.

a) Load Balancer: This controller gathers the instantaneous throughput from the SDX Switch at Starlight through flow statistics API provided by the Trema controller and switches the flow to the least congested domain based on cumulative throughput, which is an aggregate of the instantaneous throughput over time and provides a better controller performance as compared to switching based only on the instantaneous throughput. This is a good example for our vision that in the long-term our measurement infrastructure cannot only be used for analysis but also for feedback control by having the OpenFlow controller changing its behavior based on measurement data.

C. Measurement

To analyze the performance of the experiment we make use of the tools that have been developed within the scope of the Large-scale GENI Instrumentation and Measurement Infrastructure (GIMI) project. For this experiment we make use of the Orbit Measurement Framework (OMF) and Library (OML) [17] and LabWiki which is a tool used to collect and show live visualizations of network related experiments and tightly integrated with OMF. In addition, LabWiki supports the execution of large-scale experiments since the OMF Experiment Description Language (OEDL) is used to "program" an experiment. With OEDL experiments including several hundred nodes can be executed, as has been shown in cases that do not include SDXes [GEC18 Tutorial, GEC20 Demo]. This combination of tools does not only allow the measurement of an experiment, it also allows the automated execution of the experiment and repetition, if desired. These tools will also allow other experimenters to repeat this experiment. Keep in mind that the resources for the experiment described in Section V-B1 have all been requested with GENI tools and an experimenter can use these tools and the resource description (RSpec) to obtain a similar slice.

Especially for the measurement at SDXes we have instrumented the OpenFlow switch at StarLight with a measurement library that allows the observation of flow statistics. This information is gathered at the OF controller, stored in the OML database, and can be displayed live in LabWiki. The latter allows the experimenter to not only analyze an experiment after it has finished but also while it is being executed.

This measurement component is automatically integrated in the overall experiment execution, since not only the nowcast experiment itself is described through and executed by the OEDL script but also the tasks that have to be executed to perform the measurement described above. For this specific example, this means that *a)* the streaming of the radar data, the nowcasting, and the distribution of the nowcasting results; *b)* the initiation of the OpenFlow controller; and *c)* the measurement of the flow statistics are all controlled through the experiment script.

D. Future Steps

Our current plans are to further develop this SDX measurement prototype with a focus on the following topics.

First of all, we would like to support long-term measurements. Eventually, SDXes should operate reliably and with high performance for very long periods of times. To verify this requirement long-term measurements are necessary to be able to monitor the performance of an SDX over time. We will start performing long-term measurements on the prototype SDXes mentioned in this paper to study how our measurement prototype performs in such a scenario.

In Section V-B2, we mentioned that we used measurement data as input for the load-balancer OpenFlow controller. While executing this experiment we realized that the control loop operates quite slow and we have to further investigate its cause to be able to increase the response delay and make this an operational option for SDXes.

After some further testing of the prototype architecture we will make the measurement tools available to experimenters such that they can be used by the larger community for research on SDXes.

Finally, one could also think of using a selected group of SDX measurement experiments for acceptance testing of SDXes that might become online over time.

VI. Conclusion

Recently, several Software Defined Exchange prototypes have been established. These SDXes are part of the larger GENI testbed and have the goal to interconnect several SDN and non-SDN domains and make them available to experimenters. We believe that next to the SDX infrastructure the experimenter community needs tools to perform and analyze the SDX-based experiments. Motivated by that need, this paper presents the requirements for a measurement architecture that is designed for SDXes. In addition to the requirements we propose an architecture and results from an experiment executed and observed with a prototype measurement architecture.

References

[1] GENI Science Shakedown. http://groups.geni.net/geni/wiki/sol4/MultidomainShakedown.

[2] Multi-Domain SDN Deployment. https://tnc2014.terena.org/getfile/995.

[3] OpenFlow 1.3 Standard. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[4] Prototype GENI Multi-Services Network Exchange (GMNE). http://groups.geni.net/geni/wiki/sol4/GMNE.

[5] Trema. http://trema.github.io/trema/.

[6] Z. Arslan, A. Alemdaroglu, and B. Canberk. A traffic-aware controller design for next generation software defined networks. In *Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on*, pages 167–171. IEEE, 2013.

[7] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds ? Part I.

[8] D. Bhat, N. Riga, and M. Zink. Towards seamless application delivery using software defined exchanges. In *Proceedings of the Workshop on Federated Future Internet and Distributed Cloud Testbeds (FIDC)*, 2014.

[9] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12, 2007.

[10] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, and S. Shenker. Sdx: A software defined internet exchange. In *Proceedings of the ACM SIGCOMM 2014 Conference on SIGCOMM*. ACM, 2014.

[11] 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*, SIGCOMM '13, pages 3–14, New York, NY, USA, 2013. ACM.

[12] V. Kotronis, X. Dimitropoulos, and B. Ager. Outsourcing the routing control logic: better internet routing based on sdn principles. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 55–60. ACM, 2012.

[13] P. Lin, J. Hart, U. Krishnaswamy, T. Murakami, M. Kobayashi, A. Al-Shabibi, K.-C. Wang, and J. Bi. Seamless interworking of sdn and ip. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 475–476. ACM, 2013.

[14] J. Mambretti, T. A. DeFanti, and M. D. Brown. Starlight: Next-generation communication services, exchanges, and global facilities. *Advances in Computers*, 80:191–207, 2010.

[15] 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.*, 38(2):69–74, Mar. 2008.

[16] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks, 2014.

[17] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar. Omf: A control and management framework for networking testbeds. *ACM Operating Systems Review (OSR)*, 43(4):54–59, January 2010.

[18] S. Salsano, P. L. Ventre, L. Prete, G. Siracusano, and M. Gerola. Oshi - open source hybrid ip/sdn networking (and its emulation on mininet and on distributed sdn testbeds). *CoRR*, abs/1404.4806, 2014.

[19] V. Sivaraman, T. Moors, H. Habibi Gharakheili, D. Ong, J. Matthews, and C. Russell. Virtualizing the access network via open apis. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 31–42, New York, NY, USA, 2013. ACM.

[20] J. P. Stringer, C. Corr?a, J. Bailey, D. Pemberton, Q. Fu, C. Lorier, R. Nelson, and C. E. Rothenberg. Cardigan: Deploying a distributed routing fabric. In *19th IEEE Symposium on Computers and Communications (ISCC), June 2014*. IEEE, 2014.

[21] J. P. Stringer, Q. Fu, C. Lorier, R. Nelson, and C. E. Rothenberg. Cardigan: Deploying a distributed routing fabric. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 169–170. ACM, 2013.

[22] M. Su, L. Bergesio, H. Woesner, T. Rothe, A. Kpsel, D. Colle, B. Puype, D. Simeonidou, R. Nejabati, M. Channegowda, M. Kind, T. Dietz, A. Autenrieth, V. Kotronis, E. Salvadori, S. Salsano, M. Krner, and S. Sharma. Design and implementation of the {OFELIA} {FP7} facility: The european openflow testbed. *Computer Networks*, 61(0):132 – 150, 2014. Special issue on Future Internet Testbeds ? Part I.