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.

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

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

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).

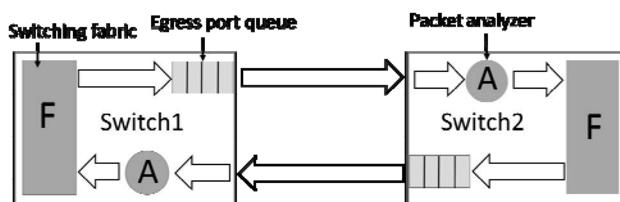

Fig. 1. Scheme of switch interaction.

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. $X/T \le C$ and $Y/T \le C$. Thereby, both output queues are empty and one-way delay in each link direction is equal to a half of RTT.
2. $X/T \ge C$ and $Y/T \le C$. 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. $X/T \ge C$ and $Y/T \ge C$. 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.

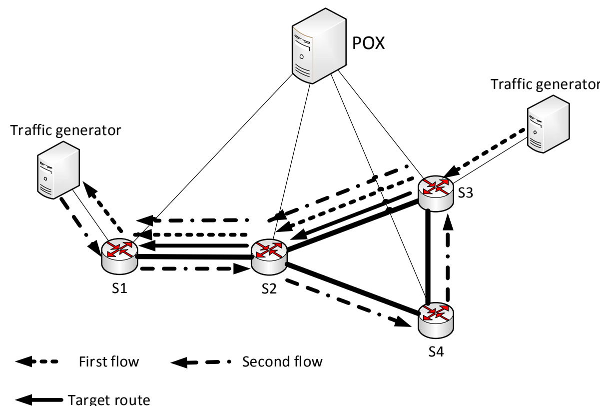

Fig. 2. Delay measurement experiment topology with generated flows and target flow.

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.

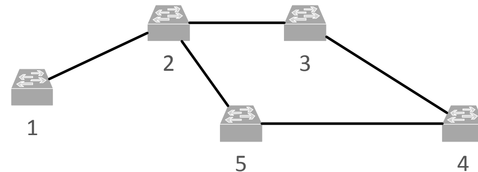

Fig. 3. Delay map construction experiment topology.

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