Security Analysis of Forwarding Strategies in Network Time Measurements Using Openflow

Alex M. S. Orozco*, Charles V. Neu†, Regio A. Michelin‡ and Avelino F. Zorzo§
*Sul-rio-grandense Federal Institute (IFSul),
†University of Santa Cruz do Sul (UNISC),
‡Federal Institute of Rio Grande do Sul (IFRS),
*†‡§Pontifical University of Rio Grande do Sul (PUCRS)
Email: *orozco@sapucaia.ifsul.edu.br, †charles1@unisc.br,
‡regio.michelin@restinga.ifrs.edu.br, §avelino.zorzo@pucrs.br

***Abstract*—Reliable network time measurement tools are important to ensure that monitoring network systems work properly, but their development do not consider security as a concern and, for example, delay attacks could compromise those tools effectiveness. Indeed, nowadays the network time measurement is not always reliable. Some researches do propose to increase network time measurement reliability using Openflow. Nonetheless, those researches do not consider the impact of altering some of the Openflow controller algorithms in their analysis. On one hand, this paper investigates how the provided POX Openflow Controller packet forwarding strategies could be applied to compromise network time measurement reliability. On the other hand, this paper also shows that the way those strategies are applied could prevent against new attacks that need to trust on network time measurement. Therefore, some experiments were performed to show the impact of the POX packet forwarding algorithms on network time measurement, either to compromise or to help in protecting the network.**

***Index Terms*—SDN, security, delay attack, inference attack, RTT forwarding, Openflow, POX.**

I. Introduction

Nowadays the complexity imposed by the wide options of network topologies implies on several strategies to increase network security. For instance, time is one of the aspects that could be used to improve network security, or other network attributes such as performance. Furthermore, in some complex scenarios, such as mesh or cloud networks, usually time, measured for some data to be sent from one node to another, may be used to detect network problems, for example, host inaccessibility, broken link, or bottlenecks caused by high traffic or attacks. Therefore, due the importance of the network time measurement, several tools were developed in the past (see, for example, Ping, Traceroute, Pathchar, Iperf [1]). Hence, the development of those tools focus on functional requirements, i.e. just to measure the time a packet takes to travel from one node to another. However, quality requirements usually are not considered [2], for example, they do not consider that someone could be, intentionally, alter time measurement.

One of the ways to solve the mentioned problem could be to use Software Defined Networks (SDN) [3], which became a large academic and industrial research topic[4]. The main functional protocol for SDN is called Openflow [5] [6], which was developed by Openflow Network Foundation (ONF). This protocol separates the control plane from the data plane in an ordinary network router or switch. The control plane is located and managed from a network entity called Controller. The Controller can communicate with many Openflow enabled network switches and manages the traffic according to policies or administrators decisions and rules. The data plane remains in the network switch.

Moreover, during the development of Openflow, new security challenges have emerged. Hence, researchers are engaged in developing new technologies and mechanisms to prevent attackers from controlling or compromising the network. Besides, if the attacker are successful they may affect the main security pillars: availability, confidentiality and integrity [7].

Indeed, the Openflow protocol could contribute to increase security [8], and consequently the reliability of network time measurements. One of the algorithms, implemented in Openflow, that has influence on network time measurement is the packet forward algorithm. This algorithm is responsible for choosing a route for any packet. In some Openflow Controllers, such as POX [9], it is possible to choose the network packet forwarding algorithm.

Some researches propose to increase network time measurement reliability using Openflow [3], or how network time measurement could be used as a way to facilitate attacks on Openflow [10]. Nonetheless, those researches did not consider the impact of altering the packet forwarding algorithm in his analysis.

Therefore, on one hand, this paper investigates how the provided POX Openflow Controller packet forwarding strategies could be applied to compromise network time measurement reliability. On the other hand, this paper also shows that the way those strategies are applied could prevent against new attacks that need to trust on network time measurement. Hence, some experiments were performed to show the impact of the POX packet forwarding algorithms on network time measurement, either to compromise or to help in protecting the network. Thus, this work presents as main contribution the package forwarding algorithm choice and how this choice could compromise the Openflow support on trustworthy network measurement field.

The remain of this paper is organized as follows. Section II

978-1-908320/73/5/$31.00 © 2016 IEEE

presents some related work that help understanding the current trends and challenges as well as some approaches related to SDN security. Section III describes the environment setup and the methodology applied to the experiments. The results and analysis of our experiments are explained in Section IV. Section V brings the conclusion and suggestions for future directions.

II. Background

The measurement of some network features, such as time, bandwidth or jitter, is an essential concern in the design of current networks. Network measurements are crucial for the operation and security of the networks, since a network manager can verify whether the bandwidth is being stressed by normal usage or is being attacked by some malicious entity (adversary), for example. Hence, if the measurement of some of the features is compromised, an adversary could exploit that in order to mask, or even perform, an attack. Thus, that could make the whole system vulnerable, and important data could be compromised, or even worse, the reputation of a company could be damaged.

Some attacks, such as delay attacks [11], exploit the end-host bandwidth bottleneck measurement mechanism. In order to solve this vulnerability, Karame *et alii* [3] discuss the Openflow contribution on network measurements security, based on bottleneck bandwidth estimation and network coordinate measurement. To achieve that, in their work, they take into consideration two entities: the prover and the verifier. The path between those entities traverses a network domain managed by an Openflow Controller. The bottleneck bandwidth estimation is obtained using the packet-pair technique [11]. The verifier requests that the prover inserts, on his header, a pre-defined flag, announcing to the Openflow-operated switches on the path that these packets correspond to bandwidth measurement packets. Then, the verifier informs the prover IP address to the Controller. The Controller propagates a rule to the outermost switch that connects to the prover, requesting all packets with the flag from the prover. Any additional delay inserted by the transition of packet-pairs is removed in this step. Then, the Controller requests to the switch all the packets that would be forward to the verifier, before they are sent to the verifier. After analyzing the packets, the Controller then sends the packets to the verifier. This process allows to find out if the bottleneck link is located between the verifier and the Openflow switch or on the provers side of the Openflow switch. After that, it is possible to measure in a more accurate way the times in the network based on the mean of round trip time (RTT) latencies. This may be used by the hosts to discover their network coordinates relative to the other hosts, *i.e.* to establish the network topology.

Although the Openflow protocol could be useful to offer a more reliable network measurement, an attack exploring that combination was proposed by [10]. This attack is a sophisticated inference attack that an adversary can use to learn more about a network. In this attack, a stream of timing probes and a stream of testing packets are sent simultaneously to the target network. Timing probes are Ping packets to a possible server in the network that passes trough the control plane, and RTT depends on the control planes load. Test packets are spoofed packets, *i.e.* packets in which the source or destination IP, or MAC, addresses could be altered. If a stream of test packets causes the probe RTTs to increase, the adversary can infer that the control plane is processing the testing packets, for example, to discover a new rule for that packet. The action of sending different test packet streams can offer enough information about the network behavior. For example, considering the scenario in which an attacker sends packets to a target destination host from different source hosts. On one hand, if the RTT for all packets are similar, then it is very likely that the target destination is a server (or an important host in the network), since all source hosts know the route to the target host. On the other hand, if RTT is really different for most of the packets, it might mean that the control plane is processing some of the packets to discover new rules for those packets. Hence, based on the forwarding rules that match on both the source and destination, the attacker can build a graph that represents the network topology. Hosts that have several edges in the graph may be critical to the network and an ideal target for Denial of Service (DoS) attacks, while hosts with few edges may be less frequently used and an ideal target for the adversary to infect while minimizing disruption to the network.

The forwarding rules aforementioned are intrinsically related to the packet forwarding algorithm used by a Controller to manage the network. Vaghani's research [12] analyzes the main forwarding algorithms available on the POX Controller. The Controller is responsible for network control policies, including how packets will be forwarded.

POX Controller offers a set of algorithms to configure packet forwarding [12]. These algorithms are described next:

a) L2_Learning: The L2_learing algorithm behaves as a layer 2 switch. As soon as it learns L2 addresses, it installs flows that are exact-matches on as many fields as possible [9], *i.e.*, different installed flows are different TCP connections results.

b) L3_Learning: The L3_learning behaves in the same way as the L2_learning for forwarding. However, this algorithm has a functionality to reply ARP requests [9]. It keeps a table that maps IP to MAC and the corresponding ports. After that, when the switch receives an ARP request, if the entry exists on the table, then it will solve and answer the ARP request; otherwise, it will flood it.

c) L2_Pairs: L2_pairs algorithm makes Openflow switches behave like a type of layer 2 switch. Differently from L2_Learning, this algorithm installs rules based on MAC addresses.

d) L2_Flowvisor: L2_flowvisor installs rules in the same way as L2_pairs. However, this algorithm uses a spanning tree component to find and update the spanning tree. After that, it conducts flooding for the selected ports from the spanning tree. The L2_flowvisor uses a module called Discovery, which is responsible for triggering link status events.

e) L2_Multi: L2_multi "learns" MAC addresses across the entire network and selects shortest path(s) between them [12]. This algorithm also uses the Discovery module to learn the topology of the entire network [9].

This set of packet forwarding algorithms was applied to the experiments of the Section III in order to evaluate the impacts of Openflow configuration on the reliability of network time measurement and also to reduce possible security attacks.

Since the Openflow Controller is a central point in an SDN, it becomes a vulnerability that can be exploited by DoS attackers [13]. In [13], researchers show how a DoS can be performed against the Controller, and proposes a scoring system to classify the incoming traffic in the Controller. In order to achieve that, they use a Support Vector Machine (SVM) algorithm [14] to avoid the malicious package processing. The SVM algorithm learns network behavior from the incoming traffic in order to identify which traffic is malicious.

III. Environment Setup and Evaluation Methodology

This section describes the set of experiments that represents a scenario to show that our claims, i.e. that different strategies to packet forwarding can affect network time measurement, and on one hand, this can mask some types of attacks, but on the other hand this can be used to avoid some types of attacks. Our simulation is based in a similar way of the one presented in [12], in which they use a mesh network composed of loops and redundant links. The simulated network had to be complex enough to be able to represent a situation in which Openflow configuration would affect network time measurement. In the simulated topology, each switch was linked to the Controller and a host, as illustrated in Figure 1.

The experiment was simulated on Mininet 2.2.1 [15], using an Intel Core 2 Duo CPU P8700 @2,53GHz with 4GB of RAM. In addition, Mininet was installed on Ubuntu 15.04.5, a Linux 3.19.0-28-generic kernel, and a POX 0.2.0 Controller also was used.

Twelve switches, structured in a mesh network[16], composed the topology, as represented by Figure 1. They were configured to operate on Openflow v.1.0 [17].

To evaluate the communication among nodes the mean of the RTT is used [18]. This mean of RTT is calculated as follows. First the packet transmission time T is calculated using Equation 1. This time represents the time between sending a packet and receiving an acknowledgment that the packet was received. This time is measured in milliseconds (round trip transmission).

$$T = N/R \tag{1}$$

In Equation 1, N represents the packet size (bits), and R is the data rate bandwidth (bit/second). Hence, to calculate RTT, Equation 2 is used:

$$RTT = S * RTT_{old} + (1 - S) * T \tag{2}$$

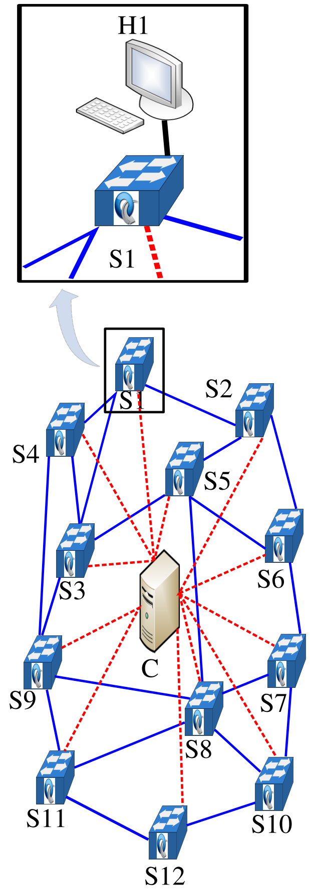

Fig. 1. Network Topology

In Equation 2: RTT is the mean RTT (ms); S is the smoothing factor, which is a factor to remove extreme variations on RTT measures; RTT_{old} is the old RTT; and, T is the packets new round trip transmission time (Equation 1). It is possible to measure RTT values using the Ping tool, in which S is equal to 0.875.

To perform the experiments, the links illustrated in Figure 1 were configured with a bandwidth of 10 Mbps and 5ms of propagation delay [12]. In the simulation, a host connected to switch S1 will communicate with the host connected to switch S11, i.e. the goal is to simulate communication that traverse the whole network (one of the longest routes in the simulated network).

For each experiment, the POX Controller was configured to forward packets based on each forwarding algorithm explained in Section II. Each experiment was performed during 370 seconds to allow *idle_timeout* and *hard_timeout* events of flow removal [18]. For *idle_timeout*, while matching packets, the flow entry will not be removed unless *hard_timeout* is reached, then the flow entry will force to be removed. Actually, the experiment sends 370 Ping packets, with 1 second interval time between each packet to achieve those measures.

To verify the impact of the packet size, on each round of 370 packets the payload was increased from 56B, 4KB, 8KB, 16KB, 32KB and 65KB. From each round, the RTT average and standard deviation values were collected. The experiment was repeated 6 times, resulting on 133,200 values. The RTT average and standard deviation values were calculated for each packet size, producing groups with values relative to each packet forwarding algorithm.

The experiment also measured the impact of each forwarding algorithm on the inference attack presented in Section II. First a packet, when a packet is sent to a switch, a switch verifies whether the information in the packet matches or not any existing rule in that switch, for example, to drop a packet or to forward that packet. If there is no matching rule, then the packet is sent to the Controller. Then the Controller decides a rule for that packet and sends that new rule to all switches, each switch will receive a new rule corresponding the route for that type of packet in that switch. After that the packet is forward to the first switch of the route. The difference between a packet that already has a rule in the switch and the packet that is sent to the Controller is the basis for the inference attack [10]. Therefore to analyze the impact of the forwarding algorithms in this type of attack, the experiment was replicated 10 times for each algorithm and also for the different sizes of packets. Different from the previous experiment, in which we analyzed the impact of the forwarding algorithm in the network time measurement, in this experiment, we reset the environment for each packet that was sent. The main objective of this resetting was to analyze the worst case for each packet, i.e. for each packet the creation of new rules and routes were analyzed in a bare situation.

IV. Results

In this section we showed the results of our simulation and then relate those results to the problems mentioned in the previous sections. The main idea is to verify whether the different algorithms can influence network time measurement and also what is their influence on inference attacks.

Table I presents the main results collected from running each algorithm described in Section II. The values in the table represent the average of RTT in milliseconds. For each algorithm, the payload varied from 56B to 65KB. As can be seen in Table I, the average response time for a 56B payload using L2_Learning is 13.22% slower than when using L2_Multi, and for a 65KB payload the difference increases to 64.17% considering the same algorithms. Therefore, the choice of the algorithm that will be used in the Controller can, definitively, influence the RTT times in an SDN.

In order to better visualize the results shown in Table I, the data from the experiments were organized in a graph as shown in Figure 2. Each algorithm was categorized based on packet size and RTT in milliseconds. Each bar represents the mean RTT value and the lines at the side of the bar indicate the amount of variation from the mean standard deviation. As can be seen in the figure, on the one hand, when the packet size is small, the RTTs are similar for all algorithms. On the other hand, when the packet size increases the RTT value from L2_learning and L3_learning algorithms are equivalent, but the L2_pairs, L2_flowvisor and L2_multi RTT values are more than twice smaller than L2_learning and L3_learning. This, also, clearly, shows that depending on the packet size, for big packets, the influence of the algorithms are significant in the RTT.

Algorithm	56B	4KB	8KB	16KB	32KB	65KB
L2_Learning	63.06	150.36	225.01	288.47	337.71	436.69
L3_Learning	61.28	143.06	219.63	282.52	282.60	401.94
L2_Pairs	60.97	63.48	70.67	82.79	109.70	163.33
L2_Flowvisor	60.95	63.54	70.65	82.82	109.73	163.33
L2_Multi	54.73	56.68	64.20	76.21	103.34	156.48

TABLE I
Average duration time in milliseconds for each algorithm

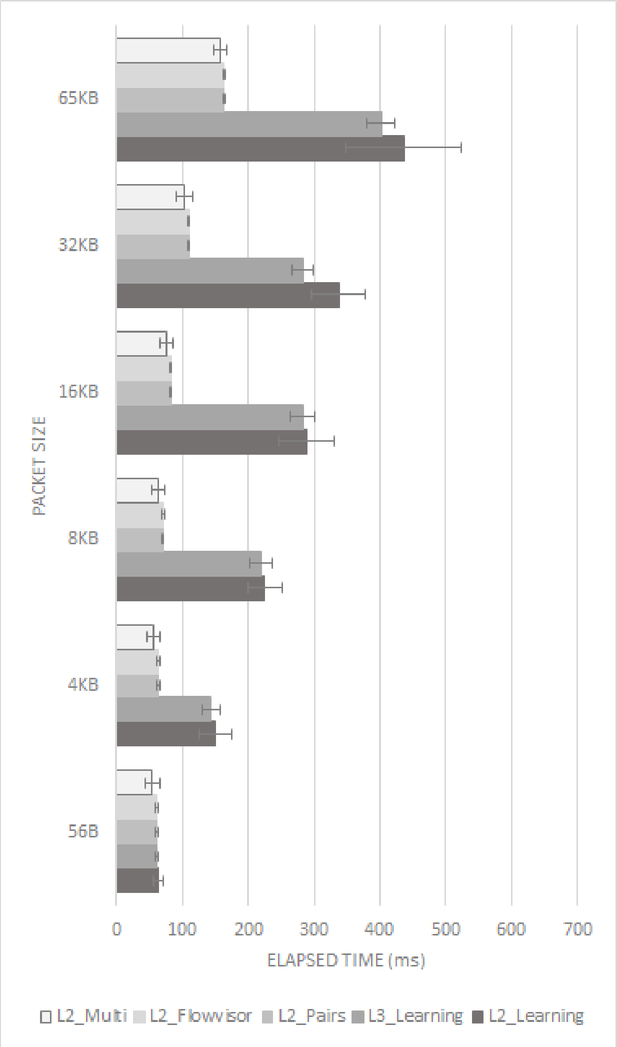

Fig. 2. Packet Forwarding Algorithms x Mean RTT

Based on those results, supposing that an Openflow network was a way to prevent delay attacks, as claimed by previous research [3], and the Controller was configured with L2_learning or L3_learning packet forwarding algorithm, the change on the algorithm to L2_pairs, L2_flowvisor or L2_multi could mask a delay attack. For example, if someone is considering the time

978-1-908320/73/5/$31.00 © 2016 IEEE

t_l for a packet to travel from Host A to host B, when using the L2_learning, if an attacker manages to change the algorithm to the L2_multi, which takes t_m (smaller than t_l - see Table I or Figure 2), then now the attacker has $t_a = t_l - t_m$ to alter the packets that are being sent from A to B, and neither A or B will detect that. Hence, despite what has been claimed in the past, regarding the use of Openflow to overcome delay attacks, they are still possible if the configuration of the Controller is not performed in a careful manner. This algorithm change could also be caused by an attack on the Controller, i.e., a DoS attack could reset the Controller and a new algorithm could be started [13] [19]. The algorithm change could be transparent to hosts, because switches remain working with old flow tables, allowing that communication continues working normally.

The results from the second part of the experiments are related to discover the RTT for a packet that has an unknown rule to a switch. In this situation, when the switch does not know a rule for a packet, the switch sends the packet to the Controller, which is responsible for establishing a rule for that packet. This new rule, could be, for example, to drop such packets or a set of rules for all the switches in the route, from host A to B, for forwarding those type of packets. Once the new rule was established by the Controller, the switch will receive it and then executes the rule for that packet. The whole time it takes for the packet, for example, to be sent from host A to B, in such situation is called, in this paper, Peak time. Figure 3 shows the measured times for different packets sizes (the same sizes as in Figure 2). As can be seen in the figure, when the packet size is small (56B), all algorithms except L2_multi produce an equivalent time. However, bigger packets impact directly on L2_learning and L3_learning performance.

As mentioned in Section III, the times shown in Figure fig:exampleFig4 are used to understand the impact of our measurements to detect when the inference attack is possible, and when it is not. In Figure 4, we show the times from Figure 2 are compared to times from Figure 3. Notice that, in Figure 4 the Peak time is shown right under the mean RTT for each algorithm. As can be seen in the figure, it is possible to realize that, for big packets size (65KB), Peak time and RTT are similar when using the L2_multi, L2_flowvisor and L2_pairs algorithms. Therefore, the inference attack [10] will not work properly, since, as explained in Section II and in Section III, the attacker will not be able to distinguish between the time a packet that is known by the switch and a packet that is unknown to the switch takes to travel from host A to host B. Hence, an attacker will have to use small packets to be able to execute an inference attack, since the times for RTT and Peak time are very different (see Figure 4) for small packets. Therefore, if someone wants to detect a possible inference attack in an Openflow network, a large number of small packets from different sources to a same target destination might indicate that that attacks is being performed. Besides, this needs to be analyzed only during the time interval that the rules are stored in the switches (see Section III). Hence, an IDS/IPS tool could be configured to trigger alarms when it detects stream of small packets during that time interval.

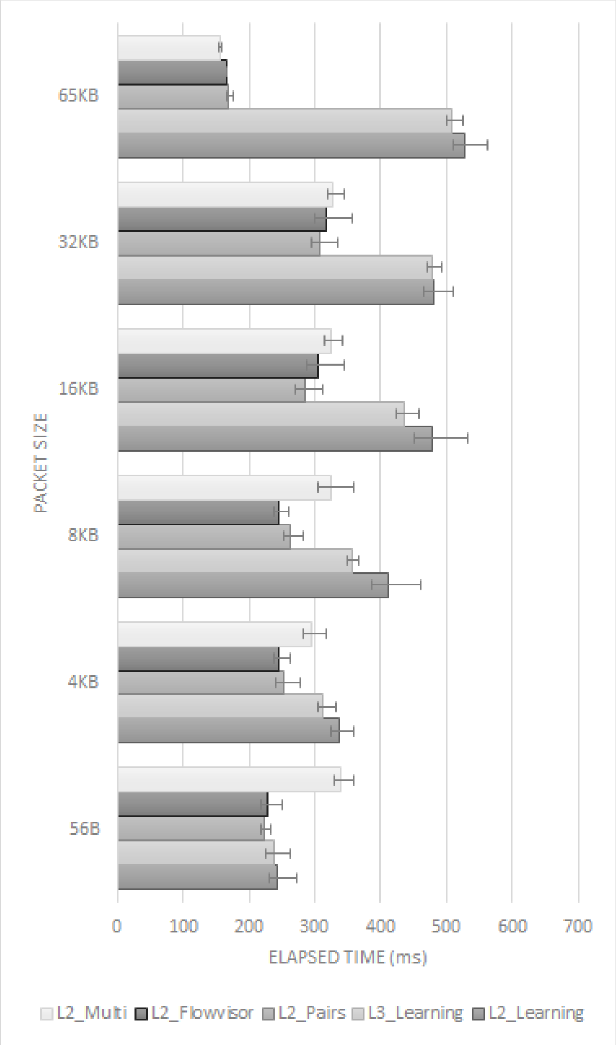

Fig. 3. Packet Forwarding Algorithms x Network Configuring time

V. Conclusion

Several applications rely on network time measurements to work properly. However, some situations imply on fluctuations on those values. It can be caused, for example, by a typical network use or a delay attack. To protect the network from malicious interference, the literature suggests the use of Openflow [3]. However, as Openflow is a new approach, new attacks to this technology have been developed and some of them rely on network time measurements, e.g., inference attack [10].

This paper presented a set of experiments to verify the impact of each packet forwarding algorithm on network RTT measurement. The results indicate that a wrong packet forwarding choice could affect significantly network measurements. Furthermore, this decision could enable a delay attack to be masked, and Openflow security contributions on trustworthy network measurement could be compromised. To solve this issue, the development of a network measurement monitoring system that establish a communication with the Controller, to detect any situation that will influence network

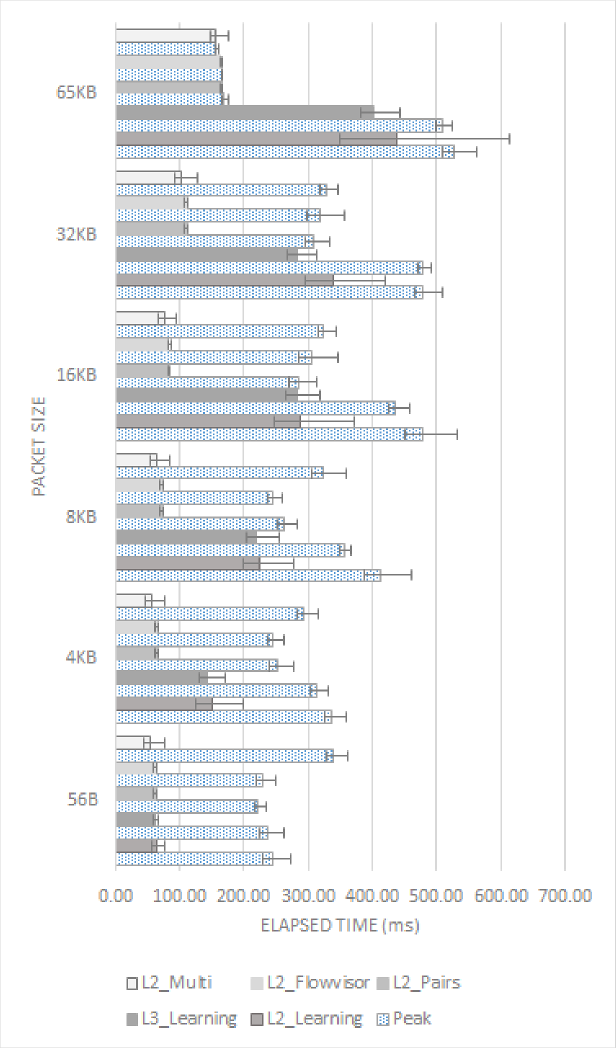

Fig. 4. Comparison between first and second experiments

time measurement is needed. The results also indicate that an attack that rely on network time measurements (inference attack [10]) can be detected and, in some circumstances, mitigated with the right forwarding algorithm choice. To achieve that, an IDS/IPS system needs to be configured to alarm stream of small packets.

References

[1] N. V. Mnisi, O. J. Oyedapo, and A. Kurien, “Active throughput estimation using rtt of differing icmp packet sizes,” in *Broadband Communications, Information Technology Biomedical Applications, 2008 Third International Conference on*, Nov 2008, pp. 480–485.

[2] G. Karame, B. Danev, C. Bannwart, and S. Capkun, “On the security of end-to-end measurements based on packet-pair dispersions,” *Information Forensics and Security, IEEE Transactions on*, vol. 8, no. 1, pp. 149–162, 2013.

[3] G. Karame, “Towards trustworthy network measurements,” in *Trust and Trustworthy Computing*, ser. Lecture Notes in Computer Science, M. Huth, N. Asokan, S. apkun, I. Flechais, and L. Coles-Kemp, Eds. Springer Berlin Heidelberg, 2013, vol. 7904, pp. 83–91.

[4] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and openflow: From concept to implementation,” *Communications Surveys Tutorials, IEEE*, vol. 16, no. 4, pp. 2181–2206, 2014.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] A. Lara, A. Kolasani, and B. Ramamurthy, “Network innovation using openflow: A survey,” *Communications Surveys Tutorials, IEEE*, vol. 16, no. 1, pp. 493–512, 2014.

[7] P. Mell, K. Scarfone, and S. Romanosky, “Common vulnerability scoring system,” *IEEE Security Privacy*, vol. 4, no. 6, pp. 85–89, Nov 2006.

[8] S. Scott-Hayward, G. O’Callaghan, and S. Sezer, “Sdn security: A survey,” in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, 2013, pp. 1–7.

[9] NOXRepo.org. Pox documentation. http://www.noxrepo.org/pox/documentation/, Visited on 18 September 2015.

[10] J. Sonchack, A. J. Aviv, and E. Keller, “Timing sdn control planes to infer network configurations,” in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks &#38; Network Function Virtualization*, ser. SDN-NFV Security ’16. New York, NY, USA: ACM, 2016, pp. 19–22. [Online]. Available: http://doi.acm.org/10.1145/2876019.2876030

[11] G. Karame, D. Gubler, and S. Čapkun, *On the Security of Bottleneck Bandwidth Estimation Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 121–141. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-05284-2_8

[12] R. Vaghani and C.-H. Lung, “A comparison of data forwarding schemes for network resiliency in software defined networking,” *Procedia Computer Science*, vol. 34, pp. 680 – 685, 2014.

[13] R. Kokila, S. Thamarai Selvi, and K. Govindarajan, “Ddos detection and analysis in sdn-based environment using support vector machine classifier,” in *Advanced Computing (ICoAC), 2014 Sixth International Conference on*, 2014, pp. 205–210.

[14] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, Jul 1998.

[15] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, “Using mininet for emulation and prototyping software-defined networks,” in *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, June 2014, pp. 1–6.

[16] R. Bruno, M. Conti, and E. Gregori, “Mesh networks: commodity multihop ad hoc networks,” *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, March 2005.

[17] O. N. Foundation, “Openflow switch specification, version 1.0.0,” [Online] https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf, 2009.

[18] P. Pupatwibul, A. Banjar, A. Sabbagh, and R. Braun, “A comparative review: Accurate openflow simulation tools for prototyping,” *Journal of Networks*, vol. 10, no. 5, 2015.

[19] M. Mushi and R. Dutta, “Data-driven study of network administration in the evolving landscape of software defined networking,” in *Proceedings of the 2014 Workshop on Human Centered Big Data Research*. New York, NY, USA: ACM, 2014, pp. 14:14–14:18.