Performance Evaluation of the ONOS Controller Under an DDoS Attack

Sebastián Gómez Macías
Universidad de Antioquia
Medellín, Colombia
sebastian.gomez3@udea.edu.co

Juan Felipe Botero
Universidad de Antioquia
Medellín, Colombia
juanf.botero@udea.edu.co

***Abstract*—Distributed denial-of-service (DDoS) has been identified as one of the biggest threats in Software Defined Networking (SDN) architecture due to it is highly effective, hard to detect and easy to deploy characteristics that exploit the vulnerabilities that this new architecture still present. This paper discusses one of the ways to deny the services of an ONOS controller, exhausting their resources when a huge number of packets_in with spoofed source mac address are sent to the controller from different zombie hosts. Metrics as ONOS controller CPU consumption and traffic latency on the network are measured to demonstrate the attack consequences.**

***Index Terms*—SDN, Security, DDoS, ONOS, CPU, Latency**

I. Introduction

Software Defined Network (SDN) has been cataloged as a promising architecture to supply the limitations of current networks, thanks to the logical centralization of the network control. This new architecture has brought several advantages that ease network management due to a global network view and the possibility to define network behaviour by applications developed into the control plane [1].

In spite of the great improvements that this new architecture provides, the following vulnerabilities and security issues have been identified: a lack of authentication mechanisms of the packet sender, the controller as a single point of failure, the OpenFlow reactive mechanism to install flow rules [2] [3], among others. These vulnerabilities have allowed different approaches attempting to attack the availability, integrity and the confidentiality of both the controller and the end devices. Some of these attacks are the well-know denial of service (DoS) and distributed denial of service (DDoS) attacks. Both have been adapted to the SDN architecture targeting the controller in order to stop the processing of the requests that come from the switches. If the controller is compromised by any of these attacks, the forwarding consequences will be noticed in the entire network due to the centralization of the network control, resulting in a greater impact in comparison with one in traditional networks.

Existing literature shows that DoS and DDoS attacks have already been targeting different SDN controllers as FloodLight, NOX, POX, among others. ONOS is considered one of the newest and robust controllers for SDN environments. To the best of our knowledge, there is not any rigorous study about DoS and DDoS attacks deployed over ONOS controller. Therefore, the main goal of this paper is to measure how much impact the DDoS attack can cause in this specific controller.

The remainder of this paper is structured as follows. In section II, we discuss recent work related to DoS and DoSS attacks in SDN architecture. In section III, we explain OpenFlow protocol fundamentals. In section IV we describe the proposed DDoS attack deployed in our testbed. In section V we show the testbed used and perform a deep analysis of the obtained results. In section VI we conclude the paper with final remarks and perspective for future work.

II. Related Work

In [4] authors has developed a DoS attack able to fill up the switch table of a **FloodLight** controller, which saves the identification of each switch (DPID) connected to the controller. According to the authors, the attack sends persistently spoofed OpenFlow messages to the controller with different DPID values, therefore the controller adds a new switch table entry by each spoofed message received. This attack causes that the switch table consumes all memory resources available in the controller resulting in the disconnection of the linked switches from the controller.

Kandoi et al. [5] take advantage of the vulnerabilities that the OpenFlow reactive mechanism has in order to install new flow rules at the switches. These vulnerabilities were used by the authors to deploy a DoS attack in a SDN topology with a **NOX** controller. The authors has studied two different effects that this attack can cause: exhaust the bandwidth of the control plane and fill up the flow rule tables of the switches. The attack was deployed using the *hping3* tool to send a huge quantity of UDP packets with spoofed sources. This heuristic has the objective of ensuring that all sent packets must be processed by the controller. This paper also shows the increase of dropped packets when the idle_timeout field of the flow rules and the control plane bandwidth are increased.

Mousavi et al. [6] took advantage of the same vulnerabilities used in [5] to deploy DoS attack to a **POX** controller. In this case, authors used the *Scapy* python library to create and send a lot of UDP packets with spoofed source IP address. This attack makes the controller unreachable for the newly arrived legitimate packets and may bring the controller down causing the loss of the SDN architecture.

978-3-903176-23-2

In [7] authors mention the possibility to deploy a DoS attack against the **ONOS** controller resources using a technique called the half-open SYN flood attack. On the other hand, in [8] authors claim that sending a lot of big packets to be processed by ONOS, the memory could be totally consumed. We consider that both references do not present metrics that prove the fall of the controller and other consequences of the attack. Our contribution in this paper is to choose one of the possible forms to exhaust the ONOS controller (DoS or DDoS attacks) and prove the success of the attack showing consumption and latency metrics.

III. OpenFlow Fundamentals

OpenFlow is the most used protocol in the southbound interface that allows software applications to program flow tables of the network devices to control different tasks like the packets forwarding or collect flow statistics. The OpenFlow architecture is formed basically by three main components: switches OpenFlow, a secure channel and a controller [9]. The devices, better known as switches, have a list of flow entries in their flow tables. Each entry has several match fields for identifying an specify flow, counters and actions. A flow can be defined as a group of packets belonging to the same time interval and sharing the same specific features. The secure channel is the interface that exchanges control information between the controller and the switches via TCP with an optional encryption mechanism, named transport layer security (TLS). The controller is responsible for managing the switch's flow tables. OpenFlow allows the controller to request information of switches or sends instructions to add or remove entries in the flow tables.

The process to forward packets in a OpenFlow switch is depicted in Fig.1. When the packet arrives to the switch, the header fields are extracted in the parser process for matching purposes. Then the header fields are compared in descendent order with the flow rules stored in the flows table looking for a match. In this step two events might happen: first, the matching system finds a flow rule that match the packet with a flow already defined, in this case, the switch applies the action specified. The second probably event is the matching system does not find a match, in this case the switch sends packet information into an OpenFlow packet called PACKET_IN to the controller to be processed. Finally, the controller instances a new rule in the switch indicating the action to apply to the packets belong to this flow [9].

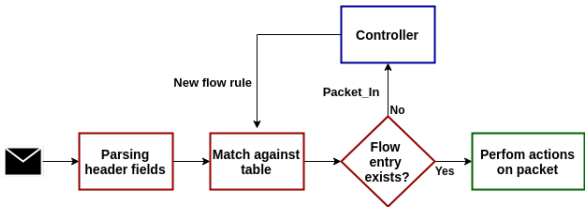


Fig. 1: Forwarding packets with OpenFlow

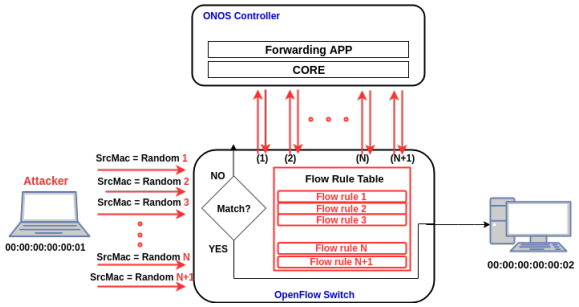


Fig. 2: Attack deployed

IV. Distributed denial of service attack in SDN

A DDoS attack is a denial of service attack (DoS) launched by multiple compromised hosts (agents) called zombies, targeting in this case the ONOS controller. The owner of the attacking device is usually unaware of the malicious program that is running on his/her device installed before by the attacker due to the found vulnerabilities [10].

The DoS attack designed, leverages the limited resources of the controller and the OpenFlow reactive mechanism to deploy the attack. The heuristic used tries to send a huge quantity of spoofed packets toward the controller to be processed and this way exhaust the resources (see Fig.2). We have noticed that for each new packet_in processed, the ONOS forwarding application instances a new flow rule specifying the physical port in, the source and destination MAC addresses in the match criteria fields.

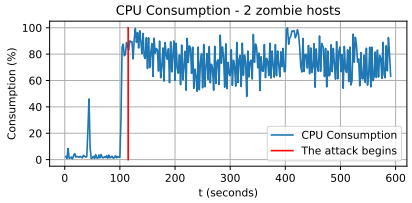
Knowing that, one python script was developed using *Scapy* library to create and send several ICMP packets per second with spoofed source MAC address, that means, each packet will have a random source MAC address. When those packets arrive to any switch, all of them must be sent to the controller into a packet_in because is highly probable that in the flow rule table does not exist any flow rule with the same random mac address in its match criteria. Deploying this attack from various compromised hosts, it is possible to reach enough number of spoofed packets arriving to the controller per second and exceed the overhead limit. ICMP packets were chosen instead UDP packets due to scapy spends less time creating and sending them getting higher rate.

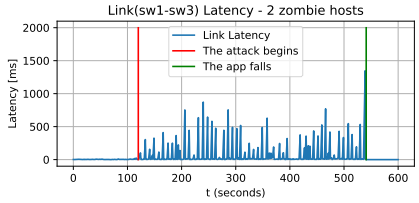
V. Evaluation

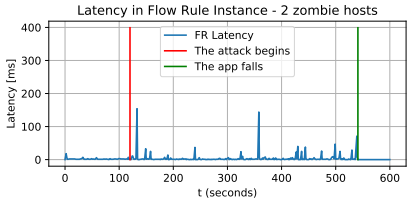
A. Testbed

To do tests in an emulated environment, we used a laptop with the next specifications: Core i5 5200, 8 GB RAM, Ubuntu 18.04 LTE, OpenFlow 1.4 and mininet 2.2.2. The testbed is depicted in Fig.6. The ONOS controller was located in a virtual machine with 2 cores and 3 GB RAM delegated. The topology defined in mininet is 3 switches, 4 hosts and 2 principal links with 100 MB of bandwidth.

In each attacking host, the attack was deployed using 4 different threads, this set-up permitted to reach a maximum rate in average equal to 100 packet/sec, that is 4.2 kilobytes/sec per attacking host. In spite of the *scapy* library is slow for





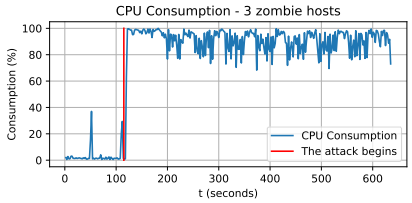


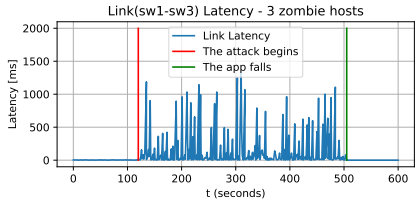
Average CPU consumption	
Before Attack	**After Attack**
4,10%	75,66%

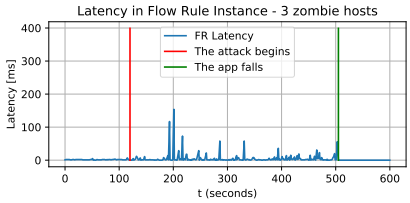
Average Link Latency	
Before Attack	**After Attack**
3,7 ms	82,2 ms

Avg flow rule instance Latency	
Before Attack	**After Attack**
1,9 ms	4,2 ms

Fig. 3: Metrics measured with 2 attacking hosts.





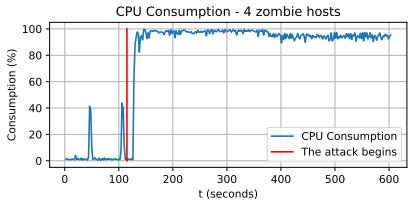


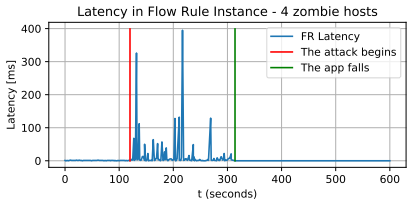
Average CPU consumption	
Before Attack	**After Attack**
4,06%	91,39%

Average Link Latency	
Before Attack	**After Attack**
4,29 ms	156,7 ms

Avg flow rule instance Latency	
Before Attack	**After Attack**
1,6 ms	5,2 ms

Fig. 4: Metrics measured with 3 attacking hosts.





Average CPU consumption	
Before Attack	**After Attack**
4,13%	95,83%

Average Link Latency	
Before Attack	**After Attack**
2,6 ms	304.4 ms

Avg flow rule instance Latency	
Before Attack	**After Attack**
0,9 ms	16,3 ms

Fig. 5: Metrics measured with 4 attacking hosts.

creating and sending spoofed packets, it is enough for seeing the DDoS attack consequences in this testbed.

The performance metrics were measured based on the number of devices deploying the DoS attack. The measures were taken over 10 minutes from 2, 3 and 4 hosts deploying the attack at the same time. The DDoS attack always started at the second minute of the simulation time. The average rate of packet_in arriving to the controller to be processed in each attack scenario is described in the table I.

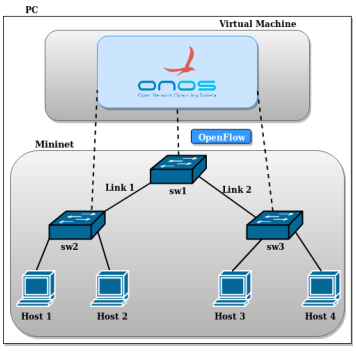


Fig. 6: Evaluation Scenario

	Before Attack	After Attack
2 attacking	3 pkt/sec	647 pkt/sec
3 attacking	7 pkt/sec	970 pkt/sec
4 attacking	7 pkt/sec	1218 pkt/sec

TABLE I: Average of Packet_In quantity

B. Metrics

The metrics chosen to measure the attack performance are: **ONOS controller CPU consumption**, **the link latency** and **flow rule instance latency**. To measure the CPU consumption was used the *psutil* python library where each 2 seconds the consumed CPU percentage in that moment, is taken and stored. To measure the other 2 metrics was created a ONOS application. The link latency is the time spent by one packet to pass a link; in this paper just was measured the link latency into the link 2 (see Fig.6). The strategy developed into the ONOS application to measure the link latency is depicted in the Fig.7. Each 2 seconds a UDP packet with an arbitrary Eth_Type value is sent from the controller to pass through the link 2. when the package finishes passing the link, it is returned to the controller due to the flow rule instanced before at the switch. Knowing the total time spent by the UDP packet

turning around (Ttotal) and the T1 and T2 times, the link latency is calculated.

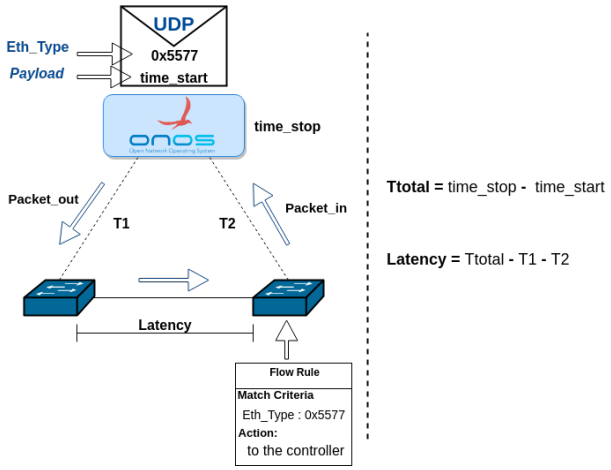


Fig. 7: Link Latency Calculation

C. Results

In the Fig.3, 4 and 5 depict the performance metrics under the DDoS attack depending of the number of attacking hosts. When the attack is running we can clearly see an increase in the average values of the metrics when the number of attacking hosts in the DDoS attack increase. On the other hand, as the link latency and flow rule instance latency were measured by a ONOS application, we have noticed that the application is not recording data during all 10 minutes of simulation time because in any moment the application falls due to a jam in the controller (see green vertical line). This was a sign to know the moment that the controller begins to fail. Knowing that, with 4 attacking hosts we got the lowest time around 5 minutes.

Analyzing the CPU consumption in the three cases (left hand side of Fig.3, 4 and 5), in the moment that the number of attacking hosts are increased, the consumption fluctuation decreases and it tends to achieve the constant maximum value, saturating the controller. The case with 4 attacking hosts, the controller has reached 95% of average CPU consumption, being our best case. On the other hand, the time spent by one UDP packet to pass through of the link 2 is increased significantly under the attack (center side of Fig.3, 4 and 5), that means a huge delay in the packets circulation on the network, especially when the attack is deployed from 4 attacking hosts. For instance, if we had a audio communication in this testbed we would have a distorted communication because the latency accepted over IP protocol is 200 ms and the average link latency obtained with 4 attacking hosts is around 364 ms. Analyzing the last metric (right hand side of Fig.3, 4 and 5), the time spent by the controller to instance a new flow rule at any flow rule table suffers a notorious increase in some measures taken into the 10 minutes but the increases do not happen as frequent as in the link latency metric. As these average latency value are not significant, the attack (either by 2, 3, or 4 attacking) does not affect enough the flow rule instance processes.

VI. Conclusions and Future Work

In this paper we have defined and deployed a DDoS attack into a SDN architecture to deny the services of an ONOS controller. The evaluation results show that the CPU consumption in the controller and the traffic delay into the network are severely affected by the DDoS attack. The CPU consumption graphs demonstrated how the overhead in the ONOS controller is increased when the number of packet_in processed per seconds increased, getting the best results with 4 hosts. Summarizing, we can conclude that the success of the DDoS attack, in any proposed SDN setup, depends of the ONOS controller resources and the attack power (number of zombie hosts) in order to be able to stop the ONOS controller operation.

The future work is to develop at the controller an intrusion detection system (IDS) based on software to detect the attack in an early stage. This module will use machine learning algorithms to classify the traffic into normal and anomalous using the features extracted periodically from the network flow statistics. The traditional mechanisms to extract the features as OpenFlow functionality or sFlow technology, may generate overhead at the controller and low accuracy. For this reason, the strategy that might be used is based on the implementation of the P4 language at the network switches to extract the features and decide whether it is necessary to send them to the controller to be classified or not, reducing the communication overhead between data and control planes.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[2] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3514–3530, Dec 2017.

[3] K. Kalkan, G. Gur, and F. Alagoz, "Defense mechanisms against ddos attacks in sdn environment," *IEEE Communications Magazine*, vol. 55, no. 9, pp. 175–179, Sep. 2017.

[4] J. M. Dover., "A switch table vulnerability in the open floodlight sdn controller." urlhttp://dovernetworks.com/wp-content/uploads/2014/03/OpenFloodlight-03052014.pdf, 2017.

[5] R. Kandoi and M. Antikainen, "Denial-of-service attacks in openflow sdn networks," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 1322–1326.

[6] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2015, pp. 77–81.

[7] O. ADENUGA-TAIWO and S. SHAH HEYDARI, "Security analysis of onos software-defined network platform," Tech. Rep., 2016.

[8] R. K. Arbettu, R. Khondoker, K. Bayarou, and F. Weber, "Security analysis of opendaylight, onos, rosemary and ryu sdn controllers," in *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*, Sep. 2016, pp. 37–44.

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

[10] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004. [Online]. Available: http://doi.acm.org/10.1145/997150.997156