JOURNAL OF INFORMATION AND COMMUNICATION TECHNOLOGY
http://jict.uum.edu.my

How to cite this article:
Polat, H., & Polat, O. (2021). An intelligent software defined networks controller component to detect and mitigate denial of service attacks. *Journal of Information and Communication Technology, 20*(1), 57-81.

AN INTELLIGENT SOFTWARE DEFINED NETWORKING CONTROLLER COMPONENT TO DETECT AND MITIGATE DENIAL OF SERVICE ATTACKS

Huseyin Polat & Onur Polat
Department of Computer Engineering, Gazi University, Turkey

Corresponding author: polath@gazi.edu.tr
onurpolat@gazi.edu.tr

Received: 2/5/2019 Revised: 11/6/2020 Accepted: 15/6/2020 Published: 4/11/2020

ABSTRACT

Despite many advantages of software defined networking (SDN) such as manageability, scalability, and performance, it has inherent security threats. In particular, denial of service (DoS) attacks are major threats to SDN. The controller's processing and communication abilities are overwhelmed by DoS attacks. The capacity of the flow tables in the switching device is exhausted due to excess flows created by the controller because of malicious packets. DoS attacks on the controller cause the network performance to drop to a critical level. In this paper, a new SDN controller component was proposed to detect and mitigate DoS attacks in the SDN controller. POX layer three controller component was used for underlying a testbed for PacketIn messages. Any packet from the host was incremented to measure the rate of packet according to its device identification and its input port number. Considering the rate of packets received by the controller and threshold set, malicious packets could be detected and mitigated easily. A developed controller component was tested in a Mininet simulation environment with an hping3 tool to build artificial

DoS attacks. Using the enhanced controller component, DoS packets were prevented from accessing the controller and thus, the data plane (switching devices) was prevented from being filled with unwanted flows.

Keywords: Security, DoS attack, decision making, software defined networking, POX controller.

INTRODUCTION

The idea of Software Defined Networking (SDN) was developed by Stanford University based on the OpenFlow protocol (Kreutz et al., 2015). With its architecture, SDN provides, allows, controls, changes, and manages a network via a central controller. It also separates the control and data planes that are combined in the traditional network structure and brings a new approach for computer networks. In SDN, the management mechanism is distinct from network devices. Thus, network devices become simple and inexpensive devices that simply transfer packets.

The SDN structure consists of control, data, and application planes as shown in Figure 1. These planes are described below:

- *Control Plane*: Responsible for managing transmission devices in the data plane. The SDN controller, which is the brains of the network, is found in this plane.
- *Data Plane:* Consists of communication devices (OpenFlow switching devices). Devices in this plane cannot perform high-level networking processes. This plane is programmed and managed by a control plane (Maugendre, 2015). When a new data packet arrives at the transmission device, the device sends packet header information to the controller for a route request. The controller, which has a dominance over the general network, calculates the flow route and sends the transmission rule required for the route to the transmission devices via a safe channel. Transmission devices, therefore, update their flowcharts and send the packet to the target or drop it (Bholebawa & Dalal, 2016).
- *Application Plane:* Helps the control plane with the security, routing, load-balancing, network structuring, and policy-making to be applied to the network.

With its flexible, programmable and dynamic architecture, SDN is expected to replace the traditional network. This is possible because SDN has the ability to respond to new technologies that require high bandwidth and dynamic management. Such technologies are cloud computing, big data, virtualisation, and mobile. The data forwarding and control functions of the SDN architecture are separated into different planes. SDN has three functional planes: data plane, control plane, and application plane. SDN networking devices, such as switches and routers, are located in the data plane and carry out packet transmissions based on rules determined by the controller (Kim & Feamster, 2013). Switching devices in the data plane send reports to the controller regarding the number of packets received from the controller and the number of packets transferred and dropped periodically. By using switching device reports, the controller can detect switching devices that show malicious behaviour (Stancu et al., 2015).

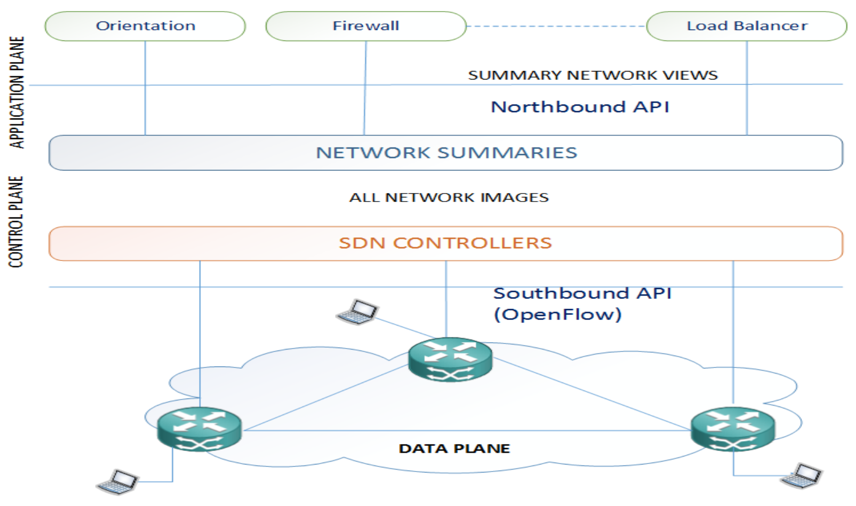


Figure 1. *SDN architecture.*

The control mechanism is abstracted from the network devices and moved to the controller that operates as the centre of the network located in the control plane. A controller, which is the brain of SDN, makes decisions on which flow is transmitted by the devices in the data plane and how the flow is organised. The controller also collects information on the states of the entire network from the network devices (Cui et al., 2016). The application plane consists of networking service applications such as security applications, analytics, business applications and others. Applications communicate with devices in the network infrastructure via the controller. Network devices are programmed using application programming interfaces (APIs) referred to as northbound and southbound interfaces (Xie et al., 2015). The controller communicates with the services working on the application plane via the northbound interface. On the other hand, the controller uses the southbound interface to communicate with the data plane devices. The most popular protocol used in this interface is the OpenFlow protocol (Cabaj et al., 2014).

SDN facilitates the implementation of innovative solutions for issues like security and virtualisation via software such as temporospatial-SDN and SDN-WAN (Xia et al., 2015). Moreover, SDN has three unique security threats, which include attacks on the control plane, attacks on the data plane, and attacks on the communication line between the control plane and the data plane. The most dangerous are those against the controller. If the attackers have access to the controller, they can take control of the entire network (Stancu et al., 2015). In addition, Denial of Service (DoS) attacks on the flow table storage capacity on transmission devices also pose a serious threat (Shang et al., 2017). The integrity of the network traffic between the switching device and the controller in SDN can be achieved if the OpenFlow protocol is consistent, accurate, trustworthy, and non-repudiate (Li, Meng, & Kwok, 2016). If the attackers have control over the OpenFlow protocol, they can access the data sent from the controller to the switching device and control the data plane.

The controller may be exposed to DoS attacks via its data plane communication line. The data plane may also be exposed to attacks via the network device flow tables (Scott-Hayward, O'Callaghan, & Sezer, 2013). These attacks are likely to happen in the form of fake traffic. The attackers have access to personal computers (PCs) and servers to create their malicious software in the network without the knowledge of the users. With that kind of network, the attackers can direct high volumes of traffic to the OpenFlow switching device in the data plane. In this case, malicious and legitimate traffics are mixed up and it is difficult to distinguish between these two traffics. If the incoming packets do not match the flow inputs in the OpenFlow switching devices, they are initially transferred to the flow buffer. Then, they are sent to the controller to write a new rule with the PacketIn message. Large amounts of PacketIn messages arrive at the controller in a very short time. In this situation, the controller's resources (memory, processor, bandwidth, etc.) are exhausted and thus, the controller reaches a state where it cannot operate (Padmaja & Vetriselvi, 2016). On the other hand, when new and legitimate packets reach the controller, the latter cannot be accessed, and thus, the SDN architecture collapses. Moreover, the bandwidth of the communication line between the controller and the switching devices under attack is also negatively affected. The performance of the network, therefore, deteriorates significantly (Shu et al., 2016).

In addition, DoS attacks on the flow table for packets from unknown or known sources are sent to the switching device first. The incoming packets are then sent to the controller via switching devices, and the controller writes the rules back to the switching devices (Ali et al., 2015). The flow table capacity of the switching device is filled up with flow rules after a short time. As a result, there is no place for legitimate incoming traffic in the flow table, and new rules cannot be written, thus packets cannot be transferred. Furthermore, because the buffer capacity is exhausted in this situation, all new packets are dropped (Kreutz, Ramos, & Verissimo, 2013). While the SDN is expected to bring about significant change by improving the underlying traditional network in terms of network reconfiguration and management, there are still some drawbacks with regard to network security (Proença et al., 2015).

To overcome the DoS attack problem in SDN network, particularly in controller and switching devices, there is a need for an intelligent mechanism with low complexities. A DoS attack that floods a large number of packets with different Internet protocol (IP) sources addresses the problem in controller and switching devices even worse. The motivation behind this study is to detect and prevent DoS attacks known as flooding attacks at the controller and switching devices in particular. An intelligent SDN controller component is proposed to detect and mitigate DoS attacks using POX controller layer 3 learning (l3_learning) component. POX controller is chosen because it allows an easy way to run OpenFlow/SDN experiments, faster development, and prototyping of new network applications.

The rest of this paper is structured as follows. The next section gives an overview of the previous studies discussing the DoS attacks in SDN network and the summary of their achievement and limitation were also given. Then, the proposed solution is presented in the following section. Performance results, evaluation, and discussion for the experiment are presented in the fourth section. Finally, the proposed solution is concluded in the last section.

RELATED WORKS

Although the new concept of SDN is supposed to improve the conventional understanding, redesigning, and management of the network, several security doors still remain open. There are several forms of attacks by malicious users targeting SDN. DoS attacks are the most frequent and most destructive of such attacks (Imran et al., 2019). Various approaches have been presented for the detection, prevention, and mitigation of the SDN-targeted DoS attacks. However, most of the presented solutions apply to a particular problem in the

SDN network due to the structure of SDN architecture. Such a solution by Zhang et al. (2016) and Tian et al. (2019) may be able to protect the control plane, but not the data plane. The following are the literature that discusses the detection and mitigation measures against DoS attacks that are deemed relevant for this work. The summarised advantages and disadvantages of this literature are given in Table 1.

In their study, Shin et al. (2013) sought to respond quickly to the attacks by the AVANT-GUARD. Two new modules called "connection migration" and "actuation trigger" were added to the switching devices. With the new mechanisms introduced, statistics were collected from the switching devices and network-related information was effectively sent to the controller. Based on the available switching devices, detailed information regarding packet routes and network status was obtained using the actuation trigger module. Using the connection migration module, the flow requests sent to the controller were limited to minimise the load on the communication line between the control plane and the data plane. The modules were created within the scope of AVANT-GUARD software application, which placed less than 1% load on the system. In addition, this module prevented unsuccessful Transmission Control Protocol (TCP) sessions without informing the control plane during the TCP SYN flood attacks. The packet was not sent to the controller for flow requests without the completion of the TCP handshake. However, AVANT-GUARD did not have a comprehensive solution since it only has mitigation for TCP-based flooding attacks. For example, no solution had been developed for protocols such as User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) (Shin et al., 2013).

Dridi and Zhani (2016) designed a three-part SDN-Guard architecture. The three parts were: (1) a flow management module that determined the flow routing and value such as timeout; (2) a rule collection module that collected malicious traffic flow inputs and reduced flow table inputs; and (3) a monitoring module that collected the flow's statistics such as a switch, link and others. In addition, the SDN-Guard was in contact with the Intrusion Detection System (IDS). The IDS analysed the PacketIn messages and informed the SDN-Guard of the possibility of a threat. The management module updated the flow routing decisions and the timeout values in the switching device flow table based on the information it gathered from the IDS. The proposed SDN-Guard architecture prevented malicious packets from reaching the controller and overloading of switching devices in the data plane with malicious flows. Nevertheless, it was difficult to differentiate malicious traffic and legitimate traffic, thus some of the malicious traffics were able to reach the controller (Dridi & Zhani, 2016).

Wang, Xu, and Gu (2015) developed FloodGuard against attacks where attackers try to exhaust control and data plane resources using PacketIn messages. FloodGuard consisted of a proactive flow rule analyser and packet migration modules. The proactive flow rule was the controller application in the control plane. When packet migration detected an attack, it sent packets that did not match the flow table to the Data Plane Cache (DPC) that was part of it. Based on the changing values, a proactive flow rule was created based on the new destination information and it was sent to the switching device. DPC gradually sent packets that did not match the flow using the "rate-limiting and "round-robin" algorithms. A scalable and effective protection mechanism against DoS attacks targeting data and control aircraft was developed. The module was created within the scope of the FloodGuard system, which caused delay during the decision-making phase and caused unnecessary loading to the controller (Wang et al., 2015).

Dao et al. (2016) suggested an Adaptive Suspicious Prevention (ASP) mechanism against DoS attacks that would target the communication line between the controller and the switching device. The key concept of this method was to determine the types of users using the Probabilistic History-Based IP Filtering algorithm. The developed mechanism aimed to analyse user traffic information and to apply policies to the incoming packets based on the type of user. Timeout and action parameters were updated and new policies were determined. Unknown users were handled suspiciously and the users were given short timeout values. Later, they were classified by the IP filtering method. If users were safe, the controller assigned a standard timeout value and updated the flow inputs. When users were found to be suspicious, the controller removed flow inputs and blocked users. The ASP mechanism for malicious packets sent put an additional burden on the switching devices in the data plane. Furthermore, classification of legitimate and malicious traffics caused an additional delay in the network. Thus, the switching devices with limited buffer capacity might fill up with queued packets and caused others to be dropped (Dao et al., 2016).

Benton, Camp, and Small (2013) stressed that if the controller was not vigilant when implementing reactive SDN rules with single control points, it could be exposed to DoS attacks with PacketIn messages. The scholars developed attack traffic from different transmissions addressed using the "l2 learning" component of the POX controller and disabled the flow table of the switching device overloading it with unnecessary flows. They recommended that concerning DoS attacks, controller vendors need to emphasise on the importance of rate-limiting and rule-generation activities with the publication of guidelines for developers. Thus, a possible solution and future work could be a mechanism that combines both OpenFlow application analysis and flow-table analysis. The solution should systematically identify the events and traffics that generate PacketIn and Flow-modification messages to warn the developer of potential DoS vulnerabilities (Benton et al., 2013).

Lawal and Nuray (2018) presented a real-time detection of the Distributed Denial of Service (DDoS) attacks on SDN and a control method based on the sFlow mitigation technology. sFlow analysed samples of packets collected from the network traffic and generated handling rules to be sent to the controller in case of attack detection. The threshold was set to 25% to make sure that the wide area network (WAN) bandwidth would not saturate or a service could not be disrupted. The attack was mitigated within a period of ~10 seconds after detection occurred and the traffic went below the threshold line. The previous study only used ICMP ping and ICMP flooding attack packets to evaluate the proposed solution (Lawal & Nuray, 2018).

Tran, Le, and Tran (2018) proposed an ODL-ANTIFLOOD, which consisted of two components including a network application for supporting decision-making and a network function for enforcing the detection and mitigation tasks. It also contributed to a multilayer attack detection mechanism and a three-phase mitigation approach to deal with the attacks. The proposed solution could detect attacks after 40 milliseconds on average, and the accuracy of the detection process was around 95%. Moreover, the proposed solution also effectively and efficiently mitigated attacks to reduce Central Processing Unit (CPU) utilisation from high 90% to 20%. An ODL-ANTIFLOOD was a more flexible and adaptive system with a simple and high accurate detection process. It had some drawbacks in detecting multiple destinations (i.e. packets with different IP destination address) attack traffic (Tran et al., 2018).

Wright and Ghani (2019) demonstrated the impact of DoS attacks on SDN elements in the NSF GENI network testbed. This work provided a key baseline and set of input data from which to develop further detection and mitigation strategies. The flagship NSF GENI testbed facility was used to evaluate the impact of DoS attacks in a live OpenFlow-based SDN network setting using the hping3 utility. They concluded that memory utilisation eventually began to decrease as the duration of the attack continued. Furthermore, for the more realistic case of randomised source/destination addresses, the shorter packet spacing consumed slightly more memory than the higher spacing packet. In both cases, however, the available (free) memory leaned downwards that increased the attack durations. The testbed did not include multiple attacking hosts, which were connected to each switching device (Wright & Ghani, 2019).

Polat, Polat, and Cetin (2020) and Sen, Gupta, and Ahsan (2020) employed machine learning algorithms to detect DDoS attacks on SDN architecture. Polat et al. (2020) used Support Vector Machine (SVM), Naive Bayes (NB), Artificial Neural Network (ANN), and K-Nearest Neighbour (KNN) classification models. A new dataset was created using feature selection methods in the existing dataset to simplify the models, facilitate their interpretation, and provide a shorter training period. The highest accuracy rate of 98.3% was obtained with the KNN classifier. On the other hand, Sen et al. (2020) utilised AdaBoosting algorithm with decision stump as a weak classifier to build the classifier model for the network. The dataset used to build the model included data for legitimate and malicious traffics for six types of protocols such as TCP, UDP, ICMP, ARP, IPv4, and SSH. The highest accuracy rate of 93% was obtained in the dataset with decision stump as a weak classifier.

Table 1

Summary of Literature Review

Authors	Advantage	Disadvantage
Shin et al. (2013)	• Restricted flow requests that were sent to the controller to avoid the load on the communication line between the control plane and the data plane. • No packet was sent to the controller for flow requests without completing TCP handshake.	• This module stops unsuccessful TCP sessions without informing the control plane during TCP SYN flood attacks. • Did not provide a comprehensive solution against TCP-based flooding attacks.
Dridi and Zhani (2016)	• Succeeded to minimise by up to 32% the impact of DoS attacks on the controller performance, switching device memory usage, and control plane bandwidth.	• Even though the number of flow rules in the table of switching devices were reduced, malicious traffic was still forwarded, which somehow consumed bandwidth.
Wang et al. (2015)	• A solution only had minor overhead as compared to other solutions. • In the software environment, the bandwidth was almost unchanged.	• The hardware environment required real OpenFlow hardware switching devices to maintain the same bandwidth.

(continued)

Authors	Advantage	Disadvantage
Dao et al. (2016)	• Reduced the average number of flow entries at switching devices and the average number of request packets up to 38% and 36%, respectively.	• Needed more time for packet processing as this problem led to high delay time for the SDN network to handle traffics. • Generated overhead when the controller requested trigger periodic reports from switching devices. • Reduced only part of the effect of DoS attacks using the short timeout.
Benton et al. (2013)	• Suggested the implementation of the Transport Layer Security (TLS) in all the controller and switching devices. • Illustrated classes of vulnerability for a network that relied heavily on PacketIn messages.	• No experimental evaluation was conducted to assess the illustrated vulnerability. • No enhanced method was proposed.
Lawal and Nuray (2018)	• Mitigated attacks within 10 seconds after detection occurred.	• Used only ICMP ping and ICMP flooding attack packets to evaluate the solution.
Tran et al. (2018)	• A flexible and adaptive system with a simple and high accurate detection process. • Detected attacks after 40 milliseconds on average. • The accuracy of the detection process was about 95%. • Effectively mitigated attacks to CPU utilisation from 90% to 20%.	• Hardly detected multiple destinations attack traffic.
Wright and Ghani (2019)	• Could be used as the baseline and set of input data from which to develop further detection and mitigation strategies.	• The testbed did not include multiple attacking hosts, which were connected to each switching device.
Polat et al. (2020) and Sen et al. (2020)	• With the KNN and AdaBoosting algorithm, it achieved an accuracy of 98.3% and 93%, respectively. • By using machine learning methods, malicious traffics could be easily classified.	• Added extra delay in detecting DDoS in the network.

PROPOSED SDN CONTROLLER COMPONENT

The POX controller had several components that supported various network behaviours such as the hub, layer two (l2) switching device, and layer three (l3) routing device. No component installed the default flow to a switching device or had a mechanism to control the PacketIn event to prevent DoS attacks from occurring. This was also true with the OpenFlow v1.0 protocol, a protocol that provided a secure communication channel between the switching device and the controller. This protocol did not provide a mechanism for detecting and preventing such types of attacks. Layer 3 learning (l3_learning) component installed a flow table after receiving a PacketIn event. In this study, an improved l3_learning component of the POX Controller was used for underlying a testbed for PacketIn messages. Random source IP address was also used to create a bunch of packets that would cause a misflow in the switching table. The purpose of these attacks as to generate as many packets as possible that would reach the controller, keep the controller occupied, and similarly fill up the switching device with a bunch of unnecessary flow tables.

In order to detect and prevent this form of attack, any packet received by the controller was incremented according to its switching device identification (i.e. DPID) and its input port number. A packet counter was used to monitor the number of packets to reach the controller within a very short time. From a packet counter, the module calculated the rate of packets reaching the controller. Let the rate of packets reaching the controller be known as R. The rate of packets reaching the controller is given in Equation 1 below.

$$R = \frac{P_n}{\Delta t} \tag{1}$$

where; P_n = Number of packets reaching the controller
$\Delta t = t_i - t_{i-1}$ = Time used for to reach the controller in seconds

In order to prevent the switching devices from saturation and flooding packets to the controller, a threshold value was chosen on the basis controller and switching device saturation. To find the value of optimal threshold, a series of experiments were run to observe the effect of DoS attacks on the controller and switching devices. The optimal threshold was obtained by comparing different rates of incoming packets from legitimate and malicious traffics. In this case, the rate was set to 2kpps (i.e. $Th =$ 2kpps).

Then, the rate of packets reaching controller was compared with a threshold. If the rate of packets exceeded a threshold value (i.e. $R > Th$), the controller first deleted all the flow tables on the switching device according to its switching device port/interface. Then, a drop flow table was added to drop all the packets from a malicious host to prevent them from reaching the controller. In this case, the host would be considered as a malicious host. However, if a threshold value was greater than packet rate (i.e. $Th > R$), an existing l3_learning component was performed to install a flow table for the destination host according to the learned packet header. In this case, a host would be considered as a legitimate host because it would be learned according to its behaviour from the switching device port/interface number and DPID.

On the other hand, the drop flow table could be removed from the switching device either by hard timeout or soft timeout due to an inactive flow table. In this case, if the attacker continued to flood packets to the controller, the controller and the entire network could be interrupted again. As a result, a timer was set to send a port or flow statistics request (*net_stats_req* and *flow_stats_req*) to check a host whether it continued to flood packets or not. If the received statistic indicated that the host as still overflowing packets, the controller must update the drop flow table. The algorithm was implemented based on the PacketIn event activated by the controller. When no PacketIn event was triggered by the controller, no drop flow tables or other flow tables would be implemented by the controller. The algorithm is displayed in the flow chart shown in Figure 2 and summarised in the pseudo-code of Algorithm 1.

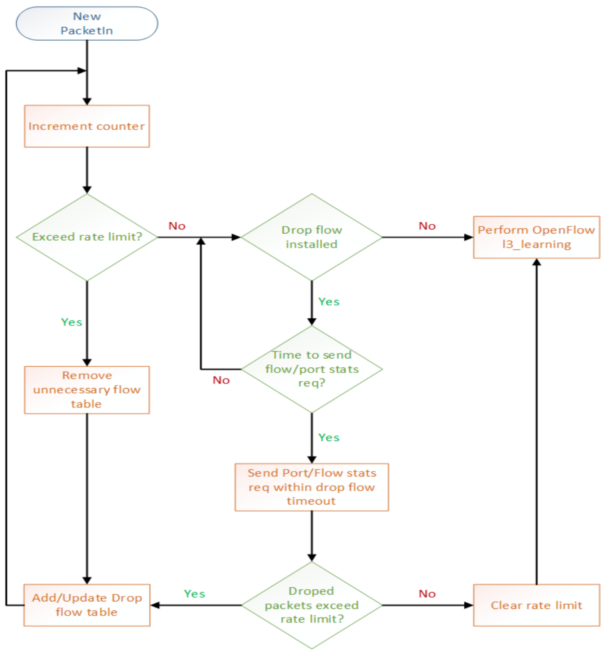

New PacketIn
Increment counter
Exceed rate limit?
No
Drop flow installed
No
Perform OpenFlow l3_learning
Yes
Yes
Time to send flow/port stats req?
No
Remove unnecessary flow table
Yes
Send Port/Flow stats req within drop flow timeout
Add/Update Drop flow table
Yes
Droped packets exceed rate limit?
No
Clear rate limit

Figure 2. *Flowchart of DoS attack mitigation on SDN controller.*

Algorithm 1: Pseudo code of DoS mitigation in SDN controller

```
Controller packet in
Increment counter (counter +=1)
If counter >=rate_limiter:
    Remove all unnecessary flows()
    Add  drop flow rule()
Else if (flow added = = True):
    If (time_to_send_stats_req == True):
        Send_stats_req()
    Handle_stats_reply()
    If (stats_received >= rate_limit):
        Update_drop flow rule()
    Else:
        Clear everything() #including counter
        Perform l3_learning()
Else
    Perform l3_learning()
```

A network topology that was used to simulate the present study is shown in Figure 3. The network topology consisted of one remote switch (POX switch) and two OVSKernel switches (s1 and s2), each controlling three hosts. These two switches were wired to a speed of 2Gbps. The connection between the hosts and the switches was set to 100Mbps. Host h100 and h200-Server were used to test the bandwidth using the iPerf command. h100 was used as a client, while h200 was used as a server. Host h1 was used to flood the packet to a switch with a random source IP address. On the other hand, h2, h4, and h5 were used as legitimate hosts. These hosts were regular hosts with hping3 tool built on the Ubuntu 14.04 operating system.

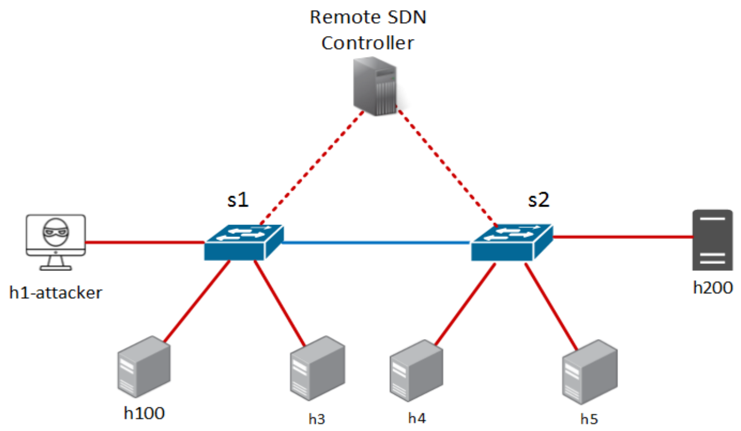


Figure 3. *Testbed network model.*

RESULTS AND EVALUATION

The topology was executed in the Mininet simulator. To simulate attacks and evaluate results, the topology consisted of a minimum number of nodes. The switch was connected to the remote POX controller (c0) by loopback address via port 6634 (i.e. 127.0.0.1:6634). DoS attacks were developed on h1 and directly connected to the Open v-switch s1. DoS traffic was generated using hping3 on the network, which was created using the Xterm command when h1 in Mininet was opened. For each experiment, the attacker sent out TCP packets with a data size of 512 bytes within 20 minutes. The rate was around 2,000 packets per second or 4mpbs for 1,000 packets per second. In the following sub-sections, various network efficiency metrics were analysed.

Average Flow Setup Latency

Applications always assume latency is low and constant. A longer time to update increases congestion, and if latency decreases, failure increases over time. The flow setup latency for l3_learning with hping3 was far too high (over 120ms all the time) (Figure 4). However, when an attack was detected and prevented, there was a high decrease in latency. This meant that the controller could add new flow table entries in a shorter time. This would have a great impact on a network that required high performance.

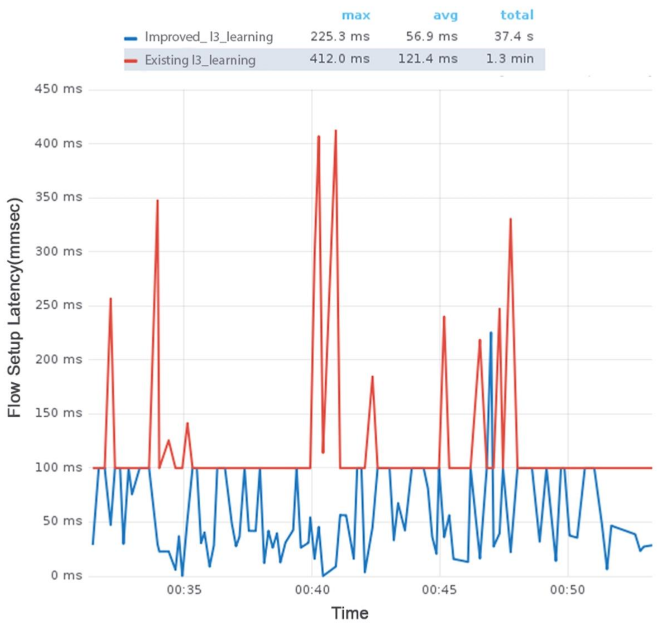


Figure 4. *Average flow setup latency.*

OpenFlow Message to Controller and Buffer Overflow

Figure 5 shows the OpenFlow messages sent from the switching device to the controller and the timeline of how many PacketIn messages the controller received. Using l3_learning, the total number of OpenFlow messages that reached the controller were 18,717. While using Improved_l3_learning, the total number of OpenFlow messages that reached the controller were 3,117. When the attack was carried out using l3_learning, numerous messages were sent to the controller and the number of messages were reduced after t = 65 sec. As a result of this attack, the Open vSwitch was unable to keep up with the speed of the incoming packets and resulted in a buffer overflow exception in the POX controller as shown in Figure 6. Moreover, this resulted in the switching device to drop certain packets, including those from genuine users. Thus, some of the packets would also not reach the controller so that the new flow tables could be added. Using Improved_l3_learning, the number of PacketIn messages were reduced by adding a drop-flow table for a host that generated an attack. In this way, several advantages were achieved, such as avoiding buffer overflow and preventing malicious request messages sent to the controller.

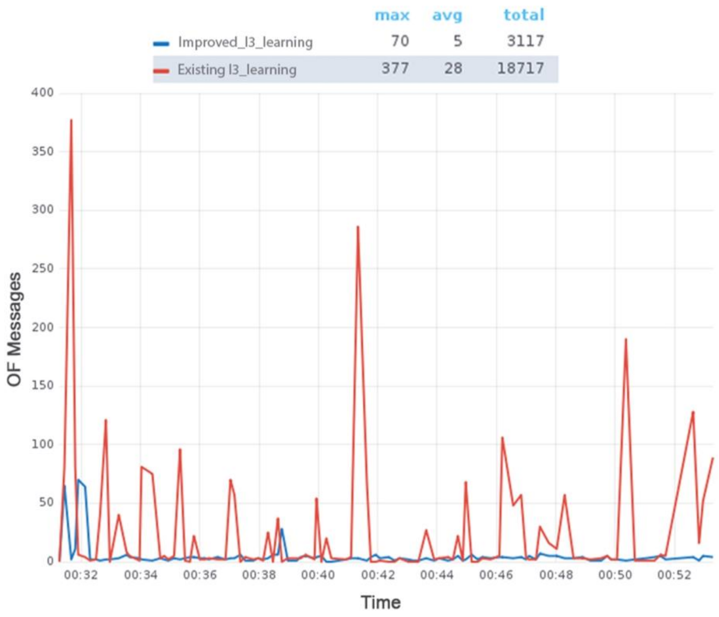
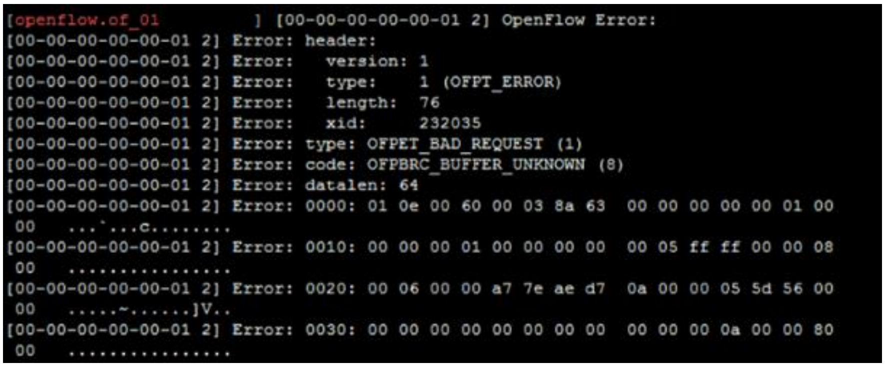


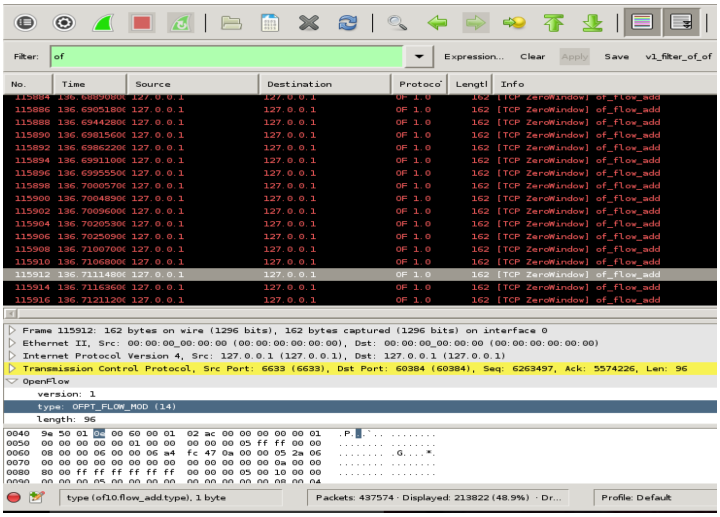
Figure 5. *OpenFlow messages to controller.*



```
[openflow.of_01          ] [00-00-00-00-00-01 2] OpenFlow Error:
[00-00-00-00-00-01 2] Error: header:
[00-00-00-00-00-01 2] Error:   version: 1
[00-00-00-00-00-01 2] Error:   type:    1 (OFPT_ERROR)
[00-00-00-00-00-01 2] Error:   length:  76
[00-00-00-00-00-01 2] Error:   xid:     232035
[00-00-00-00-00-01 2] Error: type: OFPET_BAD_REQUEST (1)
[00-00-00-00-00-01 2] Error: code: OFPBRC_BUFFER_UNKNOWN (8)
[00-00-00-00-00-01 2] Error: datalen: 64
[00-00-00-00-00-01 2] Error: 0000: 01 0e 00 60 00 03 8a 63  00 00 00 00 00 01 00
 00   ...`...c........
[00-00-00-00-00-01 2] Error: 0010: 00 00 00 01 00 00 00 00  00 05 ff ff 00 00 08
 00   ................
[00-00-00-00-00-01 2] Error: 0020: 00 06 00 00 a7 7e ae d7  0a 00 00 05 5d 56 00
 00   .....~......]V..
[00-00-00-00-00-01 2] Error: 0030: 00 00 00 00 00 00 00 00  00 00 00 0a 00 00 80
 00   ................
```

Figure 6. *Buffer overflow in data plane.*

The buffer overflow exception could also be illustrated with the TCP Zero Window as shown on the Wireshark result in Figure 7. The TCP Zero Window was obtained from the OpenFlow connection between the controller and the switching device. The amount of information that a computer could receive during a TCP session was defined as the size of the TCP Window. When the window size of the computer remained at zero for a given time, it was known as TCP Zero Window. This meant that the switching device could not receive more information, and the transmission of the TCP was stopped until the information was processed in its switching buffer.

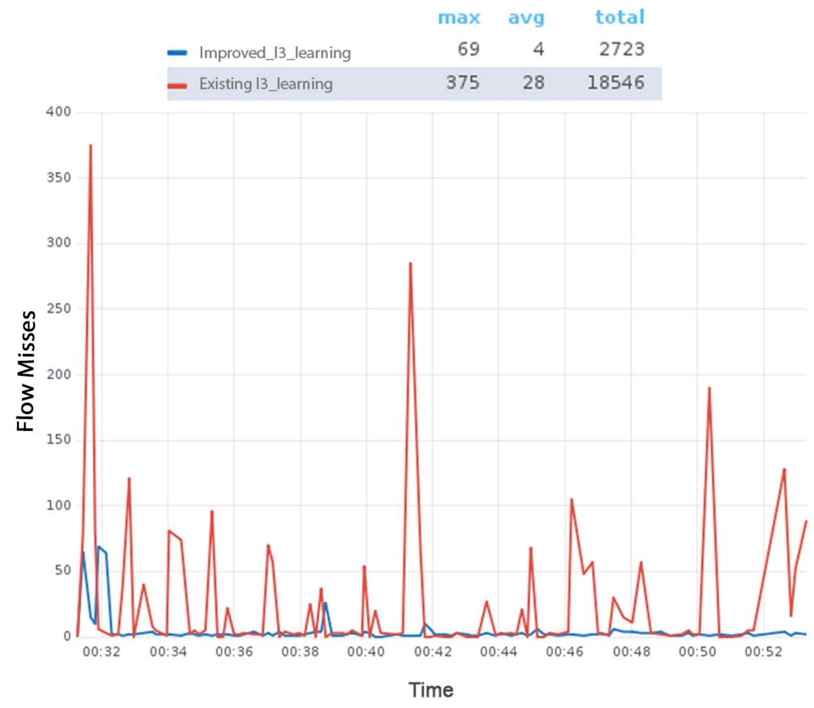


```
Filter: of

No.     Time         Source      Destination   Protoco  Lengtl  Info
115884  136.6889080( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115886  136.6905180( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115888  136.6944280( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115890  136.6981560( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115892  136.6986220( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115894  136.6991100( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115896  136.6995550( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115898  136.7000570( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115900  136.7004890( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115902  136.7009600( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115904  136.7020530( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115906  136.7025090( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115908  136.7100700( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115910  136.7106800( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115912  136.7111480( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115914  136.7116360( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add
115916  136.7121120( 127.0.0.1   127.0.0.1     OF 1.0   162  [TCP ZeroWindow] of_flow_add

▷ Frame 115912: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface 0
▷ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▷ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
▷ Transmission Control Protocol, Src Port: 6633 (6633), Dst Port: 60384 (60384), Seq: 6263497, Ack: 5574226, Len: 96
▽ OpenFlow
    version: 1
    type: OFPT_FLOW_MOD (14)
    length: 96

0040  9e 50 01 0e 00 60 00 01  02 ac 00 00 00 00 00 01   .P...`.. ........
0050  00 00 00 00 00 01 00 00  00 00 00 05 ff ff 00 00   ........ ........
0060  08 00 00 06 00 00 06 a4  fc 47 0a 00 00 05 2a 06   ........ .G....*.
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 0a 00 00   ........ ........
0080  80 00 ff ff ff ff ff ff  00 00 00 05 00 10 00 00   ........ ........
0090  00 00 00 05 00 00 00 00  00 00 00 00 00 08 00 04

type (of10.flow_add.type), 1 byte    Packets: 437574 · Displayed: 213822 (48.9%) · Dr...    Profile: Default
```

Figure 7. *TCP Zero window in Wireshark.*

Flow Misses to Controller

As explained above, before the switching device turned to a buffer overflow exception, flow misses reached a maximum of 375 packets per second as shown in Figure 8. Due to the buffer overflow exception in the switching device, the rate of flow misses was reduced when the l3_learning component was used. However, with Improved_l3_learning, the rate of flow misses was detected earlier and reduced. Since no buffer overflow occurred during the use of the Improved_l3_learning component of the POX controller, the number of flow misses was low for the whole simulation.

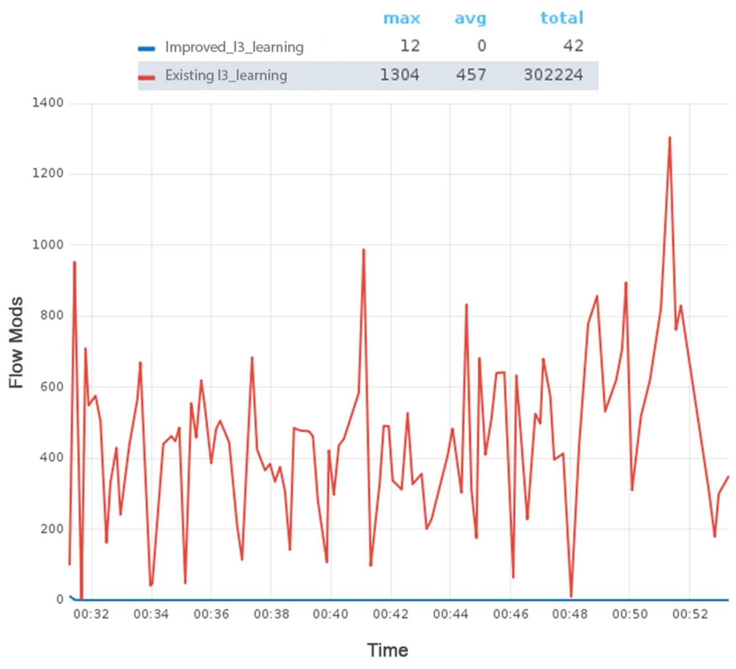


	max	avg	total
Improved_l3_learning	69	4	2723
Existing l3_learning	375	28	18546

Figure 8. *Flow misses to controller.*

Flow Modification from Controller

Due to the large number of new flows sent to the switching devices, a large number of flow rules were needed to be stored. As a consequence, the size of the switching table could be quickly flooded by DoS attacks easily. With the improved component, it was seen that the amount of flow modification messages from the controller to the switching device were reduced to normal as if there was no attack (Figure 9). The number of flow modification messages from the controller reached a maximum of 1,304 per second and 457 per second on average, which meant that the flow entries would be inserted into the switching devices. This resulted in unwanted flow table entries in the switching table. However, with the enhanced solution, the flow modification messages reached a limit of 12 per second. As a result, and on average, excessive and unnecessary flow table entries were reduced to 0.



	max	avg	total
Improved_l3_learning	12	0	42
Existing l3_learning	1304	457	302224

Figure 9. *Flows modification messages from controller.*

Maximum and Average Flow Table Entries

The number of rules stored in the switching device flow table was a very significant switching parameter since the switching device performance decreased when the number of rules in the flow table was higher. The attacker produced more than 2,000 packets per second with a random source IP address. In this case, the flow miss would be too high and this resulted in too many PacketIn messages sent to the controller requesting for the flow table modification. As shown in Figures 10 and 11, the number of rules grew over time with and without DoS attacks in the l3_learning component and the Improved_l3_learning component. The maximum and average number of flow table entries shown in Figures 10 and 11 for l3_learning reached 6,996 and 3,498 for max. and 4,817 and 2,420 for avg., respectively. However, using the Improved_l3_ learning component, the flow table entries were reduced to normal since the component attempted to delete all the unnecessary flow table entries from the attacker to give room for the switching device to store flow tables from other legitimate users.

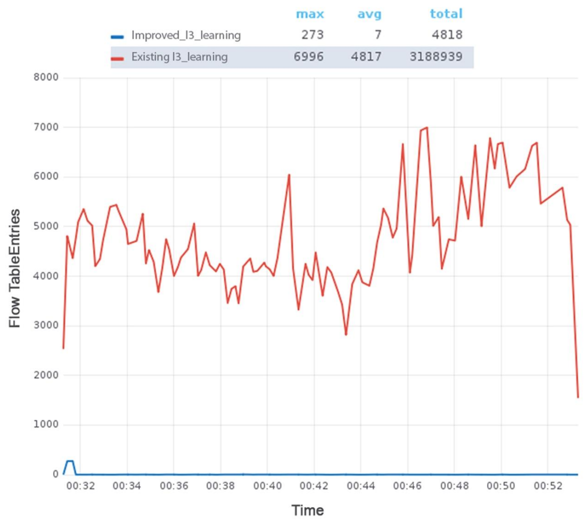


Figure 10. *Maximum flow table entries*

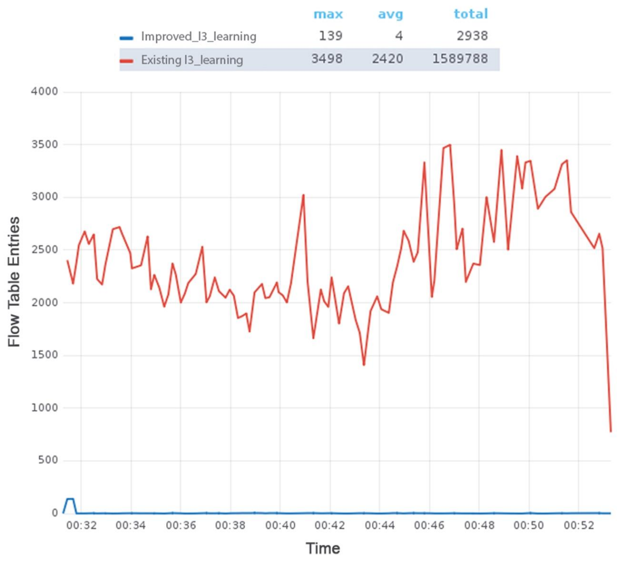


Figure 11. *Average flow table entries.*

CPU utilisation on Data Plane

CPU usage was measured when the controller was under DoS attack with the l3_learning and Improved_l3_learning components as shown in Figure 12. The DoS attack using the hping3 command had a significant impact on the increased use of the CPU. The data plane suffered from a drastic 66% rise in CPU usage. This rise in the CPU usage was due to a large number of switching device processes and packets stored in the buffer before being sent to the controller. Using the Improved_l3_learning, the increase in CPU utilization could be reduced to normal (i.e. 0% to 6%) because the data plane did not need to buffer packets and transfer them to the controller. Instead, it dropped all the attack traffic from the switching device. Before the detection and mitigation of attacks, the CPU usage was so high. Nevertheless, the proposed approach was able to prevent attacks and thus minimise excessive use of the CPU to standard.

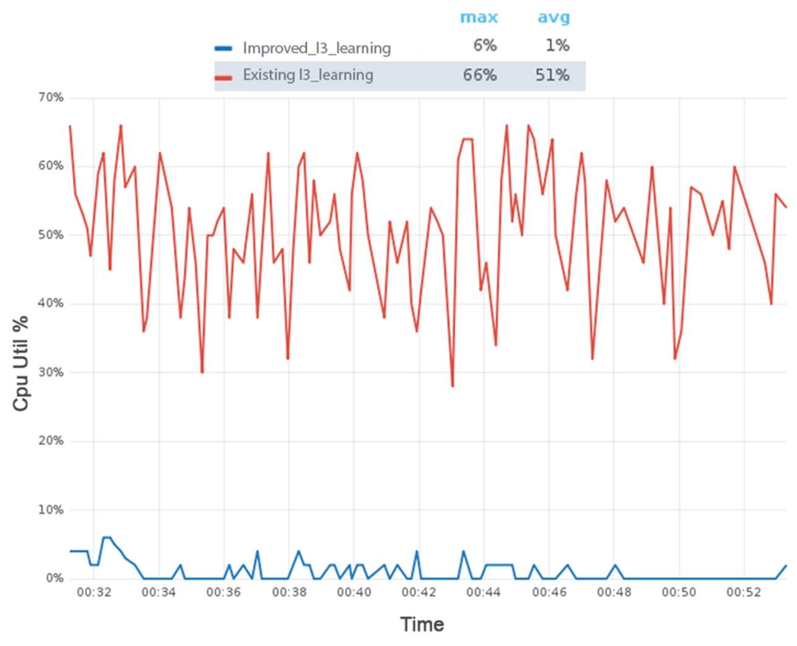


Figure 12. *CPU utilisation on data plane.*

Throughput

Source to destination bandwidth, i.e. the number of packets sent from source h100, was also measured and received at destination h200 within 6 minutes.

Before the attack, there was almost no packet loss as the bandwidth of the packets sent was equal to the packets received as shown in Figure 13. However, during the attack, with the l3_learning component, there was no route to the destination as shown in Figure 14. This was because the controller was unable to implement a new flow rule in the switching devices. With the Improved_l3_learning component, the controller was able to install new flow rules. Thus, the source host was able to communicate with the destination host. The bandwidth obtained was approximately the same as when there were no DoS attacks (by using regular ping).

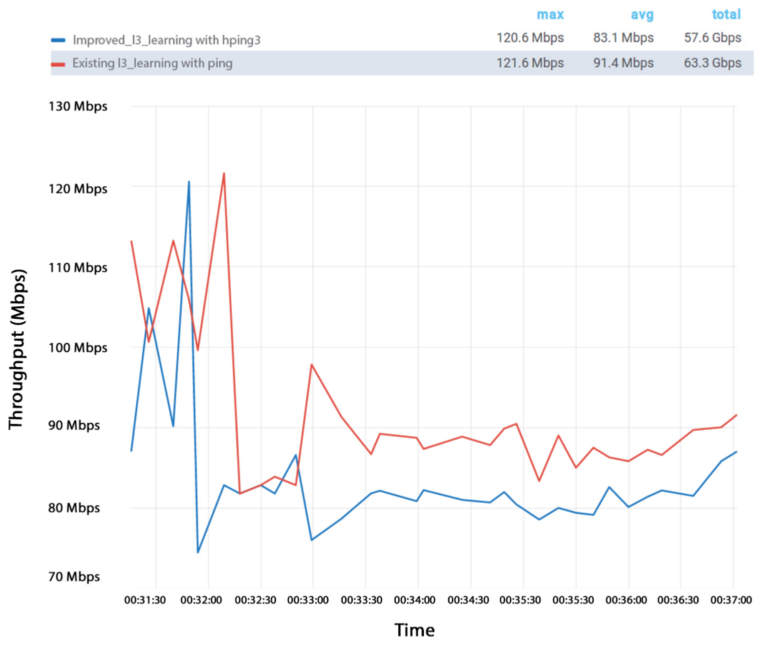
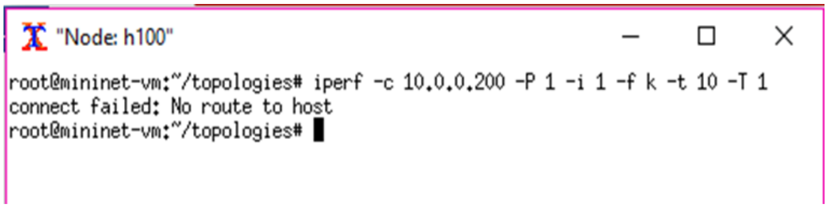


Figure 13. *Bandwidth on h100.*



```
root@mininet-vm:~/topologies# iperf -c 10.0.0.200 -P 1 -i 1 -f k -t 10 -T 1
connect failed: No route to host
root@mininet-vm:~/topologies# █
```

Figure 14. *Measured bandwidth during attack (l3_learning with hping3).*

CONCLUSION

This study aimed at detecting and mitigating a DoS attack on the SDN controller using the POX controller rate limiter packets. In addition, the effect of the DoS attack on the controller, switching devices, and bandwidth of the communicating hosts were analysed. With rapid development and prototyping, the Python-based POX controller was selected as the SDN controller platform using the l3_learning component. The proposed solution was tested in the Mininet simulation environment using the hping3 tool to build artificial DoS attacks. With this type of attack, the malicious host generated more than 2,000 packets/sec, resulting in an average of 121.4 ms of flow latency, an average of 28 of OF-messages and flow miss messages to the controller, an average of 457 flow modification messages, an average of 4,817 flow table entries, and an average of 51% CPU utilisation. As compared to the improved component, the results showed that an improvement of average flow latency of 56 ms, an average of OF-messages and flow miss messages to the controller of 4, an average of 0 unnecessary flow modification messages, an average of 7 flow table entries, and an average of 1% in CPU utilisation.

In addition, the bandwidths between the two hosts were measured under the same conditions. During the attack with the l3_learning component, the result showed that a route from h100 to h200 could not be established (i.e. “No route to the host”). This was because the switching devices were full of unnecessary flow table entries that prevented any other flow tables from being added. As a result, every packet from h100 to h200 did not reach the destination and were dropped. However, with the improved component, a route from h100 to h200 was established and the observed bandwidth was approximately equal to 80 Mbps, which was almost the same as if there were no attacks on the network. The proposed solution was thus able to detect and mitigate malicious packets to reach the controller and thus prevented the data plane (switching devices) from being filled with excessive and unnecessary flow tables’ entries and the bandwidth between the two hosts were recovered to normal.

A proposed controller component focuses on the detection and prevention of flooding (DoS) attacks on the controller and data planes. However, any malicious attacks that occurred at OF channel such as packets replication and man-in-the-middle attack are out of this scope. In the future, a possible alternative solution could be developed to combine OF channel events analysis module to systematically identify malicious events and traffic on OF channel.

ACKNOWLEDGEMENT

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

Ali, S. T., Sivaraman, V., Radford, A., & Jha, S. (2015). A survey of securing networks using software defined networking. *IEEE transactions on reliability*, *64*(3), 1086–1097. https://doi.org/10.1109/TR.2015.2421391

Benton, K., Camp, L. J., & Small, C. (2013). OpenFlow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 151–152). Hong Kong, China: ACM. https://doi.org/10.1145/2491185.2491222

Bholebawa, I. Z., & Dalal, U. D. (2016). Design and performance analysis of OpenFlow-enabled network topologies using Mininet. *International Journal of Computer and Communication Engineering*, *5*(6), 419. https://doi.org/10.17706/IJCCE.2016.5.6.419-429

Cabaj, K., Wytrebowicz, J., Kuklinski, S., Radziszewski, P., & Dinh, K. T. (2014). SDN architecture impact on network security. In M. Ganzha, L. Maciaszek, M. Paprzycki (Eds.), *Federated Conference on Computer Science and Information Systems (FedCSIS)* (pp. 143–148). Warsaw, Poland: ACSIS. https://doi.org/10.15439/2014F473

Cui, Y., Yan, L., Li, S., Xing, H., Pan, W., Zhu, J., & Zheng, X. (2016). SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks. *Journal of Network and Computer Applications*, *68*, 65–79. https://doi.org/10.1016/j.jnca.2016.04.005

Dao, N.-N., Kim, J., Park, M., & Cho, S. (2016). Adaptive suspicious prevention for defending DoS attacks in SDN-based convergent networks. *PloS one*, *11*(8), e0160375. https://doi.org/10.1371/journal.pone.0160375

Dridi, L., & Zhani, M. F. (2016). SDN-guard: DoS attacks mitigation in SDN networks. In *5th IEEE International Conference on Cloud Networking (Cloudnet)* (pp. 212–217). Pisa, Italy: IEEE. https://doi.org/10.1109/CloudNet.2016.9

Imran, M., Durad, M. H., Khan, F. A., & Derhab, A. (2019). Reducing the effects of DoS attacks in software defined networks using parallel flow installation. *Human-centric Computing and Information Sciences*, *9(1),* 16. https://doi.org/10.1186/s13673-019-0176-7

Kim, H., & Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, *51*(2), 114–119. https://doi.org/10.1109/MCOM.2013.6461195

Kreutz, D., Ramos, F., & Verissimo, P. (2013). Towards secure and dependable software-defined networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking* (pp. 55–60). Hong Kong, China: ACM. https://doi.org/10.1145/2491185.2491199

Kreutz, D., Ramos, F. M., Verissimo, P., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76. https://doi.org/10.1109/JPROC.2014.2371999

Lawal, B. H., & Nuray, A. T. (2018). Real-time detection and mitigation of distributed denial of service (DDoS) attacks in software defined networking (SDN). In *26th Signal Processing and Communications Applications Conference (SIU)* (pp. 1–4). Izmir, Turkey: IEEE. https://doi.org/10.1109/SIU.2018.8404674

Li, W., Meng, W., & Kwok, L. F. (2016). A survey on OpenFlow-based software defined networks: Security challenges and countermeasures. *Journal of Network and Computer Applications*, *68*, 126–139. https://doi.org/10.1016/j.jnca.2016.04.011

Maugendre, M. (2015). *Development of a performance measurement tool for SDN.* (Unpublished Master's thesis). Universitat Politècnica de Catalunya, Spain.

Padmaja, S., & Vetriselvi, V. (2016). Mitigation of switch-Dos in software defined network. In *International Conference on Information Communication and Embedded Systems (ICICES)* (pp. 1–5). Chennai, India: IEEE. https://doi.org/10.1109/ICICES.2016.7518925

Polat, H., Polat, O., Cetin, A. (2020). Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models. *Sustainability*, *12*(3),1035. https://doi.org/10.3390/su12031035

Proença, J., Cruz, T., Monteiro, E., & Simões, P. (2015). How to use software-defined networking to improve security - A survey. In *Proceedings of the 14th European Conference on Cyber Warfare and Security.* Hertfordshire, UK. https://doi.org/10.13140/RG.2.1.4877.1686

Scott-Hayward, S., O'Callaghan, G., & Sezer, S. (2013). SDN security: A survey. In *IEEE SDN For Future Networks and Services (SDN4FNS)* (pp. 1–7). Trento, Italy: IEEE. https://doi.org/10.1109/SDN4FNS.2013.6702553

Sen, S., Gupta, K. D., & Ahsan, M. M. (2020). Leveraging machine learning approach to setup software-defined network (SDN) controller rules during DDoS attack. In *Proceedings of International Joint Conference on Computational Intelligence* (pp. 49–60). Singapore: Springer. https://doi.org/10.1007/978-981-13-7564-4_5

Shang, G., Zhe, P., Bin, X., Aiqun, H., & Kui, R. (2017). FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications* (pp. 1–9). Atlanta, GA, USA: IEEE: https://doi.org/10.1109/INFOCOM.2017.8057009

Shin, S., Yegneswaran, V., Porras, P., & Gu, G. (2013). Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer*

& Communications Security (pp. 413–424). Berlin, Germany: ACM. https://doi.org/10.1145/2508859.2516684

Shu, Z., Wan, J., Li, D., Lin, J., Vasilakos, A. V., & Imran, M. (2016). Security in software-defined networking: Threats and countermeasures. *Mobile Networks and Applications*, *21*(5), 764–776. https://doi.org/10.1007/s11036-016-0676-x

Stancu, A., Halunga, S., Suciu, G., & Vulpe, A. (2015). An overview study of software defined networking. In *Proceedings of the 14th International Conference of Informatics in Economy.* Bucharest, Romania.

Tian, Y., Tran, V., & Kuerban, M. (2019). DoS attack mitigation strategies on SDN controller. In *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0701–0707). Las Vegas, NV, USA: IEEE. https://doi.org/10.1109/CCWC.2019.8666456

Tran, N. T., Le, T. L., & Tran, M. A. T. (2018). ODL-ANTIFLOOD: A comprehensive solution for securing OpenDayLight controller. In *International Conference on Advanced Computing and Applications (ACOMP)* (pp. 14–21). Ho Chi Minh City, Vietnam: IEEE. https://doi.org/10.1109/ACOMP.2018.00011

Wang, H., Xu, L., & Gu, G. (2015). Floodguard: A DoS attack prevention extension in software-defined networks. In *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (pp. 239–250). Rio de Janeiro: IEEE. https://doi.org/10.1109/DSN.2015.27

Wright, A. P., & Ghani, N. (2019). A testbed for the evaluation of denial of service attacks in software-defined networks. In *2019 SoutheastCon* (pp. 1–6). Huntsville, AL, USA: IEEE. https://doi.org/10.1109/SoutheastCon42311.2019.9020433

Xia, W., Wen, Y., Foh, C. H., Niyato, D., & Xie, H. (2015). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, *17*(1), 27–51. https://doi.org/10.1109/COMST.2014.2330903

Xie, J., Guo, D., Hu, Z., Qu, T., & Lv, P. (2015). Control plane of software defined networks: A survey. *Computer Communications*, *67*, 1–10. https://doi.org/10.1016/j.comcom.2015.06.004

Zhang, P., Wang, H., Hu, C., & Lin, C. (2016). On denial of service attacks in software defined networks. *IEEE Network*, *30*(6), 28–33. https://doi.org/10.1109/MNET.2016.1600109NM