



---

## Software defined network approach for layer II based distributed firewall

Ahmed Mahmud<sup>1</sup>, Kabiru Ibrahim Musa<sup>2</sup>, Usman M. Joda<sup>3</sup>

<sup>1</sup>ICT Unit, Bauchi State University, Gadau.

<sup>2</sup>Department of Management and Information Technology, Faculty of Management Sciences, Abubakar Tafawa Balewa University, Bauchi.

<sup>3</sup>Department of Mathematical Sciences, Faculty of Sciences, Bauchi State University, Gadau

Email: [mahmud.ahmed@basug.edu.ng](mailto:mahmud.ahmed@basug.edu.ng)

---

### Abstract

Controlling and managing networks has become a highly complex and specialized activity. Network operators are struggling to cope with integration of different types of networks, while meeting the challenges of increasing traffic. The traditional network tends to be rigid. Once the forwarding policy has been defined, the only way to change it is by changing the configuration of all the affected devices. In this context Software Defined Networking (SDN) is being looked upon as a promising paradigm that has the power to change the way networking is done. By centralizing control, and making forwarding nodes simple, SDN offers flexible control over the traffic flows and the policies networks use to manage these flows. Along with the excitement, there have been apprehensions regarding SDN. The perceived risks associated with SDN have prevented faster adoption so far. Spurious traffic flows can affect switches and controllers alike. An attacker, with malicious intent, can build up the infructuous flows to such an extent as to seriously overload the switches and the controller thus, led to a Denial of Service (DoS) attack. In this paper we propose a Layer II firewall for preventing such intrusions using SDN traffic flow intrusions. The Layer II firewall would not only detect the intrusions but also provide some degree of protection to the network devices the moment an attack is detected. Furthermore, it will completely prevent undesired traffic from transmitting data on the network. Thereby making the network more robust and reliable. The proposed technique uses an algorithm that control traffic variation of flow table entries and subject them to a set of rules based on source and destination MAC address in the flow header. The experimental result using Mininet virtualized network environment has shown that our proposed technique has improved performance against results obtained using an IP based firewall in terms of bandwidth, round trip time (RTT) and latency variation while preventing attack on the overall networks traffic.

**Keywords:** Software Defined Networking, Firewall, Bandwidth, Round Trip Time.

---

### 1. Introduction

Networks of the twenty first century offer immense flexibility to the business and individual users, but at the cost of higher complexity. Management and control of these type networks have become very complex and difficult activities that require specialisation. In this regard SDN is being

considered as a promising approach that has the potential of changing the networking world (Nadeau & Gray, 2013). SDN is a new networking paradigm that is introduced with the sole aim of simplifying the network management by separating the data and control planes. SDN has opened up the possibilities of

programmability in the network control plane (Azodolmolky, 2013). The separation of the control logic from networking devices, such as switches and routers, in traditional networks to a centralized unit known as the controller permits the physical network hardware to be detached from the control Plane. SDN architecture will provide a set of Application Programming Interfaces (APIs) that simplifies the implementation of some of the network services (for example, routing, multicast, security, access control, bandwidth management, traffic engineering, Quality of Service, energy efficiency, and various forms of policy management (Pujolle, 2015).

It is evident that not every small change needs to come at the cost of reconfiguring all the network devices (Nadeau & Gray, 2013). The data plane consists of devices that contain tables of flow entries. The network hardware uses the secured transport layer protocol to securely communicate with the controller about new entries that have not been populated in the flow table. Each flow entry includes matching rules and actions that guide the data plane on the action to take for the matched flow. When a packet arrives at a switch the header fields will be matched against the matching rules available in the flow table and if a match is found the action specified by the flow entry will be taken by the switch otherwise the packet header will be forwarded to the controller for further process. The controller then processes the header and constructs a flow rule to be installed in the flow tables of the switches along the chosen path (Azodolmolky, 2013). According to Khondoker (2018) Layer II switch is a type of network switch or device that works on the data link layer (OSI Layer 2) and utilizes MAC Address to determine the path through where the frames are to be forwarded. It uses hardware-based

switching techniques to connect and transmit data in a local area network (LAN).

The centralized structure of the controller could lead to many undesirable security challenges. One of such critical challenges is the impact of traffic flow intrusion on the SDN networks. These can affect switches and controllers alike. An attacker, with malicious intentions, can generate infructuous flows to such an extent as to seriously overload the switches and the controller and create a denial of service (DoS) attack (ref). At the arrival of each of the attack packets a new flow rule needs to be created and therefore the switch will need to store the packets in its memory and forward the header field to the controller. However, receiving a large number of attack packets will use up a lot of switch memory and eventually install many new flow entries in the flow tables. Thus, causing the depletion of memory and slow down the flow table lookup very quickly, and in the worst-case scenario, it may bring down the switch. Meanwhile all these packets from all over the network will be forwarded to the controller for processing. The large volume of packets sent to the controller with each of them consuming part of the memory and the processing power of the controller may also eventually break down the controller (Azodolmolky, 2013). Existing techniques did not fully address the issue of flow attacks in relation to firewall traffic control in distributed network system thus, the need for more solution that can be deploy in solving the problem. Traffic control algorithms have been introduced under different traffic pattern to monitor the network loads and to identify the exact attack paths and prevention techniques.

The remaining part of this article is organized as follow: Section 2 contains the review of existing literatures. Section 3

presents the need for traffic flow prevention technique. Section 4 provides the technique and its simulation modelling. Section 5 presents the performance evaluation and experimental results analysis. Finally, Section 6 concludes the research findings.

## 2. Related works

Exhaustive researches have been carried out for firewall traffic control using SDN. Waheed and Mufarrej (2017) proposed and implemented an SDN firewall for Network virtual function. The technique implemented by the researchers involved a single firewall and multiple firewalls to test and evaluate performance. User Datagram Protocol (UDP) traffic was used to test and evaluate the performance of the proposed firewall. However, the authors used IP address headers of the packet for the source and destination switches to deliver flows. This is susceptible to IP address spoofing and may lead to undesirable vulnerabilities in the network. Othman and Ammar (2017) implemented a POX controller based SDN firewall and analysed performance of the firewall. The research proposed a methodology for the implementation of the firewall based on the packets headers and matches them against a predefined rule in the firewall policies. They used IP address headers of the packets to match rules on the firewall policies. This technique has been inefficient due to security vulnerabilities associated with IP address matching such as IP address spoofing which could lead to denial of service and subsequently total network breach.

Morzhov and Nikitinskiy (2018) conducted a research to determine various ways to resolve anomalies in an SDN firewall. They analysed a network application designed for SDN controller. The network application was developed by

the researchers and called it PreFirewall. However, the proposed technique did not take into consideration rules set of rules that are presented in managing ports with their respective IP addresses. This makes it difficult if not impossible to implement a rule set based firewall for the SDN.

Also, Kumar and Lung, 2017 implemented a MAC address technique to filter openflow traffic. In the results obtained, the researchers were able to show that MAC based approach outperforms IP based approach. However, the researchers were not able to test the performance of the firewall with TCP and UDP traffic. This greatly affects the for network administrators to determine the usability of the scheme.

Krongbaramee and Somchit (2018) conducted a research by implementing a stateful firewall application using open Vswitch. The results obtained indicates that the firewall works effectively by block unwanted openflow traffic but with some overhead load on the switches. The limitation of this study is that the researchers did not test the scheme with multiple topologies to determine scalability and protection against SYN FLOOD attacks.

Uddin and Monir (2019) conducted a research to analyse the performance of two SDN Firewalls- POX and ODL. Results from the experiment show that the two firewalls performed the task of allowing traffic or denying it based on the rules installed in the application. The results however show evidence of data loss at time intervals. Also, the results indicate a very low transmission rate.

Ali, Imam & Kaysi (2016) proposed security architecture for the SDN networks by investigating the control plane with the possibility of inserting a proposed security

plane between the data and control plane. The proposed technique aims for an early and fast detection of an attack before the controller or host is completely swamped with a large number of malicious packets. More so, the proposed technique aims for an early and fast detection of an attack before the controller or host is completely swamped with a large number of malicious packets. The architecture has been implemented for both clients and server on a virtual network. The sever network security policies were implemented using pyretic programming language and the client policies were implemented using pox- which offers rich python APIs for network programming. However, despite the successes achieved, the technique failed to include the implementation and testing of the policy insertion, sending commands to the controller in real time to trigger (open virtual switch database) OVSDB in order to insert new policies onto the flow table.

Similarly, a study by Seungwon, Lei, Sungmin, and Guofei (2016) systematically investigated the opportunities and challenges on how SDN can benefit network security. To this end, they reviewed the new features provided by SDN, then study how these features can enhance on specific security functions. They opined that controlling network flows dynamically provides many new possibilities in network security functions. Second, they argued that SDN enables users to separate malicious (or suspicious) network flows from benign ones dynamically. This ability is quite useful when it is to differentiate security services. Also, they illustrated that if we apply SDN, we can simply build this function by controlling network flows dynamically. Further, the study investigated network programmability of SDN. In this regard, the study stated that network programmability feature of SDN offer the

possibility of programming network security functions easily. This Programming network security applications, according to the study is very useful and cost effective, because there will not need to buy additional hardware boxes or software programs to deploy network security services, but create and deploy network security applications running on a controller. The study however, did not implement any of the techniques in the development environment neither in real scenario.

Yang, YuNan, & Wei (2014) analysed security threats to SDN networks and came up with an improved framework for an SDN network they called network intrusion detection system (NIDS). The framework is based on the following rules which are part of the IP address and port number of the receiver.

```
alert tcp 192.12.1.32 :80 -> 192.168.1.12  
111
```

The first item is the conduct of the rule. It defines how the package should be handled when the package matches the rule. There are five default handling: alert, log, pass, activate and dynamic.

The second item is the agreement. The agreements that Snort can analyze are TCP, UDP and Internet Control Message Protocol (ICMP).

The third item is the IP address of the sender. It can be some IP or a range of IP address. It can be presented as 192.12.1.3:192.12.1.15.

The fourth item is the port number of the sender. It is just like the IP. There are three types of expression in the rule set. 12 means the port 12. :80 means the port number can't be greater than 80. Similar, 80: means the port number can't be less than 80. The last one is 80:100, it means

the port number must be between 80 and 100.

The fifth item is the direction operator. One-way operation (->) represents the flow direction of the data package. Two-way operation (< >) represents that the system should record and analyse the bi-directional data transferred between the two hosts.

The framework consisted of three subsystems: capture and parsing, detection and log and alarm subsystem. Experiments were conducted and results were obtained. However, the technique didn't take into consideration prevention techniques; rather it detected intrusions after it has already occurred. The system was implemented using openflow and floodlight which are not quite compatible. Also, IP addresses are used which can be unreliable if DHCP (Dynamic Host Configuration Protocol) is enabled in the network. Another study by Sezer, S., Scott-Hayward S., & Pushpinder K.C., (2013) analysed issues in security policy implementation in SDN networks. The study identified three (3) approaches to SDN security policy implementation as follows:

**Network forensics:** Facilitate quick and straightforward, adaptive threat identification and management through a cycle of harvesting intelligence from the network, analysing it, updating policy, and then reprogramming to optimize from network experience.

**Security policy alteration:** Allow you to define a security policy and have it pushed out to all the infrastructure elements, reducing the frequency of misconfiguration and conflicting policies across the infrastructure.

**Security service insertion:** Facilitate security service insertion where applications like firewalls and intrusion

detection systems (IDSs) can be applied to specified traffic according to the organization's policies. However, despite a detailed analysis of implementation issues including security considerations the technique does not scale for validation on various traffic attack issues and solutions.

Shirali-Shahreza and Ganjali (2013) Proposed FleXam which is a flexible sampling extension for OpenFlow designed to provide access to packet level information at the controller. The technique explored three information channels for OpenFlow specification which are:

**Event-based Messages:** Event-based messages are sent by the switches at events such as state change of a link or port, and usually deliver information about changes in network structure and topology;

**Flow Statistics:** Flow statistics (received packets, received bytes and duration in the current specification) are collected by the switches and pulled by the controller. This is the only way for the controller to collect information about active flows and

**Packet-in Messages:** A switch may send a packet-in message either because it did not know what to do with the packet – no matching entry found in the flow table – or as a result of a send-to-controller action in the matching flow entry. The switch may buffers the original packet and only includes part of the packet – usually the first 128 bytes – in the packet-in message. The proposed FleXam allows developers to implement security applications that need packet level data with low overhead. As a result, the application could be run directly on the controller for small networks, eliminating the need for additional monitoring machines. More complex applications for larger networks can be implemented with the help of distributed monitors, at a fraction of the

overhead of existing solutions, where all flow has to be directed to a monitor. However despite the detailed analysis of the framework, it failed to provide implementation details and experimental results.

Teenu and Jincy (2015) conducted a survey on SDN security mechanisms. The authors classified SDN security vulnerabilities into 3 categories. Control plane specific which includes attack cases against SDN control and application layer. Control channel specific traffic which includes attacks targeting to the interface (OpenFlow). Data plane specific traffic which includes the attack on network devices. The study further outlined mechanisms for security issues in SDN which include modular libraries provided by the APIs to program security policies for the SDN network. Another mechanism proposed by the study is the use of data link layer protocol. This involves applying Institute of Electrical and Electronics Engineers (IEEE) 802.1X standard and Extensible Authentication Protocol (EAP). A virtual router sends an authentication request, standardized by IEEE 802.1X (ref), and POX controller redirects it to the Authenticator (ref). The Authenticator checks the credentials of the Supplicant against a server, running the authentication method defined in EAP. If credentials are correct, the Authenticator sends a success message for Supplicant host and sends an authorization and confirmation message for POX via a SSL secure channel. The study gave an insight into existing mechanisms for handling SDN security issues. However, the study did provide guideline on how the technique was implemented which is very important in deploying security solutions for SDN networks.

Also, Govindarajan, Meng, & Ong (2013) involves a comprehensive literature review

on SDN research topics, the challenges and solutions. The study identified biggest security challenge is to protect the controller which has more intelligence for controlling the data planes. The other securities challenges identified by the study reside in the SDN based networking environment are protecting Distributed Denial of Service (DDoS) attacks and intrusion prevention. The study proposed that monitoring component is employed with active query and passive listening mechanism to aggregate the network information. It intercepts the control messages to acquire the global view of the network information. If the packet received by the switch does not match with flow table entries it sends a message to the controller, it replies to the switch for installing forwarding rule using Flow message. The switches send the Flow remove message to the controller, once the flow time is expired. Moreover, it uses the OpenFlow Read State message to know the network resource utilization. The study provided insight into mechanism for implementing security policy for an SDN network but failed to provide implementation details for such mechanisms which is important for deploying security solutions for an SDN network.

The work of Yunchun and Jutao (2015) proposed a method based on SDN network architecture for internet protocol security (IPSec) automatic configuration management to solve SDN security problems. The mechanism is based on Access authentication module is used to certify whether OpenFlow switch (IPSec Virtual Private Network VPN client application device) is SDN controller certified device, and whether it is IPSec VPN client application device certified by IPSec VPN. And it coordinates with a certification server to certify client applications. The module monitor IPSec

tunnel information, running state, the topology and performance indicators that are established by Open Flow switch (IPSec VPN client application). And the module can control IPSec tunnel connection and close. IPSec tunnel information includes the total number of IPSec tunnels, the number of remote access users, tunnel consultation mode, encryption mode, etc. Performance indicators include the number of messages received and sent by IPSec tunnel, flow/rate, and receiving/sending packets. If there is any IPSec tunnel abnormal connection, it alarms and shuts down the tunnel. Though the research proposed an impeccable security, it failed to provide data for performance evaluation and details of implementation which is paramount for deploying SDN security solutions.

Similarly, Jin and Nicol (2015) proposed a parallel simulation engine for an SDN network based on:

**Entity:** This is the base class that represents a simulation entity. Simulation experiments can actually be viewed as interactions among a number of entity objects. An entity object is a container of simulation state variables and instances of other simulation objects, such as OutChannels and InChannels.

**InChannel:** represents the endpoint of a directed communication link between entities and

**Message:** is the base class that represents events sent between entities through the communication channels.

They developed a large-scale network simulation/emulation testbed. The system architecture was extended for network testbed to support OpenFlow-based SDN simulation experiments. The study proposed an algorithm for global

simulation/emulation. The study gave a concise mechanism for simulation/emulation of SDN network environment. However, it did not take security issues into consideration which cannot be ignored in deploying SDN applications.

The inefficiency of the existing techniques in solving the challenge of network traffic attack had called for an ideal solutions that is promising.

### 3. The Need for Layer II Firewall

From the related literature, this paper has deduced that there is an existing security threat for SDN networks. This is because, as stated in the work of Seungwon et al. (2016) and Ali et al. (2016), it is possible to programme SDN network policies. This clearly gives hope for the possibility of programming security policies for SDN though no implementation details or test scenarios were conducted by the researchers. The study by Yang et al. (2014) analysed the security threat of SDN and proposed a based on subjecting every traffic to a set of rules based on matching pairs of IP addresses of the sending and receiving nodes on the network topology. This is highly inefficient if DHCP is enabled in the network. The work of Teenu and Jincy (2015) categorized security vulnerabilities into: control plane specific, control channel specific and data plane specific. The study further outlined mechanisms for security issues in SDN which include modular libraries provided by the APIs to program security policies for the SDN network using python scripts. The study by Gupta (2013) profiled security threats in SDN networks to include but not limited to: spurious traffic flows; attacks on vulnerabilities that exist in the switches; attacks on control plane communications. Therefore, the layer II firewall algorithm is

design in order prevent spurious traffic flows in an SDN network.

#### 4. Design of Layer II Firewall

The simulation and testing of the proposed method for spurious traffic flow prevention is explained through the following sections. The algorithm is implemented on the python based pox controller in the Mininet virtualized network environment (ref). The process is describe in the follows steps:

In the first step, the overall algorithm behaviour in traffic flows attacks under different legitimate UDP, ICMP and TCP traffic patterns were observed. Two traffic patterns are tested for the legitimate traffic

running in the network and the algorithm behaviour is observed under different traffic loads for the stated testbeds. Finally, more detailed analysis has been performed on the effectiveness of the algorithm in identifying the exact attack paths and prevention techniques. However, this may lead to inefficient MAC address rules on traffic flow over the network. Therefore, we introduce another subroutine stargy that can create a hash table for storing keys, values, and source MAC addresses. The general procedure of the proposed algorithms can be summarized in various steps as shown in Table 1 while the overall operational flowchart is shown in Figure 1.

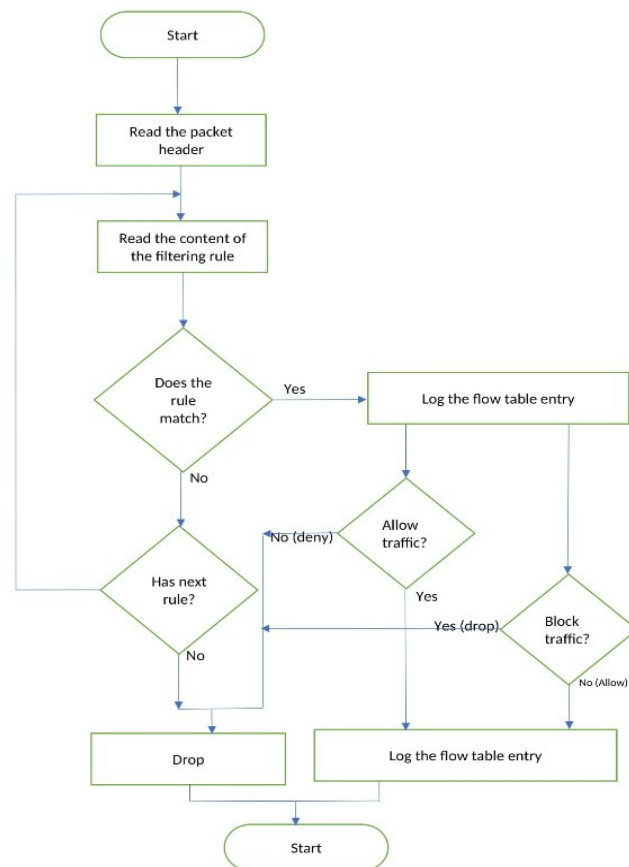


Figure 1: FlowChat of the Firewall application



Table 1: Algorithm of the Firewall application

Steps	Description
<b>Traffic Flow</b>	<b>Input:</b> (i) Use source address and switch port to update address/port table (ii) Check source MAC address against traffic flow rules <b>Process:</b> (i) Is transparent = False and either Ethertype is LLDP (Link Layer Discovery Protocol) or the packets destination address is a Bridged Filtered address? If so Drop (ii) Is destination multicast? If so FLOOD (iii) Is output port the same as input port? If yes DROP <b>Output:</b> (i) Install flow table entry in the switch so that this flow goes out the appropriate port. Send the packet out appropriate port
<b>Traffic Pattern</b>	<b>Input:</b> (i) Create a hash table for storing (keys, values) pairs (ii) Table maps (Switch, Source MAC address) to true or false <b>Process:</b> (i) Create a hash table for storing (keys, values) pairs (ii) Table maps (Switch, Source MAC address) to true or false <b>Output:</b> (i) Controller will decide to forward traffic if: (ii) There is a flow entry that maps to true

## 5. Performance Evaluation

To evaluate the performance of our proposed firewall algorithm, extensive simulations experiments have been carried out in Mininet (2016) is a tool used to simulate the SDN, allowing a simple and quick approach to create, interact and customize prototypes for SDN. The general description of the simulation setup is presented in Table 2. The performance comparison of our technique is estimated in terms of Round-Trip Time (RTT), Bandwidth and Latency Variation. The experiments are performed by varying number of packets dropped in order to evaluate the efficiency and as well the effectiveness of our proposed algorithm. First, we will set up our topology and test the connectivity between nodes in the simulated network environment. POX is

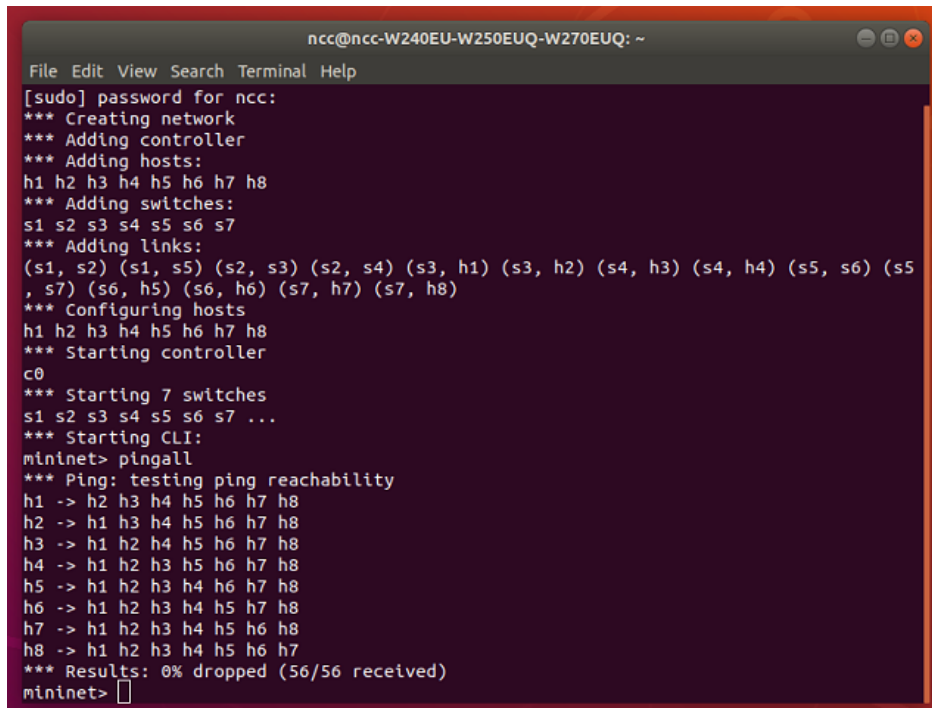
used as the controller to test this topology. After running the network topology, the controller should start to communicate with the network nodes to enable the required technology and set proper flows of traffic up for the evaluation. SDN utilities such Ping and Iperf has been used to measure response time throughput, latency for both TCP and UDP traffic. In the following section the experimental results of the proposed algorithm are discuss and analysed. Simulation setup is presented in Table 2. Ping utility will be used to measure response time.

## 6. Results Analysis and Discussion

This section presents the experimental results from the firewall experiment. Figure 2 shows the successful creation of

the topology. Seven switches and eight hosts were connected forming the desired

spanning topology (more discussion on the spanning topology).



```
ncc@ncc-W240EU-W250EUQ-W270EUQ: ~
File Edit View Search Terminal Help
[sudo] password for ncc:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7
*** Adding links:
(s1, s2) (s1, s5) (s2, s3) (s2, s4) (s3, h1) (s3, h2) (s4, h3) (s4, h4) (s5, s6) (s5, s7) (s6, h5) (s6, h6) (s7, h7) (s7, h8)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 7 switches
s1 s2 s3 s4 s5 s6 s7 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> █
```

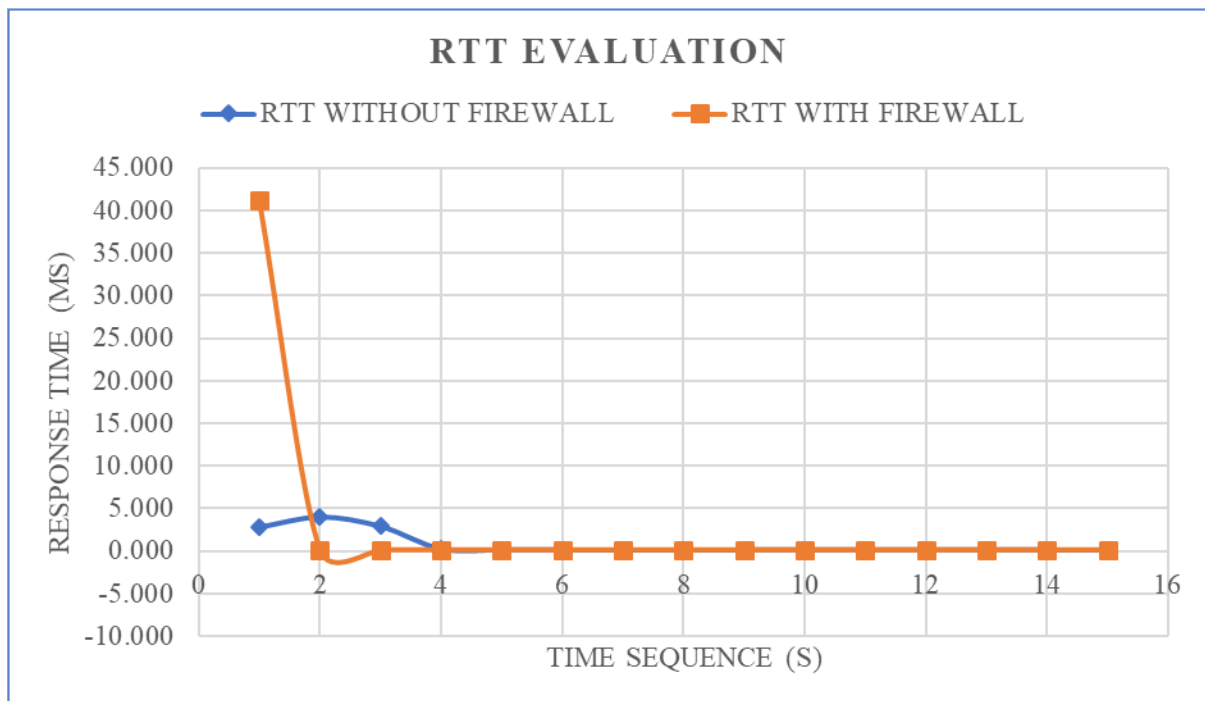
Figure 2: Creation of Mini net topology through CLI

Based on the firewall rules installed in our flow table entries, we know that h2 can ping h3. The figures below ICMP messages in 15 sequences in three streams and got responses. The RTT reduced tremendously after the first two sequences. We consider sending traffic to establish connection between two hosts by installing appropriate forwarding rules on the switches and the openflow controller. This involves exchange of openflow messages between as well as the controller updating flow table entries (Nadeau & Grey, 2013). The first few sequences involve the

learning process, because at this stage, there are no rules installed on the flow table. This accounts for the initial delay in the round-trip time from our experiment depicted in figure 2. After the initial delay, it can be seen that the RTT is almost the same for both cases of with and without the firewall (Wajdy & Hao, 2017). This means that our firewall rules in the application do not negatively affect the network performance as shown in table 3 below unlike the results obtained in the work of (Uddin & Monir, 2019) where there was a remarkable loss in data and response time.

**Table 2: ICMP Traffic from h2 to h3 with and without the firewall application running**

ICMP_SEQ	RTT WITHOUT FIREWALL	RTT WITH FIREWALL
1	2.820	41.133
2	4.036	0.130
3	2.928	0.124
4	0.244	0.124
5	0.141	0.117
6	0.126	0.123
7	0.117	0.122
8	0.120	0.125
9	0.126	0.122
10	0.141	0.127
11	0.124	0.128
12	0.128	0.120
13	0.129	0.123
14	0.131	0.126
15	0.087	0.115

*Figure 3: ICMP Output with and without the firewall application running*

To measure bandwidth, iperf utility has been used. As we did earlier with TCP test, we configured two as TCP hosts that are known to communicate with each other

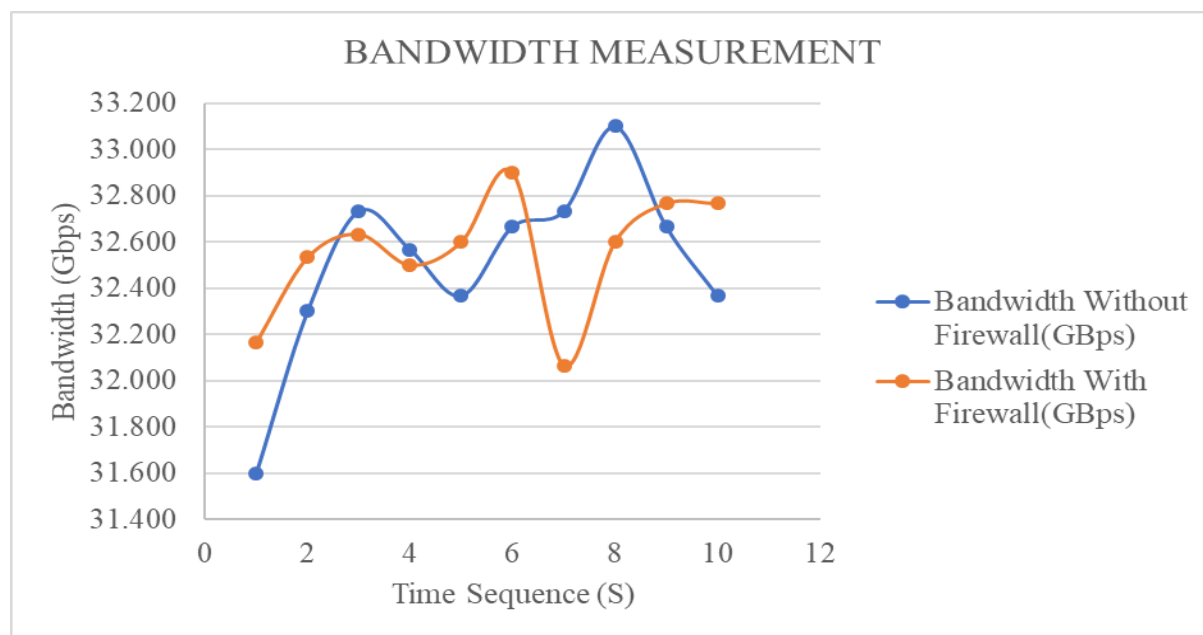
according to our firewall rules as server and client respectively. Then we initiated TCP traffic in three streams with the same TCP window size. The bandwidth

measured for the experiment without our firewall rules reaches a peak of 33.100 in the 7<sup>th</sup> sequence. This is higher than the bandwidth for all the sequences with firewall rules installed. To be able to measure TCP traffic is step further against the work of (Kumar & Lung, 2017) where

TCP and UDP could not be initiated to test the firewall application. Also, it is always a trade-off between security and performance (Nadeau and Grey, 2013). Thus, our firewall performs less in terms of bandwidth evaluation as shown in Figure 3.

**Table 3: TCP Traffic from h1 to h2 with and without the Firewall application running**

Time Interval (S)	Bandwidth without Firewall (GBps)	Bandwidth with Firewall (GBps)
0.0 - 1.0	31.600	32.167
1.0 - 2.0	32.300	32.533
2.0 - 3.0	32.733	32.633
3.0 - 4.0	32.567	32.500
4.0 - 5.0	32.367	32.600
5.0 - 6.0	32.667	32.900
6.0 - 7.0	32.733	32.067
7.0 - 8.0	33.100	32.600
8.0 - 9.0	32.667	32.767
9.0 - 10.0	32.367	32.767



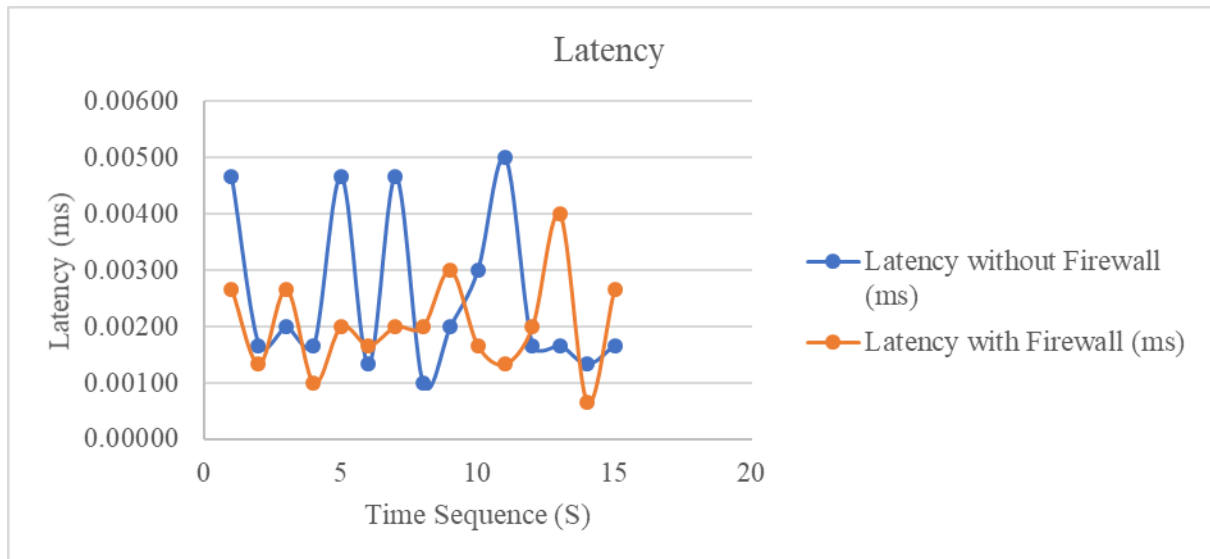
*Figure 4: TCP Output with and without the Firewall Application running*

Latency variation (jitter) delay according to Khondoker (2018) is the amount of time it takes it takes to transmit data from a source to a destination and it varies over time. To measure latency, we again configured two nodes, h1 and h2 as UDP client and server respectively. Three UDP

streams were initiated and measure the latency with the same parameters. From our experimental results shown in Figure 5 and Table 4, the network running our firewall application takes a little less time to respond than without the firewall rules applied.

**Table 4: UDP Traffic from h1 to h2 with and without the Firewall Application running**

Time Interval (S)	Latency without Firewall (ms)	Latency with Firewall (ms)
0.0 - 1.0	0.00467	0.00267
1.0 - 2.0	0.00167	0.00133
2.0 - 3.0	0.00200	0.00267
3.0 - 4.0	0.00167	0.00100
4.0 - 5.0	0.00467	0.00200
5.0 - 6.0	0.00133	0.00167
6.0 - 7.0	0.00467	0.00200
7.0 - 8.0	0.00100	0.00200
8.0 - 9.0	0.00200	0.00300
9.0 - 10.0	0.00300	0.00167
10.0 - 11.0	0.00500	0.00133
11.0 - 12.0	0.00167	0.00200
12.0 - 13.0	0.00167	0.00400
13.0 - 14.0	0.00133	0.00067
14.0 - 15.0	0.00167	0.00267



*Figure 5: UDP Output with and without the Firewall Application running*

## 7. Conclusion

This paper focuses on design and development of a Layer II firewall approach based on SDN technologies. A new algorithm has been developed and integrated into the technique that has been proposed to protect the SDN switches and controller from the destructive and undesirable effects of spurious traffic flows by adding a lightweight prevention mechanism at the controller. In evaluating the performance of our firewall application, it was discovered that the bandwidth measured for the experiment without our firewall rules reaches a peak of 33.100 in the 7<sup>th</sup> sequence. This is higher than the bandwidth for all the sequences with firewall rules installed. Though the performance in terms of bandwidth is lower when our Layer II firewall is running, it is not a bad trade-off considering the Layer II SDN firewall. The new approach relies on the fast data plane to process majority of the traffic. After testing all cases, we were able to demonstrate the ability of our firewall to filter traffic based on MAC addresses of hosts installed on our flow table entries. Also, the latency measured without our firewall application is significantly lower than the latency measured without the firewall application running. Therefore, the proposed algorithm can effectively replace expensive physical firewall layer II firewalls.

### Conflict of interest statement

The authors declare that there is no any conflict of interests

### References

Ali, H., Imam H, E., A, C., & Kaysi, A. (2016). SDN Security Plane: An Architecture for Resilient Security Services. *2016 IEEE International*

*Conference on Cloud Engineering Workshops*, 54-59.

Azodolmolky, S. (2013). *Software Defined Networking with OpenFlow*. Mumbai: Pact Publishing.

Govindarajan, K., Meng, K. C., & Ong, H. (2013). A Literature Review on Software-Defined Networking Topics, Challenges and Solutions. *IEEE 2013 Fifth International Conference on Advanced Computing (ICoAC)*, 293-299.

Gupta, L. (2013). SDN: Development, Adoption and Research Trends. *3rd Generation Partnership Project*, 1-10.

Hu, F. (2014). *Network Innovation through OpenFlow and SDN: Principles and Design*. Boca Raton: CRC Press.

Jin, D., & Nicol, D. M. (2015). Parallel Simulation and Virtual-Machine-Based Emulation of Software-Defined Networks. *ACM Transactions on Modeling and Computer Simulation*, 26 (1), 1-8.27.

Keti, F., & Askar, S. (2015). Emulation of Software Defined Networks Using Mininet in Different Simulation Environments. *IEEE Computer Society*, 4, 205-210.

Khondoker, R. (2018). *SDN and NFV: Security Analysis of Software-Defined*. Darmstad: Springer.

Krongbaramee, P., & Somchit, Y. (2018). Implementation of SDN Stateful Firewall on Data Plane using Open



- vSwitch. *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. IEEE.
- Kumar, A., & Srinath, N. K. (2016). Implementing A Firewall Functionality For Mesh Networks Using SDN Controller. *IEEE*, 168-173.
- Kumar, P. R., & Lung, C.-H. (2017). Investigation of Security and QoS on SDN Firewall Using MAC Filtering. *2017 International Conference on Computer Communication and Informatics (ICCCI -2017)*. Coimbatore, INDIA: IEEE.
- Martin, J., Rye, E., & Beverly, R. (2016). Decomposition of MAC Address Structure for Granular Device Inference. *ACM*, 78-88.
- Mininet. (2016, November 20). Retrieved from Mininet: <http://mininet.org/>
- Morzhov, & Nikitinskiy. (2018). Development and Research of the PreFirewall Network Application for Floodlight SDN Controller. *IEEE Moscow Workshop on Electronic and Networking Technologies (MWENT)*.
- Nadeau, T. D., & Grey, K. (2013). *SDN: Software Defined Networks*. Cambridge: O'reily.
- Nadeau, T., & Gray, K. (2013). *SDN: Software Define Networks*. Sebastipol, CA: O'Reily.
- Open Network Foundation. (2016, September 13). Retrieved from Open Network Foundation website: [www.opennetworking.org](http://www.opennetworking.org)
- Othman, & Ammar. (2017). Implementation and Performance Analysis of SDN Firewall on POX Controller. *2017 9th IEEE International Conference on Communication Software and Networks*, 1461-1466.
- Pena, J. G., & Yu, W. E. (2014). Development of a Distributed Firewall Using Software Deefined Networking Technology. *IEEE* , 449-452.
- Pox, R. (2016, September 13). Retrieved from Pox Repositories: [www.github.com/noxrepo/pox](http://www.github.com/noxrepo/pox)
- Prete, L. R., Schweitzer, C. M., Shinoda, A. A., & Oliveira, R. L. (2014). Simulation in an SDN network scenario using the POX Controller. *IEEE*, 978-1-4799-4340-1.
- Pujolle, G. (2015). *Software Networks: Virtualization, SDN, 5G and Security*. London: John Wiley & Sons Inc.
- Seungwon, S., Lei, X., Sungmin, H., & Guofei, G. (2016). Enhancing Network Security through Software Defined Networking (SDN). *IEEE Journal*, 978-1-5090-2279-3/16.
- Sezer, S., Scott-Hayward, S., & Pushpinder Kaur Chouhan. (2013). Are We Ready for SDN? Implementation Challenges for Software-Defined Network. *IEEE Communications Magazine*, 36-43.

- Shirali-Shahreza, S., & Ganjali, Y. (2013). Efficient Implementation of Security Applications in OpenFlow Controller with FleXam. *IEEE Computer Society*, 49-54.
- Switch, B. (2017, January 1). *Secure and Resilient SDN with Big Cloud Fabric*. Retrieved from Big Switch networks:  
[http://go.bigswitch.com/rs/974-WXR-561/images/BCF-White-Paper-Secure%20and%20Resilient%20SDN-2.pdf?\\_ga=1.112867725.86193620.1448934819](http://go.bigswitch.com/rs/974-WXR-561/images/BCF-White-Paper-Secure%20and%20Resilient%20SDN-2.pdf?_ga=1.112867725.86193620.1448934819)
- Teenu, J., & Jincy, K. (2015). Survey on SDN Security Mechanisms. *International Journal of Computer Applications* 9(3), 32-35.
- Uddin, R., & Monir, M. F. (2019). Performance Analysis of SDN Firewalls: Pox vs ODL. *5th International Conferences on Advances in Electrical Engineering (ICAEE)*. Dhaka, Bangladesh: IEEE.
- Virtual Machines*. (2016, November 20). Retrieved from VirtualBox:  
<https://www.virtualbox.org/wiki/Virtualization>
- Waheed, & Mufarrej, A. (2017 ). Implementation of Virtual Firewall Function in SDN. *9th IEEE-GCC Conference and Exhibition (GCCCE)*.
- Wajdy, O. M., & Hao, C. (2017). Implementation and Performance Analysis of SDN Firewall on POX Controller. *9th IEEE International Conference on Communication Software and Networks*, 1461-1466.
- White, R., & Tantsura, J. (2015). *Navigating Network Complexity: Next Generation Routing with SDN, Service Virtualization and Service Chaining*. Indianapolis: Addison Wesley.
- Yang, Y., YuNan, W., & Wei, Y. (2014). Security Framework based on SDN. *Advanced Materials Research* 95, 4690-4693.
- Yunchun, L., & Jutao, M. (2015). SDN-based Access Authentication and Automatic COnfiguration for IPsec. *IEEE 2015 4th International Conference on Computer Science and Network Technology (ICCSNT 2015)*, 996-999.