iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VII **Month of publication:** July 2022

DOI: https://doi.org/10.22214/ijraset.2022.45268

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

DoSMit: A Novel Way for the Mitigation of Denial of Service Attacks in Software Defined Networking

Sreejesh N. G.[1], Sabina M. A.[2], Sobhana N. V.[3]
[1]*P. G. Scholar,* [2]*Assistant Professor,* [3]*Professor, Department of Computer Science and Engineering, Rajiv Gandhi Institute of Technology, Kottayam, Kerala, India*

Abstract: ***Software Defined Networking (SDN) is a rapidly evolving technology in the field of networking. It allows the computer networks to be managed via software instead of hardware devices. It gives the benefit of simplicity in management and easiness to apply new features in the network. Thus scalability, performance and maintainability become more sophisticated compared to the legacy hardware driven networks. But the SDN has more chances of being attacked by hackers, thus compromising the entire network. One common such threat is Denial of Service (DoS). Apart from such attacks in the legacy networks, the DoS attacks can be aimed at the SDN controller, Openflow switch and the controller-switch link by exhausting their resources. This paper proposes a novel and simple method to detect and mitigate such attacks through a Heavy Packet-in Flow Mitigation feature. Also it introduces an add-on feature ARP Flood Rule to reduce the number of ARP packets during the network initial boot up time. With both the features enabled, the network can reduce the packet-in flows to the controller and rescue the controller from overwhelming with huge packet-in message processing.***

Keywords: Software Defined Networking, Denial of Service, OpenFlow, Address Resolution Protocol

I. INTRODUCTION

Software-Defined Networking (SDN) is a network architecture approach that enables the network to be intelligently managed and centrally controlled, or programmed, using software applications [1][2]. SDN makes it easier to monitor, supervise, and regulate overall network traffic by dividing conventional network architecture into control and data planes.

Traditional networking employs intermediate devices such as routers, switches, gateways, and firewalls to construct networks. These hardware devices have both the control plane and the data plane integrated into them. The control plane is responsible for the traditional learning process, route identification and all other management decisions. The data plane just obeys the decisions taken by the controller. The two planes being in a single device is difficult to manage. Also if a feature is to be added, modified or removed, the entire device has to be changed most of the time.

SDN disengages the control plane from the data plane. The network-intermediary devices are controlled by the control plane. The key element in an SDN is the SDN controller, a dedicated powerful system with the SDN controller software installed in it. Some of such software applications are RYU, NOX, POX, Beacon and Floodlight. The controller is responsible for all the control plane operations such as dynamic route finding, forwarding rule creation, ARP table management etc. Here the data plane devices are just dump switches which perform certain actions such as forward the incoming packets, drop them, etc., as per the flow rules defined by the SDN controller. There are apps above the control plane to manage and interact with the controller [2]. The architecture of an SDN infrastructure is shown in Fig. 1.

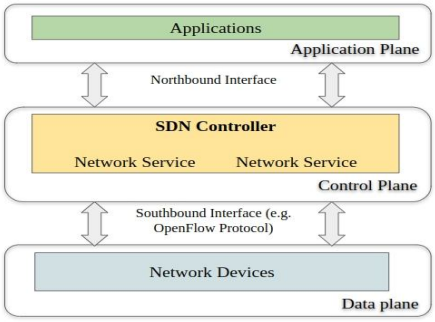


Fig. 1 SDN Architecture

The interface between the controller and the switches is called the Southbound interface and the communication happens using OpenFlow protocols [3-5]. The switches are not traditional ones but conform to OpenFlow specifications. This means that these OpenFlow switches have provisions to accept flow rules from the controller and forward the incoming packets according to these rules [4]. If no rule is found for incoming packets, such packets are encapsulated into some kind of messages called "packet-in" messages and forwarded to the controller. Then the controller will unwrap this message, analyse each packet, create an action set for it (such as to find the next hop to forward the packet, drop the packet, etc) and send back the packet to the sender switch by wrapping the actual packet and the action set inside a packet-out message. If the switch has buffered the packet, the packet-out message will have the buffer-id instead of the actual packet. The controller also sends a rule to the switch for this type of packet so that the switch can do the action without sending such similar packets to the controller again.

The Northbound interface interconnects the application plane with the control plane. This gives a network administrator access to the SDN, allowing them to configure it or retrieve data from it. This can be done through a graphical user interface (GUI) or Application User Interfaces (APIs such as REST API) which allows the controller to communicate with other applications. This helps the administrators to control the entire network infrastructure in a similar way without concerns about the complexity of the base network technology through properly designed apps and Northbound interface. Administrators can shape the network in whatever method they choose with SDNs. Administrators can also use a software interface to configure the rules and controls for the SDN.

II. ATTACKS IN SDN

The overhead of interaction between the control and data planes is high, and it may create network blockage. The centralized control of SDN makes it vulnerable to various types of attacks, e.g., flooding, spoofing, and denial of service (DoS) [6]. Today's commercial OpenFlow switches only support wired connection to the controller and the practical connection bandwidth was tested to be less than 10Mbps [7][8]. The most important attack towards SDN is the SDN-aimed Denial-of-Service (DoS) attack where the data plane, the control plane and/or the link between them become saturated. In this attack, the attacker intentionally changes some or all fields of a packet randomly so that there cannot be any match with any flow rules on the target switch. This is called "table-miss" and the attacker generates many such randomly forged packets to the network. These table-miss packets will cause a flood of messages to be sent from the victim switch to the controller, consuming the bandwidth, CPU processing time, and memory in both the control and data planes. As the controller becomes so busy that it cannot attend to the benign packets, it will lead to service denial. Thus the attacker can set up a clear DoS attack against the controller. All these point to finding a method that precisely identifies the attack, timely notify the mitigation system, and effectively defend the attack traffic with less false positive rate and possibly very small delay and overhead.

III.RELATED WORKS

AVANT-GUARD [7] came with two data plane extensions such as a connection mitigation module and an actuating module. Here a SYN proxy was used as a checkpoint for finding the resource saturation. Thus additional hardware was used to cache the incoming packets and for TCP SYN inspection. No other protocols are focused on this proposal.

NEOD [9] proposed a globally deployable Network Embedded On-line Disaster management framework for SDNs. Its major achievements were on the agility, accuracy, reliability, and scalability of the network infrastructure.

FloodGuard [8], a lightweight and protocol-independent defence framework for SDN networks against DoS attacks, used a proactive flow rule analyser by applying a runtime login on the SDN controller and applications. Here, to store the flood of packets an additional hardware cache was used. Then it would submit these packets to the OpenFlow controller using some sort of "rate limiting and round-robin scheduling".

SGuard [10] specified a method to detect the traffic, store some packets in a cache for later processing. Its "access control module" would identify the genuine source of packets and track the attacker's position so that mitigation measures can be implemented as quickly as possible. The classification module would classify benign and malicious packets and take necessary actions.

In BWManager [11], it describes the solution as to isolate data packets from different switches and then assign different time slice slots to different switches. Prioritize the packets based on "bandwidth prediction" and design an improved "weighted round-robin algorithm" for different priorities and apply the rules.

Daisy [12] would declare the heavy traffic as suspicious first. If the traffic persisted, it would block for some period of time. Daisy protected SDN from DoS attacks by analysing the acquired statistics and blocking malicious traffic from the attacker. Rather than banning an entire port or a host, the suggested solution targeted harmful traffic. In SDNGuard [13], the proposed method to detect the flooding, prevent it either by reducing the rate of packets at the controller, or by diverting the flow of packets to data plane cache for later processing. It could prevent attacks which were based on any protocols.

©IJRASET: All Rights are Reserved

FloodDefender [14] implemented separate detection and mitigation modules against DoS attacks. It could detect the heavy packet flow, diverting some packets to neighbour switches. The design would also create flow rules for the victim switches with auto expiry. To classify benign and illegal packets, an SVM based method was used in this proposed method.

IV. SYSTEM OVERVIEW

The following subsections first explain the working of SDN, define the attack scenario and then explain the methodology adopted for attack detection and mitigation.

A. Workflow of SDN

The driving elements that keep the SDN functionality going are the flow rules created by the controller and assigned on the switches. The switches keep the flow rules in one or more flow tables. The major components of the rule consist of a match part and an action part. The match field specifies a number of fields and their values to be compared with those of an incoming packet. If a proper match occurs, the corresponding action will happen. An example of an action is “forward to port 2”. Thus by carefully planning and assigning flow rules, the behavior of the SDN can be changed.

There are proactive and reactive rules [14]. Proactive rules are installed on the switch’s flow tables to manage some special packets or conditions. One of the basic proactive rules is to send an incoming packet to the controller by wrapping them in packet-in messages. This rule has the least priority and if rules are generated for specific flows later, they will have larger priorities and thus only those packets that do not match any specific rules will be forwarded to the controller.

The reactive flow rules are created and assigned into the flow tables on their requirement. One such occasion is the incoming packet at the switch port. If the specific fields of the packet match with any of the match fields of the flow table, the corresponding action will be executed. If there is more than one match, the one with higher priority is executed. If there is no match, the default action is to forward the packet to the controller by encapsulating it in a packet-in message. When this message reaches the controller, it will be unwrapped and the packet is inspected. The controller decides an action for this packet. It then creates a packet-out message encapsulating the original message and an action stating what to do with this packet. If the packet is already buffered in the switch the packet out message need not contain the original message, instead the buffer-id is placed along with the action. Then the controller sends this message back to the sender switch. The controller also specifies a rule for this type of packet and installs it in that switch’s flow table so that in future, this type of message may not be forwarded to the controller, but follow the action in the rule. Thus the switch need not send further such packets to the controller. Fig. 2 shows the normal workflow scenario in SDN.

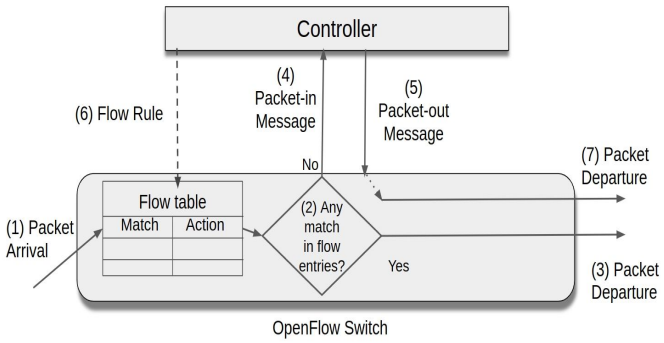


Fig. 2 - Workflow in SDN

B. DoS Attack Scenario in SDN

In reactive OpenFlow networks, the communication overhead between the control plane and data plane could be leveraged by an adversary [15][20]. An attacker creates malformed packets with spoofed fields at random, so that it is tough for them to match current flow rules in the connected switch. The attacker then generates a huge amount of table-miss packets mixed with benign ones to an OpenFlow switch. Each table-miss packet must be buffered and sent out as a message with its header by the victim switch. DoS attacks can clog the controller-switch bandwidth by flooding massive table-miss packets. This may burden the flow table of the switch by issuing unnecessary rules, and also eat up the controller's CPU resources while parsing packet-in messages. This is represented in Fig 3.

When the switch's memory is full, the result becomes extremely worse. The benign packet will not even reach the switch port as lots of spoofed packets get queued up at the switch port, causing severe network delay for them and lead to total blockage of those packets. This leads to service denial. The switch will not get any packet-out message from the controller so that the flows cannot be forwarded to other switches or ports. Even flow modification messages will not reach the controller. Also the controller itself will have a hard time processing the spoofed useless packets. Due to the heavy flow of packets, the bandwidth on the medium between the switch and the controller reaches its maximum and all through it gets stuck. Thus with a single type of attack, the switch, the controller and the link between them will get saturated leading to DoS attack.

V. SYSTEM DESIGN

The proposed system for DoS attack mitigation, called "DoSMit", stands between the controller and upper layer applications, as shown in Fig. 4. The DoSMit has two components:

1) ARP Flood Rule
2) Heavy Packet-in Flow Mitigation

While one feature handles the ARP requests effectively and thereby minimizes the number of ARP packet-in messages, the other detects and mitigates severe packet-in flows and thereby allows other flows to occur without interruption. The ARP Flood Rule feature is devised because during initial boot up of the network, the number of ARP packet-in messages will be huge in number and the system may get saturated.

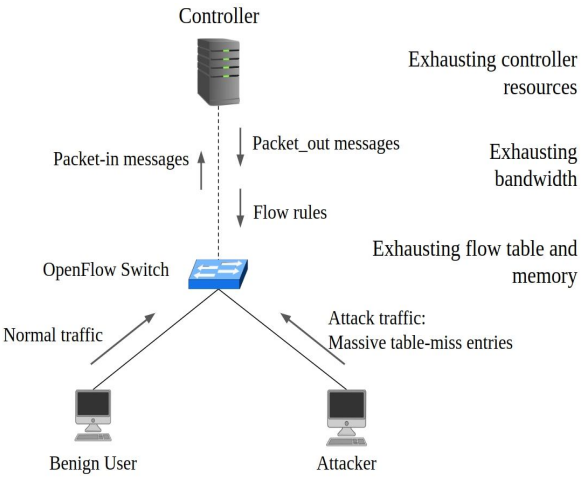


Fig. 3 SDN Attack Scenario

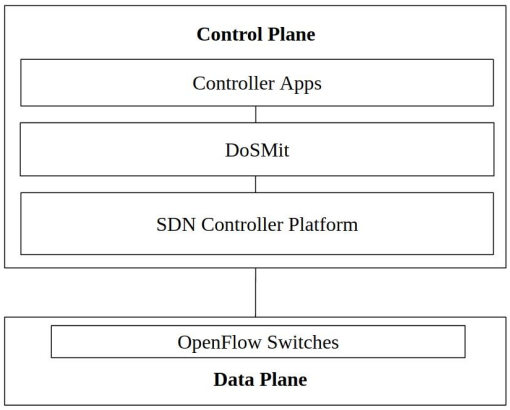


Fig. 4 Position of DoSMit

A. ARP Flood Rule Workflow

In normal conditions, when the network boots up, the devices in the network will not know any idea of the other devices in the network. Here, if one host wishes to communicate with another host, it will only have the IP address of the destination. But the device must know the MAC address of the next hop to which the packet has to send. It will not have that address. Hence it will broadcast an Address Resolution Protocol (ARP) request message. This message will have the link layer broadcast address (all f) as the destination MAC address. The OpenFlow switch will forward it to the controller through a packet-in message. The controller checks and sends this message back to the switch telling it to flood to its interfaces (in OpenFlow terms, OFPP_FLOOD). Thus this message comes to its neighbor switches which in turn forward ARP packet to the controller, and get the return packet to flood. This increases the number of ARP packet-in messages. If the number of hosts in the network is huge in number the ARP messages also will be huge and the system will think of it as an attack.

The ARP Flood Rule feature works when the first ARP packet-in message is received at the SDN controller. Before returning the packet to the switch, the controller finds the neighbor switches of the sender switch and sends a rule to all of them telling that if an ARP packet comes from this switch to each of the neighbors through their connected port (say, to_port), they all have to flood it without sending the packet-in message to the controller. Thus the OpenFlow Match will have the to_port and dst_MAC (here the broadcast MAC) as the match field entries and FLOOD as the action field entry. After all these rules are sent to the neighbors, the ARP packet is returned to the actual switch in the form of a packet-out message telling it to flood it. The neighbor switches, which have already got the rule to handle this ARP message, will flood it again to their connected ports. This will effectively reduce the number of packet-in messages. This rule lasts only 5 seconds and then will be discarded automatically. Hence these rules do not consume the switch table space for long. The ARP rule will have a slightly higher priority than the normal flow rules as the ARP requests must not be treated as normal packets. The algorithm for ARP Flood Rule feature is given below.

Algorithm 1 - ARP Flood Rule

1. *If out_port == OFPP_FLOOD and dst_MAC == "ff:ff:ff:ff:ff:ff" then*
   - 1.1. *match ← OFPMatch (to_port, dst_MAC)*
   - 1.2. *action ← OFPP_FLOOD*
   - 1.3. *For each neighbor switch*
     - 1.3.1. *Add flow (priority=5, match, action)*
   - 1.4. *End for*
2. *End if*

The workflow diagram for this feature is given in Fig. 5.

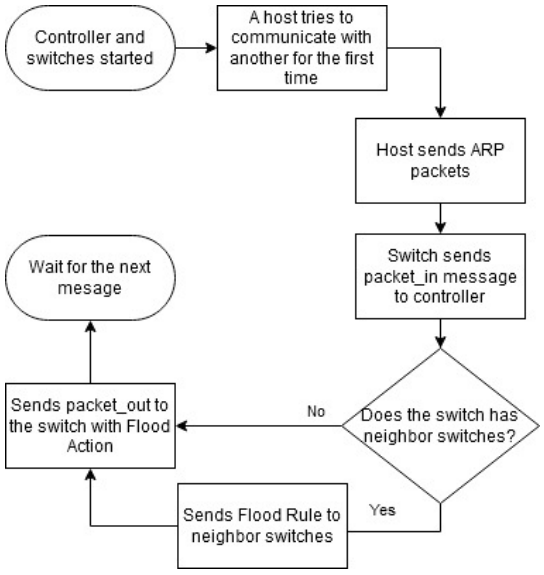


Fig. 5 Workflow of ARP Flood Rule feature

B. Heavy Packet-in Flow Mitigation

The major feature to identify and lighten the DoS attack is the Heavy Packet-in Flow Mitigation feature. For this feature to work, packet processing capacity of the controller is set as a threshold to detect whether the controller CPU has reached its saturation or not. Also, the controller keeps separate counters to count the number packet-in messages coming to it on a per-port per-switch basis.

The counters are checked at two occasions to know whether it has reached the threshold or not.

1) Whenever a packet-in message arrives.
2) At an interval of 1 sec.

When a packet-in message comes, before deciding the action for it, the controller checks whether the counter for that particular port has reached the threshold or not. If not, the controller has not reached its capacity to process it and hence it can process the message and decide actions for the packet. This is the normal situation. But when the counter has reached or crossed the threshold, the port is suspected to be under attack. If the packets for that port are processed, most probably the action for those will be to flood it, which will in turn cause the increase in packet-in messages to the controller. Hence in that situation, the controller does the following action.

a) If the switch has a buffer, the controller sends a packet-out message to it with an action to drop the packet, and hence free the buffer.
b) If the switch does not have a buffer, the controller simply drops the packet at once.

There is a separate process thread that works at 1 second intervals to check the per-port packet-in counter. Every time the thread loops through all the counters and checks whether any one has reached or crossed the threshold. If it doesn't, it will just reset the counter. But if it does, that means there is a chance of an attack. Then the thread will create a rule for the attacked port that will drop all the new packets that do not match with the existing flow rules in the switch. This drop rule has an expiry time of 60 seconds. After the expiry, if the attack still continues, the port will be assigned the drop rule again. This process continues during the entire time of huge packet flows. The advantage of this rule is that, this rule has a lower priority than the existing benign rules and hence the ongoing traffic will not be affected. Also, if the input port of this large packet-in flows is connected to another switch, that port will not be assigned this protective rule. This will help not to block the new traffic that is coming from other connected switches.
The two phases of the algorithm for Heavy Packet-in Flow Mitigation feature are given below. The first one works at every packet-in message and the second one works as a thread which wakes up at an interval of 1 second. The workflow diagrams for phase 1 is given in Fig. 6(a) and phase 2 in Fig. 6(b).

Algorithm 2 - Heavy Packet-in Flow Mitigation - Phase 1

1. *If a packet-in message comes from "switch" with packet input port "port" then*
    - 1.1. *Increment the packet-in counter [switch][port] by 1.*
    - 1.2. *If packet-in counter [switch][port] <= threshold then*
        - 1.2.1. *Process the packet normally*
    - 1.3. *Else*
        - 1.3.1. *If the packet is stored at the switch buffer then*
            - 1.3.1.1. *Send a packet-out to the switch with an action to drop the packet*
        - 1.3.2. *Else*
            - 1.3.2.1. *Drop the packet*
        - 1.3.3. *End if*
    - 1.4. *End if*
2. *End if*

Algorithm 3 – Thread for Heavy Packet-in Flow Mitigation – Phase 2

1. *For Every port of switch in the switches list do*
    - 1.1. *If packet-in counter [switch][port] <= threshold then*
        - 1.1.1. *Send a flow rule to switch with priority 5 to drop new packets for 60 seconds*
    - 1.2. *End if*
    - 1.3. *Resets the counter packet-in counter [switch][port]*

2. *End for*

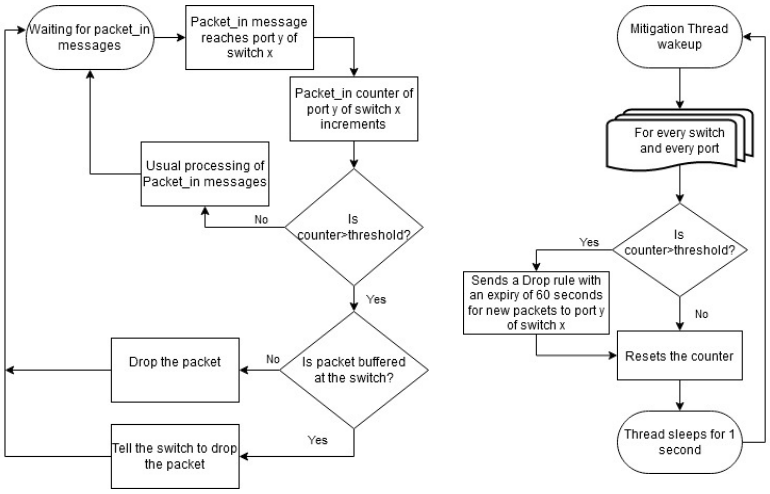


Fig. 6(a) Workflow of Heavy Packet-in Flow Mitigation Phase 1 (b) Fig. 6 Workflow of Heavy Packet-in Flow Mitigation Phase 2

VI. IMPLEMENTATION

A. *Setup*

The proposal is implemented in a computer having a processor Intel(R) Pentium(R) CPU N3710 @ 1.60GHz with memory of 4GB. The operating system is Lubuntu 18.04 LTS with Mininet [18][19] for network emulation and RYU [16][17] as SDN controller. For packet framing, hping3 [22] and Scapy [21] are used. The DoSMit features are implemented by developing the L2 learning functionality provided by the RYU. The Southbound protocol used here is OpenFlow 1.3.5 [4]. The experiments are done in software mode only, with a topology that includes a controller, 5 OpenFlow switches and 25 hosts such that the switches are connected linearly and 5 hosts are connected to each switch. The switches are directly connected to the controller. Each switch has two neighbor switches, except the first and last. For them, only one neighbor switch is there. The test network topology diagram designed in miniedit [20] is given in Fig. 7.

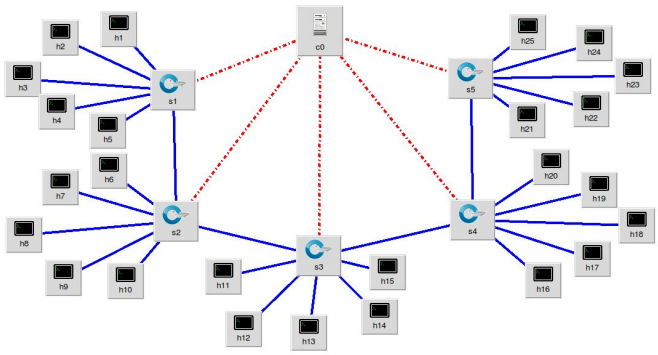


Fig. 7 Test Topology

B. *Experiment*

For the experiments and analysis after that, logging functionality for different parameters has been enabled in the app. All the parameters are written to separate *.csv* files. The measured parameters include the CPU and Memory utilization of the controller, packet-in counts per switch (especially ARP packets), per port packets and bytes statistics including the effective bandwidth.

The test was conducted using the following scenarios.

1) Normal working without implementing the new features.
2) DoS Attack without implementing the new features.
3) DoS Attack with both ARP Flood Rule and Mitigation modules enabled.

In all the scenarios, the same system and topology configurations are used. The hosts are given IP addresses from 10.0.0.1 to 10.0.0.25 and are identified by names h1 to h25. After the network is booted up, a dummy web server is set up at the host h1 to generate TCP traffic. On request from clients, it returns a dummy web page. In order to simulate normal traffic, two hosts (h5 and h6) keep on sending http requests to the web server at 1 second intervals and continuous ping requests from host h15 to h1. This setup is used for scenario 1. To simulate attack traffic for the rest of the scenarios, Scapy is used to flood packets from h16 at 50 to 200 packets per second, and with forged MAC and IP addresses. To check whether an attack has taken effect and mitigation has been activated, we use ping utility by checking the connection from h25 to h1. In an attack situation, there will not be successful ping and at the time of mitigation and under normal situations there will be successful ping between those systems.

VII. PERFORMANCE EVALUATION

By conducting the experiments under different scenarios, various factors could be identified. On flooding with spoofed packets, the switch port, the channel between the switch and the controller and the controller itself become exhausted so that the packet-in messages could not be handled by the CPU of the controller. This affects the entire network severely because the controller is the central point of decision in the network. The effect of such a DoS attack is shown in Fig. 8. While the host h16 generates attack traffic, h25 cannot ping h1.

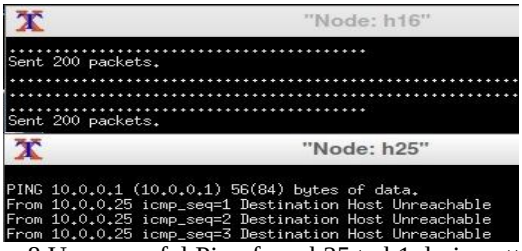


Fig. 8 Unsuccessful Ping from h25 to h1 during attack

By applying the DoSMit feature, it is seen that the system displays a warning of the heavy packet-in flow at the attacked port (here switch 4, port 3). At the same time a mitigation rule is applied at that switch to drop new flows that do not match with any of the present high priority rules. The mitigation rule has a lower priority than the specific flows in the flow table. This ensures that the existing flows are not affected. The effect of the mitigation rule is shown in Fig 9. Here attack is mitigated and h25 can ping h1.

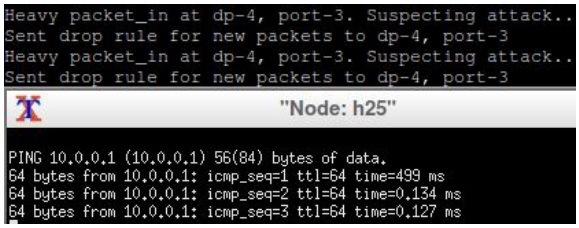


Fig. 9 Successful Ping from h25 to h1 during attack

On comparing the controller CPU usage in all the scenarios, it can be seen that the addition of the new features has not created an overhead to the normal working of the system. Though the percentage of CPU utilization without the added features shows an increase at the time of attack, there is no hike in any other scenarios. Thus the features do not have any CPU impact. The comparison of CPU usage versus time is shown in Fig. 10. This is due to the fact that when the mitigation rules are applied, the new packets are dropped at the switches and the CPU is freed from handling packets from that port.

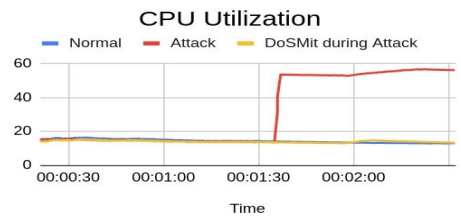


Fig. 10 Controller CPU Utilization (percentage)

Fig. 11 shows the graph of the amount of packets arrived at the controller as packet-in messages from the attack switch versus time at normal scenario and during the attack. Apart from the normal flows, the attack situations without and with DoSMit show a spike when the attack is first detected. Without the mitigation, the spike continues to be high, whereas the mitigation feature turns it down to normal. This is due to continuous table-miss packets. When DoSMit is implemented, the spike suddenly comes down to normal due to the implementation of drop rule at that port. Thus the normal flows continue.

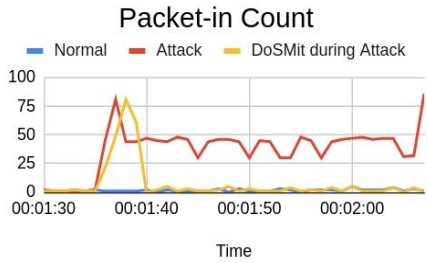


Fig. 11 Packet-in Message Count from Attack Switch

To analyze the effectiveness of the ARP Flood Rule feature, it has to be tested alone without the Heavy Packet-in Flow mitigation feature. For this the network boot phase is analyzed with and without the ARP Flood Rule feature. When the network is booted up the number of packet-in messages is higher. Then it goes down as the controller can understand different hosts and switches in the network. Later the packet-in messages is lower in number in the normal scenario. The effect of ARP Flood Rule feature is very promising that the number of packet-in massages is reduced by a considerable amount. The comparison with and without the ARP feature is given in Fig. 11.

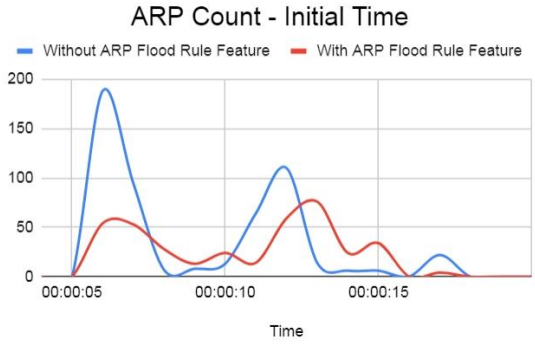


Fig. 11 Packet-in Message Count from Attack Switch

VIII. CONCLUSION

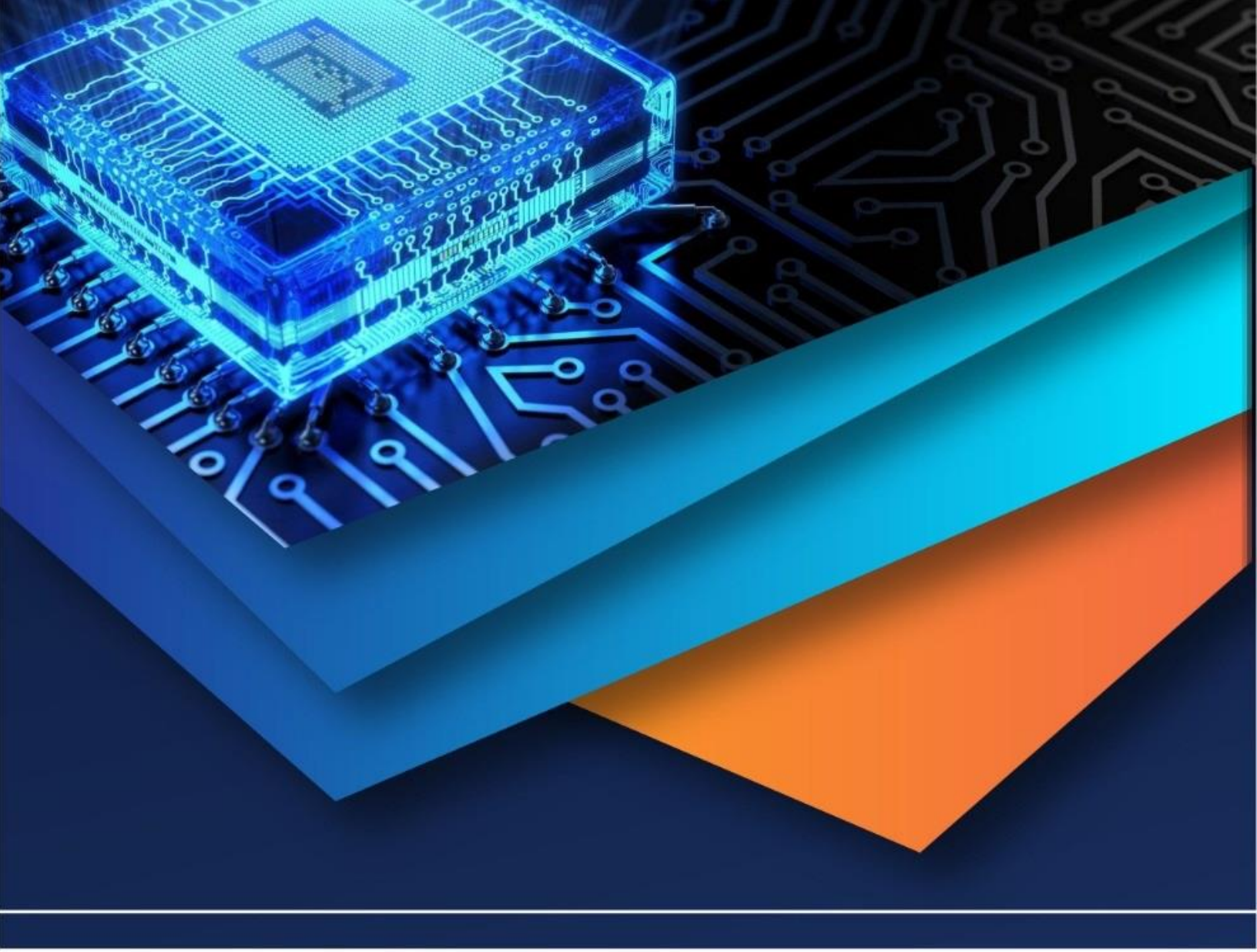
SDN is emerging as new network architecture for centralized management of the control plane through software and applications. This gives the administrator the provisions for managing network features without changing existing network hardware. Just like traditional networks, SDN also suffers from Denial-of-Service attacks. From the experiments in the software environment, it is seen that the proposed solution DoSMit can protect OpenFlow networks against SDN-aimed DoS attacks to a particular extent. It is more scalable and easier to deploy without engaging additional devices. In future the False Positive Rate (FPR) has to be lowered. Here the proposed DoSMit does not classify the packet based on protocol, but considers the amount of packets only. In future the detection and mitigation process has to be done on protocol basis as there are several types of such attacks such as ICMP ping flood and TCP SYN attacks. Such a method can reduce the FPR again.

REFERENCES

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 27-51, Firstquarter 2015, doi: 10.1109/COMST.2014.2330903.
[2] SDN Architecture [online] https://telsoc.org/journal/ajtde-v3-n4/a28
[3] N. McKeown et al., “OpenFlow: Enabling innovation in campus networks,” ACM SIG-COMM Comput. Commun. Rev., vol. 38, no. 2,pp. 69–74, Mar. 2008.
[4] OpenFlow Switch Specification [online] Available: https://opennetworking.org/software-defined-standards/specifications/
[5] OpenFlow protocol API Reference [online] Available: https://ryu.readthedocs.io/en/latest/
[6] Ahmad, Ijaz, et al. "Security in software defined networks: A survey." IEEE Communications Surveys & Tutorials 17.4 (2015): 2317-2346.

[7] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS), 2013, pp. 413–424.
[8] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw., Jun. 2015, pp. 239–250.
[9] S. Song, S. Hong, X. Guan, B.-Y. Choi, and C. Choi, "NEOD: Network embedded on-line disaster management framework for software defined networking," in Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM), May 2013, pp. 492–498.
[10] Wang, Tao, and Hongchang Chen. "SGuard: A lightweight SDN safe-guard architecture for DoS attacks." China Communications 14, no. 6 (2017): 113-125.
[11] Wang, Tao, et al. "BWManager: Mitigating denial of service attacks in software-defined networks through bandwidth prediction." IEEE Transactions on Network and Service Management 15.4 (2018): 1235-1248.
[12] Imran, Muhammad, Muhammad Hanif Durad, Farrukh Aslam Khan, and Haider Abbas. "DAISY: A detection and mitigation system against denial-of-service attacks in software-defined networks." IEEE Systems Journal 14, no. 2 (2019): 1933-1944.
[13] Maddu, Jeevan Surya, Somanath Tripathy, and Sanjeet Kumar Nayak. "SDNGuard: An Extension in Software Defined Network to Defend DoS Attack." 2019 IEEE Region 10 Symposium (TENSYMP). IEEE, 2019.
[14] Shang Gao, Zhe Peng, Bin Xiao, "Detection and Mitigation of DoS Attacks in Software Defined Networks", Vol. 28, NO. 3, Jun. 2020.
[15] R. Kandoi and M. Antikainen, "Denial-of-service attacks in OpenFlow SDN networks," in Proc. IFIP/IEEE Int. Symp. Integr. Netw. Man-age. (IM), May 2015, pp. 1322–1326.
[16] RYU SDN Framework Community. RYU Controller. [online]. Available: https://osrg.github.io/ryu/
[17] Welcome to RYU the Network Operating System(NOS) [online] Available: https://ryu.readthedocs.io/en/latest/
[18] W. Buck, A. Grammer, M. Moerike, and B. Muehlemeier, "MININET," Das Rechenzentrum, vol. 2, no. 3, pp. 137–141, Jan. 1979.
[19] Kaur, Karamjeet, Japinder Singh, and Navtej Singh Ghumman. "Mininet as software defined networking testing platform." International Conference on Communication, Computing & Systems (ICCCS). 2014.
[20] How to use MiniEdit, Mininet's graphical user interface. [online] Available: https://www.brianlinkletter.com/2015/04/how-to-use-miniedit-mininets-graphical-user-interface/
[21] Welcome to Scapy's documentation [online] Available: https://scapy.readthedocs.io/en/latest/
[22] Getting started with hping3 [online] Available: http://wiki.hping.org/94
[23] S. Gao, Z. Li, B. Xiao, and G. Wei, "Security threats in the data plane of software-defined networks", IEEE Netw., vol. 32, no. 4, pp. 108–113, Jul. 2018.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)