iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** XII **Month of publication:** December 2022

DOI: https://doi.org/10.22214/ijraset.2022.47885

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 XII Dec 2022- Available at www.ijraset.com

DDoSMitigator: An On-The-Fly Method of Mitigating Denial of Service Attack in Software Defined Networking

Sreejesh N. G.[1]
[1]*P. G. Scholar, Department of Computer Science and Engineering, Rajiv Gandhi Institute of Technology, Kottayam, Kerala, India*

Abstract: ***Started in the early 2000s, the Software Defined Networking (SDN) paradigm has become the backbone for various network based technologies. It has the advantage of being simple to manage and easy to add new features to the network. In comparison to legacy hardware-based networks, scalability, performance, and maintainability become more advanced. However, hackers are more likely to target the SDN, putting the entire network at risk. Denial of Service is a common example of such a threat. It is feasible to reduce such attacks by carefully planning and building the SDN controller as well as the network applications that administer the SDN. This paper presents a novel and efficient method applicable to various types of protocol based Distributed Denial of Service (DDoS) attacks in SDN/OpenFlow networks. It can be used to detect and mitigate Transmission Control Protocol (TCP) SYN attacks, Internet Control Message Protocol (ICMP) ping flood attacks and User Datagram Protocol (UDP) flood attacks that happen against the SDN devices and/or any host. This feature stands above the controller and conforms to the OpenFlow policy without leveraging additional devices. All the detection and mitigation are done based on a True Host list which is created by tracking the Address Resolution Protocol (ARP) requests and replies from hosts. As ARP protocol is necessarily used by all hosts, this method can be effectively utilised for true host list creation and further attack detection and mitigation.***

Keywords: Software Define Networking, Distributed Denial of Service, OpenFlow, Address Resolution Protocol.

I. INTRODUCTION

An approach to network architecture called “Software-Defined Networking” (SDN) makes it possible to use software programs to intelligently manage and centrally govern the network. SDN allows for more flexible network traffic management and control by dividing traditional network architecture into control and data planes. Intermediary devices such as routers, switches, gateways, and firewalls are used in between the hosts to create a network. These hardware devices have both a control plane and a data plane built in. The control plane governs how packets are routed and how the least cost, best cost, and second best paths for packet transfer to their destination are calculated. Depending on how the control plane is designed, the data plane transfers that traffic. Also the control plane performs several additional tasks regarding the specific device and features. It’s a headache to control these two planes with a single device. Changing the network configuration necessitates the purchase of new hardware as well as additional configuration. Legacy networks also have a hard time adapting to changing network conditions.

In SDN, the control plane is separated from the data plane, and the control plane controls several data plane devices. The SDN controller, a specialised powerful machine with the SDN controller software installed, is the most important component of an SDN. RYU, NOX, POX, Beacon, and Floodlight are examples of such software applications. All control plane functions, including dynamic route finding, forwarding rule construction, and ARP table administration, are handled by the controller. The data plane devices are merely dump switches that route incoming packets according to the SDN controller's flow rules. The application plane consists of several business specific, managing and monitoring applications above the control plane to manage and communicate with the controller. The architecture of SDN is given in Fig. 1.

There are two types of interface in the SDN; Southbound and Northbound interfaces. The Southbound interface lies between the controller and the switches and the communication through it happens using OpenFlow protocols. These switches must conform to OpenFlow [13][14] specifications. The OpenFlow switches have provisions to accept flow rules from the controller and based on which actions can be taken for the incoming packets. The Northbound interface lies between the apps and the control plane. Network administrators and management authorities access and communicate with the controller and thereby the network through Command Line Interfaces (CLI) or Application Programming Interfaces (API, for example REST API).

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue XII Dec 2022- Available at www.ijraset.com

The northbound interface allows top-level components to communicate with one another. Thus whatever be the rudimentary network technology, the operators can view the entire topology and manage it mostly from a single point using software applications. They can shape the network in any way at their wish. If there were traditional intermediate components, this task would have cost a lot of time, money and work.

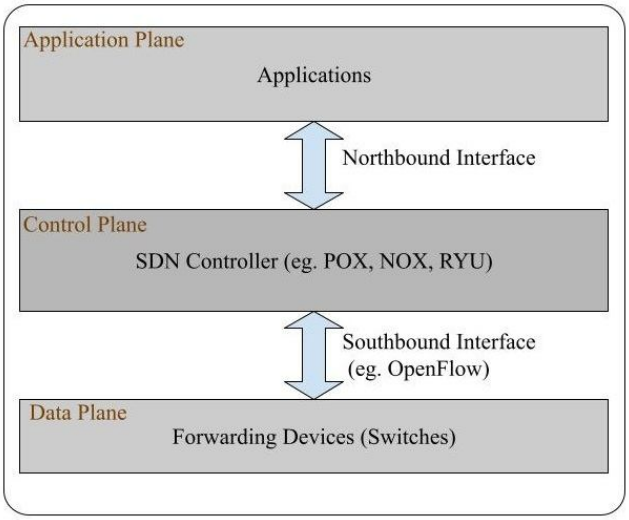


Fig. 1 SDN Architecture

A. SDN Workflow

The SDN works on a collection of flow rules that are crafted by the SDN controller and stored in one or more flow tables of the OpenFlow switches. Whenever a packet comes to the switch it will be checked against the flow table to get a matching flow rule. If no rule is found for incoming packets, such packets are wrapped inside packet-in messages and forwarded to the SDN controller. The controller will unwrap the packet-in message, analyze each packet, create rules for them (find the way to forward the packet, discard the packet, etc.) and return the packets to the corresponding switches via packet-out messages. The packet-out message contains the actual packet and an action specifying what is to be done with this packet. If the switch supports buffering of packets, the packet-out message need not contain the actual packet; instead its buffer-id will be specified. The controller also sends the new flow rules to the switch. The switch stores this rule in its flow table. From then on, the switches can manage similar such packets based on these new rules until they expire. There can be priorities for rules. This means that, if there are two rules for the same match fields, the action corresponding to higher priority will be chosen. The workflow of SDN is given in Fig. 2.

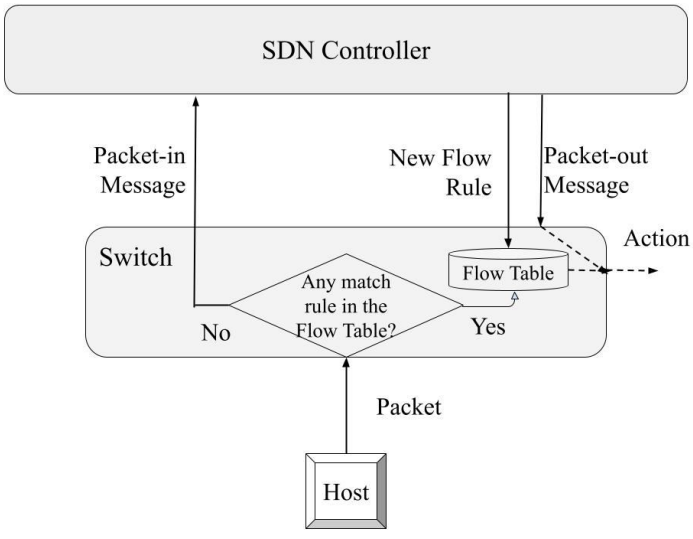


Fig. 2 Workflow of SDN

A match part and an action part are the two main parts of the rule. The values of the fields specified in the match field are to be compared to those of an incoming packet. The necessary action will take place if the proper match happens. "Output to port 2" is a type of action. As a result, the behaviour of the switches can be modified by carefully assigning flow rules. For every flow rule, there is a corresponding cookie value to identify the rule, a priority field and timers for rule expiry.

There are two types of flow rules in a broader sense. They are proactive and reactive rules. The proactive rules are pre-installed flow rules on the flow tables of the data plane switches to process the network traffic. The basic proactive rule is to forward the incoming packet to the controller by wrapping it in a packet-in message. The reactive flow rules are generated and installed on the flow tables as and when required. One such occasion is when a packet comes at the switch port. When the packet comes to the switch port for the first time, it is forwarded to the controller as usual and the controller returns the packet-out message with an action, followed by a flow rule for such types of packets.

B. Attacks in SDN

As the communication between the control and data planes goes through a very narrow path compared to the vast underlying network, it can become a bottleneck of the whole network. The centralised control feature and the narrow southbound interface of SDN makes it susceptible to various types of attacks, e.g., flooding, spoofing, Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS). Most commercial OpenFlow switches used nowadays support only cable connection to the controller [10]. The practical bandwidth of this medium was tested to be of the order of Megabits per second only. Hence saturating this link can deplete the network.

The SDN-targeted DDoS assault is the most significant attack against SDN (e.g., data-to-control plane saturation attacks). In this attack, the attacker shapes some fields of a packet at random, making it difficult to match with any flow rules on the victim switch. The attacker then floods the network with a high number of such table-miss packets. These packets will cause the victim switch to send a large amount of packet-in messages to the controller, using their communication bandwidth, CPU time, and memory in both the control and data planes. The controller becomes so busy in handling these useless packets, that it is unable to handle innocuous packets, denying the service for which the packet was intended. As a result, the attacker is able to carry out a successful DoS attack on the controller. If there are a number of systems collectively attacking the network, it can be said as a DDoS attack. Thus a DoD/DDoS attack in SDN can be done by saturating the switch, the controller or the link between the switch and the controller. The major challenges when considering the SDN-aimed DoS attacks are;

1) Precise detection of SDN-aimed threats and prompt notification to the security system when such attacks occur.
2) Efficient handling of table-miss packets, keeping small delay, small loss rate, and normal behaviour for normal packets.
3) Precise detection of attack traffic from illegal traffic without stressing computational resources.

II. RELATED WORKS

S. Shin et al. developed AvantGuard with the objectives of scalability, resilience, transparency, incremental deployment and minimum collateral impact [1]. It used a proxy to monitor the TCP 3-way handshake procedure for connection establishment. Thus it could detect the SYN flood attack, and generate flow rules to mitigate it. But attacks using other protocols could not be identified or mitigated.

NEOD (Network Embedded Online Disaster Management Framework), proposed by S.Song et al., was designed as an OpenFlow firmware extension on an OpenFlow Switch so that this can be implemented without the need of vendor specific changes[2] but focussed on finding the disaster.

Shin et al. brought a new Network Operating System (a micro-NOS) called Rosemary with an objective to prevent harmful applications from corrupting the OpenFlow controllers [3]. It could integrate key safeguards for OpenFlow applications, and also build separate sandboxes for network apps. The key objective was on handling the applications rather than the saturation attacks.

H. Wang et al. came forward with a solution called FloodGuard to defend against the SDN Saturation attack [4]. It could store the packets in a hardware cache at the switch for further processing during the attack. But adding a cache may not always be practical.

In SGuard, Wang et al. mainly focused on detecting DoS attacks with spoofed IP packets and to prevent all kinds of DoS attacks[5]. Even though the CPU and bandwidth utilisation was fine, it used an additional cache to store the packets.

BWManager was an SDN controller scheduling algorithm by Wang et al. based on bandwidth prediction to diminish DoS attacks on the SDN [6]. It used a “fuzzy synthetic evaluation decision-making model” to detect the TCP SYN flood and UDP flood.

S. Gao et al. brought a proposal called FlowKeeper to enforce port control of each OpenFlow switch when DoS or topology poisoning attacks occur [7].

The proposal DAISY (Detection And mItigation SYstem) by Imran et al. was a lightweight statistical system for blocking malicious traffic [8]. It used an anomaly detection method to identify attacks. It would first declare the packet flow as "suspicious" and if the attack continues, declare as "malicious". It could defend against SDN aimed attacks only.

SDNGuard by Maddu et al. was a framework to defend against DoS attacks [9]. It used a dataplane cache to store the packets based on a probabilistic model.

Gao et al. presented an application called FloodDefender which was fixed between the controller platform and app layer [10]. When the system detects a flood, it could divert some packet-in message through neighbour switches and thus prevent the switch from saturating. It used a Support Vector Machine for attack packet classification. But it needed to reduce false positive rate and switch buffer utilisation.

According to Varghese et al., their proposed method was the first Data Plane Development Kit (DPDK) based DDoS defence framework built on single feature anomaly detection in SDN [11]. Instead of using an OVS switch, here DPDK-OVS switches were used. However, because the DPDK was still in its early stages, many optimizations must be implemented before the full results could be obtained.

Tang et al. introduced the "Performance and Features" method for mitigating the Low-Rate TCP-Targeted DoS attacks in SDN [12]. This method used a Machine Learning model to detect the traffic as normal and abnormal. Even though the recall was slightly lower, it dramatically reduced false positives, and the total detection errors were also reduced.

III.SYSTEM DESIGN

When a DDoS/DoS attack aiming at the SDN controller occurs, the number of spoofed packets at the controller will be very high. To block this, the first thing is to detect the port from which the flood comes. After knowing this, there are two ways to mitigate the attack. The first way is blocking the attacking port which will prevent the flood, but this will affect all the running flows also. The best but most complex way is to detect and block the malicious packets only. This will allow all the benign packets and thus reduce the false positive rate. This is possible only if the incoming packets are thoroughly inspected and classified as malicious and benign. Thus the protocol by which the attack happens also needs to be identified and blocked. The major protocols that are used for such attacks are TCP (for SYN flood attack), UDP (for UDP flood) and ICMP (for ping flood attack). The prevention of such attacks needs effort compared to the first method. But the result will be more promising than the first one because only the attacking packets are blocked and thus the false positive rate will be very much less. This method is used in the proposed method, DDoSmitigator. In addition to the SDN controller, the method also protects the hosts.

The DDoSMitigator stands above the controller and inherits some of its functionalities such as the basic controller services. The location of the DDoSMitigator is given in Fig. 3. The method is an on-the-fly one which detects the packets with spoofed addresses and blocks them as early as possible so that these packets do not cause ill effects to the system. The work is done in two parts. The first one is to create a true host list consisting of all possible hosts in the network. This can be used to check whether the incoming packet is coming from a legal host or not. This allows identification of spoofed packets. The second part is the detection of a flooding attack by analysing the type of protocol (TCP, UDP and ICMP) and doing mitigation measures. The mitigation is done by blocking such spoofed flows and allowing those packets which have the addresses in the true host list. The detailed steps are given in the following subsections.

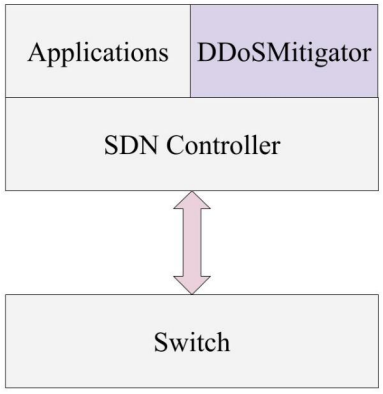


Fig. 3 Location of DDoSMitigator

A. Creating the ARP True Host List

For a packet to be classified as a benign packet, we need to create a list of true hosts. In this proposal, such a list has been made using the ARP protocol. When a host in the network needs to send a packet to another one, it needs the MAC address of the next hop in the path to the destination. During the first time transfer, the host will not have such a MAC address with it to send. Then it will generate an ARP packet and broadcast it. When the packet-in message for the same is received from the switch for that source host, the packet details are added to a temporary true host list and allows for normal operation, ie, flooding to the nearby switches. This will eventually reach the destination, which will send an ARP reply as a unicast packet to the source. When this reply reaches the actual source, it will update its ARP cache with an entry for the second host and send the packet with that MAC address. Here the first host which receives the ARP reply can be treated as a true host and added to the True Host list. The corresponding temporary entry will be removed from the temporary-true-host list. As per ARP, since only the intended host will respond to the ARP requests, the second host which replies to the ARP request can also be added to the list of True Hosts. The True Host list is implemented as a list of dictionary entries with the MAC address as the key and a list containing the switch id, the corresponding port and the ip address of the host as the values. After adding the host entry to the list, the controller will send two Layer-3 proactive rules to that host; one is to send those packets to the controller whose incoming port, MAC address and IP address matches with those in the True Host list. The other one is to send those packets to the controller if the packet has an ICMP Destination Port Unreachable message. This type of packet is generated from those hosts which get a UDP packet with an invalid port number. The steps for the ARP True Host list creation are explained in the Algorithm True-host-by-ARP. The true host list always keeps updated whenever an ARP request comes to the controller.

Algorithm 1 - True-Host-by-ARP

1. *If an ARP packet-in message comes to the controller then*
    - *1.1. If it is an ARP-request then*
        - *1.1.1. If it came to the switch through any host-connected port then*
            - *1.1.1.1. Store its identity (switch-id, inport, source MAC, source IP, destination IP) in temporary true host list.*
        - *1.1.2. End if*
    - *1.2. End if*
    - *1.3. If it is an ARP-response then*
        - *1.3.1. If it came to the switch through any host-connected port then*
            - *1.3.1.1. Add the host attributes (switch, port, MAC-addr, IP-addr) to True Host list.*
            - *1.3.1.2. Send the proactive rules to the switch.*
        - *1.3.2. Else*
            - *1.3.2.1. If it is not intended for flooding and is to be sent to a host then*
                - *1.3.2.1.1. Controller removes the corresponding entry in the temporary true host list.*
                - *1.3.2.1.2. Add the host attributes (switch, port, MAC-addr, IP-addr) to the True Host list.*
                - *1.3.2.1.3. Send the proactive rules to the switch.*
            - *1.3.2.2. End if*
        - *1.3.3. End if*
    - *1.4. End if*
2. *End if*

B. ICMP Ping Flood Attack Detection and Mitigation

The Internet Control Message Protocol (ICMP) is a network layer protocol in the TCP/IP protocol suite widely used by the system for error reporting and diagnostic purposes. One of the most common uses of ICMP is by the administrators where they use the ping utility to check connectivity between hosts. Ping utility uses the echo request-reply feature of the ICMP protocol. In this feature, the source system will send an ICMP echo request message to a destination host. The latter when receiving this packet will send back the same packet to the former system as an ICMP echo reply. This indicates the presence of the destination host and the average time to reach it.

While the ICMP ping request is helpful for administrators, it can be turned into a devil. Attackers may create spoofed ICMP echo request packets and will send hundreds of such packets to a target system.

The target will respond to these with echo reply packets. But as the source addresses are spoofed, these will wander in the network without seeing the target system or will be delivered to some systems which might not have even known that some packets were sent with their addresses. Thus the target system will not get any time to serve benign packets. This spoofed ICMP ping flood thus causes a DoS/DDoS attack in the target system or target network. In the SDN also, such spoofed ICMP requests will create problems. As the SDN controller may not know the spoofed addresses, it will tell the switches to broadcast the requests causing a lot of traffic in the network. Also the CPU and memory of the controller becomes exhausted. This will bring unavailability for benign packets.

DDoSMitigator has put forward a remedy for this. When an ICMP echo packet comes to the controller, it will check the packet details with the True Host list. If the details exist there and perfectly match, it is considered benign and treated in the normal way (ie, the packet-out is followed by a Layer 3 flow rule). If there is a mismatch in the address parameters, the controller will increment a counter ICMP-spoofed-packets for the switch. Also this packet will be dropped by the controller. If the packet is buffered in the source switch, the controller will tell the switch to drop it. Thus the spoofed packets will not reach the destination.

A thread will be executing in the background which wakes up at 2 second intervals and goes through each entry in the ICMP spoofed list. If any count exceeds a predefined threshold (say 5 packets), the controller will generate flow rules such that new ICMP echo packets coming at the attack port will be blocked. These rules will have lower priority than the proactive flow rules for forwarding genuine packets to the controller, and hence the genuine packets will be forwarded as usual and are not affected by the attack. This means that the packets from those hosts which are in the true host list will be able to send packets from that port. All other ICMP requests are dropped. This will prevent further flooding of ICMP spoofed packets. The new reactive flow rule will have a limited time period only (say 60 seconds) and after that these rules will expire to save the flow table space. This is due to the assumption that the attack may not last for a long time. If the attack persists even after the flow rule expiry, the same defence procedure happens again. All these things save the switch and controller resources. The steps for the ICMP Flood Mitigation are given in the Algorithm ICMP-Ping-Flood-Mitigation.

Algorithm 2 - ICMP-Ping-Flood-Mitigation

1. *If an ICMP echo request packet-in message comes to the controller then*
   - *1.1. If the source details are present it the True Host list then*
     - *1.1.1. Sends a packet-out message to forward the ICMP packet as usual.*
     - *1.1.2. Install an L3 flow rule to the switch for such packets.*
   - *1.2. Else*
     - *1.2.1. Make an entry for that switch and port in a spoofed-ICMP list if not there already, and increment its count.*
     - *1.2.2. If the packet is not buffered in the switch, the controller just drops it and if buffered, tells the switch to drop it.*
   - *1.3. End if*
2. *End if*

Thread for the spoof-check-ICMP

3. *For each entry of the spoofed-ICMP list, if the count exceeds a limit do*
   - *3.1. 5.1. Alerts as an attack.*
   - *3.2. 5.2. Send flow rules to the source switch to block all ICMP echo packets coming to the attack port.*
   - *3.3. 5.3. Remove the list entry.*
4. *End for*

C. UDP Flood Attack Detection and Mitigation

The User Datagram Protocol is a network layer protocol in the TCP/IP protocol suite. As it is a connectionless and unreliable service, UDP is sometimes called the best effort delivery service. As the UDP does not need any connection to the destination, attackers can effortlessly use it for flooding attacks. Generally the UDP flooding attack is created by sending hundreds of spoofed packets with random source ports aiming at the target system. On receiving these packets, the target system will check whether there is any application associated with the destination port received in the packet. Mostly there will not be any. Thus the system will reply with a "Destination Unreachable (Port Unreachable)" ICMP message to the spoofed address which will not be received at that address. As these types of request will be very large in number, benign packets will not be serviced, causing a successful DoS attack.

In an SDN environment also, this will cause problems as the controller will not be able to find the specified addresses to which the replies have to be delivered. Thus the controller CPU and memory will get exhausted with such a huge number of requests and thus leading to a successful DoS attack. If the attack is a collective one from multiple sources, it can be considered as a DDoS attack.

This spoofed UDP packets can be controlled by creating flow rules for allowing packets from True Hosts only and denying those from illegal hosts. With the mitigation feature enabled, the incoming requests will be checked with the True Host List. If any mismatch occurs, corresponding counters in the spoofed-UDP list for the switch will get incremented as seen in ICMP spoofed attack.

There is a thread for UDP also, which leaves sleep at every 2 second interval. It will check the UDP spoof count and create a flow rule to the source switch to drop all the UDP packets coming to the attack port. Since this flow rule has lower priority than the proactive flow rule for forwarding the true host packets to the controller, these benign packets are not affected by the new flow rule.

The above procedure shows how to handle spoofed UDP packets but there can be UDP attacks from genuine hosts by putting random destination ports in the packets and flooding such packets. In this situation, the target system will not have such UDP server applications at those ports and hence they will reply with "Destination Unreachable" type ICMP messages with code as "Port Unreachable". Thus if such messages come from the target system in huge numbers, it can be considered as an effect of UDP flood attack. Here the source of the UDP flood should be blocked from sending such UDP packets. This host address can be obtained from the destination address field of those ICMP messages. For each such ICMP message the system will increment a counter in its udp-flood-icmp-list for that attacking host. The same thread used to handle the UDP spoofed flood will also manage this list and if the count of such ICMP messages reaches a threshold, the controller will send flow rules to block UDP packets from that host. After a particular time, this rule will expire. If the attack still persists, the same procedure repeats. The steps for UDP DoS attack are given in the Algorithm UDP-Flood-Mitigation.

Algorithm 3 - UDP-Flood-Mitigation

1. *If a UDP packet-in message comes to the controller then*
    - *1.1. If the address details are in the True Host list then*
        - *1.1.1. Send a packet-out message to forward the UDP packet.*
        - *1.1.2. Install L3 flow rule to the switch for such packets.*
    - *1.2. Else*
        - *1.2.1. Make an entry in the spoofed-udp list for that switch and port if not already made, and increment its count.*
        - *1.2.2. If the packet is not buffered in the switch*
            - *1.2.2.1. The controller just drops it*
        - *1.2.3. Else*
            - *1.2.3.1. Tell the switch to drop it.*
        - *1.2.4. End If*
    - *1.3. End if*
2. *End if*

Thread for the spoof-check-UDP

3. *For each entry of the spoofed-UDP list, if the count exceeds a limit do*
    - *3.1. Alerts as an attack.*
    - *3.2. Send flow rules to the source switch to block all UDP packets coming to the attack port.*
    - *3.3. Remove the list entry.*
4. *End For*
5. *For each entry of the udp-flood-icmp list, if the count exceeds a limit do*
    - *5.1. Alerts as an attack.*
    - *5.2. Send an L3 rule to the attack switch to drop the UDP packets for the specific source-destination address pair*
    - *5.3. Removes the list entry.*
6. *End for*

D. TCP SYN Flooding Attack Detection and Mitigation

The Transmission Control Protocol (TCP) is a network layer protocol in the TCP/IP protocol suite. It is a connection oriented and reliable service. This means that there must be a connection between the source and the destination hosts. The way in which the connection is established is called a Three-Way Handshake. In this method, device A first sends a SYN message to another device B asking to establish a connection. B, if ready for connection, will acknowledge this and send back a connection request to A. These two are combined into one message called the SYN+ACK message. While receiving this message, A will acknowledge this by a ACK message and thus a bidirectional connection is established. This is followed by the data transfer phase. Thus a proper TCP connection establishment phase consists of SYN, SYN+ACK and ACK messages. Tracking this flow of messages can assure a proper TCP connection. After the data transfer, the connection is terminated formally by a Finish (FIN) message. For abrupt termination, a Reset (RST) message is used.

When the SYN packet is received at B, it will reserve some resources for the connection with A. This can be abstracted as making a table entry in its TCP connection table. After sending the SYN+ACK message, B will wait for the ACK message from A, keeping the table entry as such. If no ACK is received until the wait timer expires, it will again send a SYN+ACK message and wait for the ACK. This happens many times (usually 3 times). After all the attempts fail, B will delete the entry for A in its table. If A has sent the packet with a spoofed address, the SYN+ACK will be sent to that spoofed address which will never send back an ACK message. If such a spoofed address exists, that host will send back an RST message to reset the TCP connection table entries. During all this time the resources at B will be waiting for a non existing host and hence gone wasted. If there are hundreds of such requests, the TCP connection table at B will get exhausted and benign hosts will not get a chance to connect to B. Thus it is a type of DoS attack, called the SYN Flooding attack. Handling such requests in SDN will exhaust its resources also and creates a DoS attack in SDN. If the attack is from a collection of hosts, the attack becomes a DDoS attack.

The attack can be mitigated in two ways. The first way is used when the packet has spoofed address fields. This can be taken care of by the True Host List as seen in the ICMP and UDP flood mitigations. The controller uses a spoofed-TCP list to count the number of such spoofed packets. This way can be thought of as the Host Based mitigation. The second way is used if the source address is in the True Host list but that host is not sending back the ACK packet when the target sends a SYN+ACK message. Thus tracking the state of the 3-way handshake can prevent such attacks. When a SYN packet from a true host arrives at the controller, it will create an entry for that connection in a tcp-syn-only-conn list. When a corresponding SYN+ACK message comes, the controller will just tell the switch to forward the packet to the destination (which is the actual sender of the SYN packet). When a corresponding ACK packet is received at the controller, which may be piggybacked with a data packet or simply as an ACK packet, the controller considers it as a legal connection. It will then tell the switch to forward the packet as usual and install a Layer 4 flow rule with higher priority at both the source and destination switches. In all the intermediary switches, normal L3 rules are installed as if there are no mitigation features working in the controller. Along with this, the tcp-syn-only-conn list for the source-destination entry will be deleted. This is the case of normal TCP packets. In SYN flood attacks, the number of entries waiting for the ACK packets will be high. There is a thread syn-flood-check-TCP which wakes up every 2 seconds to check this list. If the count of such entries matches a threshold, the controller will install Layer 3 flow rules to block new TCP packets for that source-destination pair for some time (say, 10 seconds). As usual, this will not affect ongoing TCP traffic. The steps for this can be seen in Algorithm TCP-Flood-Mitigation.

Algorithm 5 - SYN-Flood-Mitigation

1. *If the TCP packet address details are present in the True Host List then*
    - *1.1. If the packet is a SYN packet then*
        - *1.1.1. Add the packet details to a tcp-syn-only-conn state list.*
        - *1.1.2. Tell the switch to forward the packet as usual.*
    - *1.2. Else*
        - *1.2.1. If the packet is a SYN+ACK packet then*
            - *1.2.1.1. Tell the switch to forward the packet as usual.*
    - *1.3. Else*
        - *1.3.1. If the packet is an ACK one and not RST or FIN ones then*
            - *1.3.1.1. Tell the switch to forward the packet as usual.*
            - *1.3.1.2. Install L4 flow rules at the source and destination switches.*
            - *1.3.1.3. Removes the state list entry.*

1.3.2. End If
1.4. End If
2. *End If*
3. *Else*
   3.1. Add the switch-port pair to the spoofed-tcp list.
4. *End If*

Thread for the spoof-check-TCP
5. *For each entry of the spoofed-TCP list, if the count exceeds a limit do*
   5.1. Alerts as an attack.
   5.2. Send flow rules to the source switch to block all TCP packets coming to the attack port.
   5.3. Remove the list entry.
6. *End For*

Thread For the syn-flood-check-TCP
7. *For each entry of the tcp-syn-only-conn list, if the count exceeds a limit do*
   7.1. Alerts as an attack.
   7.2. Send an L3 rule to the source and destination switches to drop new TCP packets between them.
   7.3. Removes the list entry.
8. *End For*

IV. IMPLEMENTATION

In the first subsection, the hardware and software setup used for the implementation and testing is explained. Next, the experiment conducted with the implementation is detailed and in the last subsection, the analysis of the result is explained.

A. Setup

The implementation of DDoSMitigator is done as an application on the RYU SDN controller [16], inheriting some of the functionalities of the Simple Switch 13. The application is developed, implemented and tested on a system having Intel i7 processor and 8GB RAM. The basic application used is the Switch App with L3 learning functionality and flow rule timeout set to 10 second. RYU is free, open source and written in Python language. RYU is well suited for both experimental purposes and real world installations. The protocol used at the Southbound Interface is the OpenFlow version 1.3.5 [13][14][17]. The network topology is set up using the network topology emulator Mininet [15]. The topology includes a controller, 5 OpenFlow switches (s1 to s5) and 15 hosts (h1 to h15) such that the switches are connected linearly and 3 hosts are connected to each switch. The test topology is given Fig. 4. The controller and the rest of the topology are operated in the separate operating systems (Lubuntu 18.04.5 LTS) in two virtual machines. Oracle VirtualBox [19] is used to create virtual machines.

B. Experiment

The experiment is done in a software environment. To get the exact results at the attack situations with and without the feature implementations, similar types of network traffic must be there. This is the same in normal situations also. Thus an automated script has been created which creates the network topology and generates the required traffic. The working of this automated script is as follows.

After the network is up and started running, a web server is created at the host h5 so that when an http request comes to it, a sample web page is returned. As the http uses TCP in the transport layer, this helps to test the TCP traffic and attack scenario. For normal TCP traffic, hosts h1, h7 and h12 will send http requests to the http server in some random but small intervals and get the sample web pages in return. At host h3, a UDP server is started at port 5001 in order to test the UDP transfer. A UDP packet towards h3 at port 5001 will not generate another packet, but if it is intended for some other ports, h3 will reply with a "Destination Unreachable" ICMP message with the code "Port Unreachable". Some UDP packets from hosts h4 and h11 are transmitted to this host intermittently to account for the UDP traffic. Also ICMP echo requests are transmitted from hosts h2, h6 and h8 to random destinations in the network and get back the ICMP echo reply in return.

All of these packet transfers constitute normal traffic in the test network. For testing the proposed features and analysing various comparison factors, this setup is run for half a minute with and without the presence of the new features and gets the logs for analysis. Functionality for writing various measures into CSV files is included in this proposed custom controller app. The measures consist of CPU and memory utilisation of the controller, packet-in counts per switch, ARP counts, aggregate flow statistics, and port and flow statistics per port per switch. When the features are enabled, the attack logs are also written if any attacks are detected by the controller.

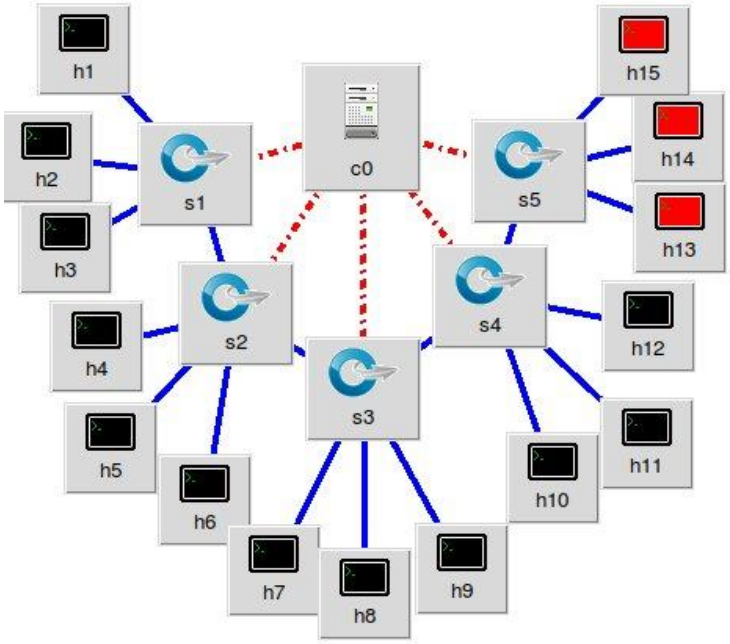


Fig. 4 Test Topology

For testing, the attacks originated from hosts h13, h14 and h15 through the switch s5. ICMP attack is generated by flooding ICMP echo packets from host h13 to h9 with spoofed source addresses for about 15 seconds. To test whether the attack has taken effect, the host h14 will try to ping h9 two times. Then the network is kept idle for 30 seconds so that it will come back to normal mode.
Then a UDP flood is generated from host h14 to h3 with spoofed source addresses targeting the port 5001 for about 15 seconds. During the attack, the host h15 will send two UDP packets to h9 targeting the port 5002 so that, if the packet teaches the controller it will reply with a Destination Unreachable ICMP message back to h15. If this message is not generated, we can assume that the UDP packet has not reached the destination. Again, 60 seconds cool off time for the network to come back to normal mode.
The next attack is a TCP SYN flooding attack from host h15 to h5 with spoofed source addresses and lasts for about 15 seconds. As h5 is a web server, the attack may overload it and benign requests will not be attended. To check this, host h13 will send http requests to h5 two times with 3 second intervals.
After an interval of 30 seconds, host h15 will flood TCP SYN packets with actual source and destination addresses to h5 for 15 seconds. During this time, host h14 will send http requests to h5 two times at 3 second intervals to check whether the server can respond to the request or not. After the attack the network will wait for some seconds and close down all the connections.
The test has been conducted as four scenarios; Normal and Attack, with and without DDoSMitigator.

C. Result

When the network runs in Normal mode with and without the DDoSMitigator, no visual difference can be seen. The network works normally with and without the new features. Hence, at a first glance, the proposed features do not have an impact on performance. The difference can be seen when there are ICMP, UDP or TCP SYN attacks. This is the situation where the features show their benefits. Without the features, during the ICMP attack, host h14 can't ping h9. This means that the switch/controller has been tightly overloaded with packet processing or the switch-controller link has been exhausted, denying the service for packets from true hosts. When the DDoSMitigator features are engaged, blocking of spoofed packets at the controller starts at the very first moment of the attack and as the number of such packets increases, mitigation rules are applied.

Here the rule is to block the ICMP echo packets for some time (here 10 seconds for all types protocols) at the port from which such packets are originated. Thus the SDN controller, the switch and also the hosts are escaped from exhaustion. The block rules are self-expiring with a limit of 10 seconds and hence will not take the flow table space for long. Here one thing to be noted is that even if the ICMP echo requests are blocked, those from true hosts can be forwarded without any restriction. This means that the False Positive Rate (FPR) is the minimum or null.

As the UDP packets do not need connection to travel towards the intended destination and there are no acknowledgements, flooding causes a huge number of packets generated from the source h14 within a very short time. These packets overwhelm the switch/controller very much faster than any other protocols and hence benign packets do not get any chance to be processed by the controller, leading to Denial of Service.

During this attack, the host h15 tries to send UDP packets to the UDP server at h3 but to a different port. If these packets had reached the server, it would have been replied with Destination Unreachable ICMP messages. But here it is not happening as the UDP packets do not reach the controller due to heavy packet flooding. Enabling the mitigation features, packets with spoofed addresses are blocked at the controller denying them from reaching the destination. If the number of such packets increases beyond a threshold, the UDP block rule for that port is applied for 10 seconds, after which the rule expires. If the attack persists, again the same rule is applied.

This will not affect any ongoing or new UDP flows from true hosts and hence FPR is again less. Here the UDP packets from h15 can reach the UDP server and it replies with Destination Unreachable message, indicating successful processing of the UDP messages by the controller and thus the reception at host h3.

TCP attack is done by flooding spoofed TCP SYN packets to the http server. Without mitigation measures, this will exhaust the switch/controller and thus subsequent http requests from host h13 do not get serviced. When the mitigation measures are included, the TCP spoofed packets are handled in the same way as that of ICMP and UDP. That is, when TCP packets with spoofed addresses come to the controller initially, they are blocked there.

If the count of such packets increases beyond the threshold within the mitigation thread interval, block rules will prevent sending spoofed packets at the switch port itself. Here also, FPR is very less as the packets from true hosts only are allowed to pass through the switch.

The last attack is the TCP SYN flood attack with packets having actual addresses. There will be only one flow rule created in the switch for the traffic but the packet transfer is huge.

The controller also does not get overwhelmed but the switches have a large number of packets to forward. As the TCP table in the server becomes full and the switch becomes heavily busy in forwarding the packets, benign requests are not serviced. But when the mitigation rules are applied, the number of first handshake packets (SYN packets) and third handshake packets (ACK packets) are noted by the controller. If the number of unattended SYN packets for a specific pair of addresses crosses a limit within a particular time interval (say, 2 seconds), an attack is announced, blocking the source from generating SYN packets to the destination for some time (10 seconds). If the attack persists, the action continues. As this will not affect normal TCP ongoing traffic, FPR is very less here also.

The on-screen log that is generated during the attacks is given in Fig. 5. This log is populated by the DDoSMitigator and is set store in a log file for later processing also.

D. Analysis

There are different logs for each of the normal flows without and with mitigation features, and for each of the attack scenarios without and with the mitigation features. All the measures are plotted against time. The first analysis is among these to know whether the mitigation features have better performance with low overhead. The next analysis compares the results of the mitigation features with some previous works in this area of focus.

1) *CPU and Memory Utilisation:* From the logs, it can be seen that under normal working, the controller CPU and memory utilisation with DDoSMitigator enabled are in line with those of the basic controller. Fig 6 (a) shows the comparison of CPU and memory utilisation of the controller without and with enabling the mitigation features under normal working. Here the CPU usage and Memory are of very low and that is due to the background normal traffic. Fig. 6 (b) shows the same at the attack time. The four peaks of the CPU usage without mitigation features indicate the four attack scenarios stated above (ICMP, UDP, TCP spoofing and TCP SYN Flood). They are greater than 40 percent. The CPU usage is much higher than that with mitigation features where it is steady. This is the same with memory consumption also.

```
(0:00:22.227795): *****ICMP FLOOD ATTACK at [switch:5][port:2]. Applying restric
tion rules for 10 seconds*****
(0:00:33.287496): *****ICMP FLOOD ATTACK at [switch:5][port:2]. Applying restric
tion rules for 10 seconds*****
(0:00:34.297871): *****ICMP FLOOD ATTACK at [switch:5][port:2]. Applying restric
tion rules for 10 seconds*****
(0:01:03.415790): *****UDP FLOOD ATTACK at [switch:5][port:3]. Applying restrict
ion rules for 10 seconds*****
    1 Unwanted service request from 00:00:00:00:00:0f/10.0.0.15 to 00:00:00:00:0
0:03/10.0.0.3
    2 Unwanted service request from 00:00:00:00:00:0f/10.0.0.15 to 00:00:00:00:0
0:03/10.0.0.3
(0:01:14.453792): *****UDP FLOOD ATTACK at [switch:5][port:3]. Applying restrict
ion rules for 10 seconds*****
(0:02:10.543440): *****TCP FLOOD ATTACK at [switch:5][port:4]. Applying restrict
ion rules for 10 seconds*****
(0:02:21.592356): *****TCP FLOOD ATTACK at [switch:5][port:4]. Applying restrict
ion rules for 10 seconds*****
(0:02:45.309535): SYN Attack from 10.0.0.15 (switch 5, port 4) to 10.0.0.5 (swit
ch 2, port 4). Blocked for 10 seconds
(0:02:48.316922): SYN Attack from 10.0.0.15 (switch 5, port 4) to 10.0.0.5 (swit
ch 2, port 4). Blocked for 10 seconds
```

Fig. 5 Test On-screen Log of Attack

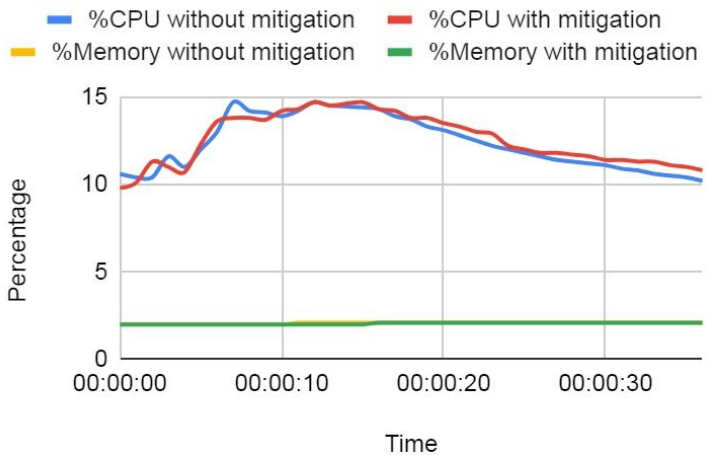


Fig. 6 (a) CPU and Memory Utilisation under Normal Conditions

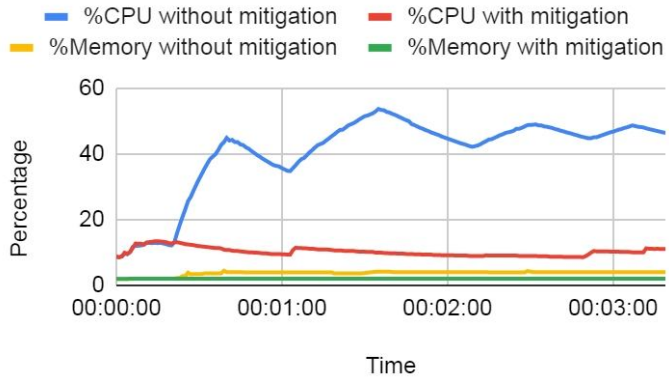


Fig. 6 (b) CPU and Memory Utilisation under Attack Conditions

2) *Packet-in Count:* This analysis is done to understand how many packet-in messages have come to the controller during the four types of attacks without and with DDoSMitigator. The graph can be seen in Fig. 7. This is worth noting that the numbers of packet-in messages are huge during the four attacks without DDoSMitigator whereas in the mitigation method, when the packet-in messages starts to increase, the countermeasures are applied and it again falls to normal. The remaining messages are that of the normal flow.

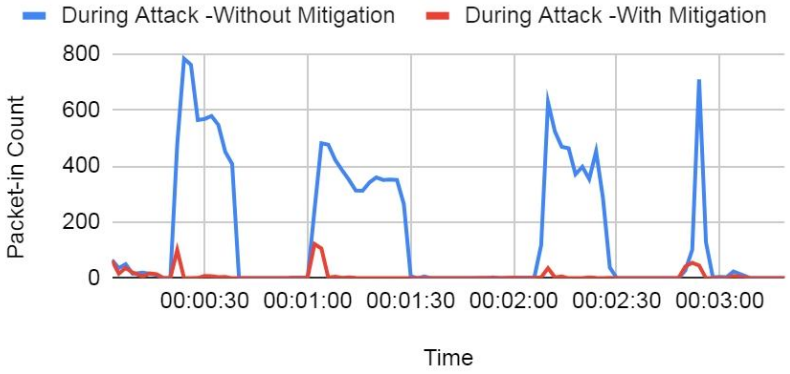


Fig. 7 Packet-in Count at the Attacker Switch

3) *Flow Table Utilisation of the Attacker Switch and the Target Switch under Different Attacks:* In all the three spoofed packet flooding attacks (ICMP, UDP and TCP), the number of flows created at the attacker switch and the target switch are huge. These are the unnecessary flow rules built for the spoofed addresses. That means the flow table is occupied with useless flow entries. As the flow table size is limited, unnecessarily increasing its usage will lead to flow table overflow. This does not happen when DDoSMitigator is applied. Here only the necessary mitigation rules are applied at the switches and hence the flow table space is very much saved. This can be seen in Fig. 8 (a), (b) and (c). In the fourth scenario (TCP SYN flood) shown in Fig. 8 (d), as the number of source and destination addresses in all the packets are the same, this does not create a lot of table entries in the basic controller app. When DDoSMitigator is applied, it will enforce flow rules to prevent such SYN floods towards the source and destination switches. But as the destination switch generates a lot of SYN+ACK packets and RST packets many times and DDoSMitigator creates some mitigation rules. This is not a big issue as it will cut down the amount of packet flow through the switches, saves their resources and helps the server from crashing.

4) *Flow Table Utilisation of All Switches:* Figure 9(a) and (b) shows the usage of flow tables of all the five switches in the test topology during the attack scenarios. It is obvious that without the mitigation system the first three attacks (ICMP, UDP and TCP Flood) fill the flow tables of all the switches using useless flow entries. Small attacks like this can make almost 3000 flow entries in all the switches. This will be very huge in the real world and will exhaust the flow tables very quickly. The graphs are irregular because there is random traffic as part of normal flows of packets in the background and the flow counts vary from time to time as each rule has auto expiry time of maximum 10 seconds. As seen earlier, the fourth attack does not increase the flow entry count but overwhelms the switch and the host. With DDoSMitigator, the peak number of flow rules is below 50. This means mitigation can be possible without any type of useless flow rules.

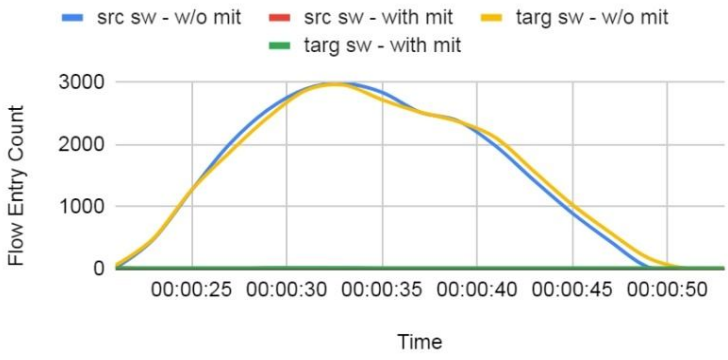


Fig. 8 (a) Flow Table Utilisation during ICMP Flood at the Attacker Switch and the Target Switch - without and with DDoSMitigator

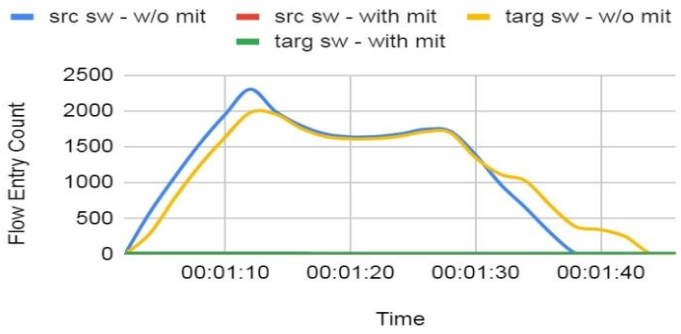


Fig. 8 (b) Flow Table Utilisation during UDP Flood at the Attacker Switch and the Target Switch - without and with DDoSMitigator

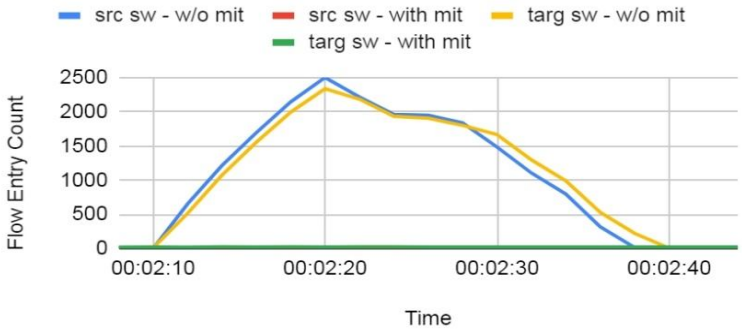


Fig. 8 (c) Flow Table Utilisation during TCP Flood at the Attacker Switch and the Target Switch - without and with DDoSMitigator

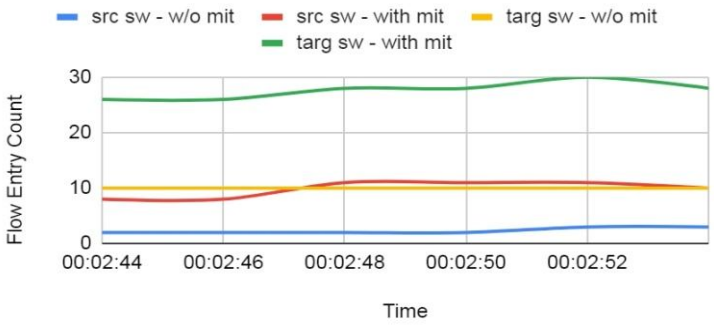


Fig. 8 (d) Flow Table Utilisation during TCP SYN Flood at the Attacker Switch and the Target Switch - Without and with DDoSMitigator

5) *Recall Rate and False Positive Rate:* Recall rate is the ratio of number of packets identified as spoofed to the number of actual spoofed packets. Since all the packets that are spoofed can be identified easily by comparing with the true host table, the recall rate is 100%. False Positive Rate (FPR) is the ratio of the number of benign packets detected as attacking packets to the total number of packets. To find the FPR in this method, ICMP spoofed packet flow at 500 packets per second is generated from a host towards another host. Meanwhile, from the same host 10 ICMP echo messages are transmitted to the target host. All the 10 have reached the destination and replies received back at the attacker host. On statistics for a time period of 10 seconds, 5546 spoofed packets and 20 benign packets have been transmitted. This means the 20 packets are identified as benign ones and all others as attacking packets. It can be assured that over a longer time period also, the FRP will be close to zero and the recall rate will be 100%. This is much better than many previous implementations. Table 1 [10] shows comparison of Recall Rate and False Positive Rate of FloodGuard [4] and FloodDefender [10] with DDoSMitigator.

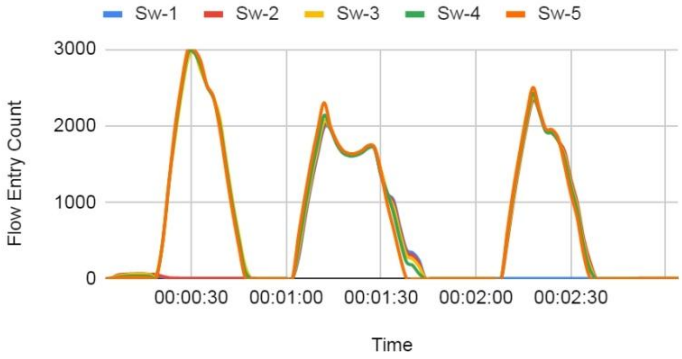


Fig. 9 (a) Flow Table Utilisation of All Switches without DDoSMitigator

Fig. 9 (b) Flow Table Utilisation of All Switches with DDoSMitigator

TABLE 1 [10]
RECALL RATE AND FALSE POSITIVE RATE

Proposal	Recall Rate	FPR
FloodGuard	100% (10/10)	10% (1/10)
FloodDefender	100% (10/10)	0% (0/10)
DoSMit	100% (10/10)	0% (0/10)

6) Time Delay: The time delay can be thought of as the time duration for a packet to reach the destination. When heavy packet flow occurs, the delay may be larger. The delay is calculated as the average of the to and fro time between the source and the destination. As the rules are quick and robust the delay is very much low in DDoSMitigator compared to that of the ordinary openflow network, FloodGuard and FloodDefender. This is due to the blocking of attacking packets at their very stages of detection. Table 2 shows the comparison of time delay of 100 packets per second (pps) UDP based attack and 3 shows that of 500 pps attack. Here the normal packets are sent at 100 pps and 500 pps respectively. In both cases, it can be found that DDoSMitigator outperforms the other proposals.

TABLE 2 [10]
TIME DELAY OF NORMAL FLOWS UNDER 100 PPS UDP-BASED ATTACKS

Delay	OpenFlow	FloodGuard	FloodDefender	DoSMit2
Min. Delay	10.9ms	0.3ms	0.3ms	0.019ms
Avg. Delay	1843ms	15.1ms	17.5ms	0.086ms
Max. Delay	TimeOut	TimeOut	4913ms	16.783ms

TABLE 3 [10]
TIME DELAY OF NORMAL FLOWS UNDER 500PPS UDP-BASED ATTACKS

Delay	OpenFlow	FloodGuard	FloodDefender	DoSMit2
Min. Delay	10.7ms	0.4ms	0.3ms	0.005ms
Avg. Delay	2038ms	29.2ms	18.7ms	0.180ms
Max. Delay	TimeOut	TimeOut	4891ms	48.071ms

7) *Scalability:* DDoSMitigator is highly scalable that with every new host only two proactive rules are added to the connected switch and thereby protects the network from the attacks that may happen from that host. It has been already found that all the other resources are utilised in the same amount as that of a basic controller app.
8) *Overall Feature Comparison:* Table 4 shows an overall comparison among different previous proposals [10] with DDoSMitigator. The benefits of DDoSMitigator can be summarised as greater attack detection and prevention, low delay, less flow table usage, very low false positive rate, no data plane extension, no additional hardware and prevent all types of attacks (ICMP, UDP and TCP, as said in previous proposals).

TABLE 4 [10]
COMPARISON OF DIFFERENT PARAMETERS

Proposals	AvantGuard [1]	FloodGuard [4]	FlowKeeper [7]	FloodDefender [10]	DDoSMitigator
Controller App		Yes	Yes	Yes	Yes
Data plane Extension	Yes				
Additional Device		Yes	Yes		
Support Protocols	TCP	All	All	All	All
Benign Traffic Delay	Medium	Mostly low	High	Low	Very low

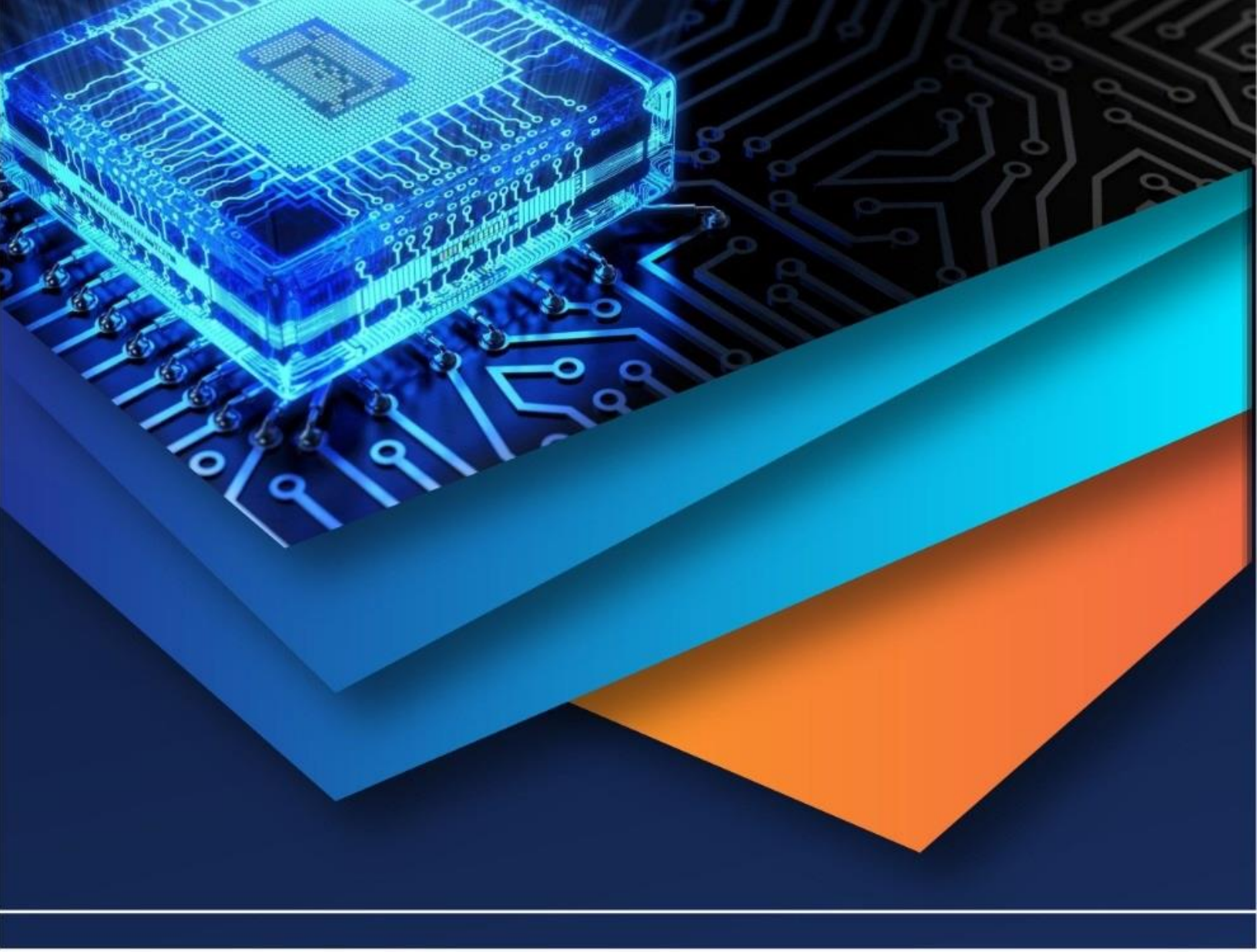
V. CONCLUSION

DDoSMitigator follows an on-the-fly mechanism for detecting and mitigating the DoS and DDoS attacks in a quicker way. On all the factors of comparison such as CPU, memory, performance, flow table utilisation and delay, this method outperforms many older proposals. With very low FPR, it can mitigate all the protocol attacks during its early stages of detection itself, saving valuable resources and providing service to all benign packets. Though the system is independent of scaling, it is to be tested in a larger environment with more number of hosts. Also there are other types of attacks which DDoSMitigator cannot address. Even though DDoSMitigator is proposed to stop DoS/DDoS attacks, with some improvements, it can be extended to prevent some other types of attacks such as intrusion and man-in-the-middle attacks. In DDoSMitigator, the true host table is constructed based on the ARP request/response pattern. It can be further modified to know the true hosts by monitoring the DHCP packets.

REFERENCES

[1] 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.
[2] 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.

[3] Shin, Seungwon, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. "Rosemary: A robust, secure, and high-performance network operating system." In Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, pp. 78-89. 2014.
[4] 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.
[5] Wang, Tao, and Hongchang Chen. "SGuard: A lightweight SDN safe-guard architecture for DoS attacks." China Communications 14, no. 6 (2017): 113-125.
[6] Wang, Tao, Zehua Guo, Hongchang Chen, and Wei Liu. "BWManager: Mitigating denial of service attacks in software-defined networks through bandwidth prediction." IEEE Transactions on Network and Service Management 15, no. 4 (2018): 1235-1248.
[7] 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.
[8] 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.
[9] Maddu, Jeevan Surya, Somanath Tripathy, and Sanjeet Kumar Nayak. "SDNGuard: An Extension in Software Defined Network to Defend DoS Attack." In 2019 IEEE Region 10 Symposium (TENSYMP), pp. 44-49. IEEE, 2019.
[10] Gao, Shang, Zhe Peng, Bin Xiao, Aiqun Hu, Yubo Song, and Kui Ren. "Detection and Mitigation of DoS Attacks in Software Defined Networks." IEEE/ACM Transactions on Networking 28, no. 3 (2020): 1419-1433.
[11] Varghese, Josy Elsa, and Balachandra Muniyal. "An Efficient IDS Framework for DDoS Attacks in SDN Environment." IEEE Access 9 (2021): 69680-69699.
[12] Tang, Dan, et al. "Performance and Features: Mitigating the Low-Rate TCP-Targeted DoS Attack via SDN." IEEE Journal on Selected Areas in Communications 40.1 (2021): 428-444.
[13] 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.
[14] OpenFlow Switch Specification [online] https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf
[15] Mininet Walkthrough [online] http://mininet.org/walkthrough/
[16] RYU SDN Framework [online] https://osrg.github.io/ryu-book/en/Ryubook.pdf
[17] OpenFlow protocol API Reference [online] https://ryu.readthedocs.io/en/latest/ofproto ref.html
[18] SDN Architecture [online] https://telsoc.org/journal/ajtde-v3-n4/a28
[19] Oracle VM VirtualBox [online] https://www.virtualbox.org/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY