AN ECOSYSTEM FOR VULNERABLE TRAFFIC ANALYSIS AND MITIGATION SERVICES IN SOFTWARE DEFINED NETWORKING

Ponmanikandan V[1], Ramprasath J[2], Rakunanthan K S[3], Santhosh Kumar M[4]

[1]*U.G. Student, Department of Information Technology, Dr. Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India*
[2]*Assistant Professor, Department of Information Technology, Dr. Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India*
[3]*U.G. Student, Department of Information Technology, Dr. Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India*
[4]*U.G. Student, Department of Information Technology, Dr. Mahalingam College of Engineering and Technology, Pollachi, Tamil Nadu, India*

---***---

Abstract - *Software Defined Networking (SDN) is an architecture that aims to make networks agile and flexible. The goal of SDN is to improve network control by enabling enterprises and service providers to respond quickly to changing business requirements. SDN consists of three layers, they are Application, Control, and Infrastructure. The application layer consists of Business Applications. The Control layer consist of the SDN Controller which can be programmed based on user needs and the Infrastructure layer consists of Switches and Gateway machines. To communicate between these layers, SDN uses northbound and southbound application program interfaces (APIs) where the northbound API communicates between the infrastructure and control layers and the southbound API communicates between the application and the control layers. The distinct advantage of Software Defined Networking is the separation of the control plane and the data plane. In this project we manage the control plane to mitigate Denial of Service (DoS) attacks by making a set of rules. These rules are obtained by passing data packets into K-Means algorithm to preprocess it and into a user defined algorithm. Denial of Service is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic or sending it information that triggers a crash. In both instances, the DoS attack deprives legitimate users (i.e. employees, members, or account holders) of the service or resource they expected. K-Means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. The user defined algorithm makes use of the preprocessed packets from the K-Means algorithm to analyze and detect any abnormal data traffic. The algorithm also updates the firewall of SDN to provide mitigation services.*

Key Words: **Software Defined Networking, DoS Attacks, Traffic Analysis, K-Means Clustering, Internet Protocol.**

1.INTRODUCTION

SDN is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services. The OpenFlow® protocol is a foundational element for building SDN solutions. The SDN Architecture is directly programmable, agile, centrally managed, programmatically configured and is open standard based and vendor neutral.

SDN consists of three layers namely Application layer, Control layer and Infrastructure layer. Application layer is open area to develop as much innovative application as possible by leveraging all the network information about network topology, network state, network statistics, etc. There can be several types of applications which can be developed like those related to network automation, network configuration and management, network monitoring, network troubleshooting, network policies and security. Such SDN applications can provide various end-to-end solutions for real world enterprise and data centre networks. Network vendors are coming up with their set of SDN applications. Control layer is the land of control plane where intelligent logic in SDN controllers would reside to control network infrastructure. This is the area where every network vendor is working to come up with their own products for SDN controller and framework. Here in this layer, a lot of business logic is being written in controller to fetch and maintain different types of network information, state details, topology details, statistics details, and more. Infrastructure layer is composed of various networking equipment which forms underlying network to forward network traffic. It could be a set of network switches and routers in the data centre. This layer would be the physical one over which network virtualization would be laid down through the control layer (where SDN controllers would sit and manage underlying physical network).

In computing, a DoS attack is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. DoS is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. The two common DoS attacks are Smurf attack and SYN flood.

In a Smurf Attack, the attacker sends ICMP broadcast packets to a number of hosts with a spoofed source IP address that belongs to the target machine. The recipients of these spoofed packets will then respond, and the targeted host will be flooded with those responses. A SYN flood occurs when an attacker sends a request to connect to the target server but does not complete the connection through what is known as a three-way handshake—a method used in a TCP/IP network to create a connection between a local host/client and server. The incomplete handshake leaves the connected port in an occupied status and unavailable for further requests. An attacker will continue to send requests, saturating all open ports, so that legitimate users cannot connect.

2. LITERATURE SURVEY

Although Software-defined Networking (SDN) enables new network applications and more flexible control in dynamic network environments, security is still an important concern as it is not yet a built-in feature in the SDN architecture. Due to this reason more and more research work is done on SDN security to this demand. These works focus on dealing with various networking attacks. Increasingly, it is expected that current security schemes will operate in near real time to face a spectrum of threats, inspecting large volumes of traffic, and providing efficient identification of various network anomalies. We present some previous SDN-related anomaly detection efforts to meet this demand.

Aleroud and Alsmadi (2016) made the anomalous events that compromise the SDN architecture to fall into three categories: (i) attacks on the control plane, (ii) compromising of communication between the control and data planes, and (iii) threats designed to attack the data plane equipment. A flood attack can directly or indirectly cover these three categories because of the volume of traffic and the increasing number of connection requests. Excessive use of these network resources can overload the controller as well as occupy all the forwarding table entries of the devices in the data plane with flows from illegitimate connections. In this manner, despite the extensive work in SDN security, most of the approaches have been designed to contain flood attacks (e.g., DDoS attacks). Early work on DDoS detection in the SDN environment was reported by Braga, Mota, and Passito (2010). The presented system continuously observed the statistical features of the flows to identify any anomalous activity. The authors used Self Organizing Maps (SOM) with the topological neighborhood described by a Gaussian function to classify each packet as benign or abnormal.

Ha et al. (2016) stated that identifying flood attacks requires detailed processing and inspection of a large volume of traffic. In that light, the authors presented a traffic sampling strategy for SDN networks that maintains the total aggregate volume sampled below the processing capacity of an Intrusion Detection System (IDS). Mousavi and St-Hilaire (2015) proposed an entropy-based mechanism to detect a DDoS attack. In case of an attack, the entropy decreases by evaluating the randomness of incoming packets' destination IP addresses. In Xiao, Qu, Qi, and Li (2015) an effective detection approach based on traffic classification with correlation analysis (CKNN) was proposed. Although the detection rate of known anomalous events is high, such methods become incapable of recognizing variations of the same attack, because signature-based detection is used. For this same reason, the techniques must/should be trained with a large labeled dataset, which is not always possible to obtain. The ecosystem proposed in this paper overcomes these limitations because the detection adapts automatically to changing traffic patterns. Several studies employ alterations in the SDN infrastructure to conduct anomaly detection. Wang, Zhang, Singh, Lumezanu, and Jiang (2013) proposed NetFuse as a mechanism to protect against traffic overload in OpenFlow-based data center networks. In order to guard the network against the effects of malicious traffic, NetFuse sits between network devices and network controller as an additional layer. Joldzic, Djuric, and Vuletic (2016) presented an alternative solution based on an SDN network topology consisting of three layers to place the IDS. The outermost, located at the network gateway, contains an OpenFlow switch responsible for dividing the incoming traffic and forwarding the generated parts to the intermediate layer. The second layer is composed of several devices called processors, which perform the detection of an attack. In the third layer, the traffic generated by the processors is aggregated and sent to the interior of the network through an OpenFlow switch. Two disadvantages are observed in this approach. First, it assumes that the network is anomaly free, which can become a problem when anomalies are launched by the internal network itself that make the SDN controller unavailable. The second disadvantage is the division of traffic among the processors, which might lead to the masking of attacks, as due to load balancing, each processor can analyze disconnected parts of the same anomaly. Chen, Junuthula, Siddhrau, Xu, and Chao (2016) applied specialized software boxes to improve the scalability of ingress SDN switches to adjust the control plane's workload during DDoS attacks.

In this paper, we focus specifically on the SDN environment, in which the controller is able to collect and analyze traffic statistics reports from switches. Our proposed ecosystem is different from previous work because it does not require any change to the SDN infrastructure. There is also no need of human intervention to stop the attack. The efficiency of the ecosystem is also increased because of dynamic grouping using K-Means algorithm.

3. PROPOSED SYSTEM

The aim of the project is to create an SDN ecosystem which has the capabilities of identifying and mitigating DoS attacks using various features of SDN and user defined algorithm. The proposed system can be classified into five steps for clear understanding. They are creation of a virtual environment using Mininet, capture the packets that flow through the virtual environment using pyshark library, pre-process the captured data packets using K-Means algorithm, input the pre-processed packets into the user-defined algorithm to check for any kinds of anomaly, if any anomaly is detected update the firewall rules of the Mininet environment to block the corresponding user to prevent further attack.

3.1 Creation of virtual environment

Mininet is a network emulator, or perhaps more precisely a network emulation orchestration system. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.) The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middlebox, with a given amount of queueing. When two programs, like an iperf client and server, communicate through Mininet, the measured performance should match that of two (slower) native machines. In short, Mininet's virtual hosts, switches, links, and controllers are the real thing – they are just created using software rather than hardware – and for the most part their behavior is similar to discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform.

For this project we create an environment with four host machines and a switch. This is known as single topology in which all the host machines are connected to a single switch. The switch is directly connected to the controller. In this case we are using a remote controller known as the POX controller. POX is a networking software platform written in Python. We can now test the created environment by using various simple commands such as ifconfig, ping, nodes, links. This marks the end of the creation

of virtual environment. The Fig 1 shows the created virtual environment.
Code: sudo mn - - mac - - topo single, 4 - - controller remote

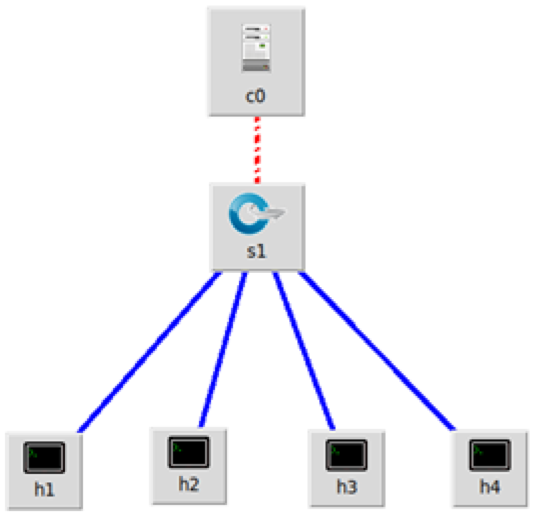


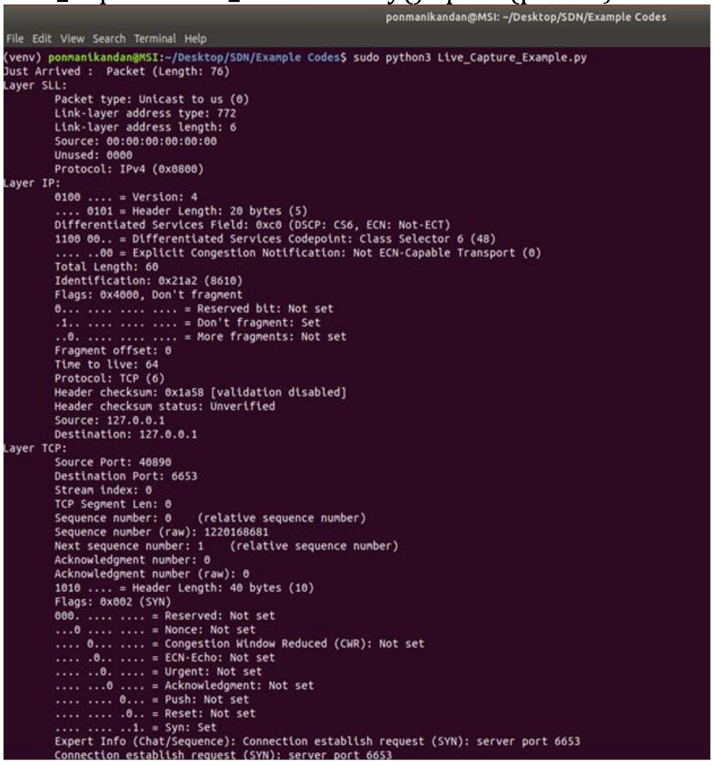
Fig -1: SDN Environment

3.2 Capturing the data packets

For the capture purpose we make use of the python environment along with pyshark library. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

PyShark is a wrapper for the Wireshark CLI (TShark), so we can have all Wireshark decoders in the PyShark. We can use PyShark to sniff an interface or we can analyze the pcap files. PyShark allows two types of packet analysis. They are file capture and live capture. The file capture mechanism requires a pcap file as an input to start the analysis process whereas the live capture is an advanced mechanism that allows real time capturing of data packets. The live capture requires the interface as an input and supports multiple filters for analysis. We now connect the Mininet interface to the PyShark to start the capture of data packets. The fig _ shows a captured data packet.

Code: Live_Capture = pyshark,LiveCapture(interface = 'any') for packet in Live_Capture.sniff_continuously() : print(packet)


ponmanikandan@MSI: ~/Desktop/SDN/Example Codes
File Edit View Search Terminal Help

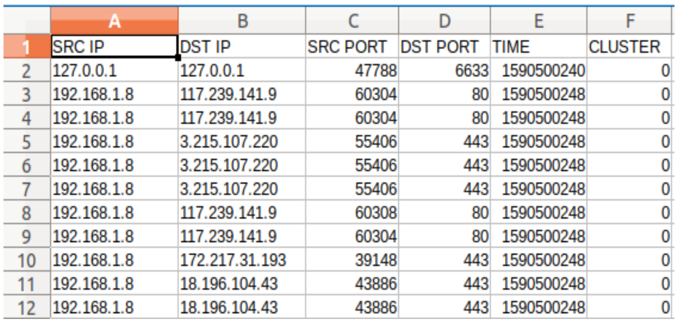
```
(venv) ponmanikandan@MSI:~/Desktop/SDN/Example Codes$ sudo python3 Live_Capture_Example.py
Just Arrived :  Packet (Length: 76)
Layer SLL:
        Packet type: Unicast to us (0)
        Link-layer address type: 772
        Link-layer address length: 6
        Source: 00:00:00:00:00:00
        Unused: 0000
        Protocol: IPv4 (0x0800)
Layer IP:
        0100 .... = Version: 4
        .... 0101 = Header Length: 20 bytes (5)
        Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
        1100 00.. = Differentiated Services Codepoint: Class Selector 6 (48)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
        Total Length: 60
        Identification: 0x21a2 (8610)
        Flags: 0x4000, Don't fragment
        0... .... .... .... = Reserved bit: Not set
        .1.. .... .... .... = Don't fragment: Set
        ..0. .... .... .... = More fragments: Not set
        Fragment offset: 0
        Time to live: 64
        Protocol: TCP (6)
        Header checksum: 0x1a58 [validation disabled]
        Header checksum status: Unverified
        Source: 127.0.0.1
        Destination: 127.0.0.1
Layer TCP:
        Source Port: 40890
        Destination Port: 6653
        Stream index: 0
        TCP Segment Len: 0
        Sequence number: 0    (relative sequence number)
        Sequence number (raw): 1220168681
        Next sequence number: 1    (relative sequence number)
        Acknowledgment number: 0
        Acknowledgment number (raw): 0
        1010 .... = Header Length: 40 bytes (10)
        Flags: 0x002 (SYN)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...0 .... = Acknowledgment: Not set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..1. = Syn: Set
        Expert Info (Chat/Sequence): Connection establish request (SYN): server port 6653
        Connection establish request (SYN): server port 6653
```

Fig -2: Live Capture

3.3 K-Means Clustering

The K-Means algorithm clusters data by trying to separate samples in n groups of equal variances, minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large number of samples and has been used across a large range of application areas in many different fields. The k-means algorithm divides a set of N samples into K disjoint clusters C, each described by the mean of the samples in the cluster. The means are commonly called the cluster "centroids"; note that they are not, in general, points from X, although they live in the same space. The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

We can make use of the SCIKIT K-Means library for doing the packet pre-processing. The library is very easy to import and implement and provides the cluster label for each entry. This data is then loaded from a CSV file using pandas library. The CSV file only contains the required packet details for faster processing. The fields include Source IP, Destination IP, Source PORT, Destination PORT, Timestamp. The K-Means algorithm makes use of the Port values and the timestamp values for clustering because IP addresses cannot be used for k-means clustering. The final CSV is created with the above columns along with a new column cluster. The cluster column consists of the cluster label for each data packet. The fig 3 shows the final CSV output.

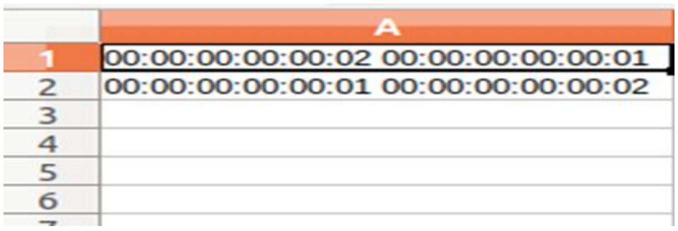


	A	B	C	D	E	F
1	SRC IP	DST IP	SRC PORT	DST PORT	TIME	CLUSTER
2	127.0.0.1	127.0.0.1	47788	6633	1590500240	0
3	192.168.1.8	117.239.141.9	60304	80	1590500248	0
4	192.168.1.8	117.239.141.9	60304	80	1590500248	0
5	192.168.1.8	3.215.107.220	55406	443	1590500248	0
6	192.168.1.8	3.215.107.220	55406	443	1590500248	0
7	192.168.1.8	3.215.107.220	55406	443	1590500248	0
8	192.168.1.8	117.239.141.9	60308	80	1590500248	0
9	192.168.1.8	117.239.141.9	60304	80	1590500248	0
10	192.168.1.8	172.217.31.193	39148	443	1590500248	0
11	192.168.1.8	18.196.104.43	43886	443	1590500248	0
12	192.168.1.8	18.196.104.43	43886	443	1590500248	0

Fig -3: K-Means Output

3. 4 User Defined Algorithm

The output of the K-Means algorithm is given as the input to the user defined algorithm. This algorithm is a complex IF condition which makes use of K-Means label for faster processing. The algorithm only takes up the cluster with maximum number of packets for processing because if an attack happens, all the packets will originate from the same user which intern increase a cluster size many folds. Now the large cluster is taken as input and processed with the help of Source IP, Destination IP and the Timestamp. If there are unreasonable number of data packets from a source to destination during a short time window, then the algorithm notes the IP address of both the source and destination. This is cross referenced with the MAC index to know their respective hardware addresses.

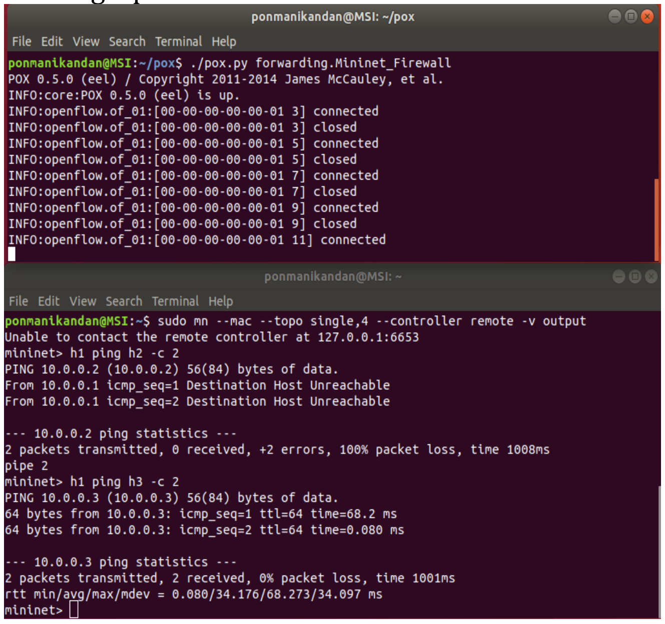


	A
1	00:00:00:00:00:02 00:00:00:00:00:01
2	00:00:00:00:00:01 00:00:00:00:00:02
3	
4	
5	
6	

Fig -4: Firewall Rules

3. 5 POX Firewall

The hardware address that is obtained by the previous step is stored in a CSV file. This CSV file is given as input to the POX controller during 'ConnectionUp' and 'PacketIN' event. The POX controller reads the CSV file which contains the MAC address and sends it to all the switches. This is stored in the flow table of the switches. Every packet is cross checked with the flow table before exiting the switch. If the switch identifies a packet from the flow table it will block the packets, preventing attacks. fig __ shows the CSV and fig 5 shows the POX controller blocking a packet.



```
ponmanikandan@MSI:~/pox$ ./pox.py forwarding.Mininet_Firewall
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] closed
INFO:openflow.of_01:[00-00-00-00-00-01 5] connected
INFO:openflow.of_01:[00-00-00-00-00-01 5] closed
INFO:openflow.of_01:[00-00-00-00-00-01 7] connected
INFO:openflow.of_01:[00-00-00-00-00-01 7] closed
INFO:openflow.of_01:[00-00-00-00-00-01 9] connected
INFO:openflow.of_01:[00-00-00-00-00-01 9] closed
INFO:openflow.of_01:[00-00-00-00-00-01 11] connected
```

```
ponmanikandan@MSI:~$ sudo mn --mac --topo single,4 --controller remote -v output
Unable to contact the remote controller at 127.0.0.1:6653
mininet> h1 ping h2 -c 2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1008ms
pipe 2
mininet> h1 ping h3 -c 2
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=68.2 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.080 ms

--- 10.0.0.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.080/34.176/68.273/34.097 ms
mininet>
```

Fig -5: Firewall Blocking

The process from step 2 to step 5 is repeated every 10 second window to effectively capture and analyze packets. Thus, DoS attacks are identified and mitigated in an SDN environment using various SDN features, K-Means Algorithm, and a User Defined Algorithm.

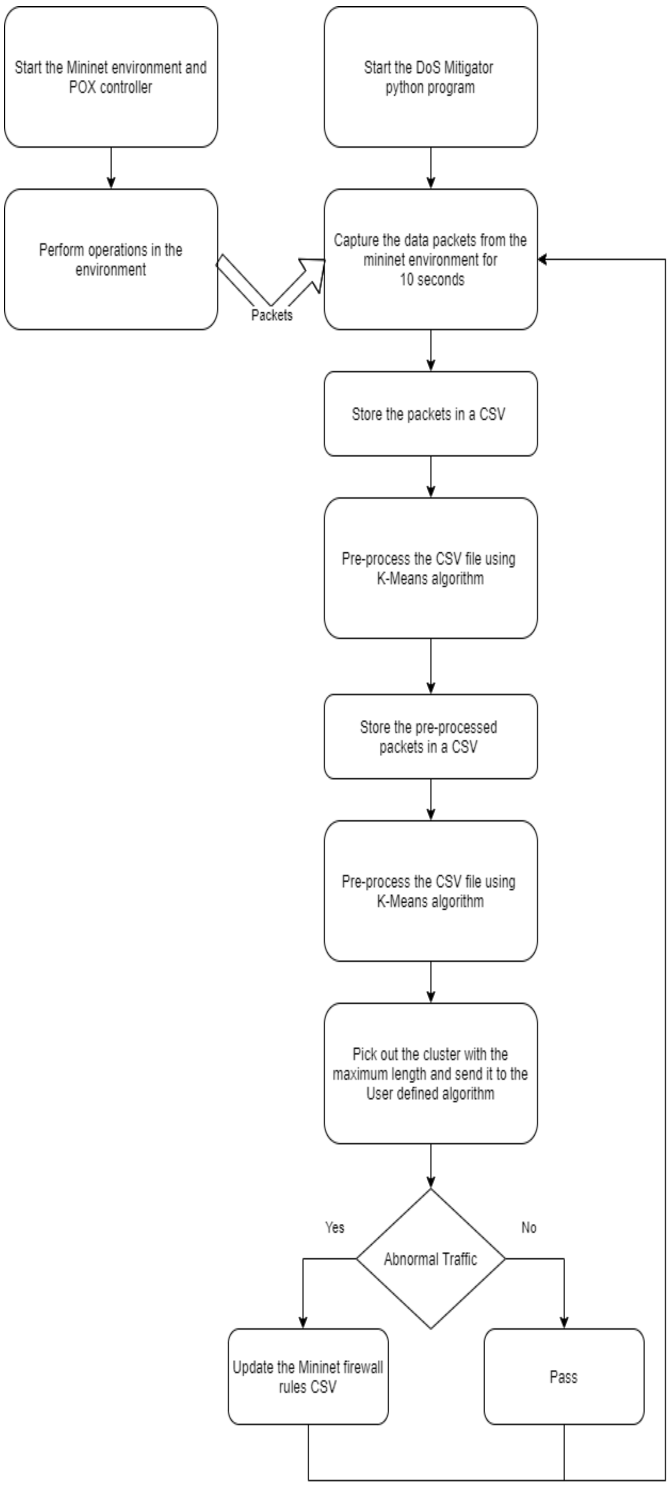

Start the Mininet environment and POX controller
Start the DoS Mitigator python program
Perform operations in the environment
Packets
Capture the data packets from the mininet environment for 10 seconds
Store the packets in a CSV
Pre-process the CSV file using K-Means algorithm
Store the pre-processed packets in a CSV
Pre-process the CSV file using K-Means algorithm
Pick out the cluster with the maximum length and send it to the User defined algorithm
Yes
No
Abnormal Traffic
Update the Mininet firewall rules CSV
Pass

Fig -6: DoS Mitigator workflow

4. ALGORITHMS AND PSEUDOCODE

4.1 K-Means algorithm

```
Step 1: Initialize cluster centers
Step 2: Assign observations to the closest cluster
        center
Step 3: Revise cluster centers as mean of assigned
        observations
Step 4: Repeat step 2 and step 3 until convergence
```

4.2 K-Means pseudo code

```
Input: k (number of clusters)
 D (a set of lift ratios)
Output: a set of k clusters
Method:
Arbitrarily choose k objects from D as initial cluster centers;
Repeat:
        1.(re)assign each object to the cluster to which object is the most similar, based on mean value of the objects in the cluster;
        2.Update the cluster means, i.e., calculate the mean value of the objects for each cluster
Until no change
```

4.3 User defined algorithm

```
Step 1: Input the K-Means pre-processed CSV
Step 2: Select the packets of cluster with maximum
        length
Step 3: Order the packets with timestamp
Step 4: For each timestamp group the SRC and DST IP
        to find its size
Step 5: If the size is above a normal value (10 in this
        case) report the IP
Step 6: Find the MAC address from the MAC index
        using reported IP
Step 7: Output the abnormal MAC address into the
        Firewall rules CSV
```

4.4 User defined pseudo code

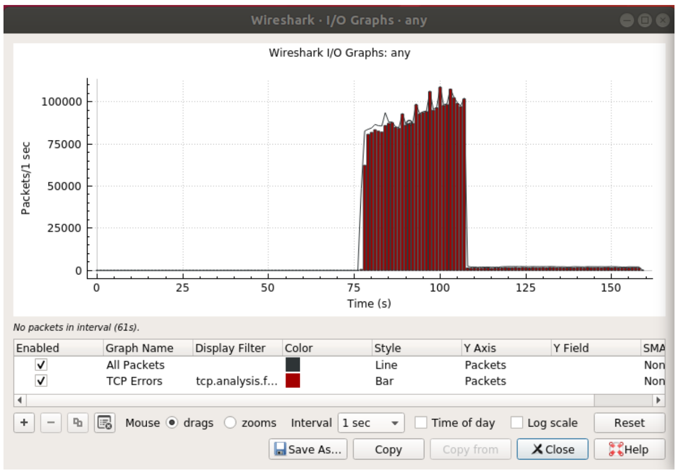
```
Input: CSV (K-Means pre-processed)
Output: CSV (updated mininet firewall rules)
Method:
Select the cluster with maximum packets
Order the packets in that cluster
Repeat:
        1.Group the source and destination IP and
        find its count
        2.Check if the count is above a normal value
```

3.If yes, add the IP in an array
Until no more timestamp
Update the values in the array into the mininet firewall CSV

5. RESULT

5. 1 I/O Graph Analysis

Wireshark IO Graphs will show you the overall traffic seen in a capture file which is usually measured in rate per second in bytes or packets (which you can always change if you prefer bits/bytes per second). In default the x-axis is the tick interval per second, and y-axis is the packets per tick (per second). It's mostly useful for troubleshooting seeing spikes and dips in your traffic, btw, to look into the traffic closer you can click on any point on the graph and it will focus on that packet and display the information in the background packet list window.



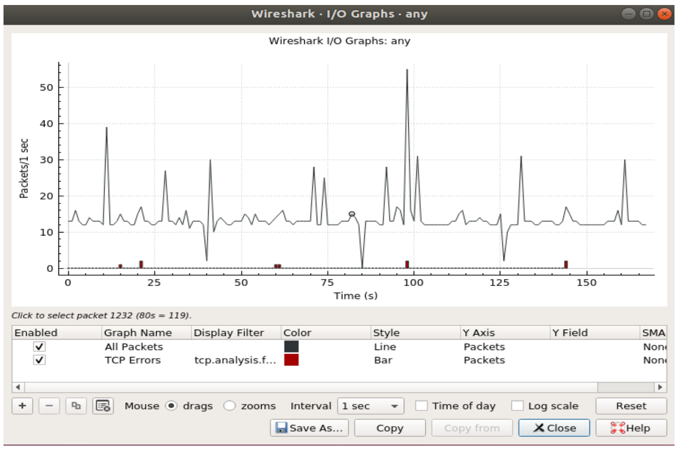
No packets in interval (61s).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA
✓	All Packets		■	Line	Packets		Non
✓	TCP Errors	tcp.analysis.f...	🟥	Bar	Packets		Non

Interval 1 sec ☐ Time of day ☐ Log scale

Fig -7: MK1 I/O Graph

The above figure shows that packet traffic of MK1 is more because there were not any mitigation services available to analyze the traffic. The large traffic also makes the recipient machine unresponsive for an indefinite time.



Click to select packet 1232 (80s = 119).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA
✓	All Packets		■	Line	Packets		Non
✓	TCP Errors	tcp.analysis.f...	🟥	Bar	Packets		Non

Interval 1 sec ☐ Time of day ☐ Log scale

Fig -8: MK2 I/O Graph

The above figure shows that packet traffic of MK2 is normal irrespective of the DoS attack initiated by the host h1.

5.2 CPU Load Analysis

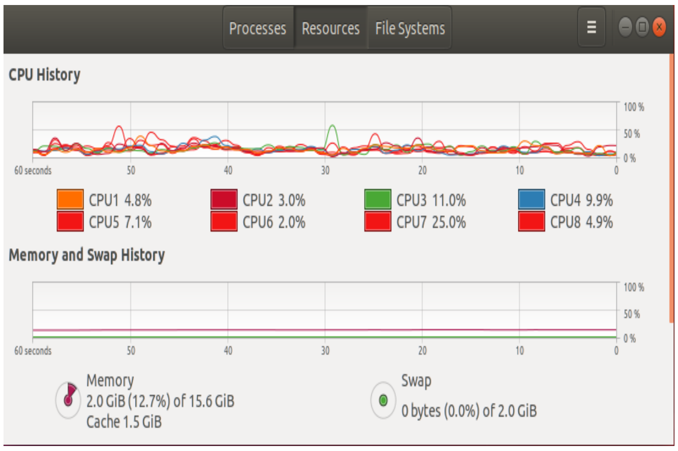


Fig -9: Normal CPU Load

The CPU load is a measure of the amount of computational work that a computer system performs. The load average represents the average system load over a period of time. It conventionally appears in the form of three numbers which represent the system load during the last one-, five-, and fifteen-minute periods.

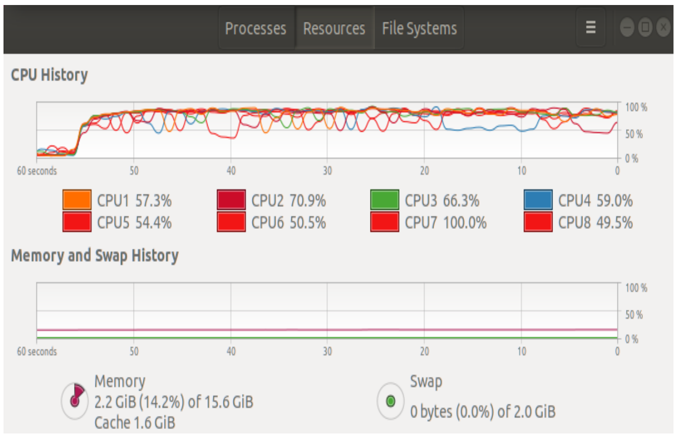


Fig -10: MK1 CPU Load during attack

The above figure shows that the CPU load is directly proportional to the packet traffic. This causes usage of multiple processors to read and ACK the upcoming traffic. This in turn causes the system to slow down and unresponsive for other legit users.

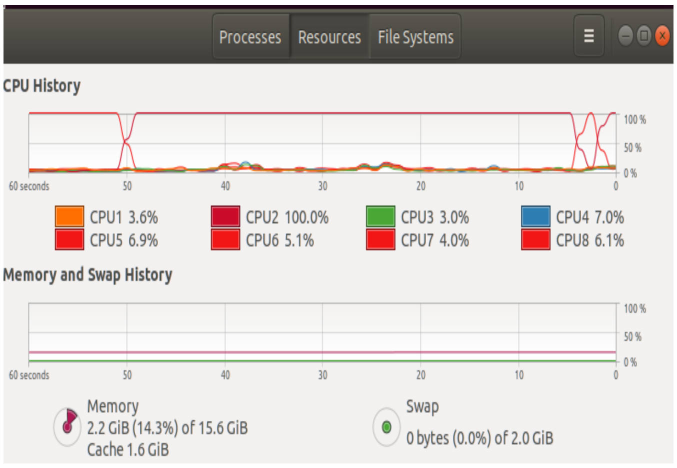
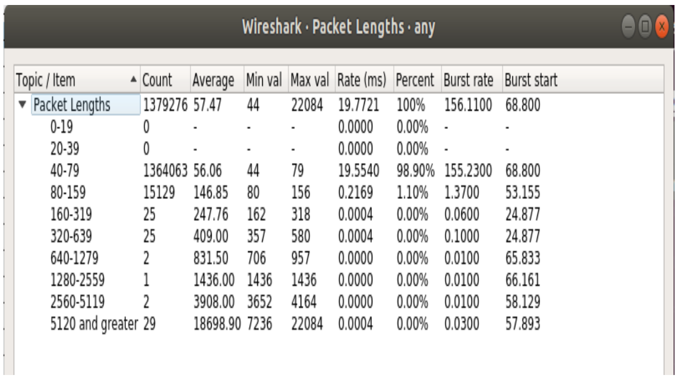


Fig -11: MK2 CPU Load during attack

The above figure shows that like the MK2's I/O Graph the CPU load is low when compared to MK1 due to the availability of mitigation services. The single core (CPU 2) is high because it is used for the attacking purpose.

5. 3 Packet Length

Network packet is a formatted unit of data carried by a packet-switched network. A packet consists of control information and user data, which is also known as the payload. Control information provides data for delivering the payload, for example: source and destination network addresses, error detection codes, and sequencing information. Typically, control information is found in packet headers and trailers. By using Wireshark, we can find the packet size of the attacking packets, in this case it is 40 – 79 bytes.



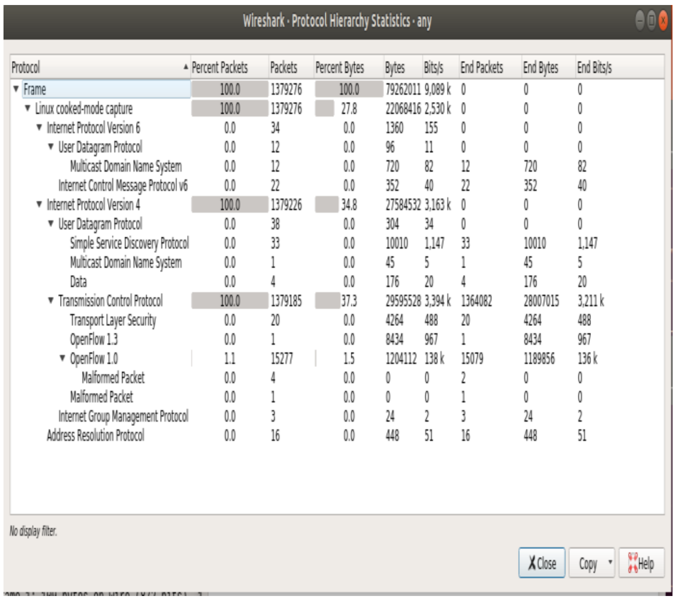
Wireshark · Packet Lengths · any

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	1379276	57.47	44	22084	19.7721	100%	156.1100	68.800
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1364063	56.06	44	79	19.5540	98.90%	155.2300	68.800
80-159	15129	146.85	80	156	0.2169	1.10%	1.3700	53.155
160-319	25	247.76	162	318	0.0004	0.00%	0.0600	24.877
320-639	25	409.00	357	580	0.0004	0.00%	0.1000	24.877
640-1279	2	831.50	706	957	0.0000	0.00%	0.0100	65.833
1280-2559	1	1436.00	1436	1436	0.0000	0.00%	0.0100	66.161
2560-5119	2	3908.00	3652	4164	0.0000	0.00%	0.0100	58.129
5120 and greater	29	18698.90	7236	22084	0.0004	0.00%	0.0300	57.893

Fig -12: Packet Length

5.4 Protocol Hierarchy

The communication between the computers in the Internet is defined by different protocols. The protocols TCP and IP build the basis of the communication in the Internet. The combination of the TCP and the IP protocol is known as TCP/IP protocol that represents the standard system used in most large networks. In this case all the attacking packets follow TCP protocol.



Wireshark · Protocol Hierarchy Statistics · any

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
▼ Frame	100.0	1379276	100.0	79262011	9,089 k	0	0	0
▼ Linux cooked-mode capture	100.0	1379276	27.8	22068416	2,530 k	0	0	0
▼ Internet Protocol Version 6	0.0	34	0.0	1360	155	0	0	0
▼ User Datagram Protocol	0.0	12	0.0	96	11	0	0	0
Multicast Domain Name System	0.0	12	0.0	720	82	12	720	82
Internet Control Message Protocol v6	0.0	22	0.0	352	40	22	352	40
▼ Internet Protocol Version 4	100.0	1379226	34.8	27584532	3,163 k	0	0	0
▼ User Datagram Protocol	0.0	38	0.0	304	34	0	0	0
Simple Service Discovery Protocol	0.0	33	0.0	10010	1,147	33	10010	1,147
Multicast Domain Name System	0.0	1	0.0	45	5	1	45	5
Data	0.0	4	0.0	176	20	4	176	20
▼ Transmission Control Protocol	100.0	1379185	37.3	29595528	3,394 k	1364082	28007015	3,211 k
Transport Layer Security	0.0	20	0.0	4264	488	20	4264	488
OpenFlow 1.3	0.0	1	0.0	8434	967	1	8434	967
▼ OpenFlow 1.0	1.1	15277	1.5	1204112	138 k	15079	1189856	136 k
Malformed Packet	0.0	4	0.0	0	0	2	0	0
Malformed Packet	0.0	1	0.0	0	0	1	0	0
Internet Group Management Protocol	0.0	3	0.0	24	2	3	24	2
Address Resolution Protocol	0.0	16	0.0	448	51	16	448	51

No display filter.

Fig -13: Protocol Hierarchy

6. CONCLUSIONS

SDN is built on logically centralized network topologies, which enable intelligent control and management of network resources. DoS is an attack meant to shut down a machine or network, making it inaccessible to its intended users. DoS attacks accomplish this by flooding the target with traffic or sending it information that triggers a crash. Thus, DoS attack can prevent the users from accessing various network resources of SDN. By using this project, we introduced a system that can identify DoS attack and mitigate its effects by dynamic packet grouping and analysis. The effectiveness of this project can be analyzed by using various method listed in the result. This project can be improved in future by extending its mitigation services for other malicious network attacks such as Spoof, Distributed Denial of Service, Man in the Middle.

REFERENCES

[1] Aleroud, A., & Alsmadi, I. (2016). Identifying dos attacks on software defined networks: A relation context approach.

[2] Braga, R., Mota, E., & Passito, A. (2010). Lightweight DDoS flooding attack detection using NOX/OpenFlow.

[3] Celyn Birkinshaw, Elpida Rouka, Vassilios G. Vassilakis, implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks (2019).

[4] Ha, T., Kim, S., An, N., Narantuya, J., Jeong, C., Kim, J., & Lim, H. (2016). Suspicious traffic sampling for intrusion detection in software-defined networks.

[5] Joldzic, O., Djuric, Z., & Vuletic, P. (2016). A transparent and scalable anomaly-based dos detection method.

[6] Luiz Fernando Carvalhoa, Taufik Abrão b, Leonardo de Souza Mendes c , Mario Lemes Proença Jr, An ecosystem for anomaly detection and mitigation in software-defined networking (2018).

[7] Mousavi, S. M., & St-Hilaire, M. (2015). Early detection of DDoS attacks against SDN controllers.

[8] Rishikesh Sahay a,b , Gregory Blanc a,b , Zonghua Zhang b,c, Hervé Debar, ArOMA: An SDN based autonomic DDoS mitigation framework (2017).

[9] Thuy Vinh Tran, Heejune Ahn, Challenges of and solution to the control load of stateful firewall in software defined networks (2017).

[10] Xiao, P., Qu, W., Qi, H., & Li, Z. (2015). Detecting DDoS attacks against data center with correlation analysis.

[11] Mininet: http://mininet.org/walkthrough/

[12] PyShark: https://github.com/KimiNewt/pyshark

[13] Hping3: https://linux.die.net/man/8/hping3

[14] Pandas: https://pandas.pydata.org/docs/

[15] Scikit-learn: https://scikit-learn.org/stable/modules/clustering.html#clustering

[16] J Ramprasath, M Aswin Yegappan, Dinesh Ravi, N Balakrishnan, S Kaarthi, Assigning Static Ip Using DHCP In Accordance With MAC, International Journal for Trends in Engineering & Technology, Volume 20, Issue 1, (Feb 2017).

BIOGRAPHIES

Ponmanikandan V, perusing bachelor's degree in Dr. Mahalingam College of Engineering and Technology, India. My areas of specialization are Software Networking and .NET Programming.

Ramprasath J, working as an Assistant Professor in the Department of Information Technology at Dr. Mahalingam College of Engineering and Technology, India He pursues Ph.D. in Information and Communication Engineering from Anna University, Chennai, India. He has presented 6 papers in conferences and he published 3 papers in the international journal.

Rakunanthan K S, perusing bachelor's degree in Dr. Mahalingam College of Engineering and Technology, India. His areas of specialization are Cloud Database and Software Networking.

Santhosh Kumar M, perusing bachelor's degree in Dr. Mahalingam College of Engineering and Technology, India. His areas of specialization are Database Management, Web Development.