Journal of Cyber Security and Risk Auditing

https://www.jcsra.thestap.com/

JOURNAL OF CYBER SECURITY AND RISK AUDITING
Smart Technologies Academic Press
STAP

Enhancing DDoS Attack Detection and Mitigation in SDN Using Advanced Machine Learning Techniques

Nathaniel Frederick[1], Aitizaz Ali[1]

[1] *Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU), Malaysia*

ARTICLE INFO

Article History

Received: 01-11-2024
Revised: 30-11-2024
Accepted: 05-12-2024
Published: 06-12-2024

Academic Editor:

Prof. Youakim Badr

Vol.2024, No.1

DOI:
https://doi.org/10.63180/jcsra.thestap.2024.1.4

ABSTRACT

The introduction of Software-Defined Networking (SDN) as a new infrastructure has demonstrated significant advantages over traditional networks in terms of scalability, flexibility, and security. However, SDN networks are also more susceptible to Distributed Denial of Service (DDoS) attacks, which can lead to a loss of system availability. Therefore, in this research, a machine learning-based model is developed to detect and prevent DDoS attacks in SDN environments. Our approach extends traditional flow-based features by incorporating additional parameters such as average flow packet size and recent flow history, among others, to enhance detection accuracy. Six machine learning models—Logistic Regression (LR), Naïve Bayes (NB), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Tree (DT), and Random Forest (RF)—were evaluated using the CIC-DDoS2019 dataset. The results show that the Random Forest model achieved the highest detection rate with the lowest false positive rate compared to the other models, while also having minimal impact on normal traffic. The proposed system functions as an Intrusion Prevention System (IPS) by sampling flow parameters from Open Flow switches at intervals. Upon detecting an attack, the system applies traffic policing measures. Experimental results confirm that the Random Forest model achieved a high F1-score of 99.87%, making it a promising candidate for real-time DDoS detection and mitigation in SDN networks.

Keywords: Software-Defined Networking (SDN), DDoS, Intrusion Prevention System (IPS), Random Forest, Machine Learning, Network Security.

How to cite the article
Frederick, N., & Ali, A. (2024). Enhancing DDoS Attack Detection and Mitigation in SDN Using Advanced Machine Learning Techniques. Journal of Cyber Security and Risk Auditing, 2024(1), 23–37. https://doi.org/10.63180/jcsra.thestap.2024.1.4

*Corresponding author. Email: aitizaz.ali@apu.edu.my

1. Introduction

One of the most critical and rapidly growing areas of business within the field of cybersecurity is network infrastructure, due to the increasing relevance of secure digital communication and commercial transactions [1]. As these digital activities continue to expand in scale and complexity, the threat landscape has also matured and become more widespread. To maintain secure operations, it is essential to detect and contain a range of malicious activities—such as unauthorized access, confidentiality breaches, and Distributed Denial-of-Service (DDoS) attacks [2]. In this context, innovative approaches, particularly those made possible by advanced technologies such as machine learning, are becoming increasingly valuable and popular for enhancing Network Intrusion Detection Systems (NIDS). These systems play a critical role in predicting potential attacks and recommending appropriate security measures to safeguard corporate enterprises [3].

Among the various attack strategies, DDoS attacks are particularly challenging to defend against and are among the most commonly employed methods of cyberattack [4]. The difficulty in mitigating DDoS attacks stems from their distributed nature—attack traffic originates from multiple IP addresses simultaneously, making it difficult to identify and block the source. DDoS attacks can compromise a wide range of internet-connected devices and are typically targeted at web servers or critical network infrastructure [5]. A Distributed Denial of Service (DDoS) attack involves using a large number of compromised computers to launch a coordinated assault, overwhelming one or more target systems with traffic. By leveraging these unsuspecting devices—often referred to as botnets—attackers can exhaust system resources such as CPU, bandwidth, and memory, eventually leading to service disruption and denial of access for legitimate users. The primary goal of such an attack is to exhaust the target's capabilities, rendering it unable to respond to legitimate requests and thereby disrupting service availability [6],[7].

2. Literature Review

Priya et al. [6] employed machine learning techniques to create a DDoS detector that can function on any standard hardware. The precision of the outcomes is 98.5 percent. The researchers employed two variables, delta time and packet size, to differentiate DDoS packets from ordinary packets using three classification algorithms: K-Nearest Neighbors, Random Forest, and Na¨ıve Bayes. The detector has the ability to detect several types of DDoS assaults, such as ICMP floods, TCP floods, and UDP floods, among others. Certain systems may necessitate a substantial number of attributes to accurately identify Dis- tributed Denial of Service (DDoS) attacks in older systems, whereas other systems may re- quire a multitude of features to effectively detect DDoS attacks in older systems. Certain sys- tems may exclusively operate with particular protocols. Their proposed method, however, addresses these issues by detecting any sort of DDoS attack without the need for a specialised protocol that relies on fewer distinguishing features.

Doshi et al. [7] were motivated to develop novel methods for automatically identifying mali- cious network traffic targeting consumer Internet of Things (IoT) devices. The study demon- strated that utilising IoT-specific network behaviours to inform the selection of features can lead to accurate detection of DDoS attacks in IoT network traffic. This can be achieved by various machine learning techniques, such as neural networks. These findings indicate that by using inexpensive machine learning methods and flow-based, protocol-independent traffic data, home gateway routers or other network middleboxes can automatically identify the lo- cal sources of DDoS attacks from IoT devices.
Aysa et al. [8] utilised pandemic modelling tools to analyse IoT networks consisting of Wire- less Sensor Networks (WSNs). They provided a structure for recognising and diagnosing ab- normal defensive actions. There are substantial problems based on the influence of IoT- specific factors such inadequate processing capacity, power restrictions, and node density on the creation of a botnet.

The slang of the wrapper methods terms builds on attributes provided by GA and classifica- tion approaches of ANN and MLP are employed to enhance the DDoS detection using the attributes from ANN. The result shows the proposed strategy is very effective along a dual purpose: the ability to prevent False Negatives and identify a DDoS. A 'machine learning based DDoS detection system' of Sudugala, et al. [11] is proposed, which has a low false positive rate and a high accuracy rate based on testing data. The scheme includes signature that were issued through traffic off the network trace while network traffic flow was conduct- ed in the prior stage in order to draw certain conclusions. With four different machine learn- ing algorithms such as support vector machines utilized with the same dataset as benchmark. The articles arrived at the highest possible precision and additionally presented a comparison of their findings with the results of the other artificial intelligence applications. Zecheng et al.

[12] Suggested the use of source-oriented machine learning algorithms in which they develop a cloud-based detection system to tackle attacks from undesirable sources. The VM-level data coming from both the virtual machines and the hypervisor switch this disconnection from the network extends to the outside world.

3. Research Methodology

3.1 System Development Methodology

The system was implemented using two methodologies namely; the Rapid Application Development (RAD) and the Spiral Methodology. RAD allowed for fast cycles of design with the Spiral Methodology setting up risk assessments and refining the design through prototyping with subsequent cycles as shown in Figure 1. Data collection and preprocessing is the first great step in the journey of a successful data science project. Information for this study was obtained from the CIC-DDoS2019 dataset due to the sample's balance of normal and attack traffic.

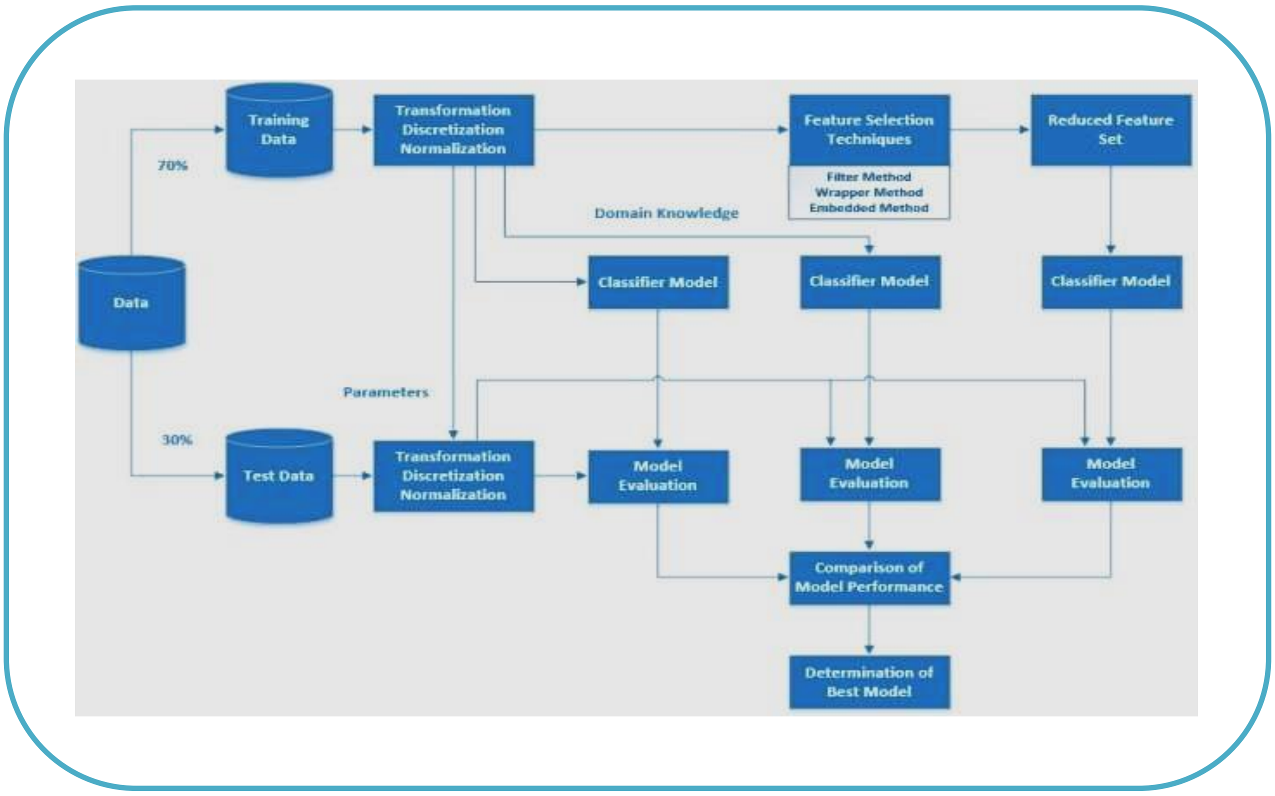


Figure 1. Model Architecture for DDoS attack detection

Key preprocessing steps included: (1) Data Cleaning: Excluding missing or unnecessary pieces of information. (2) Normalization: FIRS digitizes values to standardize, and guarantee that its values are in line with the international standards as currently practiced. (3) Feature Engineering: Extending the list of metrics that can be used in model training, such as the average size of the packet, the history of the flow, etc. In addition, three machine learning algorithms were evaluated for DDoS detection are K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest (RF) as shown in Figure 2.

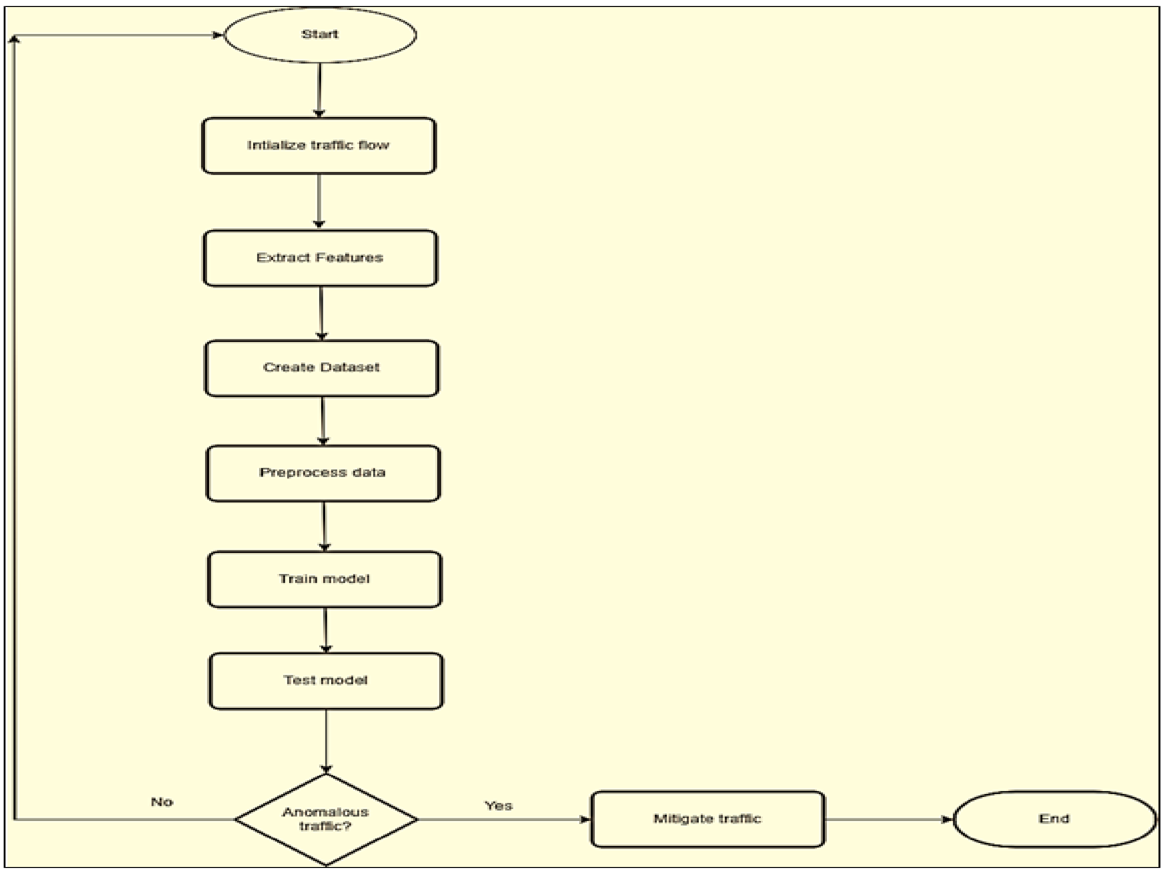


Figure 2. Machine Learning Module Methodology

The implemented system consists of two core modules: a detection module and a mitigation module; the detection and the mitigation of DDoS attacks in the context of SDN environment. In reality, to train the detection model, a dataset is created through emulating network traffic in the specifically designed SDN topology in Mininet environment. The traffic of this simulation includes regular traffic, and other instances like DDoS attacks. As an SDN controller, the RYU controller collects relevant traffic features from the network packets such as source IP, destination IP, ports, and timestamps.

Once the statistical reports from the flow tables have been received, the Attack Detection Module uses the formulas highlighted in the Statistical Method section of this document. It then analyzes the statistical results against an input dataset using many algorithms like KNN, RF, or SVM to classify whether the network traffic flow has a threat or not or Support Vector Machines (SVM), to analyze the statistical results against an input dataset, determining whether the network traffic flow poses a threat. When the flow of traffic is categorized as normal, the packets are just allowed to pass through the system without reroutes or checks. However, if the traffic flow is considered to be a threat the specifics of this flow are transferred to the Mitigation Module.

For instance, if this hazardous traffic flow is going to switch ID 1 on port 1 the Mitigation Module will immediately send the controller a request to tell switch ID 1 to block the traffic flows emanating from port 1. Same as that, the Mitigation Module will produce real-time alerts to the system administrator for constant monitoring. Upon completion of attack flow, commonly around 60 seconds, the system will re-connection to port 1 where the process will continue. The system's global packet processing workflow is provided in the form of a diagram in Figure 3 to capture an overall perspective of the detection and prevention process.

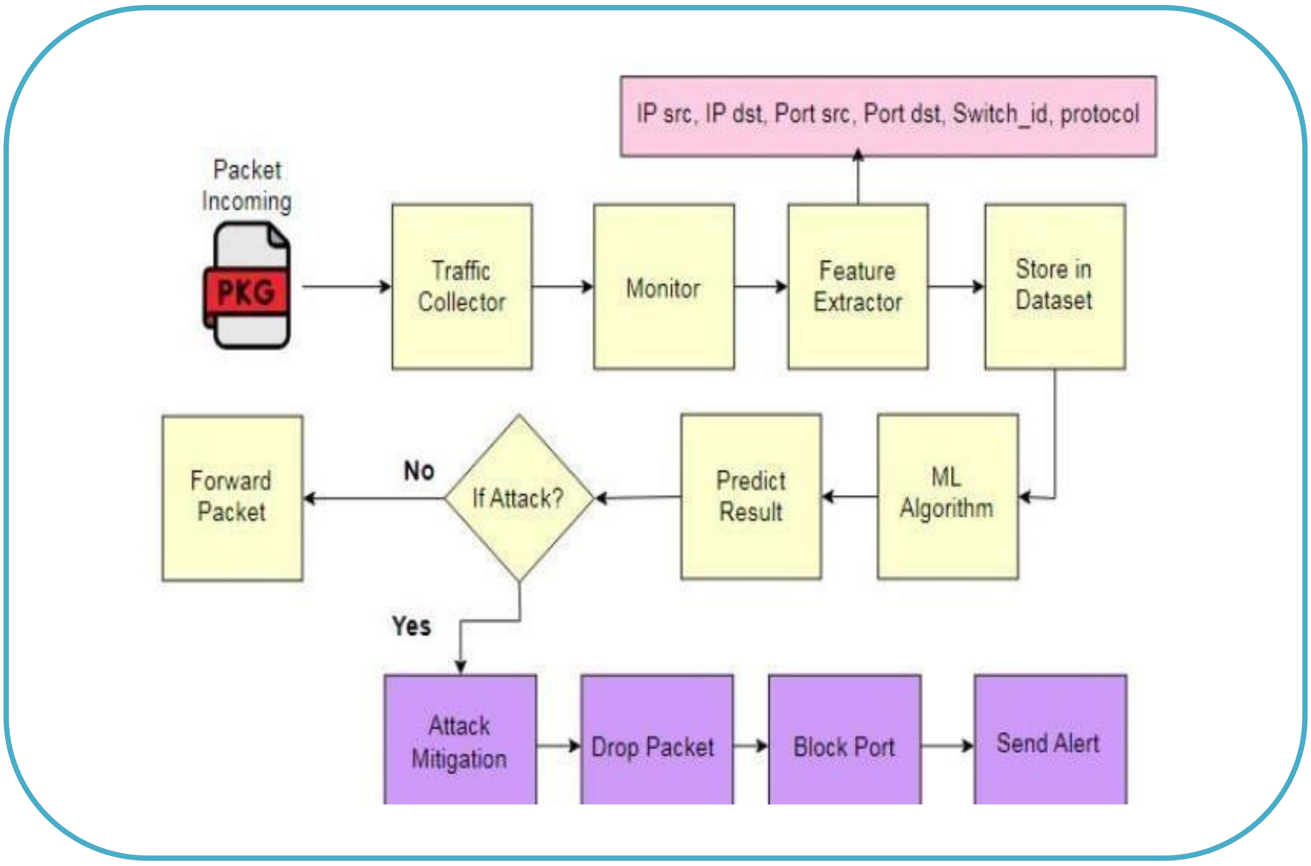


Figure 3. The packet processing workflow of the system

To enhance this system for the detection and prevention of DDOS attacks, we recommend a proper design of an SDN Network system as shown below in Figure 10. By the standard of a SDN network infrastructure, the network infrastructure is comprehensively divided into three layers. Control Layer: This layer is regarded as the key control layer of the whole system and it consists of the Controller device. The proposed attack prevention modules labelled as uitSDNDDoSD will be embedded directly into the Controller. This module act as a reservoir that is used to aggregate all or select traffic flows from the OpenFlow protocol for statistics computation. When the module recognizes an attack on the system it informs the Controller thus forcing the OF Switch to drop the packets. To this end, this research employs the RYU software in the emulation of a Controller device in the system.

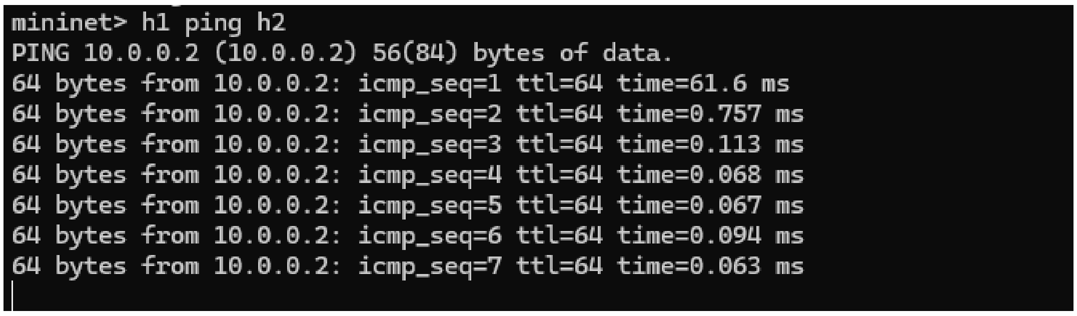
4. Analysis and Results

From the experimental analysis in Table 1, it is very clear that the proposed Random Forest (RF) model had better performance than K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) in terms of accuracy, precision, recall, and F1-score. The RF model was the most accurate model to detect DDoS attacks with an accuracy of 99.92%, a precise model of 99.64% and a perfect F1 score of 100%. Further, it took the shortest time to train, only about 40 seconds, which is still much shorter than the 3600 seconds needed for the training of the SVM model. Furthermore, the RF model involved less computational power, the CPU usage being 50%, and RAM usage being 70% making it ideal for surveillance in live networks.

Since network systems demand rapid response and have low computational complexity in terms of system demands, the RF algorithm becomes the most fit for DDoS attack detection. Its high accuracy is therefore enhanced by the low time it takes in training coupled with the optimized resource utilization, thus it is suitable for use in environments where early identification and response to threats are paramount.

Table 1. Analysis of the results of machine learning algorithms

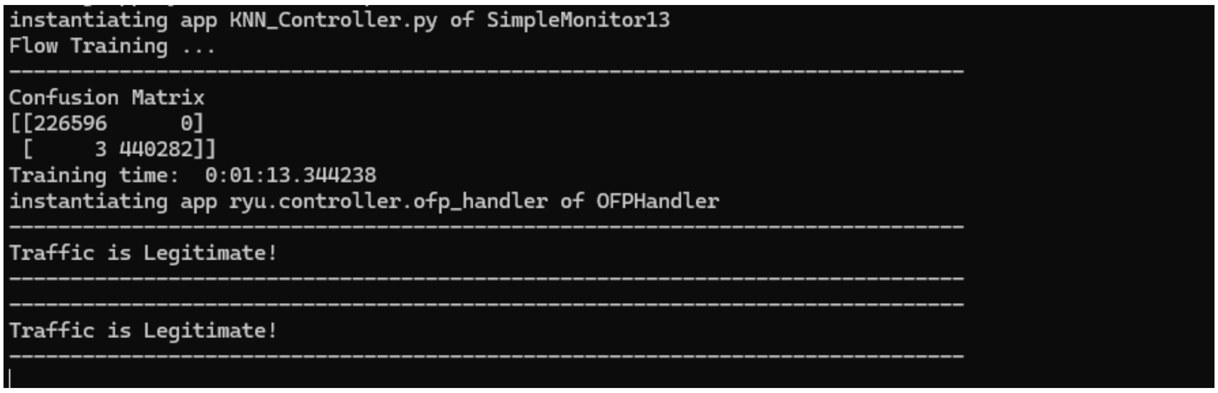
Algorithm (%)	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**Model Training Time (s)**	**CPU Usage (%)**	**RAM Usage (%)**
KNN	97.65	98.04	97.84	78	51	80
RF	99.92	99.64	100	40	50	70
SVM	87.30	88.23	87.77	3600	70	90



```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=61.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.757 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.094 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.063 ms
```

Figure 4. Successfully pinging between h1 and h2 in the Mininet network

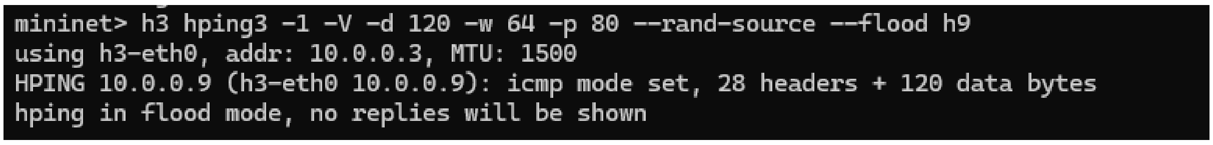
In this case, I am using the Mininet CLI to test the link between two hosts, h1 and h2 through ICMP ping request invention. With the ping test it is estab- lished that h1 is indeed able to send messages to h2 and get corresponding re- plies with different response time. The first ping has a very high packet re- sponse time (180 ms, which may be caused by the network initialising) but subsequent pings have a low standard range of .057 – .783 ms. This shows normal network operation where the hosts are communicating with one an- other with negligible delays.



```
instantiating app KNN_Controller.py of SimpleMonitor13
Flow Training ...
------------------------------------------------------------------------------
Confusion Matrix
[[226596      0]
 [     3 440282]]
Training time:  0:01:13.344238
instantiating app ryu.controller.ofp_handler of OFPHandler
------------------------------------------------------------------------------
Traffic is Legitimate!
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Traffic is Legitimate!
------------------------------------------------------------------------------
```

Figure 5. System confirms that the monitored network traffic is legitimate

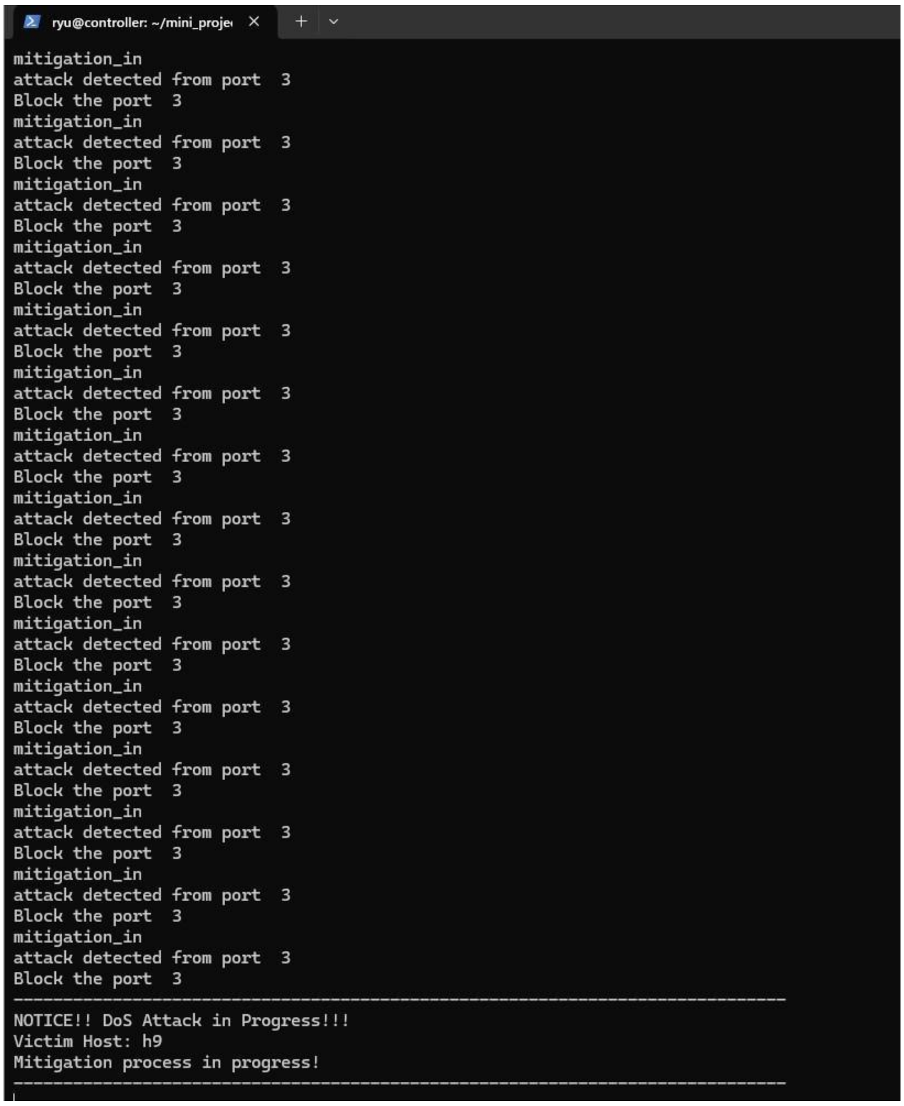
The system has decided that the network traffic is legitimate, this is that no DDoS attack has been established. When the current traffic was evaluated by the machine learning model incorporated with the Ryu controller it was concluded that the traffic did not pose to be malicious and all the communication was normal. This message is another message that gets through in the course of the traffic monitoring in the DDoS detection system showing that normal traffic, or benign traffic, continues uninterrupted.



```
mininet> h3 hping3 -1 -V -d 120 -w 64 -p 80 --rand-source --flood h9
using h3-eth0, addr: 10.0.0.3, MTU: 1500
HPING 10.0.0.9 (h3-eth0 10.0.0.9): icmp mode set, 28 headers + 120 data bytes
hping in flood mode, no replies will be shown
```

Figure 6. Simulating a DDoS attack from host h3 to host h9 using hping3 in flood mode

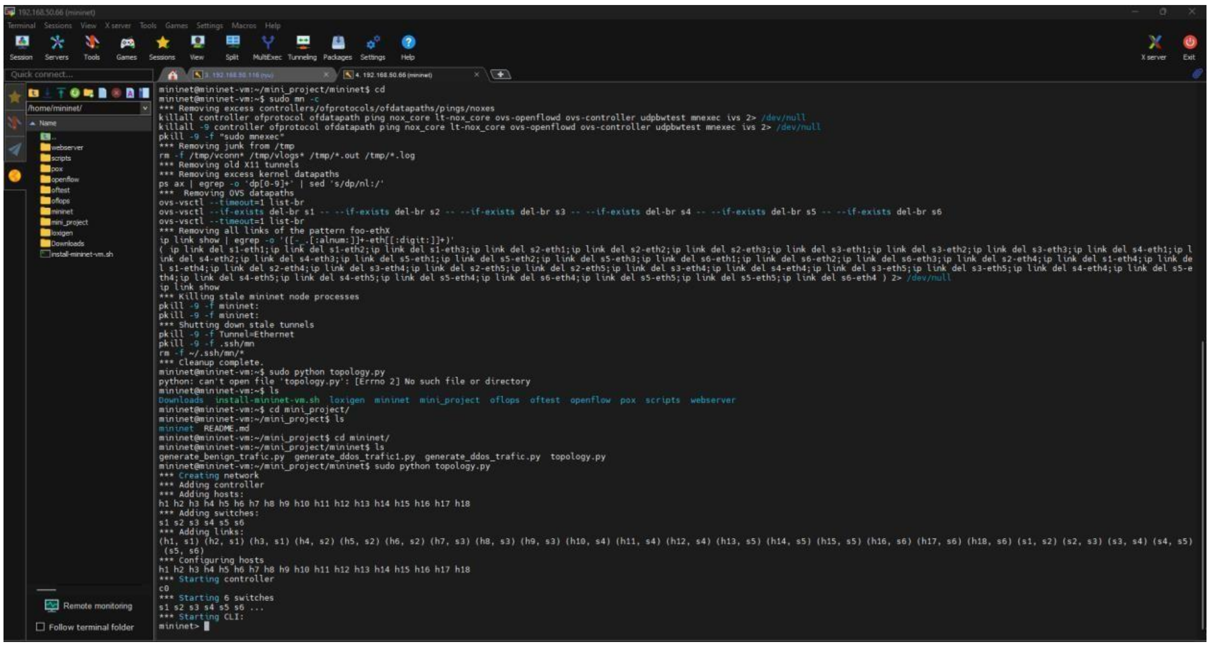
In this case, I am assuming host h3 and I mimic a DDoS against host h9 using hping3. The command is still able to launch the flood mode which transmits ICMP packets within a short period with random source IP address to the target. Using the --flood option makes a very fast transmission of packets and the --rand-source option gives source IP addresses of packets randomly like a normal distributed DDOS attack. This test falls under the DDoS detection project to experiment with different reactions of the system under attack by malicious traffic intended on the network.



```
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
mitigation_in
attack detected from port  3
Block the port  3
--------------------------------------------------------------------------------
NOTICE!! DoS Attack in Progress!!!
Victim Host: h9
Mitigation process in progress!
--------------------------------------------------------------------------------
```

Figure 7. Identified an attack targeting host h9 from port 3

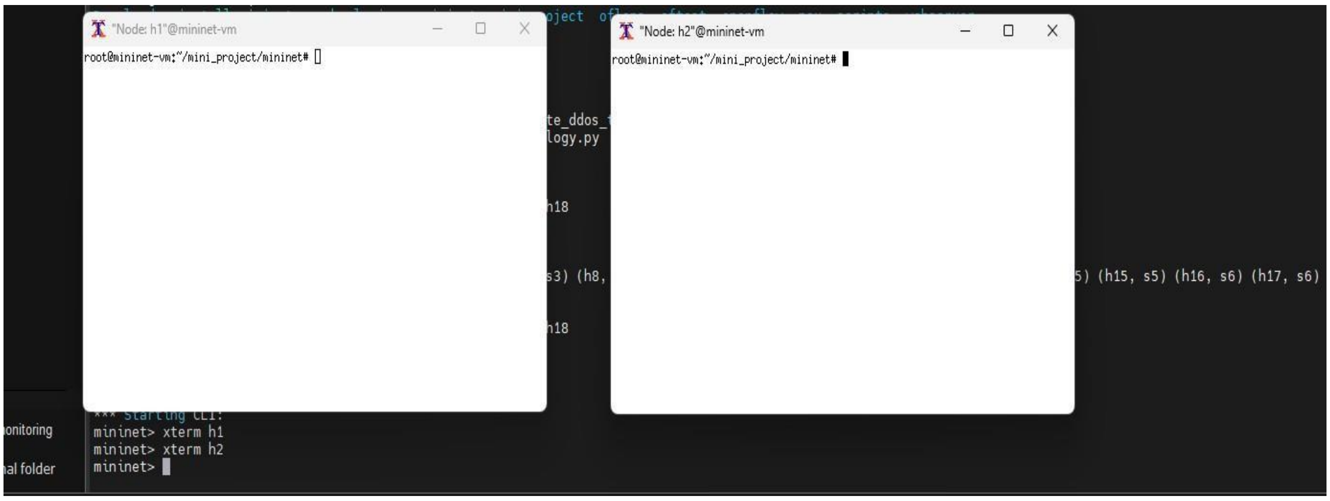
In Figure 8 a DDoS attack has been detected from port 3 of the system on host h9. Several alerts prove the identification of the attack and the beginning of countermeasures, which is to philtre out the attack traffic on port 3 to avoid hitting the target host constantly with the malicious traffic. This continuous detection and mitigation cycle shows that the system can dynamically detect and counter DDoS attacks in general, keep on isolating the attack traffic while allowing legitimate traffic to pass through the system.



```
mininet@mininet-vm:~/mini_project/mininet$ cd
mininet@mininet-vm:~$ sudo mn -c
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-controller udpbwtest mnexec ivs 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --if-exists del-br s1 -- --if-exists del-br s2 -- --if-exists del-br s3 -- --if-exists del-br s4 -- --if-exists del-br s5 -- --if-exists del-br s6
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
( ip link del s1-eth1;ip link del s1-eth2;ip link del s1-eth3;ip link del s2-eth1;ip link del s2-eth2;ip link del s2-eth3;ip link del s3-eth1;ip link del s3-eth2;ip link del s3-eth3;ip link del s4-eth1;ip l
ink del s4-eth2;ip link del s4-eth3;ip link del s5-eth1;ip link del s5-eth2;ip link del s5-eth3;ip link del s6-eth1;ip link del s6-eth2;ip link del s6-eth3;ip link del s2-eth4;ip link del s1-eth4;ip link de
l s1-eth4;ip link del s2-eth4;ip link del s3-eth4;ip link del s2-eth5;ip link del s2-eth5;ip link del s3-eth4;ip link del s4-eth4;ip link del s3-eth5;ip link del s3-eth5;ip link del s4-eth4;ip link del s5-e
th4;ip link del s4-eth5;ip link del s4-eth5;ip link del s5-eth4;ip link del s6-eth4;ip link del s5-eth5;ip link del s5-eth5;ip link del s6-eth4 ) 2> /dev/null
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
mininet@mininet-vm:~$ sudo python topology.py
python: can't open file 'topology.py': [Errno 2] No such file or directory
mininet@mininet-vm:~$ ls
Downloads  install-mininet-vm.sh  loxigen  mininet  mini_project  oflops  oftest  openflow  pox  scripts  webserver
mininet@mininet-vm:~$ cd mini_project/
mininet@mininet-vm:~/mini_project$ ls
mininet  README.md
mininet@mininet-vm:~/mini_project$ cd mininet/
mininet@mininet-vm:~/mini_project/mininet$ ls
generate_benign_trafic.py  generate_ddos_trafic1.py  generate_ddos_trafic.py  topology.py
mininet@mininet-vm:~/mini_project/mininet$ sudo python topology.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (h6, s2) (h7, s3) (h8, s3) (h9, s3) (h10, s4) (h11, s4) (h12, s4) (h13, s5) (h14, s5) (h15, s5) (h16, s6) (h17, s6) (h18, s6) (s1, s2) (s2, s3) (s3, s4) (s4, s5)
 (s5, s6)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...
*** Starting CLI:
mininet>
```

Figure 8. MobaXterm to launch a custom Mininet network topology

In this case, I am getting connected to the Mininet VM through an SSH session using MobaXterm. Once I connect to the server and start working as root, I cd into the directories for the mini_project then into the mininet where there are Python scripts for creating both normal traffic and DDoS traffic and the network topology file (topology.py). Approximately six switches and eighteen Hosts are connected to a remote controller C0, all connected through the topology.py script that I execute. After the host and switches are configured the Mininet command-line interface provides an interactive shell for the network's interaction for the traffic stimulation and DDoS setup test.



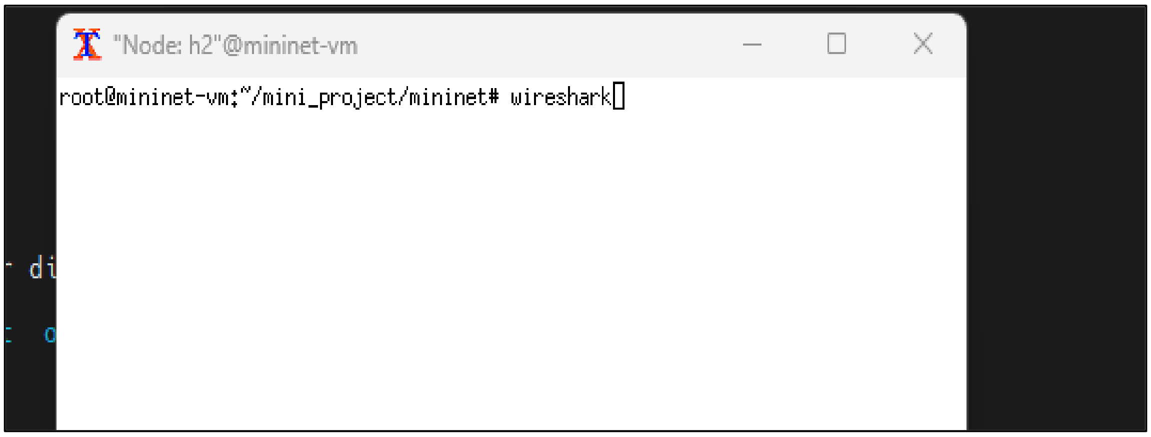
```
"Node: h1"@mininet-vm
root@mininet-vm:~/mini_project/mininet#
```

```
"Node: h2"@mininet-vm
root@mininet-vm:~/mini_project/mininet#
```

```
*** Starting CLI:
mininet> xterm h1
mininet> xterm h2
mininet>
```

Figure 9. Opening xterm windows for hosts h1 and h2 in Mininet

Using the Mininet CLI to start xterm for the two hosts, h1 [sudo mn --ver – xterms and h2 [sudo mn –ver –xterms. This let me directly communicate with these hosts within different terminal sessions, I can perform commands/operate traffic be- tween the hosts. These xterm windows are also used to perform different tests, for example, to generate any sort of traffic, including benign or DDoS, and to analyse the response of the network in real time.

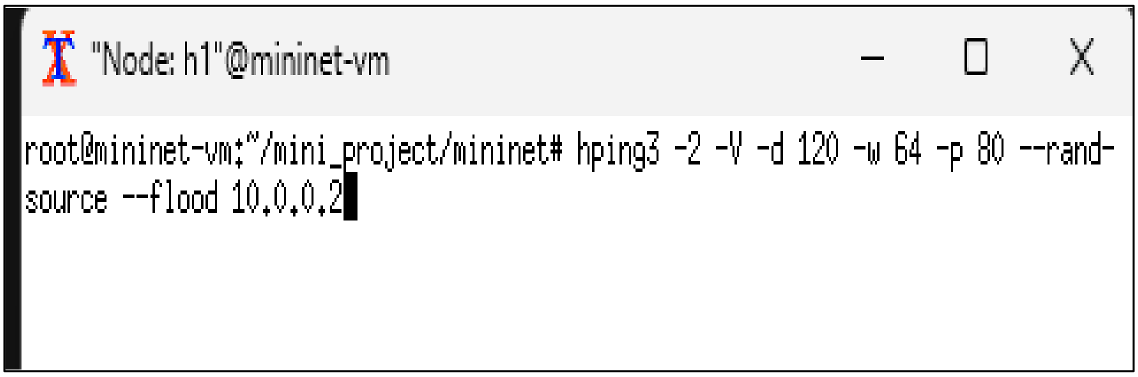


```
root@mininet-vm:~/mini_project/mininet# wireshark
```

Figure 10. Mininet VM Terminal Session for Network Traffic Analysis

The figure illustrates the terminal interface of a Mininet virtual machine when I logged into the network at a particular node (h2). The terminal command analyses shows that the user is trying to execute a programme called Wireshark in order to sniff the network. The "mini_project/mininet" directory path is displayed in current working directory in command prompt. The ses- sion is a preview of the user's environment before they go to analyse the traf- fic in the context of their DDoS detection task.

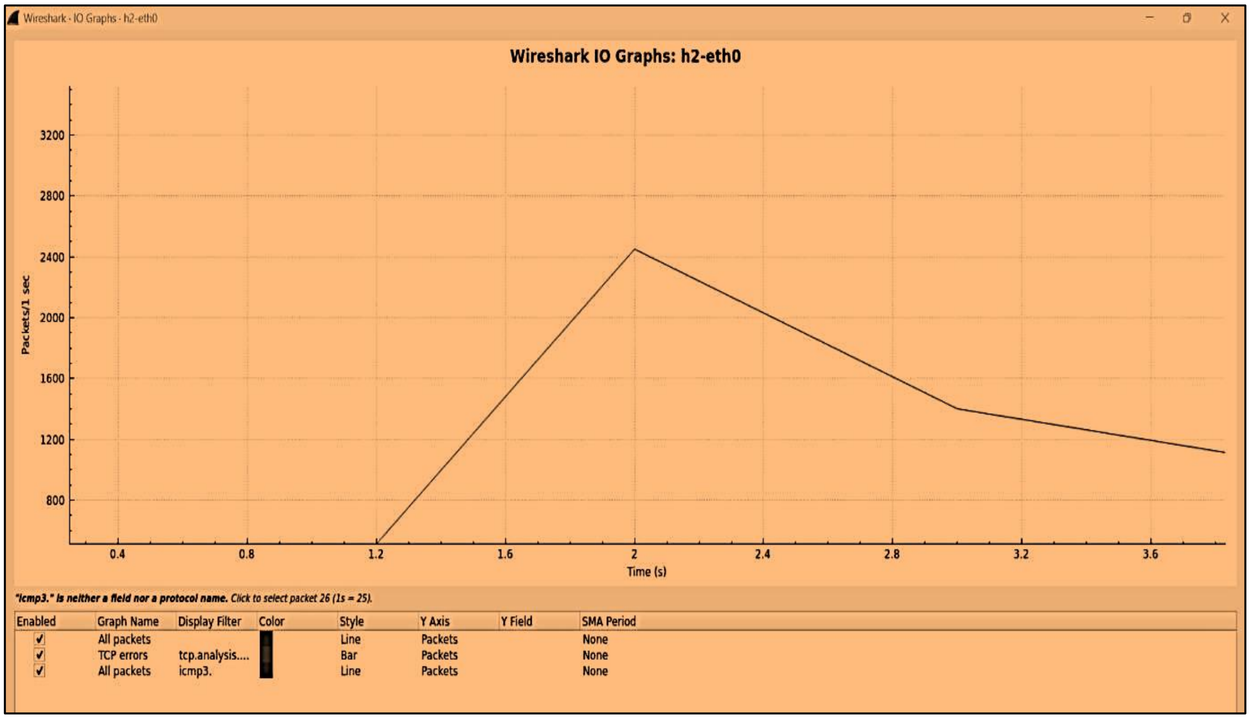
UDP attack



```
root@mininet-vm:~/mini_project/mininet# hping3 -2 -V -d 120 -w 64 -p 80 --rand-
source --flood 10.0.0.2
```

Figure 11. DDoS Attack Simulation Command Execution on Mininet VM

The figure illustrates a terminal session of the current project in a virtual ma- chine by using Mininet where I logged in to the node h1. This is the command that is typed to launch the programme with the purpose of launching a DDoS test utilizing hping3. This command defines other characteristics of the packets to be sent for testing, the data size to be transmitted, and the target port. The command is to open a flood on the target by sending packets with random source IP addresses set to 10.0.0.2, creating what is known as a Distributed Denial-of-Service (DDoS).

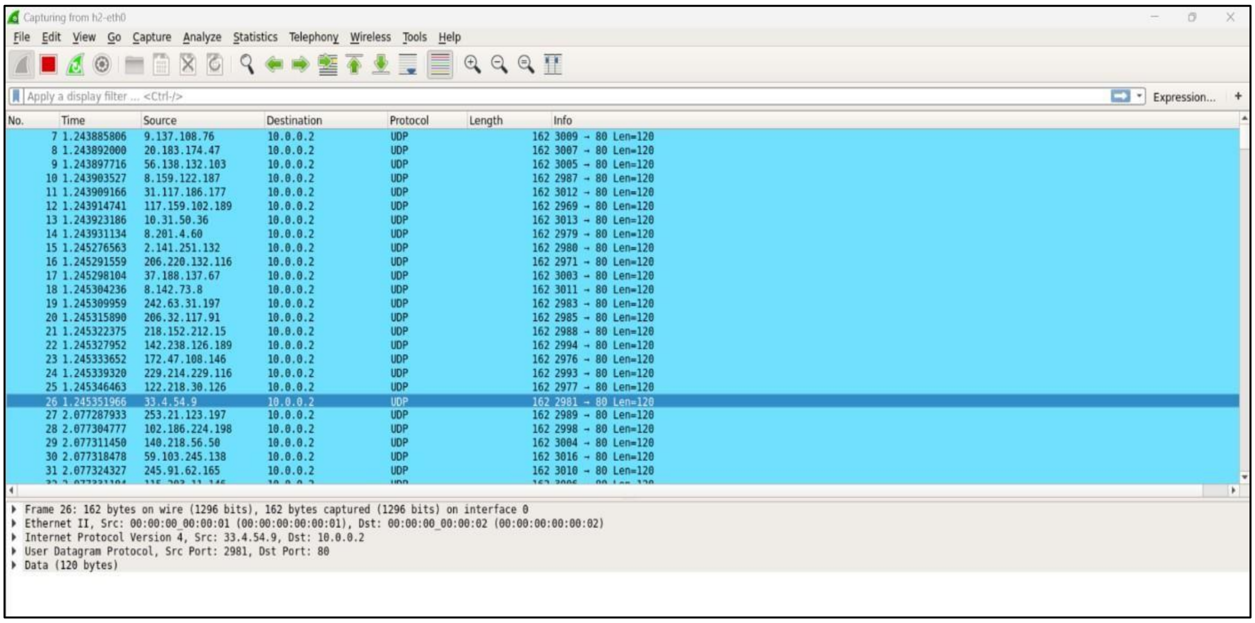


"icmp3." is neither a field nor a protocol name. Click to select packet 26 (1s = 25).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All packets		■	Line	Packets		None
✓	TCP errors	tcp.analysis....	■	Bar	Packets		None
✓	All packets	icmp3.	■	Line	Packets		None

Figure 12. Wireshark IO Graphs for Network Traffic Analysis on h2-eth0

The figure demonstrates the IO Graphs option of Wireshark and more specifically the network interface h2-eth0. The given graph shows packets which are representations of the amount of information flowing in a network, with Y axes as a number of packets/second, while the X- axis covers the time in seconds. The data in the graph shows oscillations in the activity of networks, which can most probably relate to the simulated DDoS attack.

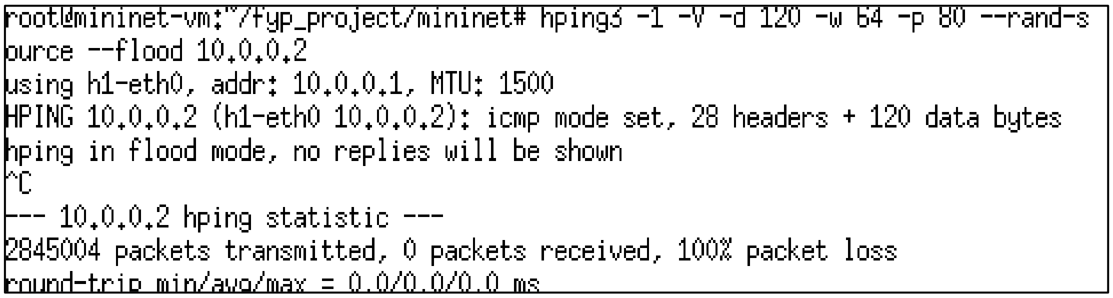

Capturing from h2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time          Source            Destination   Protocol   Length   Info
    7 1.243885806   9.137.108.76      10.0.0.2      UDP        162 3009 → 80 Len=120
    8 1.243892000   20.183.174.47     10.0.0.2      UDP        162 3007 → 80 Len=120
    9 1.243897716   56.138.132.103    10.0.0.2      UDP        162 3005 → 80 Len=120
   10 1.243903527   8.159.122.187     10.0.0.2      UDP        162 2987 → 80 Len=120
   11 1.243909166   31.117.186.177    10.0.0.2      UDP        162 3012 → 80 Len=120
   12 1.243914741   117.159.102.189   10.0.0.2      UDP        162 2969 → 80 Len=120
   13 1.243923186   10.31.50.36       10.0.0.2      UDP        162 3013 → 80 Len=120
   14 1.243931134   8.201.4.60        10.0.0.2      UDP        162 2979 → 80 Len=120
   15 1.245276563   2.141.251.132     10.0.0.2      UDP        162 2980 → 80 Len=120
   16 1.245291559   206.220.132.116   10.0.0.2      UDP        162 2971 → 80 Len=120
   17 1.245298104   37.188.137.67     10.0.0.2      UDP        162 3003 → 80 Len=120
   18 1.245304236   8.142.73.8        10.0.0.2      UDP        162 3011 → 80 Len=120
   19 1.245309959   242.63.31.197     10.0.0.2      UDP        162 2983 → 80 Len=120
   20 1.245315890   206.32.117.91     10.0.0.2      UDP        162 2985 → 80 Len=120
   21 1.245322375   218.152.212.15    10.0.0.2      UDP        162 2988 → 80 Len=120
   22 1.245327952   142.238.126.189   10.0.0.2      UDP        162 2994 → 80 Len=120
   23 1.245333652   172.47.108.146    10.0.0.2      UDP        162 2976 → 80 Len=120
   24 1.245339320   229.214.229.116   10.0.0.2      UDP        162 2993 → 80 Len=120
   25 1.245346463   122.218.30.126    10.0.0.2      UDP        162 2977 → 80 Len=120
   26 1.245351966   33.4.54.9         10.0.0.2      UDP        162 2981 → 80 Len=120
   27 2.077287933   253.21.123.197    10.0.0.2      UDP        162 2989 → 80 Len=120
   28 2.077304777   102.186.224.198   10.0.0.2      UDP        162 2998 → 80 Len=120
   29 2.077311450   140.218.56.50     10.0.0.2      UDP        162 3004 → 80 Len=120
   30 2.077318478   59.103.245.138    10.0.0.2      UDP        162 3016 → 80 Len=120
   31 2.077324327   245.91.62.165     10.0.0.2      UDP        162 3010 → 80 Len=120

▸ Frame 26: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▸ Internet Protocol Version 4, Src: 33.4.54.9, Dst: 10.0.0.2
▸ User Datagram Protocol, Src Port: 2981, Dst Port: 80
▸ Data (120 bytes)
```

Figure 14. Wireshark Packet Capture Log during DDoS Attack Simulation

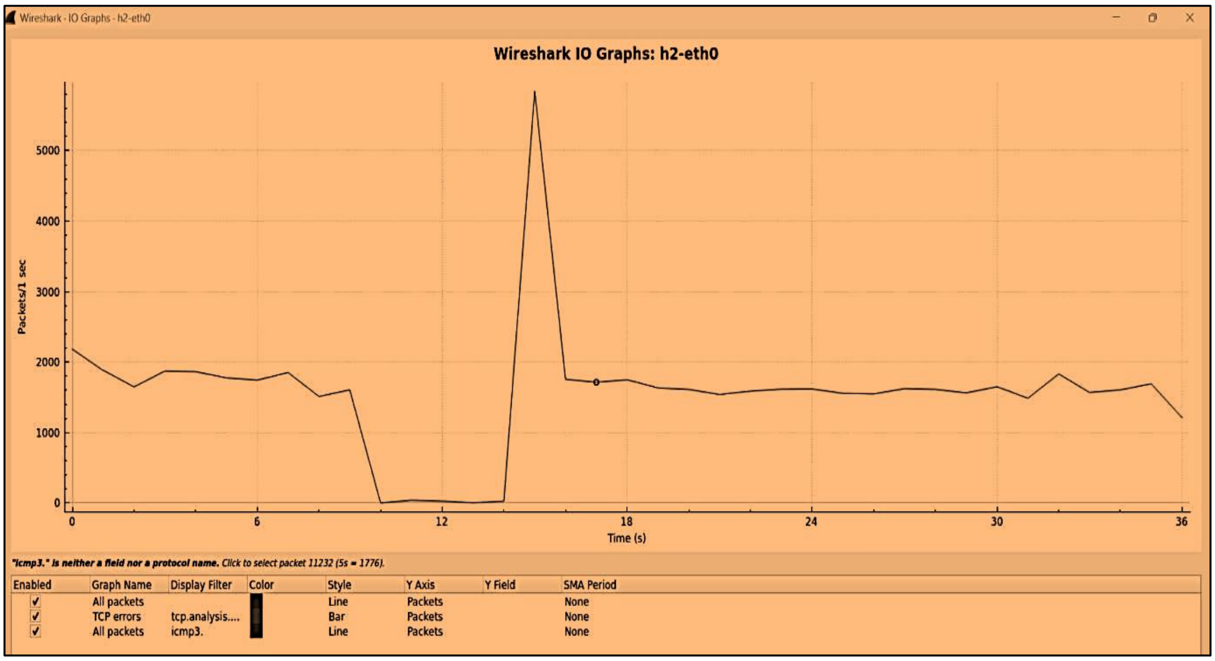
The figure shows the Wireshark interface which philters the captured network packets from the h2-eth0 interface during a simulated DDoS attack. In the packet capture log several UDP packets were recorded, and the information in the table contains information on the source and destination IP addresses, ports, and lengths. Every submission also contains useful data such as the used protocol (UDP) and packets per packet size. The data captures illustrate the large amount of packets in the network, which is typical for the artificially induced DDoS attack scenario.



```
root@mininet-vm:~/fyp_project/mininet# hping3 -1 -V -d 120 -w 64 -p 80 --rand-s
ource --flood 10.0.0.2
using h1-eth0, addr: 10.0.0.1, MTU: 1500
HPING 10.0.0.2 (h1-eth0 10.0.0.2): icmp mode set, 28 headers + 120 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.2 hping statistic ---
2845004 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 14. DDoS Flood Attack Simulation Using hping3

The one Figure shows a terminal on a Mininet Virtual Machine where I am typing a command using hping3 to mimic flood attacks on the network. The command sends packets of a particular size, using ICMP protocol and a random source IP address. From the terminal output, it can be observed that a total of 2,845,004 packets were sent out but no packets where received hence this if a 100% packet drop rate

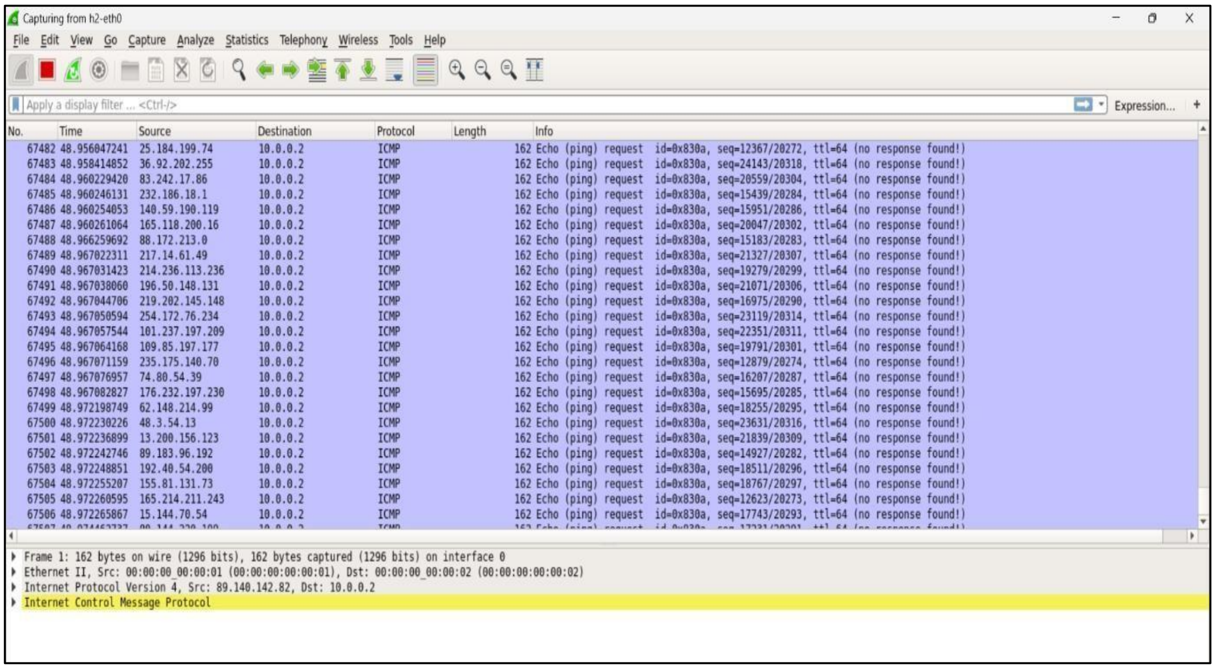


"icmp3." is neither a field nor a protocol name. Click to select packet 11232 (5s = 1776).

Enabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All packets		■	Line	Packets		None
✓	TCP errors	tcp.analysis....	■	Bar	Packets		None
✓	All packets	icmp3.	■	Line	Packets		None

Figure 15. Wireshark IO Graphs during ICMP Attack Simulation on h2-eth0

The figure shows IO Graphs feature of Wireshark – the first option presents the traffic from interface h2-eth0 during the DDoS attack simulation. When analyzing the characteristics of the packets relativize to the time, the number of packets per second is shown, which has concentrated the sharp peak in the load at the moment of flooding attack, approximately at the time point 12. This spike is due to a high packet transmission rate of the accessed Web sites while undergoing the DDoS attack. After this peak, the packet rate comes to a second state of a lower level indicating the effect of that particular attack in the network traffic.

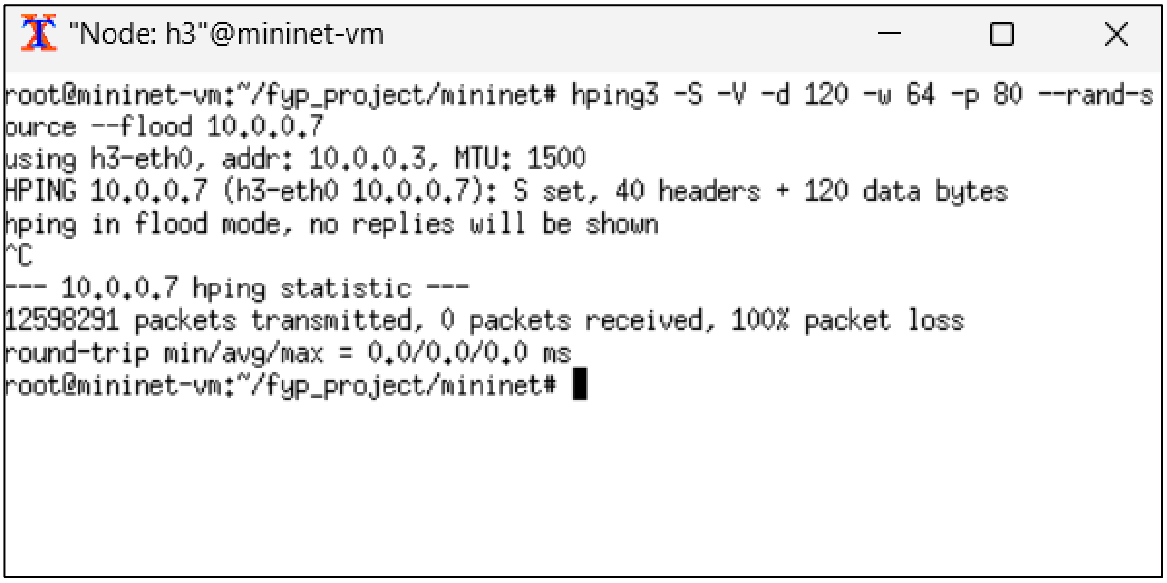

Capturing from h2-eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time         Source           Destination  Protocol  Length  Info
67482 48.956047241 25.184.199.74    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=12367/20272, ttl=64 (no response found!)
67483 48.958414852 36.92.202.255    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=24143/20318, ttl=64 (no response found!)
67484 48.960229420 83.242.17.86     10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=20559/20304, ttl=64 (no response found!)
67485 48.960246131 232.186.18.1     10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=15439/20284, ttl=64 (no response found!)
67486 48.960254053 140.59.190.119   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=15951/20286, ttl=64 (no response found!)
67487 48.960261064 165.118.200.16   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=20047/20302, ttl=64 (no response found!)
67488 48.966259692 88.172.213.0     10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=15183/20283, ttl=64 (no response found!)
67489 48.967022311 217.14.61.49     10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=21327/20307, ttl=64 (no response found!)
67490 48.967031423 214.236.113.236  10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=19279/20299, ttl=64 (no response found!)
67491 48.967038060 196.50.148.131   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=21071/20306, ttl=64 (no response found!)
67492 48.967044706 219.202.145.148  10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=16975/20290, ttl=64 (no response found!)
67493 48.967050594 254.172.76.234   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=23119/20314, ttl=64 (no response found!)
67494 48.967057544 101.237.197.209  10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=22351/20311, ttl=64 (no response found!)
67495 48.967064168 109.85.197.177   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=19791/20301, ttl=64 (no response found!)
67496 48.967071159 235.175.140.70   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=12879/20274, ttl=64 (no response found!)
67497 48.967076957 74.80.54.39      10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=16207/20287, ttl=64 (no response found!)
67498 48.967082827 176.232.197.230  10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=15695/20285, ttl=64 (no response found!)
67499 48.972198749 62.148.214.99    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=18255/20295, ttl=64 (no response found!)
67500 48.972230226 48.3.54.13       10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=23631/20316, ttl=64 (no response found!)
67501 48.972236899 13.200.156.123   10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=21839/20309, ttl=64 (no response found!)
67502 48.972242746 89.183.96.192    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=14927/20282, ttl=64 (no response found!)
67503 48.972248851 192.40.54.200    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=18511/20296, ttl=64 (no response found!)
67504 48.972255207 155.81.131.73    10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=18767/20297, ttl=64 (no response found!)
67505 48.972260595 165.214.211.243  10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=12623/20273, ttl=64 (no response found!)
67506 48.972265867 15.144.70.54     10.0.0.2     ICMP      162 Echo (ping) request  id=0x830a, seq=17743/20293, ttl=64 (no response found!)

▸ Frame 1: 162 bytes on wire (1296 bits), 162 bytes captured (1296 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 00:00:00_00:00:02 (00:00:00:00:00:02)
▸ Internet Protocol Version 4, Src: 89.140.142.82, Dst: 10.0.0.2
▸ Internet Control Message Protocol
```

Figure 16. Wireshark Capture of ICMP Packets during DDoS Attack Simulation.

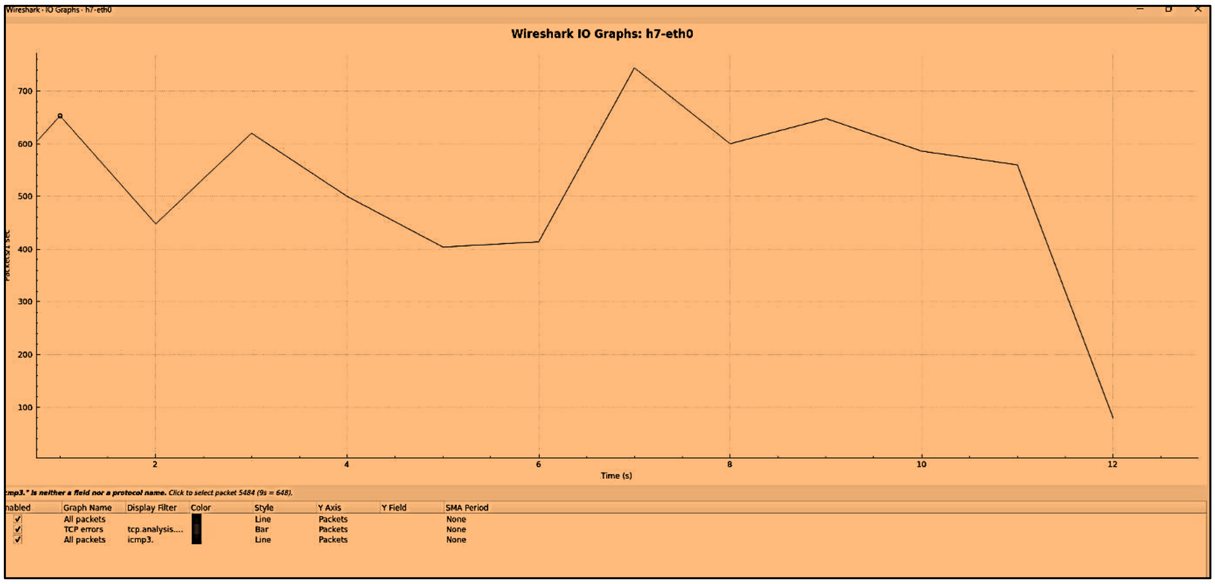
The figure demonstrates the interface of Wireshark while capturing ICMP packets during the dummy DDoS attack. From the captured log, the innumerable ICMP echo request packets listed with different source IP addresses sent to the destination IP (10.0.0.2). Each entry provides information like serial number, packet size and response indication which reveals that no response was received against the echo request. This captures the essence of the flood attack in the sense that there is going to be a massive number of packets going out hoping to flood the target system.



```
root@mininet-vm:~/fyp_project/mininet# hping3 -S -V -d 120 -w 64 -p 80 --rand-s
ource --flood 10.0.0.7
using h3-eth0, addr: 10.0.0.3, MTU: 1500
HPING 10.0.0.7 (h3-eth0 10.0.0.7): S set, 40 headers + 120 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.7 hping statistic ---
12598291 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@mininet-vm:~/fyp_project/mininet# █
```

Figure 17. SYN Flood Attack Simulation Using hping3

The figure depicts a terminal session of a Mininet virtual machine that I am using to perform a simulation of an attack, using hping3 command, SYN flood attack. It has a packet size of packet and works by utilizing TCP packets which have various characteristics such as protocol, not to mention the randomized source IP address. The terminal signifies that in total 1,259,821 packets were sent yet none were received hence a 100 % packet drop out. The following output represents the execution of the attack called the SYN flood whose aim is to inundate the targeted server with SYN requests that have not been finished with the handshake process.



mp3." Is neither a field nor a protocol name. Click to select packet 5484 (9s = 648).

nabled	Graph Name	Display Filter	Color	Style	Y Axis	Y Field	SMA Period
✓	All packets		■	Line	Packets		None
✓	TCP errors	tcp.analysis....	■	Bar	Packets		None
✓	All packets	icmp3.	■	Line	Packets		None

Figure 18. Wireshark IO Graphs during SYN Flood Attack Simulation on h7-eth0

Specifically, the figure illustrates the IO Graphs function of Wireshark and demonstrates the traffic originating from the h7-eth0 interface during a SYN flood attack emulation. This is made clear by the diffusion slope of the graph that shows the number of packets received per second and puts into evidence periodic oscillations of the traffic. However, it is most significant here to point to the fact that there is the highest rate of packet transmission here, which shows that the SYN flood attack was intense at these times. This graph acts as a logical representation of how the attack affects the flow of packets with period of high packet flow representative of SYN flood attack.

5. Discussion

Specifically, the Random Forest Classifier was selected due to its high accuracy to adjust the network traffic as either good or bad. The confusion matrices also showed how the model can successfully discern between DDoS attacks with very few false positives and false negatives. This level of accuracy should in particular not forgo further traffic as to damage the reliability and trust of the participating users. Nonetheless, specific parameters are more important: precision and recall as the results of the confusion matrix underline. Precision evaluates the number of correct identifications as a percentage of all identified positive while recall estimates the ability of a destination to pick out all the real positive identities. The F1 score of the system was 99.87% proving that the system tested high reliability and accuracy in the detection of DDoS attacks.

The presented system was examined in real-time on a dedicated SDN network constructed in Mininet using live emulation. Thus during these simulations, the system was very successful in identifying the malicious hosts and in blocking the traffic on the compromised ports. Such threats were promptly addressed to prevent major havoc, as the prognosis might have signified to different stability of the networks. Thus these results support the possibility of using the system in dynamic network environments, where timely response is crucial. The adaptability of the system and its performance under attack is well explained by the agility afforded by the system to dynamically modify flow rules on OpenFlow switches to mitigate the ongoing attack.

6. Conclusion

The objective of this study were met and this work developed and deployed an accurate detection and mitigation system for a DDoS attack in an SDN environment using the Random Forest Classifier approach with an accuracy of 0.997%. The system was shown to have practical utility in network emulation using Mininet and traffic analysis with Wireshark due to the ability to efficiently neutralize attack traffic in real-time as well as avoid overloads and guarantee service availability. The presented project may prove useful in many ways, firstly, as a resource for academicians and students interested in DDoS mitigation, and secondly, for industries, especially cybersecurity and cloud computing to implement a scalable, flexible, and efficient solution against DDoS threats. The modularity, centralized control with software- defined networking, low false positives and high detection rates mark it as strong and flexible.

Corresponding author

Aitizaz Ali
aitizaz.ali@apu.edu.my

Acknowledgements
I would also like to express my sincere appreciation to my supervisor, Dr. Aitizaz Ali for his support, direction, and motivation during this research. His comments, knowledge, and suggestions have been a great source of guidance as well as a significant help in gaining valuable feedback for my projects. I have been motivated by Dr. Ali's commitment and love for his work, which has encouraged me to pursue the highest level of quality in my studies. I am very grateful for his support and guidance throughout this project. Thank you for entrusting me with my project and for showing me that I can develop myself with your guidance.

Funding
NA

Contributions
N.F; A.A; Conceptualization, N.F; A.A Investigation, N.F; A.A Writing (Original Draft), N.F; A.A; Writing (Review and Editing) Supervision, N.F; A.A; Project Administration.

Ethics declarations
This article does not contain any studies with human participants or animals performed by any of the authors.

Consent for publication
Not applicable.

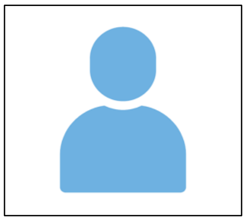
Competing interests
All authors declare no competing interests.

References

[1] Cai, J., Qian, F., Yu, R., & Shen, L. (2020). Output feedback control for pneumatic muscle joint system with saturation input. *IEEE Access*, *8*, 83901–83906. https://doi.org/10.1109/access.2020.2991729
[2] Choo, K.-K. R. (2011). The cyber threat landscape: Challenges and future research directions. *Computers & Security*, *30*(8), 719–731. https://doi.org/10.1016/j.cose.2011.08.004
[3] Douligeris, C., & Mitrokotsa, A. (2004). DDoS attacks and defense mechanisms: Classification and state-of-the-art. *Computer Networks*, *44*(5), 643–666. https://doi.org/10.1016/j.comnet.2003.10.003
[4] Gal-Oz, N., Gonen, Y., & Gudes, E. (2019). Mining meaningful and rare roles from web application usage patterns. *Computers & Security*, *82*, 296–313. https://doi.org/10.1016/j.cose.2019.01.005

[5] Hashmi, U. S., Zaidi, S. A., Darbandi, A., & Imran, A. (2018). On the efficiency tradeoffs in user-centric cloud RAN. In *2018 IEEE International Conference on Communications (ICC)*. https://doi.org/10.1109/icc.2018.8422228
[6] Kolias, C., Kambourakis, G., Stavrou, A., & Voas, J. (2017). DDoS in the IoT: Mirai and other botnets. *Computer, 50*(7), 80–84. https://doi.org/10.1109/mc.2017.201
[7] Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review, 34*(2), 39–53. https://doi.org/10.1145/997150.997156
[8] Morvant, E. (2015). Domain adaptation of weighted majority votes via perturbed variation-based self-labeling. *Pattern Recognition Letters, 51*, 37–43. https://doi.org/10.1016/j.patrec.2014.08.013
[9] Pareek, G., & B.R., P. (2021). Secure and efficient revocable key-aggregate cryptosystem for multiple non-predefined non-disjoint aggregate sets. *Journal of Information Security and Applications, 58*, 102799. https://doi.org/10.1016/j.jisa.2021.102799
[10] Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys, 39*(1), 3. https://doi.org/10.1145/1216370.1216373
[11] Rahman, Md. A., Asyhari, A. T., Bhuiyan, M. Z., Salih, Q. M., & Zamli, K. Z. (2018). L-CAQ: Joint link-oriented channel-availability and channel-quality based channel selection for mobile cognitive radio networks. *Journal of Network and Computer Applications, 113*, 26–35. https://doi.org/10.1016/j.jnca.2018.03.022
[12] Spanos, G., & Angelis, L. (2016). The impact of information security events on the stock market: A systematic literature review. *Computers & Security, 58*, 216–229. https://doi.org/10.1016/j.cose.2015.12.006
[13] Swain, G. (2016). A steganographic method combining LSB substitution and PVD in a block. *Procedia Computer Science, 85*, 39–44. https://doi.org/10.1016/j.procs.2016.05.174

Biographies



Nathaniel Frederick, Department School of Technology Networks, Security, Forensic Asia Pacific University of Technology & Innovation (APU). nathanielfrederick76@gmail.com

Aitizaz Ali received the master's degree in computer systems engineering (with distinction) from GIK Institute, Topi, Khyber Pakhtunkhwa, Pakistan, and the Ph.D. degree in cybersecurity and blockchain technology from the School of IT, Monash University Malaysia, Jaya, Malaysia. He is a Lecturer with the School of IT, UNITAR International University, Petaling Jaya, Malaysia. He is the author of several Journal papers and international Conferences. He has authored or coauthored more than 20 research papers, including in highquality journals. His research interests include blockchain, cloud Ccomputing, cybersecurity, cryptography, deep learning, AI, and healthcare systems. Moreover. He was the Reviewer of IEEE Internet of Things Journal, IEEE Transactions on Network Science and Engineering, IEEE Access, IET, and Human-centric Computing and Information Sciences Journals for several years. aitizaz.ali@apu.edu.my