Design and Development of a Smart Framework for Proactive Defense Against Distributed Reflection Denial of Service Attacks in Software Defined Networking Environment

THESIS

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

SHAIL SAHARAN
2017PHXF0404P

Under the Supervision of

Prof. Vishal Gupta
Associate Professor
Department of Computer Science and Information Systems
Birla Institute of Technology & Science, Pilani – 333 031(Rajasthan)

BITS Pilani

Pilani | Dubai | Goa | Hyderabad | Mumbai

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI - 333031 (RAJASTHAN) INDIA
2024

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI - 333031 (RAJASTHAN) INDIA

CERTIFICATE

This is to certify that the thesis entitled **"Design and Development of a Smart Framework for Proactive Defense Against Distributed Reflection Denial of Service Attacks in Software Defined Networking Environment"** submitted by **Shail Saharan** ID. No. **2017PHXF0404P** for the award of Ph.D. of the Institute embodies original work done by her under my supervision.

Signature of the Supervisor	
Name in capital letters	: **Prof. Vishal Gupta**
Designation	: Associate Professor, Department of CSIS Birla Institute of Technology & Science PILANI - 333031(Rajasthan)

Date: ________________________

Acknowledgments

I want to express my gratitude to Professor V Ramgopal Rao, Vice Chancellor, BITS Pilani, for providing all the necessary facilities to successfully complete the study. I am highly grateful to Prof. Sudhir Kumar Barai, Director, BITS Pilani, Pilani campus, for his inspiration. I sincerely thank Prof. Shamik Chakraborty, Associate Dean, Academic Graduate Studies and Research Division (AGSRD), and Prof. Navneet Goyal (Head, Department of CSIS) for providing me with all the research facilities. I gratefully acknowledge the support received from Dr. Shashank Gupta, DRC Convenor, in completing all the formalities smoothly.

I sincerely thank my supervisor Prof. Vishal Gupta, Associate Professor, Department of CSIS, for his valuable guidance and support in my research work. His patience, cooperation, and suggestions helped me in every step of the accomplishment of the study. His skillful supervision enriched this study higher than my expectations.

I am incredibly thankful to Prof. K. Haribabu and Dr. Shashank Gupta for providing valuable inputs as Doctoral Advisory Committee members and helping me improvise my research throughout the process. I am also grateful to all the department's faculty members and research scholars for their support and encouragement.

I want to express my profound gratitude to my parents, Mr. Bhagat Singh Saharan and Mrs. Madhu Choudhary, for providing me with unfailing support, patience, and continuous encouragement throughout my academic life. I am grateful to my sisters and brother, Dr. Anushansa Singh, Ms. Lisa Sagrohe, and Mr. Nikhil Choudhary, for their love, affection, and constant support. I thank my husband, Dr. Chinmay Choudhary, for always being there through thick and thin. I am thankful to all my family members for always being a significant source of support.

Getting through my thesis required more than academic support, and I feel indebted to Arshveer Kaur, Stuti Chug, and Pratibha Saini for listening to me and, at times, having to tolerate me over the past four and a half years. I cannot begin to express my gratitude and appreciation for their friendship and unwavering personal support while I was at the University.

Finally, my greatest gratitude to God for letting me through all the difficulties and making me a humble and compassionate human.

SHAIL SAHARAN

With the proliferation of Internet activities in one's personal and professional life, network security has become one of the most important topics for discussion. It deals with protecting the integrity, confidentiality, and accessibility of computer networks and data. Countless security breaches attempting to steal user data and hack into private spaces have made this topic even more critical. One such category of a breach is the Distributed Denial of Service (DDoS) attack. It aims at disrupting the flow of genuine traffic by overwhelming the resources of the targeted network. Attackers send a huge amount of network traffic in a distributed fashion to (mainly) a single victim. Its primary goal is to overwhelm the resources of the targeted network, thus making it and its provisioned services (if any) unusable for genuine users.

Distributed Reflection-based Amplification Denial of Service Attacks (DRDoS) is a type of DDoS attack. It exploits the client-server communication model wherein a client sends a request packet to the server and receives a response packet. The response packets in this attack are much larger in size in comparison to the request packets. The attacker spoofs the source Internet Protocol (IP) address to that of the victim in the request packet. Since the underlying Internet architecture does not validate the source IP address while forwarding the packets, as a result, all responses go to the victim. One such protocol often exploited for DRDoS attacks is a Domain Name System (DNS) protocol. The attacker uses various zombie computers to send massive DNS query packets to DNS servers. The packets' inscribed source address is set to the victim's address so that when the DNS server responds to the queries, the victim receives many respective response packets, and, in turn, the system crashes. This type of attack is amplification-based because DNS response packets are generally exponentially larger than DNS request packets. Thus, amplification in the packet's size consumes the victim's bandwidth and, in the worst-case scenario, crashes the system by overwhelming it with the packets. This study deals with providing defense against DRDoS attacks.

Because of the legitimate character of the traffic, detecting and mitigating DRDoS attacks becomes more challenging. Distinguishing between attack traffic and legitimate traffic is a formidable task. Even, if possible, the victim's resources are overwhelmed during attack detection and mitigation. Therefore, attack prevention is considered a better defense approach. In the thesis, we have provided prevention techniques against DRDoS attacks in the Software Defined Networking (SDN) environment. Utilizing the flexibility and programmability aspects of SDN, via this study, we intend to make the underlying network smart enough to prevent attacks. More specifically, we have designed, developed, and validated an SDN-based framework that will introduce "appropriate intelligence" to enhance the functionality of OpenFlow-enabled L2/L3 switches so that the underlying network itself can prevent DDoS attacks. This thesis proposes eight different techniques to prevent/detect DRDoS attacks.

SYMSDN, IP-Switching, PortMergeIP, and Port-Mapping techniques focus on modifying reverse forwarding rules. In these techniques, the response from the server goes back to the attacker even when the attacker spoofs the source IP of the victim; hence, the attacker is penalized for the attack. SYMSDN and Port-Mapping use symmetric routing as the underlying approach, thus requiring modification in the entire core network. PortMergeIP and IP-Switching depend on the Internet Service Provider (ISP) to modify the IP address of the request packet; hence, they require changes in only the edge networks.

RDPID, again a prevention approach, uses Path Identifiers (PIDs) which are primarily used in Information-Centric Networks (ICN) to forward response packets on PIDs. As against static PIDs, we use Reliable Dynamic PIDs (RDPIDs) to refrain the attackers from learning

these PIDs and launching the attack. These PIDs are proposed to be stored within the packet so that the response packet can follow the same path as the respective request packet.

PoDIBC, another proposed prevention technique, does source authentication to prevent IP spoofing. It is achieved by using signatures to authenticate the sender's identity. PoDIBC uses the Barreto, Libert, McCullagh, Quisquater (BLMQ) signature scheme of Identity-Based Cryptography (IBC) to prevent the targeted victim from the attack packets. Since IBC uses the packet identity as a public key (which in PoDIBC is the source-IP address), it eliminates the infrastructure required for public key certificate distribution.

RF-SDN is a Machine Learning (ML) approach used in the SDN environment to prevent DDoS attacks. With the advancement of artificial intelligence (AI), (ML) algorithms have become sophisticated enough to classify traffic as malicious or benign based on distributional differences between malicious and benign packets. We use random Forest model in the SDN environment to detect DDoS attacks, primarily focusing on Portmap, DNS, UDP, UDP-lag, and SYN datasets available in the CICDDoS 2019 attack dataset. Through the prediction made by the model as either attack traffic or legitimate traffic, the attack is detected and blocked. The approach is validated to detect a DDoS attack in near real-time and prevent the attack traffic from reaching the victim.

Finally, we propose DDoS detection using entropy. Entropy is used to measure randomness or uncertainty in a network's traffic. As this randomness decreases, i.e., one type of traffic dominates the network, the entropy value declines. This abbreviation leads to the possibility of an attack on the network. The proposed approach employs a dynamic threshold mechanism to distinguish between regular and attack traffic effectively.

All the prevention approaches are validated through detailed experimentation by creating topologies involving attackers, victims, and a server to generate amplification attacks.

The results show that because of the proposed approaches, none of the attack traffic reaches the victim, or approximately 1 % (in the case of RF-SDN).

List of Tables

S. No.	Abbreviation	Definition
1.	DoS	Denial of Service
2.	DDoS	Distributed Denial of Service Attacks
3.	DRDoS	Distributed Reflection Denial of Service
4.	MITM	Man In The Middle
5.	RTT	Round-Trip Time
6.	IP	Internet Protocol
7.	MAC	Media Access Control
8.	TCP	Transmission Control Protocol
9.	TTL	Time To Live
10.	UDP	User Datagram Protocol
11.	BDP	Bandwidth Delay Product
12.	ISP	Internet Service Provider
13.	NAT	Network Address Translation
14.	OS	Operating System
15.	AS	Autonomous System
16.	AD	Accountability Domains
17.	BR	Border Router
18.	RM	Resource Manager
19.	NM	Network Manager
20.	DNS	Domain Name System
21.	ARP	Address Resolution Protocol
22.	HTTP	Hypertext Transfer Protocol
23.	SMTP	Simple Mail Transfer Protocol
24.	NTP	Network Time Protocol
25.	BGP	Border Gateway Protocol
26.	AIP	Accountable Internet Protocol
27.	CLDAP	Connection-less Lightweight Directory Access Protocol
28.	CSP	Cloud Service Provider
29.	IoT	Internet of Things
30.	ICN	Information-Centric Network
31.	ML	Machine Learning
32.	DL	Deep Learning
33.	AI	Artificial Intelligence
34.	ONF	Open Networking Foundation
35.	SDN	Software Defined Networking
36.	NFV	Network Function Virtualization
37.	VM	Virtual Machine
38.	PID	Path Identifier
39.	DPID	Dynamic Path Identifier
40.	RDPID	Reliable Dynamic Path Identifier
41.	NID	Network Identity
42.	IBC	Identity-based Cryptography
43.	IBS	Identity-Based Signature

44.	IBE	Identity-Based Encryption
45.	BLMQ	Barreto, Libert, McCullagh, Quisquater
46.	OOB	Out Of Bag
47.	IPS	Intrusion Prevention System
48.	IDS	Intrusion Detection System
49.	NIDS	Network Intrusion Detection System
50.	HIDS	Host Intrusion Detection System
51.	IPDS	Intrusion Prevention Detection System
52.	CSP	Cloud Service Provider
53.	RSU	Road Side Unit
54.	WSN	Wireless Sensor Network
55.	VANET	Vehicular Ad-hoc Networks
56.	MANET	Mobile Ad-hoc Networks
57.	MAEC	Multi-Access Edge Computing
58.	OS	Operating System
59.	MPLS	Multi-Protocol Label Switching
60.	LAN	Local Area Network
61.	WLAN	Wireless Local Area Network
62.	VLAN	Virtual Local Area Network
63.	LISP	Location/ID separation protocol
64.	TR	Tunnel Routers
65.	RPF	Reverse Path Forwarding
66.	FIB	Forwarding Information Base
67.	M-Boxes	Middle Boxes
68.	SVM	Support Vector Machine
69.	SOM	Self-Organised Map
70.	PCA	Principal Component Analysis
71.	LDA	Linear Discriminant Analysis
72.	ANN	Artificial Neural Network
73.	TPA	Third-Party Auditor
74.	VRF	Virtual Router Firewall
75.	SAVI	Source Address Validation Improvement
76.	SAVA	Source Address Validation Architecture
77.	SAVE	Source Address Validity Enforcement
78.	FCFS	First Come First Served
79.	DHCP	Dynamic Host Configuration Protocol
80.	SEND	Secure Neighbor Discovery
81.	MAAM	Mixed Address Assignment Methods
82.	WIDIP	Wireless Distributed IPS
83.	PKES	Public Key Exchange Server
84.	IDEA	International Data Encryption Algorithm
85.	RDPF	Route-based Distributed Packet Filtering
86.	SOS	Secure Overlay Service
87.	API	Application Programming Interface
88.	TOS	Time Of Service
89.	DTLS	Datagram Transport Layer Security
90.	CGA	Cryptographically Generated Address
91.	CA	Certifying Authority
92.	MI	Mutual Info

93.	GE	General Entropy
94.	SPRT	Sequential Probability Ratio Test

CHAPTER 1- Introduction

1.1 DDoS Attacks

With the abundance of computers in our day-to-day activities and their need to interface with the Internet, many organizations, directly or indirectly, depend upon Internet infrastructure for proper and reliable functioning. The services provided by various organizations primarily depend upon two aspects—first, the correct functioning of their respective applications, and second, the underlying Internet itself. If the organization's application is faulty, the individual organization will only be blamed. But whom to blame if the organization's service model is disrupted due to the vulnerabilities of the underlying Internet architecture? For example, it would be chaos, hassle, and revenue loss if any ticket booking (flight, train, movie, etc.) website could be made inaccessible using the vulnerabilities of the underlying Internet architecture. Attackers are always looking for such vulnerabilities to launch different cyber-attacks. Distributed Denial of Service (DDoS) attack is one type where an attacker uses multiple zombie machines and generates enormous traffic to attack the victim. For example, one such attack was observed in 2016 when a series of DDoS attacks targeted a Domain Name System (DNS) provider organization named DYN (2016 Dyn Cyberattack, 2021). As a result, it caused significant Internet platforms and services unavailable to a large part of Europe and North America.

DDoS attacks are launched for many different reasons. These are business rivalry, political mileage, taking revenge, monetary gain, etc. In such attacks, by generating an enormous number of network packets, the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting the services of a host connected to the Internet. The attackers target and make inaccessible the services of a website or any online service directly or through that service's dependencies. The attacker often chokes the available bandwidth, making resources unavailable to legitimate users.

1.1.1 DDoS Attacks a Significant Threat

A report by Amazon suggests that a 100-millisecond delay in response time can potentially drop their overall sales by approximately 1% (Greg, 2006). The various security reports generated by Arbor Networks (Netscout, 2019), Cloudflare (Famous DDoS Attacks | The Largest DDoS Attacks of All Time, 2021), and Secure List (B. Kupreev Oleg et al., 2021) suggest that DDoS attacks are a significant threat. According to Cloudflare, five of the most impressive DDoS attacks are (a) the attack on Github, (b) the attack on DYN servers, (c) the Spamhaus attack, (d) the attack on Google, and (e) the 2020 AWS attack (Famous DDoS Attacks | The Largest DDoS Attacks of All Time, 2021). A10 Networks classified the attacks based on amplitude (Paul, 2020). The top five among these are (a) The Google attack (2.5 Tbps), (b) The AWS DDoS attack in 2020 (2.3 Tbps), (c) The DYN attack (1.5 Tbps), (d) The attack on GitHub (1.35 Tbps), and (e) attack on Occupy Central (500 Gbps). Many DDoS-for-hire services are available as paid services that take the responsibility of attacking on the attacker's behalf (Booters, Stressers and DDoSers, 2021). As they have a whole infrastructure on their side, they are more capable of making a full-fledged attack. According to the Q3 report of 2019 (B. E. Kupreev Oleg & Alexander, 2019) by Secure List, even when the FBI took down many DDoS-for-Hire sites, new ones sprung up in their place, and the number of attacks using these services increased by 400% from the previous quarter. As highlighted, following is a brief about some of the deadliest DDoS attacks to understand more about the threat and the methodology associated with DDoS attacks.

a) The attack on Google- The DDoS attack on Google peaked at 2.5 Tbps in September 2017 and took place over six months (Menscher, 2020). The attack was a reflection attack that used about 180,000 servers, such as Connection-less Lightweight Directory Access Protocol (CLDAP), DNS, and Simple Mail Transfer Protocol (SMTP), to send amplified responses to Google servers (Cimpanu, 2020b). The spoofed requests were 167 Mbps and are said to be originated from several Chinese Internet Service Providers (ISPs). After three years, Google disclosed this deadliest attack to raise awareness about this huge threat, i.e., DDoS attacks. It also stated that these attacks would intensify with increased bandwidth in the coming years.

b) The AWS 2020 attack- Amazon Web Services (AWS) is a primary Cloud Service Provider (CSP). In the first quarter of 2020, AWS reported a DDoS attack that was 44 % larger than any other volumetric attack detected by AWS Shield (Threat Landscape Report – Q1 2020, 2020). The attack peaked at 2.3 Tbps. It was also a reflection attack that used CLDAP servers to launch the attack. AWS did not disclose the customers to whom the attack was targeted but only said that it was an "elevated threat" for three days for AWS Shield staff (Cimpanu, 2020a).

c) The attack on GitHub- According to the Institute for Research on Internet Society, this attack was launched in March 2018 (DDOS Attacks and the GitHub Case, 2018). It used Memcache, an open-source, simple, and robust distributed memory caching system. Memcache reduces database load through caching, which helps to increase the speed of dynamic web applications. It works in the form of a client-server application. The list of Memcached servers is made available through the client software. The client's requested object is first checked in Memcache; if it is there, the value is returned; otherwise, a request to the database is made. Various institutions own about 100,000 Memcache servers (Newman, 2018). Anyone can use them to send special request commands to which these servers respond with a much larger reply. The attack was launched on GitHub using this feature. It was also a reflection attack (Ghoshal, 2018). The attacker just requested the data from Memcache servers and spoofed their source IP addresses to that of GitHub, resulting in an attack of 1.35 Tbps.

d) DYN attack- The attack occurred on October 21, 2016, and was on American and European Internet infrastructure (Famous DDoS Attacks | The Largest DDoS Attacks of All Time, 2021). A total of 85 major sites suffered as the main target was DYN, a DNS service provider company. The major sites affected were Airbnb, Netflix, PayPal, Visa, Amazon, The New York Times, Reddit, and GitHub, as they all used services from DYN (2016 Dyn Cyberattack, 2021). The attack was launched by producing a large number of DNS requests for DYN servers through an army of bots. The unique feature of this attack was that the bots were created from tens of millions of compromised Internet of Things (IoT) devices, such as cameras, radios, etc., using malware named Mirai. It took about one day to mitigate the attack.

e) Spamhaus attack- Spamhaus, a nonprofit organization, works in threat intelligence. Its services protect about three billion users' mailboxes from spam messages, malware, phishing, etc. As Spamhaus works against Internet abusers, it is on the target list of many cybercriminals. Among many attacks on Spamhaus, the biggest one occurred on March 18, 2013. It was a DNS amplification attack. Generated by open DNS resolvers, the attack started from 10 Gbps and reached 90 Gbps on March 19, 2013. On March 22, the attack reached 120 Gbps, as reported by Cloudflare, which finally mitigated the attack (Prince, 2013a). In this attack, the attackers generated requests for DNS zone files for ripe.net (Prince, 2013b) to open DNS resolvers. The request packet size was 36 bytes, and the response packet size was estimated to be 3000 bytes in size. Thus, approximately 100 times amplification factor. Furthermore, the source IP address of the request packets was spoofed with Spamhaus; hence, all the responses from over 30,000 unique DNS resolvers went to Spamhaus, creating such massive traffic on Spamhaus servers.

1.1.2 Defense against DDoS Attacks

From all the attack statistics mentioned in section 1.1.1, DRDoS attacks were among the deadliest attacks. Due to their legitimate nature, it is tough to differentiate whether the traffic is a legitimate one or an attack; hence, they are harder to detect. These attacks generally exploit the client-server architecture. In DRDoS attacks, the attacker spoofs the source IP address of request packets to that of the victim, resulting in all the server response packets going to the victim instead of the attacker. Many techniques in the literature claim to protect from DDoS attacks. The defense against DDoS attacks can be classified as attack detection, mitigation, and prevention. These are formally defined as follows-

1.1.2.1 DDoS Attack Detection -

Strategies falling under this realm are equipped for detecting the attack with probability P, where P depends upon the particular detection mechanism. In pure detection techniques, the host/network is allowed to be susceptible to DDoS attacks. Once an attack has started, the primary focus is to maximize the probability of detection based on different heuristics. These heuristics vary from network traffic analysis, entropy, anomalous behavior of data traffic, etc. (J. Singh & Behal, 2020). Once an attack is detected, remedial measures can be taken.

1.1.2.2 DDoS Attack Mitigation -

Once an attack is detected, strategies falling under this realm can mitigate it with probability P_1 and within time T. Again, the value of P_1 and T depends upon the particular mitigation mechanism. Once the attack is detected, DDoS mitigation techniques or tools are used to mitigate the impact of such an attack. These techniques are primarily based on rate-limitation (J. Singh & Behal, 2020), tracing the attacker, blocking it, etc.

1.1.2.3 DDoS Attack Prevention -

In general, prevention means not permitting something from occurring in the first place. Thus, in the context of DDoS attacks, strategies falling under this realm should not allow DDoS attacks from happening. Prevention means strategizing defense so that DDoS attacks can be prevented from happening in the first place, and even if they occur, the victim should remain as unaffected as possible by the attack.

1.2 Contribution of Thesis

The primary focus of activities aimed at countering DDoS attacks revolves around two main aspects: detecting the attacks, followed by mitigating the attack. With this approach, the defense strategies are reactive, leaving the underlying network vulnerable to DDoS attacks until they occur. Once an attack takes place and exhausts the victim's bandwidth and resources, only then do the detection and mitigation measures come into play.

Rather, a more promising strategy involves a shift toward attack prevention. This proactive approach aims to either prevent the attacks from being generated in the first place or, if they are generated, ensure that the attack traffic is unable to reach the victim. In contrast, the detection and mitigation methods always involve the attack traffic reaching the victim, thus disrupting its normal functioning.

Considering the benefits of prevention and its proactive nature, it is regarded as the primary defense technique in this study against DDoS attacks.

Limiting the scope to the Internet architecture, we have identified three distinct categories of DDoS prevention techniques based on their effectiveness in safeguarding the victim. These categories are named Ideal Prevention, True Prevention, and Partial Prevention, as illustrated in Figure 1.1.

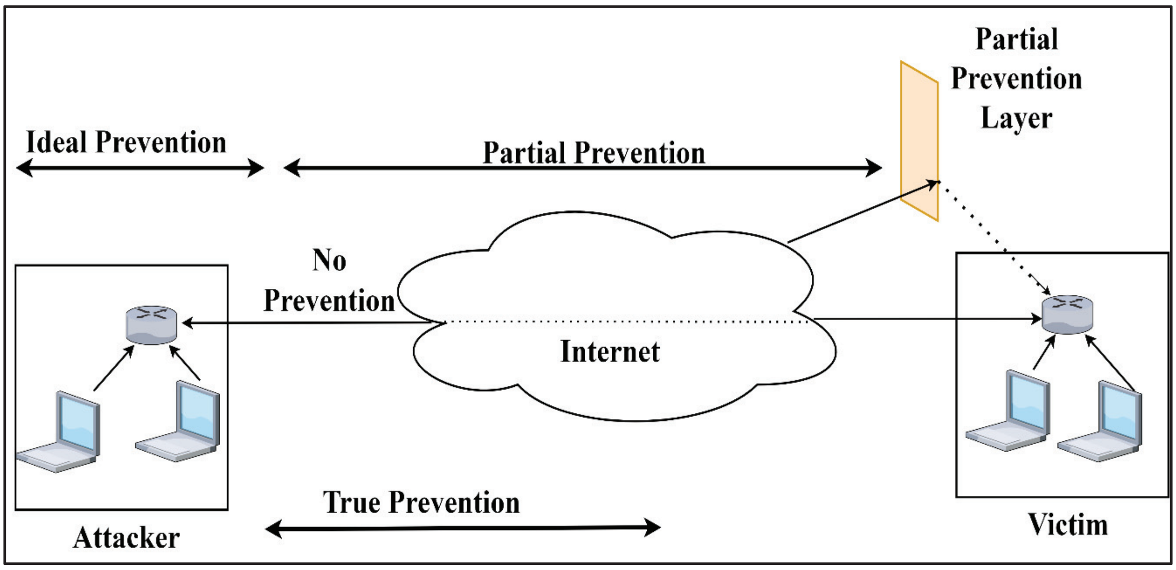


Figure 1.1 : Prevention techniques

a) **Ideal Prevention-** This prevention technique completely prevents the attack traffic from entering the core network. In this case, the attack traffic generated by an attacker (or botnets in control of the attacker) cannot leave their respective access network.

b) **True Prevention-** This prevention technique allows the attack traffic to leave the attacker's network and enter the core network but is wholly prevented from reaching the victim's network.

c) **Partial Prevention-** This prevention technique initially allows the attack traffic to leave the attacker's network, enter the core network, and even reach the victim's network. However, the attack traffic is stopped from reaching the victim upon successful detection of an attack. Generally, an additional prevention layer between the attacker and the victim is responsible for this detection or diversion. The exact location of this prevention layer varies and may be anywhere in between the gateway router of the attacker's network and that of the victim's network.

Implementing the prevention technique mentioned in section 1.2 requires certain changes in the Internet architecture. These changes can be achieved in two ways: first, by enforcing predefined policies in all edge networks (access networks) to prevent attackers from launching attacks. Second, by enhancing the intelligence of the underlying core network responsible for traffic forwarding to minimize the probability of attack traffic reaching the victim. While it may be impractical to entirely stop attackers from launching attacks due to the need for widespread restrictions and upgrades across access networks, preventing attack traffic from reaching the victim's network is achievable by enhancing the intelligence of the underlying core network. This could involve introducing additional functionalities in core routers, incorporating extra layers in the core network, or having ISPs provide add-on functionalities to offer protection against DDoS attacks.

In this study, we have utilized Software Defined Network (SDN) as the underlying network architecture to demonstrate the necessary intelligence and changes required for effective DDoS prevention and detection techniques. SDN is a promising network paradigm that decouples the forwarding and control planes, enabling a programmable network architecture. Leveraging the flexibility and programmability of SDN, the study aims to enhance the intelligence of OpenFlow-enabled L2/L3 switches within an SDN-based framework to enable the underlying network itself to prevent, detect, and mitigate DDoS attacks.

Furthermore, among all the attacks discussed in section 1.1.1, DRDoS attacks were identified as the most severe. Due to their legitimate nature, distinguishing between legitimate traffic and an attack is challenging, making them harder to detect. Therefore, this study's main focus is prevention against DRDoS attacks.

1.2.1 Research Objectives

Based on the above discussion and literature review, the following objectives have been formulated for the study:

a) To study and analyse existing techniques in literature to defend against DDoS attacks and keep the primary focus on prevention.

b) Design a framework to introduce appropriate intelligence in underlying SDN-enabled network switches/controllers and assess its security against DRDoS attacks. Such switches may be a part of the core network or a separate barrier network through which the traffic will pass. The following two hypotheses are researched –

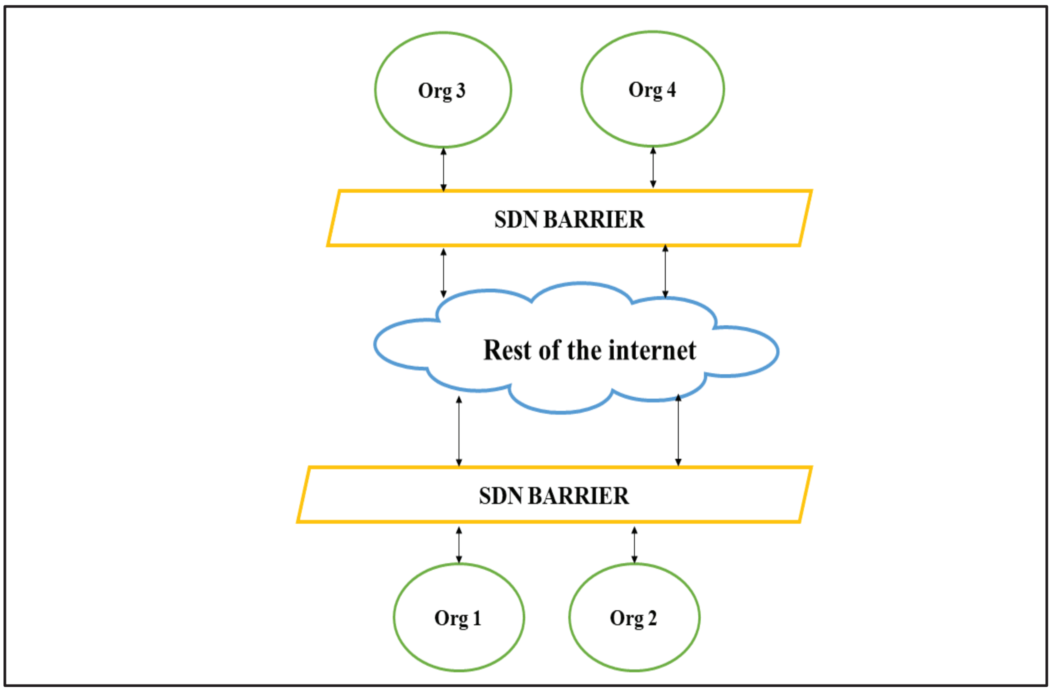


Figure 1.2: SDN barrier

- *Assuming that the entire network is SDN-enabled*

  In this scenario, we assume the entire network, including the core and edge networks, is SDN-enabled. It means that all the L2/L3 switches between the client and the server must be SDN-enabled. Hence, we propose modified routing algorithms for the SDN environment, which are capable of preventing/mitigating DRDoS attacks.

- *Assuming that the Internet core network is not SDN-enabled*

  A separate SDN-based barrier is proposed in this scenario, as shown in Figure 1.2. It is called a barrier, as all the traffic to an organization will pass through it. It can be implemented in two ways: either by a separate module connected through an SDN-enabled switch or by making only the edge network SDN-enabled (not the entire core network). It will be capable of preventing, detecting, and mitigating DRDoS attacks.

c) Implement and validate the proposed framework in SDN lab setup and measure its effectiveness to prevent DDoS attacks against various network metrics.

1.2.2 Organization of Thesis

The thesis comprises nine chapters, each serving a specific purpose.

Chapter 1 presents the introduction to the thesis, covering an overview of DDoS attacks and the significant threats they pose. The chapter also delves into DDoS defense techniques, emphasizing the importance of prevention against DDoS attacks. Furthermore, the contribution and research objectives of the thesis are highlighted.

Chapter 2 entails an extensive literature survey of existing prevention techniques, categorizing them into three approaches: Ideal Prevention, True Prevention, and Partial Prevention. These techniques form the basis for proactive defense against DRDoS attacks, which are the main focus of this research.

Chapter 3 briefly summarizes related theory, aiming to offer a comprehensive understanding of all proposed prevention techniques. The chapter briefly covers SDN and discusses the foundational techniques utilized in the prevention approaches: Identity-based Cryptography (IBC), random forest classifier, and symmetrical routing.

Chapters 4 and 5 elaborate on the proposed prevention techniques, categorized based on their underlying approach. These chapters center around modifying reverse forwarding rules such that the attacker attacks itself. Chapter 4 details four prevention techniques: SymSDN, IP-Switching, PortMergeIP, and Port-Mapping. While SymSDN and Port-Mapping approaches assume SDN-enabled Internet Core and edge networks, PortMergeIP and IP-Switching assume only the edge network to be SDN-enabled. Chapter 5 introduces a prevention approach using Path identifiers (PIDs), predominantly used in Information-Centric Networks (ICNs), enabling response packets to follow the same path as request packets.

In Chapter 6, the proposed prevention technique utilizes an Identity-Based Signature (IBS) scheme named Barreto, Libert, McCullagh, Quisquater (BLMQ), referred to as PoDIBC. This scheme employs source authentication to prevent IP spoofing by using signatures for sender identity authentication. PoDIBC employs the BLMQ signature scheme of IBC to protect the targeted victim from attack packets, eliminating the need for public key certificate distribution.

Chapter 7 incorporates the use of the Machine Learning (ML) algorithm, random forest, in the SDN environment for real-time DDoS attack detection and prevention. This defensive approach leverages the differences between malicious and legitimate traffic to identify and block attacks. With the advancement of Artificial Intelligence (AI), ML algorithms can effectively classify traffic as either malicious or benign based on distributional differences, enabling accurate predictions and timely action against attacks.

Chapter 8 focuses on the detection of DDoS attacks using entropy. Entropy is used to measure randomness or uncertainty in a network's traffic. As soon as this randomness decreases, i.e., one type of traffic dominates the network, the entropy value decreases. This abbreviation leads to the possibility of an attack on the network. The proposed approach employs a dynamic threshold mechanism to effectively distinguish between normal and attack traffic. Finally, Chapter 9 lists the conclusions of the thesis.

********** End of Chapter **********

2.1 Introduction

DDoS defense mechanisms are techniques or tools primarily used to defend networks or hosts attached to the Internet against DDoS attacks. These defense mechanisms are based on multiple different techniques such as traffic monitoring, various types of traffic analysis, traceback techniques, packet characterization techniques, etc. (Bhatia, S., Behal, 2018). We can classify any defense mechanism as prevention, detection, and mitigation. As mentioned in the previous chapter, prevention is a better defense technique than detection and mitigation to keep the victim from harm as much as possible. Hence, this chapter has a detailed literature survey of all the existing prevention techniques.

2.1.1 DDoS Defense Taxonomy

DDoS defense mechanisms can be looked upon from the perspective of their applicability, i.e., whether the mechanism is applied before the attack takes place (i.e., proactive defense mechanisms) or after the attack takes place (i.e., reactive defense mechanisms). Correspondingly, a taxonomy for DDoS defense is highlighted in Figure 2.1.

2.1.1.1 Reactive Techniques

Reactive techniques react to the DDoS attack, i.e., it allows the attack to happen, and then the detection or mitigation is done. In such a case, the victim starts detecting the attack when it is under attack. As soon as the victim detects the attack, an appropriate mitigation technique is applied. Here, the attack traffic constantly consumes the victim's network resources.

2.1.1.2 Proactive Techniques

Prevention means the act of stopping something from happening (Prevent). In general, a technique or a set of techniques that claims to "prevent an event" should not allow the same event to happen in the first place. For example, deadlock prevention methods do not allow deadlocks to happen (Deadlocks). Proactive techniques are primarily DDoS prevention techniques. Rather than reacting to a DDoS attack, techniques falling under this realm focus on preventing the DDoS attack traffic from reaching the targeted victim.

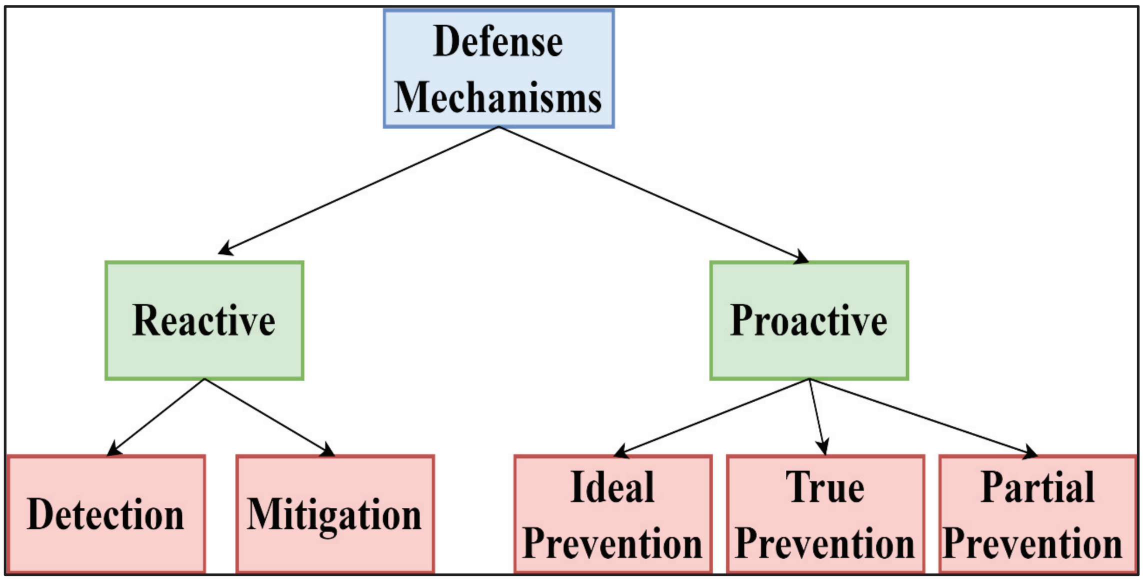


Figure 2.1: DDoS defense mechanisms

Depending upon the scope and meanings associated with various "Prevention" techniques, proactive techniques can further be classified into Ideal Prevention, True Prevention, and Partial Prevention. These further classifications can also be explained with the help of Figure 1.1, which shows an attacker, a victim, and a third-party layer responsible for mitigating the attack connected across the Internet. Figure 1.1 illustrates that in Ideal Prevention an attack packet cannot leave the attacker's network, and in True Prevention, attack traffic can leave the attacker's network but will be mitigated in the network; hence, it cannot reach the victim network. In Partial Prevention, the attack will be mitigated by a prevention layer from reaching the victim, but some percentage of attack reaches the victim.

a) Ideal Prevention

These types of prevention techniques prevent the attack traffic from entering the core network of the Internet. In this case, the attack traffic generated by an attacker (or botnets in control of the attacker) cannot leave their respective access networks. More formally, as defined by (Gupta et al., 2010), Ideal Prevention can be considered as "Attack prevention methods which try to stop all well-known signature-based, and broadcast-based DDoS attacks from being launched in the first place or edge routers keep all the machines over Internet up to date with patches and fix security holes." Later, the authors argue that DDoS attacks are always vulnerable to attack types for which signatures and patches do not exist in the database with this definition of prevention. By this definition, Ideal Prevention includes attack prevention methods that try to stop a type of DDoS attack (well-known signature-based attacks, broadcast-based DDoS attacks, reflection-based attacks, etc.) from being launched in the first place. We call it "Ideal Prevention" because such methods are ideal solutions if feasible and applied across the network.

b) True Prevention

These types of prevention techniques entirely prevent the victim from attacking traffic. As formally described by (Saharan & Gupta, 2021), the attack traffic can leave an attacker's access network and enter the core network of the Internet but can never reach the target victim's network. These types of techniques are attack prevention methods that introduce sufficient "intelligence" within the network and make the network self-sufficient. Here, self-sufficiency means that the network itself is capable of stopping/diverting the attack and does not allow the attack traffic to reach the victim's network. The attacker can attack, but the victim will not be affected in any manner.

c) Partial Prevention

These types of prevention techniques partially prevent the victim from the attack traffic. In this case, the attack traffic initially reaches the victim but the attack traffic is stopped from reaching the victim upon successful detection of an attack. A third-party prevention layer is placed between the attacker's and victim's networks. The job of this third party is to detect the attack that is taking place and stop it from reaching the victim's network as soon as possible. It may also be responsible for mitigating the attack. The exact location of this intermediate prevention layer varies and may be anywhere between the gateway router (which connects an attacker to the core network) and the victim's network. Partial Prevention is called partial because some of the attack reaches the victim's network.

2.2 DDoS Prevention: Proactive techniques

This section shows a detailed literature survey of DDoS prevention techniques, which fall under the realm of Ideal Prevention, Partial Prevention, and True Prevention. Also, DDoS prevention techniques in the literature are not just limited to Internet architecture. Instead, such techniques are also available for network architectures like Mobile Ad-hoc Networks (MANETS), Vehicular Ad-hoc Networks (VANETS), etc. However, we have classified all the prevention techniques in Table 2.1 for completeness.

Table 2.1 : Classification of prevention techniques

Author	**DDoS defense technique**	**Type**
(Freiling et al., 2005)	Botnet Tracking	Ideal Prevention
(Shafi & Basit, 2019)	Preventing botnet creation	Ideal Prevention
(H. Luo et al., 2013)	Identifier/locator separation	Ideal Prevention
(Baker & Savola, 2004)	Ingress filtering	Ideal Prevention

(Ndibwile et al., 2015)	Protection using ML and traffic authentication	Partial Prevention
(Zhuotao Liu et al., 2018)	M-boxes between client and server, as prevention layer	Partial Prevention
(Subbulakshmi et al., 2013)	Network monitors as prevention layers	Partial Prevention
(Lad et al., 2014)	DDoS Prevention Module between client and server	Partial Prevention
(Dhanapal & Nithyanandam, 2019)	A prevention layer in cloud, classifying request messages into different zones	Partial Prevention
(Jaber et al., 2018)	Swarm intelligence to be used in an Intrusion Prevention System (IPS) before the cloud server	Partial Prevention
(Saxena & Dey, 2015)	Third-party auditor acts as a shield before the victim cloud	Partial Prevention
(Sahi et al., 2017)	A third party, known as CS_DDoS, is a classifier method between cloud and the user	Partial Prevention
(Shridhar & Gautam, 2014)	The idea of a honeypot, as a prevention layer, between the client and server	Partial Prevention
(Navaz et al., 2013)	Cloud Service Provider (CSP) maintains a third party to monitor the network.	Partial Prevention
(Somasundaram & Meenakshi, 2021)	A 3-layer filtering mechanism between attacker and cloud	True Prevention
(Huong & Thanh, 2017)	An SDN gateway to filter network traffic	Partial Prevention
(Z. Ahmed et al., 2019)	SDN controllers use blockchain to share information about the attack	Partial Prevention
(A. K. Singh et al., 2020)	NFV and SDN to develop a defense model against DDoS attacks	Partial prevention

(Harikrishna & Amuthan, 2021)	Use of Self Organised Maps (SOMs) for accurate detection of DDoS attacks	Partial Prevention
(François et al., 2012)	IPSs form virtual protection rings around the hosts to defend.	Partial Prevention
(Z. Liu et al., 2018)	ISP-based service with three layers to defend against DDoS attacks	Partial Prevention
(Y. Kim et al., 2006)	The network keeps a score of packets	Partial Prevention
(Kalkan & Alagöz, 2016)	The network keeps the score of Packets	Partial Prevention
(Mirković et al., 2003)	Traffic monitoring by gateway routers	Partial Prevention
(Poongodi et al., 2019)	ReCAPTCHA controller, to prevent large botnets-based attacks	Partial Prevention
(Malhi & Batra, 2016)	Road Side Units (RSUs) as prevention layer	Partial Prevention
(Islam et al., 2018)	Both RSUs and controllers for prevention against DDoS attacks	Partial Prevention
(Grover & Mittal, 2016)	Both RSUs and controllers for prevention against DDoS attacks	Partial Prevention
(Timcenko, 2014)	Separate Intrusion Detection System (IDS) nodes for prevention	Partial Prevention
(Nagar et al., 2017)	Dedicated IPS nodes to detect and block the attack	Partial Prevention
(Kaushal & Sahni, 2016)	Attack prevention nodes to control the transmission of each node in Wireless Sensor Network (WSN)	Partial prevention
(Jingle & Rajsingh, 2014)	Intrusion Prevention Detection System (IPDS) nodes in the network	Partial Prevention
(Dao et al., 2018)	Multi-access edge computing controller (MAEC)	Partial Prevention
(Bhardwaj et al., 2018)	Edge computing in IoT	Partial Prevention

(Misra et al., 2011)	Middleware between application level and technological infrastructure to detect and prevent an attack	Partial Prevention
(X. Chen et al., 2021)	Packet sampling at multiple control points	Partial Prevention
(Dao et al., 2021)	Fog-shield, the endpoint defender and Orchestrator	Partial Prevention
(Wu et al., 2013)	Source validation	True Prevention
(H. Luo et al., 2017)	Dynamic PIDs (DPIDs)	True Prevention
(Al-Duwairi et al., 2020)	DPID with Get message logging	True Prevention
(X. Liu et al., 2008)	A new encrypted header field in the IP header	True Prevention
(Hu et al., 2017)	Sender's encrypted ID in the IPv6 extension header	True Prevention
(Nadeem et al., 2021)	Intermediate routers perform two levels of verification	True Prevention
(Andersen et al., 2008)	Accountability of Autonomous Systems (Ass)	True Prevention
(Pappas et al., 2016)	Forwarding accountability between Ass	True Prevention
(Y. Liu et al., 2015)	Encrypted network identities to nodes	True Prevention
(Park & Lee, 2001)	Allowed IP packets on a link	True Prevention
(J. Li et al., 2002)	Source periodically informing neighboring nodes	True Prevention
(Duan et al., 2018)	Route mutation to protect critical links from attack	True Prevention
(Keromytis et al., 2004)	Secure overlay points and secret servlets	True Prevention
(Osanaiye, 2015)	Operating System (OS) fingerprinting	True Prevention
(Goncalves et al., 2017)	Information about blacklisted nodes is propagated through wireless routers.	True Prevention

2.2.1 Ideal Prevention

An attacker generally carries out a chain of processes to launch an attack. Ideal Prevention techniques target and prevent such processes from being executed, preventing an attacker from launching an attack. Therefore, such techniques do not allow the attack traffic to enter the network. One of the prominent ways to launch a DDoS attack requires multiple hosts acting in a coordinated fashion. These attack hosts are called bots (or zombies); they belong to networks called botnets of compromised hosts and are remotely controlled by an attacker. The preventive approach proposed by (Freiling et al., 2005) identifies and infiltrates this remote-control network mechanism and aims at shutting it down. It is done by deploying honeypots that attract the traffic of malicious actions of the botnet maintainers and facilitates forensic analysis of this traffic. It helps in the detection of malware which is responsible for remote-control mechanism and prevent botnets from being activated.

Qaisar Shafi et al. prevent botnets creation in the Internet of Things (IoT) networks (Shafi & Basit, 2019). The proposed scheme uses SDN and blockchain to detect botnets among IoT networks in a distributed fashion. This detection is claimed to be automatic without the need for manual intervention. Thus, if an attacker is prevented from creating and controlling the botnets, DDoS attacks cannot be launched. Along similar lines, the technique proposed by (H. Luo et al., 2017) uses identifier/locator separation and thwarts DDoS attacks by preventing bot creation. As against the current Internet architecture, which uses only an IP address to represent both identity and location of a network node, the identifier/location separation technique separates the two. It was initially proposed to address the Internet’s routing scalability problem. Location/ID separation protocol (LISP) is one of many routing protocols which uses location/identifier separation. Using the modified DNS request/response mapping, a set of mapping servers for each provider network, and tunnel routers (TRs), the proposed technique makes it very difficult for attackers to find vulnerable hosts to act as zombies (or bots). As through this approach, TRs only provide locators of hosts that provide a service. For the hosts that do not provide any service, the TRs cannot find locators. Hence the attackers cannot find vulnerable hosts to create bots. It also creates a hindrance in sending attack commands to these zombies, even if somehow, they are created.

Although it seems feasible, identifying and infiltrating the botnets is always impossible. With many botnets present across the Internet, automating the infiltrating and analysis process is far from practical. Also, the attack is still possible through booter services (Booters, Stressers and DDoSers, 2021).

Ingress filtering is yet another effective way to implement Ideal Prevention. Ingress filtering is implemented at the ISP level, and it prevents DDoS attacks by not allowing spoofed addresses to access the network. It also helps in tracking the source of the attack. Attacks can be somewhat prevented if the traffic leaving an edge network and entering an ISP can be limited to the traffic it is legitimately sending. There are at least five ways to implement ingress filtering, with varying impacts (Baker & Savola, 2004).

- Ingress Access Lists
- Strict Reverse Path Forwarding (RPF)
- Feasible Path RPF
- Loose RPF
- Loose RPF ignoring default routes

Ingress Access Lists check every message's source address against a list of acceptable prefixes before the packet enters the network and drops the packet whose source IP is not in the list. Strict RPF is similar to access lists; the only difference is that the access list is dynamic. Forwarding Information Base (FIB) is used to check source addresses. The technique states that if the packet is forwarded on the same interface the packet is coming upon, that packet will be accepted. A better version than Strict RPF is Feasible Path RPF. In this, alternative paths are also looked upon; if any alternative path is matched, it is accepted; otherwise, the packet is dropped. The mechanisms in Feasible RPF need to be defined more clearly, like where it will work and where it will not. The packet is checked against FIB in Loose RPF mode and is dropped when it will not match any incoming interface. The term used for this is the existence of the path on that router. However, this is useless because, like Loose RPF sacrifices directionality, it loses the ability to limit an edge network's traffic to send legitimate traffic sourced from that network. The fifth technique, Loose RPF ignoring default routes, is like Loose RPF; the only difference is that the source check-list default routes are excluded.

All the methods mentioned above prevent DDoS attacks by preventing IP Spoofing and allowing acceptable prefixes of IP addresses to pass through the network interface. But these techniques have some issues. To further elaborate, two terms, multihoming and symmetrical routing, must be understood. Symmetrical routing is a routing protocol that forces the request and corresponding response packets to follow the same path. It is not feasible to enforce symmetrical routing for entire internet traffic. Strict RPF requires a path to be symmetrical; it cannot work for an asymmetrical path. Multihoming means making multiple connections in an organization for better reliability and connection to the Internet. It has two types -classical multihoming and multihoming with multiple addresses. In classical multihoming, the organization that uses multiple network providers for reliability has its range of IP addresses, which it will announce to all network providers. The organization has an address for each provider in multihoming, with multiple addresses. Ingress filtering does not work correctly for multihoming networks. As the range of addresses belonging to a particular network or source increases, the range of acceptable addresses becomes more complex and dynamic. It becomes harder for ingress filtering to deal with this, and there is always a risk that legitimate packets can be dropped if the access lists are not up to date.

To apply ingress filtering against multihoming networks, the ways are as follows-

- Applying an appropriate form of Loose RPF.
- Ensure that each ingress filter at each ISP is complete in terms of information about change and interfaces so that legitimate packets are not dropped.

Even if these changes are made, modifications will be needed in complex networks containing too many uplinks and peers. An approach where IP spoofing can be prevented and validation of source IP can be done without dropping the packets is needed.

2.2.2 Partial Prevention

This section reviews DDoS prevention techniques that initially may allow the attack traffic to reach the victim. Later, this traffic is detected with the help of an additional prevention layer between an attacker and the victim.

Once detected, the attack traffic is stopped from reaching the victim. Figure 2.2 provides a general architecture of the partial prevention layer in the network. The prevention layer can either be situated near the source of the attack, in the edge routers of the ISP, or near the victim of the attack. The purpose of the prevention layer is to detect and mitigate the attack as soon as possible. This prevention layer generally classifies the traffic as an attack or legitimate and blocks it. Due to some false positives from the classification, legitimate traffic can be deemed attack traffic and blocked.

Various authors have proposed adding this prevention layer at different points within the network and using different technologies. (Ndibwile et al., 2015) have proposed using decoy servers and bait servers as prevention layers to protect web servers from DDoS attacks. All the regular traffic first passes through the bait server. From this, authenticated traffic is allowed to pass to the actual web server, and the unauthenticated traffic passes to the decoy server, which uses an additional layer of authentication to remove the number of false positives.

The decoy and bait web servers use the same IP address, so the attacker does not know where the traffic is going.

(Zhuotao Liu et al., 2018) have proposed a concept of middle-policing between the client and server as a prevention layer. It can be deployed in the existing Internet architecture. This task is performed by Middle-Boxes (M-Boxes), which are placed between the victim and the network. These M-Boxes work as a prevention layer. Regardless of how sophisticated a DDoS attack is, as long as middle-police can effectively enforce victim-defined traffic control policies to forward victim-preferred traffic, the impact the DDoS attack imposes on a victim is minimized. (Subbulakshmi et al., 2013) have proposed using network monitors to detect and prevent attacks. First, non-spoofed IPs are detected using enhanced Support Vector Machine (SVM), and spoofed IPs are detected using hop-count filtering. After this, appropriate mitigation approaches, such as rate-limiting, dropping the packets, etc., are applied based on attack strength.

(Lad et al., 2014) proposed an approach for REST-based web services. The DDoS prevention module does the prevention against DDoS attacks. Every request packet from the client will go through this module to the server. The server tries to validate the client through a token; if it has a valid token, the request is checked for valid IP; otherwise, the registration module is called to generate the token. If the IP of the request packet is valid, it is accepted; otherwise, it is dropped.

Numerous web applications use cloud platforms for more flexibility, additional security, reduced operating cost, etc. These web applications hosted on cloud servers are also prone to DDoS attacks. Various techniques which filter out attack traffic are proposed to prevent DDoS attacks on cloud platforms. Often, these techniques can be considered a third-party prevention layer responsible for detecting and mitigating the attack traffic. The technique proposed by (Dhanapal & Nithyanandam, 2019) uses a prevention layer within the cloud infrastructure to safeguard web servers against internal and external slow Hypertext Transfer Protocol (HTTP) DDoS attacks. This prevention layer classifies client request messages into different zones. These zones are monitoring orange, red, and green state zones. The monitoring state zone is responsible for monitoring all the HTTP requests and, based on different heuristics, classifies them into the orange, green, or red zone. If the request is classified within the red zone, it is blocked. The prevention layer thus monitors and blocks the packets from reaching the web server after the attack has started.

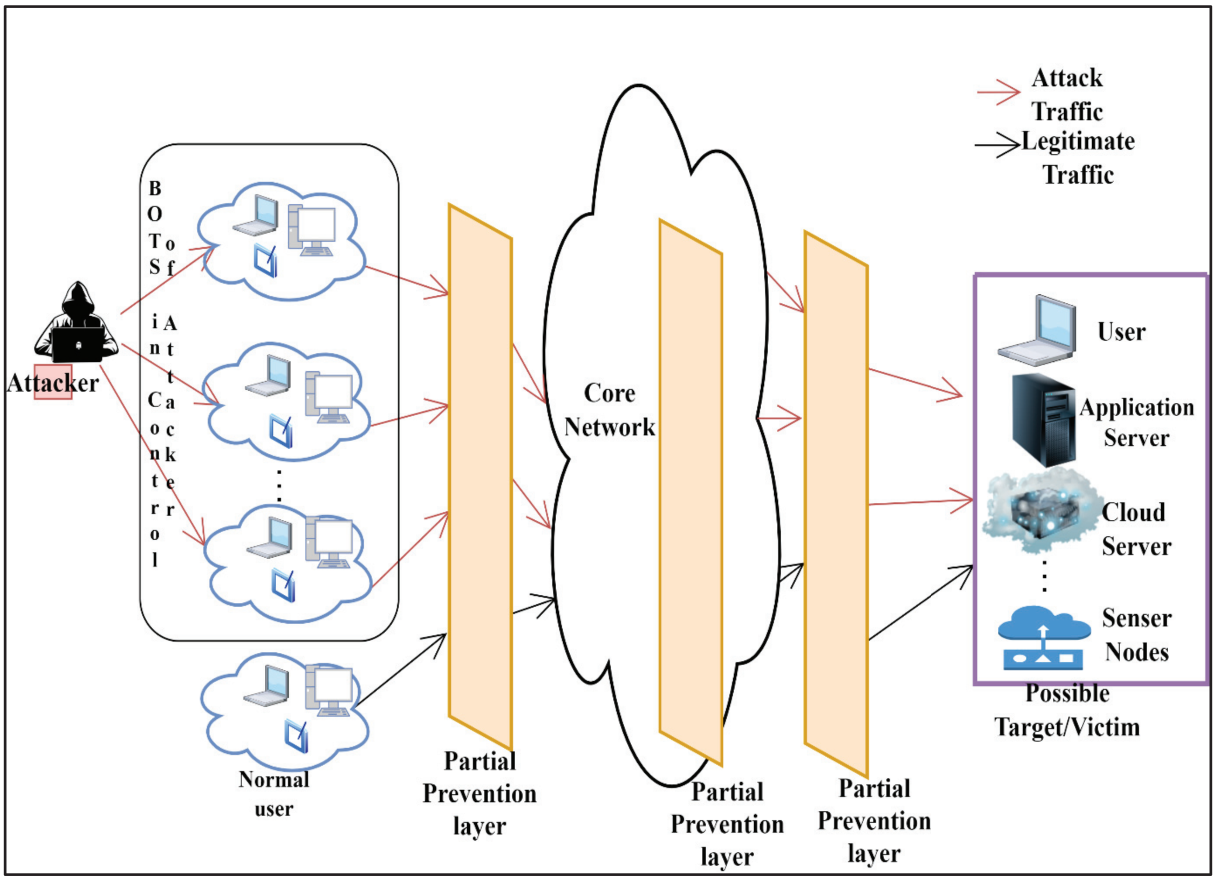


Figure 2.2 : Partial Prevention

(Jaber et al., 2018) proposed a host-based IPDS in the hypervisor before the cloud server. The packets are captured in this hypervisor before it reaches the cloud. Swarm intelligence and data analysis are used for better detection of DDoS attacks. There are two phases inside the hypervisor, i.e., IDS and IPS. IDS uses Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA); IPS uses the Artifical Neural Network (ANN) classifier to classify attack traffic and regular traffic. These are both used to filter the attack traffic. Using a Third-Party Auditor (TPA) between a victim cloud and an attacker is proposed by (Saxena & Dey, 2015). This TPA, known as Cloud-Shield, is responsible for detecting and preventing DDoS attacks. All legitimate and malicious packet information is logged into the Cloud-Shield on behalf of cloud servers. After analyzing all the packets, Cloud-Shield can trace back the attacks' source based on Dempster Shafer Theory (Siaterlis & Maglaris, 2003). (Sahi et al., 2017) have also proposed using a third party known as CS_DDoS, a classifier method, between cloud and the user. The detection phase stores the records of the incoming packets and classifies them as malicious or legitimate based on classification results. The malicious detected packets are stopped using the cloud service, and the source IP is blacklisted for further communication. The approach proposed by (Shridhar & Gautam, 2014) discusses the idea of a honeypot as a prevention layer between the client and server. The approach assumes that the attack packets should be known and forwarded to the honeypot, and the normal packets will go to the server. Honeypots should be created to resemble the original server for this to work, and attack packets should be known before the attack. In the approach proposed by (Navaz et al., 2013), entropy is used to detect the attack. A third party is maintained by CSP, which monitors the network and generates alerts for a user if an attack is detected. The router placed at the cloud server calculates entropy; if it is less than a certain threshold, it is considered an attack. The entropy is calculated using a port number, IP address, and flow size as inputs. A three-layer filtering layer between the attacker and cloud to prevent DDoS attacks is proposed by (Somasundaram & Meenakshi, 2021). The first layer filters out unauthenticated requests, the second layer prevents users from accessive use of resources, and the third layer removes spoofed requests finally only legitimate requests reach the cloud servers. So, with this only authenticated user can connect to the cloud and the malicious users also cannot demand for resources more than their limit.

Various authors have used SDN switches and controllers as a prevention layer to prevent the attack traffic from reaching the victim during an attack. (Huong & Thanh, 2017) proposed an SDN-based gateway to protect application servers against DDoS attacks. This gateway filters the network traffic to a security analyzer that sends traffic indicator results to the SDN controller. This security analyzer is capable of detecting attacks on the fly. The controller then decides whether the indicator results correspond to an attack. In case of an attack, appropriate flow entries are pushed in the SDN gateway to drop corresponding single packet flows. This way, the proposed technique claims to save the application servers against DDoS attacks. (Z. Ahmed et al., 2019) have used blockchain and SDN to prevent the victim from DDoS attacks. The SDN controller oversees the detection and mitigation of the attack. All the communication takes place through the Ethereum blockchain. Based on a threshold calculated from incoming packets, the controller decides if a source is legitimate or an attacker. The controller shares this information with other controllers using blockchain. (A. K. Singh et al., 2020) have used the functionality of Network Function Virtualization (NFV) and SDN to develop a defense model against DDoS attacks. NFV helps to manage network functions on-demand, and SDN for redirecting the flows and managing the rules. The proposed technique monitors the network traffic periodically. Due to an attack when the server is overloaded with traffic, its traffic is directed to another Virtual Machine (VM) which analyses the attack traffic to know the source of the attack. Also, the users connected to that server are connected to a different server whose IP addresses are also spoofed. (Harikrishna & Amuthan, 2021) have proposed a partial prevention scheme for SDN-based clouds. A prevention layer in the form of a Virtual Router Firewall (VRF), which connects the internet with the SDN-based clouds' physical infrastructure, is proposed. It has the benefit of rival penalized SOMs and constant learning rate which results in rapid and accurate detection of DDoS attacks. It categorizes data flows into normal or malicious based on weights of neuron whose weight vector is very close to euclidian distance of considered IP vector (deviation in actual data traffic from expected traffic).

A prevention layer can also be added within the ISP's network. (François et al., 2012) have proposed creating a prevention layer in an IPS at the ISP level. The IPSs form virtual protection rings around the hosts to defend and collaborate by exchanging selected traffic information. This service is provided as close to the attack source as possible and as far as possible from the victim. Belief scores on potential attacks are computed and shared among networks to detect DDoS attacks.

Similarly, (Z. Liu et al., 2018) proposed an ISP-based service to defend against DDoS attacks on the victim side. This defense architecture has three layers. First is the flood throttling layer- which handles large application-layer attacks such as Network Time Protocol (NTP). This layer minimizes the effect of amplification flooding attacks, which exploits specific network service protocols for launching an attack. It uses a weighted fair queuing technique for the same. The second layer is the congestion-resolving layer, and it is used to stop more subtle and sophisticated attacks. It also punishes attackers by blocking them. Therefore, users who overlook packet losses and continuously inject packets are held accountable for the enduring congestion during DDoS attacks. The third layer is user-specific, and the goal of adding this user-specific defense layer is to provide the flexibility for the victim to enforce self-interested traffic control policies that are most suitable for their business logic. (Y. Kim et al., 2006) (Kalkan & Alagöz, 2016) have proposed using a scoring mechanism of packets to differentiate between regular and attack packets. The approach proposed by (Kalkan & Alagöz, 2016) is collaborative and proactive. According to the authors, "attack prevention efficiency measures how early the network can get rid of the attack packets." It generates a score for network packets, called PacketScore, to distinguish between attack packets and legitimate ones. The following attributes are considered for scoring the packets: IP address, port number, protocol type, packet size, Time To Live (TTL) value, and TCP flag. This packet-based analysis starts when congestion is detected within the network. Finally, upon detecting the attack packets, the network drops these, thus preventing the attack. (Y. Kim et al., 2006) have given an approach in which a packet score is kept for each packet, and a packet is called a legitimate packet if its value is below a threshold. Otherwise, the packets are blocked. The attributes used for scoring are packet size, TTL values, protocol-type values, source IP prefixes, TCP flags, and server port numbers. This approach can't filter low-volume traffic, and the attacker can mimic the characteristics of a legitimate packet. It is partial Prevention.

D-WARD, a DDoS defense mechanism proposed by (Mirković et al., 2003), is deployed at source-end networks and automatically detects and stops attacks originating from it. The gateway router between the source network and the rest of the Internet monitors the behavior of each peer with which the hosts communicate. Periodically, the traffic statistics are compared to that of regular traffic. In case of substantial deviation crossing certain pre-defined thresholds, the traffic is rate limited in proportion to their aggressiveness.

In addition to the Internet architecture, Partial Prevention techniques are also proposed for different types of networks, i.e., VANETS, WSNs, wireless mesh networks, 5G, and IoT networks. VANETS, a subclass of MANETS, has two types of communication nodes (i.e., Vehicles and Road Side Units (RSU's) and three types of communication links among nodes, i.e., Vehicles to Vehicles, Vehicles to RSUs, and RSUs to RSUs. A DDoS prevention layer is generally added within RSUs or as an additional controller operating between the communicating nodes. (Poongodi et al., 2019) proposed an additional controller, called reCAPTCHA controller, to prevent large botnets-based attacks. This reCAPTCHA controller filters specific IP addresses or ports through source integrity checks. It uses a challenge-response mechanism to calculate the entropy associated with the data from the covariance matrix. If it crosses the threshold, the node is blocked. (Malhi & Batra, 2016) proposed that the framework for DDoS Prevention uses a prevention module within RSUs as a prevention layer. These RSUs periodically calculate the fitness of every connecting node using a genetic algorithm. The fittest nodes are allowed to communicate, and the worse nodes are discarded by ending their communication with other nodes. (Islam et al., 2018) and (Grover & Mittal, 2016) have proposed using both RSUs and controllers for prevention against DDoS attacks. (Islam et al., 2018) leverage edge-computing, SDN, and NFV for their prevention architecture. RSUs are extended with micro-boxes to act as security gateways. They capture the traffic and provide analysis capabilities to the controller, which is located in the cloud. This controller is responsible for identifying threats and controlling all the micro-boxes. (Grover & Mittal, 2016) have proposed grouping nodes to detect the attack. RSUs first monitor and make a group of nodes based on a few select parameters. Once a group is formed, a group leader is selected, who will work as a group controller. This controller will now help to detect and block the malicious node. (Timcenko, 2014) have proposed prevention in MANETS using IDS nodes. These nodes, different from the communicating nodes, manage the MANET infrastructure. An attack profile is created based on the forensic analysis of log files of packets in the detection phase. In the prevention phase, all nodes are updated about the attacker's profile, and attackers are blacklisted.

In a WSN, sensor nodes are deployed in primarily hostile areas to monitor various network parameters for inspecting the traffic. (Nagar et al., 2017) proposed a methodology to detect and block the malicious nodes in a WSN to prevent them from generating DDoS attack traffic. Dedicated IPS nodes work as the third-party layer and scan the neighbors regularly, detect and find a node involved in frequent message passing, block it, and inform the honest nodes in its vicinity to change the routes. (Kaushal & Sahni, 2016) proposed an approach for preventing DDoS flooding attacks in the WSN. The methodology includes limiting the number of transmissions of each node. This limit is based on the number of neighboring nodes (transmission is hop by hop). Each node will decide and tell its neighbor count to an examiner node, set a threshold based on that, and work as an attack-prevention node. If any node sends more packets than a threshold, it is compared with the packet delivery ratio of the neighboring nodes. An attacker involved in the attack will have an abnormal packet delivery ratio. Hence it will be detected. But normal packets can also suffer due to a limit in transmission. Similarly, (Jingle & Rajsingh, 2014) proposed a DDoS detection and mitigation technique for wireless mesh networks by proposing an IPDS. These IPDS nodes work collaboratively at various strategic points within the network. This IPDS system admits a node and keeps track of the traffic generated by it to detect any IP spoofed flooding attack in a mesh network. There are four components of the IPDS in this technique. First is the admission controller responsible for bandwidth allocation. The second is the traffic analyzer, which consists of a timer monitor and bandwidth monitor, and finally, the mitigation manager is responsible for attack mitigation. A single IPDS cannot handle all nodes; hence, the use of multiple IPDS is promoted. In all these techniques an IPS node or controller is used as a prevention layer.

The technique proposed by (Dao et al., 2018) proposes a MAEC controller to prevent attacks. It helps in the detection of attacks in an early stage. MAEC is an emerging 5-G technology. It is helpful against DDoS attacks as it is implemented near clients. Whenever the first edge nodes detect malicious activity, they inform the central MAEC-X controller, generating policies for the edge nodes to handle this suspicious traffic, but before this, some percentage of the attack reaches the victim.

DDoS attacks also threaten IoT infrastructure due to their open nature. (Bhardwaj et al., 2018) have proposed Shadownet, a prevention layer to detect DDoS attacks in IoT infrastructure. It uses edge computing and a shadow net web service. Edge computing works as the first line of defense, where edge functions will sketch IoT traffic profiles. After this, it sends shadow packets to the shadow web service. Shadow packets contain locally-driven info about the IoT traffic. The attack detection and appropriate mitigation actions will be decided at the shadow web service. (Misra et al., 2011) have proposed a service oriented architecture, which works as a prevention layer against DDoS attacks in the form of middleware between the application level and technological infrastructure. The detection and prevention phases use learning automata to optimize the utilization of all its resources. In the detection phase, a threshold value for each layer is defined based on the resources available at each layer. A DDoS attack is detected if a request for these resources at any layer exceeds the threshold limit. In the defense phase, packets from an identified attacker are discarded. (X. Chen et al., 2021) also propose a partial prevention scheme with multiple control points to defend against IoT-based DDoS attacks. Packet sampling is done at these control points, and based on that, the attacker's and defender's costs for attack will be calculated, and control strategies will be made to stop the attack. To secure heterogeneous IoT, (Dao et al., 2021) proposed a prevention layer in the form of a fog-shield endpoint defender and orchestrator. Multiple endpoint defenders are located at the border of each homogeneous IoT system. These endpoint defenders contain SOMs to classify traffic, and they send a report to the centralized orchestrator. The main task of the orchestrator is to cooperate with training results and policies among endpoint defenders. It is also responsible for analyzing and generating a policy for endpoint defenders.

2.2.3 True Prevention

Based on the literature, True Prevention can be categorized into techniques involving Source Validation, PIDs, Route Mutation, Route-based Packet Filtering, Secure Overlay

Points, Encryption, and OS Fingerprinting, as shown in Figure 2.3. Validating the source IP address is one of the promising ways to prevent DDoS attacks. Source Address Validation Improvement (SAVI) (Wu et al., 2013) is a promising way to implement this in the Internet architecture. Source validation protects from IP Spoofing, thus preventing DRDoS attacks. It is purely a network-based technique and has no dependency on end hosts. Following is a three-step model followed by SAVI instances:

- Using specialized packets (monitoring packets), legitimate and valid source IP addresses are identified for a host.
- Bind the legitimate source IP address and a link-layer property of the host's network attachment. The link-layer property is chosen such that this binding, called a binding anchor, is harder to spoof and is verifiable in every packet. The binding anchor varies from IEEE unique identification, security association between host and base station for wireless links, or Ethernet port of a switch to which the host attaches (Wu et al., 2013)
- Validate the source IP using the binding anchor.

The SAVI instances should be positioned as close as possible to the host. Ideally, it should be located in the host's default router. The issues with SAVI instances may be that they can face reliability issues due to the loss of bindings in SAVI devices through the restart of SAVI devices or binding information for a new link not reaching the SAVI device. If the SAVI device's physical location is known, the attacker can change or surpass it by connecting through a non-SAVI device. SAVI devices can also be attacked, e.g., a DDoS attack on them. Several SAVI documents have been standardized based on the different address assignment techniques e.g., First Come Fist Served (FCFS) SAVI (Nordmark et al., 2012), Dynamic Host Configuration Protocol (DHCP) SAVI (Bi et al., 2015), Secure Neighbor Discovery (SEND)

SAVI (Bagnulo & Garcia-Martinez, 2014), and Mixed Address Assignment Methods (MAAM) SAVI (Bi et al., 2017).

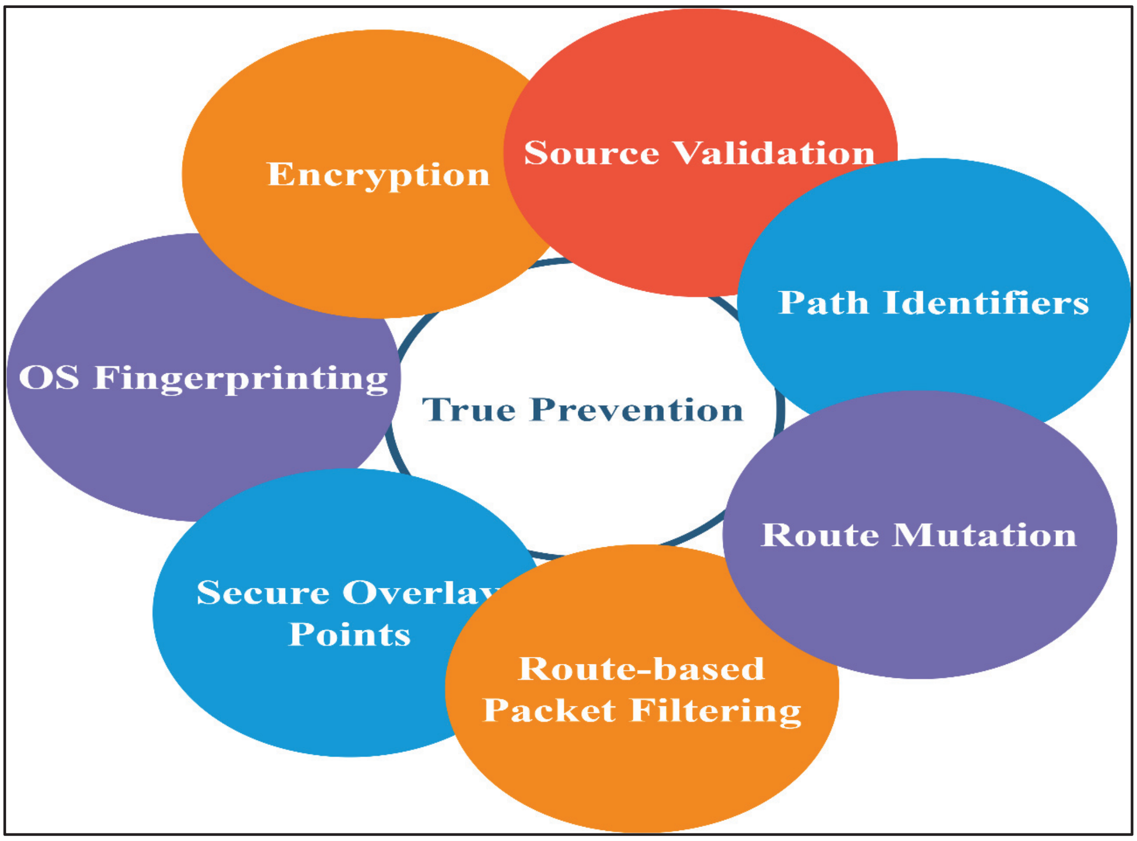


Figure 2.3 : True Prevention

Jessica et al. (Goncalves et al., 2017) have proposed a Wireless Distributed IPS (WIDIP), and its main focus is to protect the internal network from attacks. The attackers are identified and blacklisted in this work using IP and Media Access Control (MAC) addresses. If there is more than one IP address for a MAC, that address is blacklisted, and this information is propagated to other wireless routers. Hence, the attacker is detected at the source and thus prevented from making the attack. The problem with this technique is that innocent machines controlled by bots are also blacklisted for a very long time and won't avail of any service as they are blacklisted.

Instead of using IP addresses as routing parameters, PIDs are used in the content-centric network to enforce prevention. PIDs are identifiers that identify the path between network entities as inter-domain routing objects. (H. Luo et al., 2017) have proposed prevention against DDoS attacks with DPIDs. In DPIDs, two adjacent domains periodically update and install new PIDs into the data plane for packet forwarding. Even if the attacker obtains the PIDs to its target and sends the malicious packets successfully, they will become invalid after a particular configurable time. Therefore, the network will discard the subsequent attack packets. (Al-Duwairi et al., 2020) have also used the concept of DPID with Get-message logging. This approach is mainly for ICNs where users request information using Get messaging. The reason for logging Get-messages for DDoS attack detection is that normal users correspond to a Get message while an attacker does not. Here, the ICN routers log Get-request messages using bloom filters. Bloom filters will help in comprehensive logging; they don't even take up much space. This approach is claimed to give better results against DDoS prevention than DPID. The limitations of this technique are modification of underlying Internet architecture is required for its implementation, and if a link breaks between the communication, the response packet is lost.

One way to provide prevention is by authenticating the source IP address using different encryption techniques. Many authors have used encrypting the packet header to authenticate the source address. (X. Liu et al., 2008) proposed a new field (captioned as Passport) with an IP header in their proposed approach. When the packet leaves its originating AS, the border/egress router will attach message authentication code to the passport header of the packet. A secret key is shared in advance between source AS and each AS between the source and destination. With the help of this message authentication code, the inter-between ASs can verify whether the incoming packet belongs to that particular source address. The limitation of this technique is that Diffie-Hellman is used for exchanging the keys, which in itself is not secure. (Hu et al., 2017) proposed to store the sender's undeniable and reliable identity using the IPv6 extension header. A slight change in legacy networks is proposed by implementing this technique in two steps. For inter-domain accountability, SAVI is used, and before the packet leaves the network, the gateway router/switch embeds users' credibility information in the packet. This L3 switch is named a SuperFlow switch, which can be an SDN/OpenFlow; hence, different granularities of the flow can be controlled. The user's private keys will be stored in the extension header of the IPv6 protocol. The public keys of users will be stored in a Public Key Exchange Server (PKES). The receiver can verify the sender's signature through its local PKES. The limitation in this technique is SAVI devices and also PKES on which Denial of Service (DoS) attacks can be made. User identification as a mode of verification for the cloud is also proposed by (Nadeem et al., 2021). The authors propose intermediate routers do attack prevention by using two levels of verification. The first is a user ID and password; the second consists of port numbers assigned to different countries. Routers work as Network IDS (NIDS) and Host IDS (HIDS) to prevent attackers from generating the attack. If somehow the attacker comes in possession of a username and password, the attack can be launched. This technique also causes an accessive delay for a legitimate user to access the cloud. The Accountable Internet Protocol (AIP) proposed by (Andersen et al., 2008) provides self-certify addresses without depending on a third party. This technique proposes Accountability Domains (ADs) (like ASs), and each host in that AD is given a unique End-point Identifier (EID) for authentication so that each host will have a combination of AD: EID. The problem with this technique is the deployability and refurbishment of Internet protocol. (Pappas et al., 2016) proposed the approach, named Forwarding Accountability, for Internet reputability. It works by incentivizing ISPs and ASs. The idea is forwarding accountability, in which the receiver in the communication decides security policies that the sender must follow. The inter-between ASs are also responsible for sending the traffic as they mark the packets that pass through them. These transit ASs insert cryptographic markings. The limitation of this technique is open recursive resolvers like that in DNS amplification attacks. A new IPv6 address generation algorithm is proposed by (Y. Liu et al., 2015). The basis of this algorithm is time and Network Identity (NID). The authenticity of source addresses is achieved with the help of Source Address Validation Architecture (SAVA) . Still, dependency on SAVI devices can be a hurdle because a DOS attack can be made on SAVI devices. The proposed approach has three steps after SAVI. First, a scalable structure of NID is designed. After this, the second step involves the address generation algorithm for IPv6. The concatenation of NID and time is encrypted using the International Data Encryption Algorithm (IDEA) algorithm (Daemen et al., 1994), (Lai & Massey, 1991), generating an address assigned to the host. The final step is the implementation of the algorithm and traceback.

In the approach proposed by (Park & Lee, 2001) and (J. Li et al., 2002), the packets allowed on a particular link are controlled. It is Route-based Packet Filtering. The approach Route-based Distributed Packet Filtering (RDPF) by (Park & Lee, 2001) ensures True Prevention, followed by traceback. Prevention is based on a pre-defined range of IP packets allowed on a link. This range is based on the source-destination IP address pair. The packets on a particular link are allowed if they belong to the allowed pair. The corresponding router of that AS link will block the packet for every other packet. This technique can't prevent intelligently spoofed IP addresses and doesn't support dynamic changes in the topologies. If Border Gateway Protocol (BGP) is used for configuration, then hijacking in a BGP session can mislead routers. In the method proposed by (J. Li et al., 2002), called Source Address Validity Enforcement (SAVE), a dynamic routing problem is resolved. In this proposed approach, the source address periodically informs about itself by sending messages to all destination nodes and solves the RDPF dynamic routing problem. Each router will know valid IP addresses that can arrive on it through this approach. Routers receive valid addresses from incoming tables (sent previously). Hence, it prevents attacks from invalid IP addresses. Still, this technique cannot stop attacks from valid IP addresses and causes an increase in memory and computational costs in routers.

Focusing mainly on DDoS attacks that exploit network infrastructure design, (Duan et al., 2018) proposed a proactive Routing Mutation technique. Analysis reveals that many public servers have only a few critical links, and congesting these critical links can cause 60-90% service degradation in most cases (Duan et al., 2018). Once such links are known, the attacker selects links with the highest flow density to the victim servers as target critical links. The proposed approach includes developing a susceptibility metric to quantify the potential bandwidth degradation ratio and severity level of attack. Based on these metrics, the proposed solution provides proactive route mutation to reduce susceptibility and improve the availability and resilience of critical servers. This methodology targets a victim directly, not links, so even if the flow is switched to non-critical links, it won't matter as the victim will still be affected. (Keromytis et al., 2004) proposed an excellent method for preventing DOS attacks, known as Secure Overlay Services (SOSs). Authentication of the packets is done at secure overlay points, which forward the packets through overlay nodes to beacon nodes. Beacon nodes forward these packets to a secret servlet (remote node). These secret servlets send the packets to destination nodes. Scalability is a big issue with this proposed solution because the state for each target must be maintained at the secret servlets, beacons, and access points. (Osanaiye, 2015) proposed OS Fingerprinting for DDoS prevention against IP spoofing. It is implemented to detect spoofed packets and not forward them. The detection scheme is based on matching the OS of the spoofed IP packet with the OS of the trustworthy Source. The fingerprinting can be passive or active. It only works if the spoofed source and actual source have different OSs and have extra communication overhead to check for OS.

2.3 Limitations of Prevention Techniques

The benefits and limitations of all the prevention techniques, i.e., Ideal, Partial, and True Prevention, are listed in Table 2.2

Table 2.2: Advantages and limitations of existing prevention techniques

Author	**Benefits**	**Limitations**
(Freiling et al., 2005)	Prevents activation of botnets by infiltrating the remote-control network mechanism.	DDoS attacks can still happen with booter services (*Booters, Stressers and DDoSers*, 2021).
(Shafi & Basit, 2019)	The attacker is prevented from creating and controlling the botnets; therefore, DDoS attacks cannot be launched.	DDoS attacks can still happen with booter services. (*Booters, Stressers and DDoSers*, 2021).
(H. Luo et al., 2013)	This technique makes it difficult for attackers to find vulnerable hosts to create bots by separating identity from a location.	DDoS attacks can still happen with booter services. (*Booters, Stressers and DDoSers*, 2021).
(Baker & Savola, 2004)	Ingress filtering stops DDoS attacks by not allowing spoof packets to enter the network; hence the victim will remain unaffected.	• Problematic to implement in complex networks containing too many uplinks and peers. • The ingress filter at each ISP should be complete in terms of information about change and interfaces so that legitimate packets are not dropped.
(Ndibwile et al., 2015)	Bait servers and decoy servers are used before the actual web servers so that only the validated traffic reaches the actual web servers.	• Due to false positives, legitimate traffic also gets blocked • The new type of DDoS attacks, whose characteristics are unknown, will be hard to stop.
(Zhuotao Liu et al., 2018)	Middle-police allows only victim-preferred traffic to pass, and these M-boxes can be implemented in the existing Internet infrastructure.	Victim-preferred traffic, e.g., DNS amplification attacks, can reach the victim.
(Subbulakshmi et al., 2013)	Network monitors detect the attack traffic in the network itself so it cannot reach the victim.	• Due to false positives (from SVM), legitimate traffic also gets blocked

		• Legitimate packets may use asymmetrical paths, which result in different hop counts causing false detection.
(Lad et al., 2014)	A DDoS prevention token only allows packets with valid tokens to reach the server.	If the prevention model's bandwidth is choked from all the packets needing validation, then legitimate packets will be dropped.
(Dhanapal & Nithyanandam, 2019)	The monitoring zones block the attack traffic from reaching the server. Four different zones are used for better classification.	Attacks that have no signature or mimic legitimate traffic will be hard to stop.
(Jaber et al., 2018)	The prevention module detects and stops the DDoS attack from reaching the server. For detection, it uses swarm intelligence.	• The DDoS attack can be made on the prevention module. Because of this, legitimate traffic will also suffer.
(Saxena & Dey, 2015)	Cloud-Shield logs all packets, and after analyzing them, attack traffic is prevented from reaching the cloud servers.	• All packets pass through the Cloud-Shield, the delay introduced due to this is not shown. • This approach can be classified as Partial Prevention, hence some percentage of attack traffic can reach the victim before it is detected. The exact percentage of this occurrence has not been quantified.
(Sahi et al., 2017)	A third party classifies and blocks the attack packets from cloud servers. The IP address of attackers is also blocked.	Detection is done using classification, which can result in blocking legitimate IP addresses due to false positives.
(Shridhar & Gautam, 2014)	The honeypot used between the client and server will act as a decoy, and all the attack packets will go toward the honeypot, and the server will remain unaffected.	The approach assumes that the attack packets should be known and forwarded to the honeypot.
(Navaz et al., 2013)	Using entropy, a third party detects the attack and prevents it from reaching the server.	The DDoS attack can reach the victim if the threshold for detection is not

		crossed, for example, low-rate DDoS attacks.
(Somasundaram & Meenakshi, 2021).	A 3-layer filtering mechanism between attacker and cloud, allowing only legitimate requests to reach the cloud.	• Legitimate users can also suffer because of the number of requests allowed to access the server. • Low-rate DDoS attacks are still possible.
(Huong & Thanh, 2017)	A security analyzer detects the attack on the fly and provides a fast response for attack mitigation.	• The security analyzer cannot detect low-rate DDoS attacks, resulting in no attack detection. • To detect different DDoS attacks, the number of parameters used for attack detection is not sufficient.
(Z. Ahmed et al., 2019)	Multiple controllers communicate through blockchain to provide attack information to all networks to stop the attack from reaching the victim.	Due to delays in information exchange among controllers, some parts of attack traffic can reach the victim.
(A. K. Singh et al., 2020)	After an attack happens and the server is overloaded, all users connected to it are transferred to another server.	With the increase in the number of users and servers, the processing delay in removing users and changing their IP addresses increases, causing a delay in preventing DDoS attacks.
(Harikrishna & Amuthan, 2021)	Use of SOMs for accurate detection of DDoS attacks.	• The delay caused in normal traffic by applying ML techniques on routers. • The percentage of regular traffic reaching the cloud server is also uncertain.
(François et al., 2012)	Virtual protection rings are created near the attack source and as far as possible from the victim.	• Some part of the attack reaches the victim before these rings detect the attack, and attack information is shared among these rings. This percentage of attack traffic reaching victims is not shown. • Due to false positives, legitimated IPs can be blacklisted and blocked, and no approach to unblock these IPs is proposed.
(Z. Liu et al., 2018)	It punishes the attackers responsible for the congestion by blocking them.	Delay is introduced for regular traffic to reach the victim because of three

		prevention layers placed by ISP in the network.
(Y. Kim et al., 2006)	A scoring mechanism to differentiate between attack packets and normal traffic	• This approach can't filter low-volume traffics, and the attacker can mimic the characteristics of a legitimate packet. • Attributes to decide the score of packets can take up too much storage.
(Kalkan & Alagöz, 2016)	A scoring mechanism to differentiate between attack packets and normal traffic	• This approach can't filter low-volume traffics, and the attacker can mimic the characteristics of a legitimate packet. • Hard to decide the correct attribute pairs to be used for scoring to detect an ongoing attack.
(Mirković et al., 2003)	The proposed technique is applied near the source of the attack to be prevented at the source only.	• The authors themselves say that detection can be unreliable, hence proposed a rate-limiting mitigation technique, but due to this, legitimate traffic also suffers, as that is prone to rate-limiting also.
(Poongodi et al., 2019)	Its reCAPTCHA controller is capable of detecting low-rate DDoS attacks also.	Normal traffic frequency and entropy will also be checked by the reCAPTCHA controller, causing unnecessary delay.
(Malhi & Batra, 2016)	The detection accuracy of attack packets increased due to the use of a genetic algorithm.	Increased processing overhead on RSUs to implement a genetic framework for the detection of DDoS attacks.
(Islam et al., 2018)	Both controllers and RSUs are used for the prevention	The DDoS attack can be made on the controller located in the cloud.
(Grover & Mittal, 2016)	Both controllers and RSUs are used for the prevention	The DDoS attack can be made on the group leader, which is responsible for controlling other nodes.
(Timcenko, 2014)	Separate IDS nodes are created to prevent attacks so that overhead on normal nodes can be avoided.	Legitimate nodes used by attackers for spoofing attacks will also be blacklisted.
(Nagar et al., 2017)	Dedicated IPS nodes for continuous scanning of the network to detect attacks	The DDoS attack can be made on the IPS nodes so that attacks on WSN nodes cannot be prevented.

(Kaushal & Sahni, 2016)	Examiner nodes detect the attacker based on the information provided by WSN nodes. These sensor nodes inform about their neighbors and the amount of traffic they should receive.	Normal packets can also suffer due to a limit in transmission by the nodes.
(Jingle & Rajsingh, 2014)	Various IPDS nodes located strategically work collaboratively to detect attacks.	Communication overhead in the network due to collaborative information sharing among IPDS nodes.
(Dao et al., 2018)	MAEC is implemented very near to clients to detect DDoS attacks.	The DDoS attack can be made on MAEC-X central controller.
(Bhardwaj et al., 2018)	Edge computing is used as the first line of defense, leading to attack detection as near the source as possible.	Deployment of Shadownet in multiple networks for effective detection compromises the fast-path assumptions and is not feasible.
(Misra et al., 2011)	Learning automata is used for intelligent sampling of packets to categorize them into attack or legitimate ones.	This technique does not guarantee that all the attack packets can be dismissed.
(X. Chen et al., 2021)	Packet sampling at multiple control points	For mitigation and control strategies to be implemented, classifying what bots and normal users are, is unclear.
(Dao et al., 2021)	Multiple endpoint defenders classify traffic, and centralized endpoint defenders analyze this traffic and generate policies.	• The channel between the endpoint and the centralized orchestrator must be secure; otherwise, detection might not be possible. • Extra functionalities for attack detection must be added at each homogeneous system's border.
(Wu et al., 2013)	• Detects attack as near to the source as possible. • Entirely network-based, with no dependency on end hosts.	• DDoS attacks on SAVI devices can be made. • Can have reliability issues due to the loss of bindings in SAVI devices
(H. Luo et al., 2017)	• DPIDs are introduced that can prevent flooding attacks also. • Attack packets cannot reach the victim.	• It is limited to ICNs. • The DPID sharing and updation time can be improved.

(Al-Duwairi et al., 2020)	• Bloom filters are used for the effective logging of GET messages. • Attack packets cannot reach the victim.	It only works for ICNs where a Get request is used.
(X. Liu et al., 2008)	Using encryption inter-between ASs can verify that the packet belongs to a source and prevents the packet from reaching the victim.	Diffie-Hellman is used for key exchange, which itself is not secure.
(Hu et al., 2017)	The sender's identity is stored inside the IPv6 extension header only.	• Separate PKES needs to be maintained • DOS attacks can be made on PKES and SAVI devices
(Nadeem et al., 2021)	Intermediate routers perform authentication before the cloud can be accessed.	• Internal users can launch an attack on the cloud • Accessive delay is caused for a legitimate user accessing the cloud.
(Andersen et al., 2008)	This technique uses self-certifying addresses without any dependency on a third party.	Deployability and refurbishment of IP
(Pappas et al., 2016)	ISPs and ASs are incentivized so that security policies can be enforced to prevent DDoS attacks.	The attack can happen from open recursive resolvers like that in DNS amplification attacks.
(Y. Liu et al., 2015)	To prevent spoofing, the user is identified with two more identities, NID and time identity.	Dependency on SAVI devices can be a hurdle because a DOS attack can be made on SAVI devices.
(Park & Lee, 2001)	The router of a link blocks the packet if that packet's source-destination IP pair doesn't belong on that link.	It doesn't support dynamic changes in the topologies.
(J. Li et al., 2002)	The source address periodically shares information with neighbors to self-validate.	• This technique can't stop the attack from valid IP addresses. • Increase in memory and computational costs in routers.

(Duan et al., 2018)	Critical links are identified proactively, and attack is prevented by route mutation.	• For attacks targeting a victim directly, not links, even if the flow is switched to non-critical links, it won't matter as the victim will be affected.
(Keromytis et al., 2004)	With the help of secret servlets, the attack is prevented from reaching the victim.	Scalability is a big issue with this proposed solution because the state for each target must be maintained at the secret servlets, beacons, and access points.
(Osanaiye, 2015)	Prevent IP spoofing of packets.	• It only works if the spoofed and actual sources have different OSs. • Extra communication overhead to check every time for OS.
(Goncalves et al., 2017)	The attacker is detected at the source and prevented from making the attack.	• Innocent machines controlled by bots are also blacklisted for a very long time and won't be able to avail of any service as they are blacklisted.

2.3.1 Research Gaps

Based on the literature survey following are the identified research gaps which require further research:

a) The dependability of applying prevention techniques should be minimized for the end-users. Ideally, the end-users should have no role in DDoS prevention. The underlying forwarding network should be made self-sufficient. It should not allow attack traffic to be generated; if attack traffic generation cannot be stopped, it should not allow this traffic to reach the victim.

b) The ideal technique for prevention is Ideal Prevention. The primary issues with achieving Ideal Prevention are-

i) It includes methods that try to stop all well-known signature-based and broadcast-based DDoS attacks from being launched in the first place, or edge routers keep all the machines over the Internet updated with patches and fix security holes. But with this, networks are always vulnerable to attack types for which signatures and patches do not exist in the database.

ii) Ideal Prevention includes methods to prevent botnet creation and causing hindrance in remote-control mechanisms. But DDoS attacks are always possible from DDoS for hire services (*Booters, Stressers and DDoSers*, 2021).

iii) Ingress filtering a source validation scheme is difficult to apply for multihomed networks. To apply ingress filtering against multihoming networks, the ways are-

- Applying an appropriate form of Loose RPF.
- Ensure that each ingress filter at each ISP is complete in terms of information about change and interfaces so that legitimate packets are not dropped.

Even if these changes are made, modifications will be needed in complex networks containing too many uplinks and peers.

c) Partial Prevention defines a prevention layer between the victim and the attacker, preventing the victim from attack traffic.

i) This layer collects traffic statistics from the network and classifies the traffic as legitimate or attack based on some parameters. Due to some false positives from the classification, legitimate traffic can be deemed attack traffic and blocked.

ii) In some techniques, the prevention layer itself is vulnerable to DDoS attacks.

The limitations found in individual techniques are listed in Table 2.2.

********* End of Chapter *********

3.1 Introduction

To provide proactive defense against DRDoS attacks, the prevention techniques proposed in this research work use some existing paradigms and techniques. To better understand all the proposed prevention techniques, this chapter describes all behind-the-scenes techniques. As mentioned in Chapter 1, we will implement our strategies using SDN. So, first, we give a detailed description of SDN. After this, we describe the base techniques used in our prevention approaches, i.e., IBC, random forest classifier, and symmetrical routing.

But before providing defense against DDoS attacks, we first need to understand DDoS attacks. What are the different types of DDoS attacks, and how do they work? So, before dwelling on prevention, let us first understand the types of DDoS attacks.

3.2 Classification of DDoS Attacks

Different taxonomies for classifying DDoS attacks have been proposed in the literature, which helps to understand the problem and its solution space. Among the various ways, one way to classify DDoS attacks is the process through which the attack is generated. Accordingly, a taxonomy is proposed in section 3.2.1.

3.2.1 Types of DDoS Attacks

Different authors in the literature have proposed different classifications/types of DDoS attacks. As shown in Table 3.1, the categorizations are as follows-

- A two-dimensional view of DDoS attacks is discussed by (Bhatia, S., Behal, 2018). One is the high-rate flooding attacks generated from the high number of requests targeting a victim by various bots. The other is semantic attacks that exploit an application or protocol's implementation or design flows.

- (Bawany et al., 2017) characterized prevalent DDoS attacks into three categories: reflection-based, protocol exploitation flooding, and reflection and amplification-based. Reflection-based DDoS attacks include Smurf attacks and Fraggle attacks. Protocol exploitation flooding attacks include SYN flooding attacks and UDP fragmentation attacks. Finally, reflection and amplification-based DDoS attacks have DNS and NTP amplification attacks.
- Similarly (Swami et al., 2019) divided DDoS attacks into volumetric attacks and application layer attacks. Volumetric attacks were further classified into flooding attacks and amplification attacks.
- The classification done by (Yan et al., 2015) is based on TCP/IP protocol stack, i.e., Application, Transport, and Internet layer.
- (Mirkovic & Reiher, 2004) presented two taxonomies to classify DDoS attacks and defenses based on the essential features of attack strategies and design decisions of DDoS defense mechanisms. It also states how DDoS attacks take place on the global Internet.
- (Zeebaree et al., 2018) state that DDoS attacks are of two types: application layer and network/transport layer attacks. Application layer attacks can be further divided into resource-exhausting and bandwidth-consumption attacks.
- According to US-Cert, layer-3 (Network layer) and layer-4 (Transport Layer) DDoS attacks are volumetric attacks, and layer-1 (Application layer) attacks are semantic-based attacks (*DDoS Quick Guide*, 2020).

Table 3.1 shows the above-listed DDoS attack classifications.

Table 3.1: Different classifications of DDoS attacks

Author	Classification
(Bhatia, S., Behal, 2018)	Flooding attacks Semantic attacks
(Bawany et al., 2017)	Reflection attacks Protocol exploitation attacks Reflection and amplification attacks
(Swami et al., 2019)	Volumetric attacks (flooding, amplification), Application layer attacks
(Yan et al., 2015)	Application layer attacks, Transport layer attacks, Internet layer attacks
(Mirkovic & Reiher, 2004)	Source address vulnerability Exploiting vulnerability Attack rates dynamics Based on the impact of the attack and the recovery of the victim
(Zeebaree et al., 2018)	Application layer attacks (resource exhausting, bandwidth consumption attacks) Network/Transport layer attacks
US-CERT (*DDoS Quick Guide*, 2020)	Volumetric attacks (Transport layer, Network layer) Semantic attacks (Application layer attacks)

Based on the above classifications, a general taxonomy for types of DDoS attacks is depicted in Figure 3.1; mainly, the attacks are categorized as volumetric, semantic, and reflection attacks. As shown in Figure 3.2, volumetric attacks are generated from the high number of requests targeting a victim, while semantic attacks are the ones that exploit the implementation or design flows of an application or protocol.

For reflection attacks, the attacker spoofs the source IP address of request packets to that of the victim, resulting in all the server response packets going to the victim instead of the attacker. When these responses are amplified, like DNS amplification attacks, they become reflection amplification attacks. This generalized taxonomy can cover all the attack types mentioned in Table 3.1. Some attacks can also overlap in multiple categories, e.g., a flooding attack is mainly a volumetric attack, but the HTTP flood attack is also a layer 7 attack.

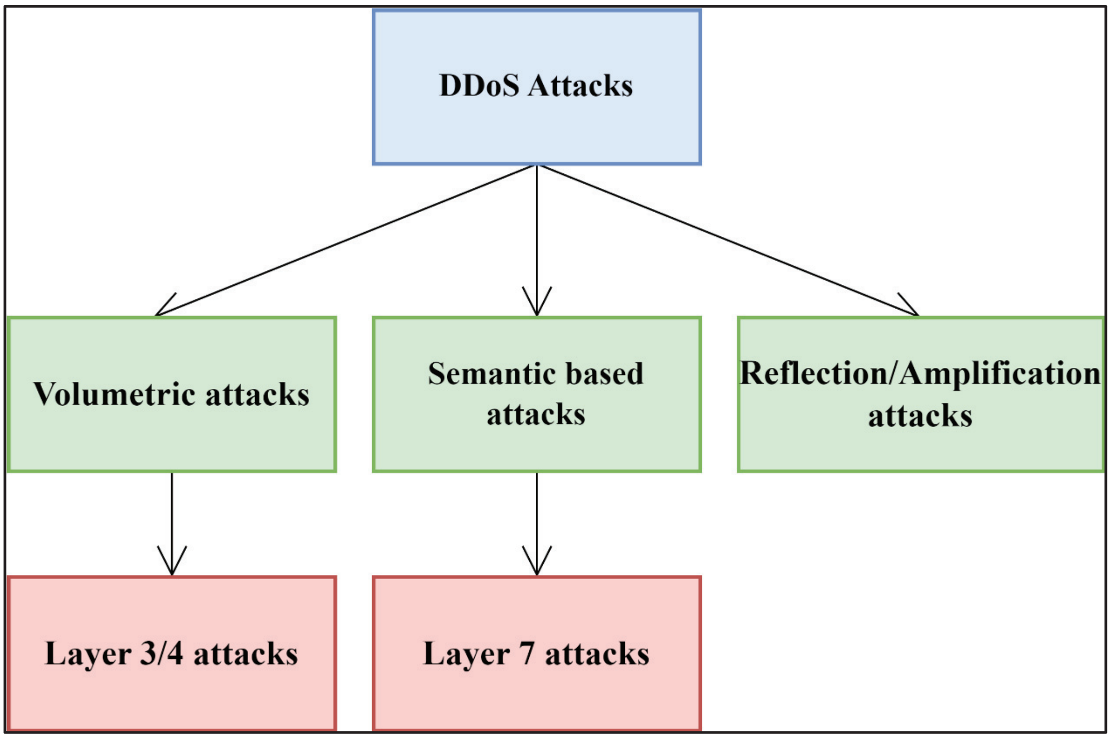


Figure 3.1: Taxonomy of DDoS attacks

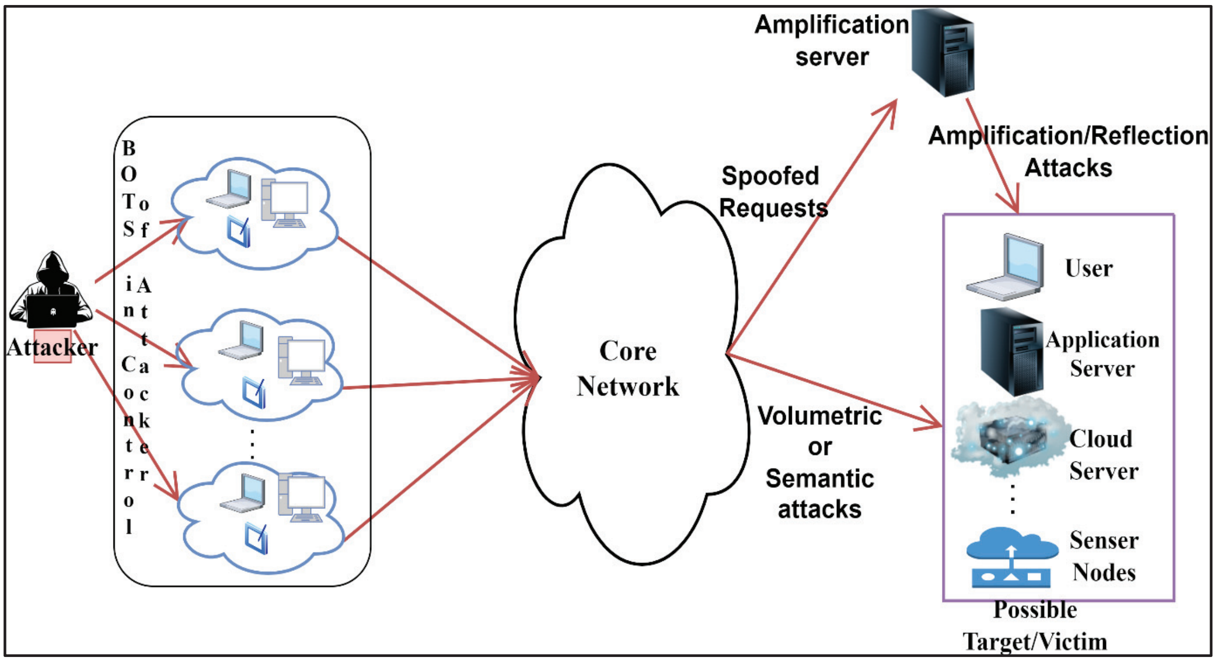


Figure 3.2 : Various DDoS attacks

3.3 Software Defined Networking (SDN)

Due to the dawn of technologies such as Cloud Computing, Big Data, Virtualization, and the increase in the use of mobile devices, a need for high network capacity and network scaling became necessary (Rowshanrad et al., 2014). The network devices became more complex to support changing traffic patterns, and it was difficult for network administrators to configure each device individually. Because of this reason, the idea of programmable networks was introduced, and SDN came into existence. SDN is a promising network paradigm in which the forwarding and control planes are decoupled. It is a network architecture that breaks vertical integration by separating the data plane, i.e., the network devices that forward traffic, from the control plane, i.e., the software logic that controls how traffic will be forwarded through the network (Kreutz et al., 2015). This decoupling feature facilitates many innovations by making the network programmable and logically centralized.

SDN infrastructure has two parts:

a) Software control plane
   - It is the network's brain
   - It can be run separately from devices
   - It computes the logic of how traffic will be forwarded.

b) Programmable data plane
   - Typically, programmable hardware
   - Controlled by the control plane.

The advantages of SDN over conventional-network are-

- SDNs are easier to coordinate, i.e., the network operator can easily write a program that helps coordinate different network devices.
- Easier to evolve because of network automation.

- Apply programming languages as conventional computer science approaches, as the controller can give new flow rules based on the type of functionality needed.
- SDN is logically centralized but not physically; hence there can be more than one controller distributed across the globe to balance the load.

3.3.1 OpenFlow Protocol

OpenFlow protocol is responsible for interaction/communication between the SDN architecture's control plane and data plane ("Software-Defined Networking: The New Norm for Networks," 2012). There is a non-profit industry known as Open Networking Foundation (ONF), which is prominent in normalizing the critical elements of SDN architecture like OpenFlow Protocol and also in the development of SDN. In an SDN architecture, OpenFlow is the first standard communications interface defined between the forwarding layer and the control layer. ("OpenFlow Switch Specification Version 1.5.1 (Protocol Version 0x06)," 2015) OpenFlow 1.0 got released on December 31, 2009, and formally introduced how control and data plane can be separated. It introduced flow tables as forwarding structures in a network switch and allowed 12 different fields on which incoming packets can be matched against flow entries. OpenFlow 1.1 added new features in the form of multiple flow tables, group tables, Multi-Protocol Label Switching (MPLS) tags, Virtual Local Area Network (VLAN) tags, and virtual ports. OpenFlow 1.2 added generic and extensible packet matching capability via OpenFlow extensible match descriptors. It allowed any header fields used in matching for Ethernet, VLAN, MPLS, IPv4, and IPv6 to be used in flow tables. Then came the significant milestone, i.e., OpenFlow 1.3. Today, many of the commercially available networks have switched to support this version. It extended the features by now allowing per-flow meters, cookies, and auxiliary connections, which helps in entries. Experimentor modes permitted within OpenFlow allow experimenting with application layer flow definitions. OpenFlow 1.3.1

was released on September 06, 2012, and the main change was to add a bitmap of version numbers for hello messages during negotiation. OpenFlow 1.3.2 was released on April 25, 2013. OpenFlow 1.3.3 on September 27, 2013, OpenFlow version 1.3.4 on March 27, 2014, and OpenFlow version 1.3.5 was released on March 26, 2015, with many major changes, and few of them are defining oxm_len for OpenFlow extensible match IDs in table features to have payload length, IPv6 flow label was made maskable, control channel maintenance section was added, and some new modification to MPLS. OpenFlow version 1.4.0, released on August 5, 2013, modified many other parts of protocols with tag, length and value structures, improving extensibility. This additional extensibility will also help in easily extending the Experimenter extension Application Programming Interface (API). Likewise, with new modifications, OpenFlow version 1.4.1 was released on March 26, 2015, and OpenFlow version 1.5.0 on December 2014. Then came OpenFlow version 1.5.1 on March 26, 2015, which added a new error OFPBAC_BAD_METER for the wrong meter in flow-mod, clarifications for spelling, grammar, and other types were given, and a few other changes were also made. On December 2016, a new version of 1.6 came, but it is only accessible to ONF members.

3.4 Techniques Used for Prevention

In this section, the techniques used for Prevention against DDoS attacks are explained. These include IBC, RF, and symmetrical routing

3.4.1 Identity-Based Cryptography (IBC)

It is a kind of public-key cryptography in which an identity unique to the participating entity is used as a public key with a corresponding private key. It eliminates the need for separate servers (e.g., PKES servers) to distribute public key certificates. IBC incorporates both IBS and Identity-Based Encryption (IBE) (Long & Xiong, 2020). (Boneh & Franklin, 2001), (Cocks, 2001), and (Sakai, R., Ohgishi, K., and Kasahara,) introduced their respective works on IBE. The work of (Boneh & Franklin, 2001), and (Sakai, R., Ohgishi, K., and Kasahara) is primarily based on bilinear mapping on elliptic curves, and bilinear Diffie-Hellman is the underlying principle for the security of their proposed schemes. Since crafted by Boneh et al. (Boneh & Franklin, 2001) and Sakai et al. (Sakai, R., Ohgishi, K., and Kasahara), the pairing-based IBC is proven to be very efficient with a proven security model. Since its inception, IBC has been applied in various areas (Anggorojati & Prasad, 2018),(Salman et al., 2016), (Drias et al., 2017), (Suganthi, S.D., Anitha, R., and Thanalakshmi). To further promote its application, IEEE has also specified a standard based on pairing-based IBC ("IEEE Standard for Identity-Based Cryptographic Techniques Using Pairings,"). In this, a pairing-based signature scheme is specified, named BLMQ (Barreto et al., 2005) (named after the surnames of its proposers, i.e., Paulo S.L.M. Barreto, Benoit Libert, Noel McCullagh, and Jean-Jacques Quisquater). BLMQ is based on the work of Sakai and Kasahara (SAKAI & KASAHARA, 2003).

3.4.2 Random Forest

Random forest, trademarked by Leo Breiman (BREIMAN, 2001) and Adele Cutler, is a standard ML Algorithm (Yiu, 2019). Random forest algorithm can be used as both a classifier and a regressor. This technique consists of multiple decision trees to output a single result of prediction. To understand random forests, we first need to understand decision trees.

3.4.2.1 Decision Trees

The basis of the random forest model is decision trees. As the name suggests, decision tree is a tree in which a node splits into different nodes based on a condition. For example, the condition can be that the Sun rises in the east, so that this node will be split into two depending on the answer yes or no. Following this principle, a prediction is made when we reach the final leaf node.

3.4.2.2 Ensemble Methods

A number of algorithms, such as decision trees, are combinedly used in ensemble learning techniques, and their predictions are combined to determine the most common outcome (*Random Forest*). The ensemble technique bagging (Breiman, 1996) is used in random forest. Bagging is also known as bootstrap aggregation. This method selects a random sample of data from a training set with substitution, which allows for multiple selections of the individual data points. Following the generation of several data samples, these models are individually trained to result in more accurate predictions.

3.4.2.3 The Random Forest Algorithm

The random forest algorithm is created using a group of decision trees. Ensemble learning is a popular technique that improves the accuracy of predictive models by combining multiple individual models. One such ensemble method is the decision tree ensemble, which comprises several decision trees, each trained on a bootstrap sample of the data with replacement.

To further enhance the diversity and reduce correlation across decision trees, feature bagging is applied, introducing another randomization instance.

To evaluate the performance of the decision tree ensemble, an Out Of Bag (OOB) sample, consisting of one-third of the training data not used in building each tree, is used as a test set. For classification tasks, the predicted class is determined by taking the majority vote on the individual trees' outputs, while for regression tasks, the individual trees' outputs are averaged.

Finally, the OOB sample is used for cross-validation to estimate the accuracy of the model and to finalize the prediction. This approach helps to ensure that the model generalizes well to unseen data.

3.4.3 Symmetrical Routing

In computer networking, symmetric routing is a configuration in which traffic between two hosts follows the same path in both directions. It means that the network traffic from Host A to Host B takes the same path as that from Host B to Host A.

Symmetric routing is important because it ensures that the network traffic flows efficiently and with the same level of security in both directions. If the routing is asymmetric, i.e., traffic follows different paths in each direction, it can cause issues with performance and security. Additionally, asymmetric routing can make it difficult to apply security policies consistently to both directions of traffic. In summary, symmetric routing is essential for ensuring efficient and secure network traffic flow between hosts.

********** End of Chapter **********

CHAPTER 4- Prevention of DRDoS Amplification Attacks by Penalizing the Attackers in a Software-Defined Networking Environment

4.1 Introduction

In this chapter, we present four mechanisms to defend against DRDoS attacks. The proposed techniques are based on the philosophy of attack prevention, which conforms to the definition of True Prevention. To the best of our knowledge, no technique can completely prevent any attack traffic from entering the Internet's core network in the first place. It would require controlling the attacker's network infrastructure itself. Instead, we argue that the responsibility of prevention should lie within the underlying network infrastructure, i.e., the underlying network functionality should be amended and should be equipped with enough rules to prevent such attacks. Presently, Internet architecture adheres to the following design principles:

a) **Destination IP-based routing**: Routing algorithms populate the router's forwarding tables, which forward each network packet. This forwarding decision is solely based on the destination IP address.

b) **No source-IP validation**: The primary design goal of the Internet was reliability, distributed management, cost-effectiveness, and support for multiple varieties of networks and different types of communication services. Security was not the primary design principle. Accordingly, source IP validation is not done by the network. In the client-server model, the source IP and destination IP fields are swapped in the corresponding response packet when the packet reaches the destination. This packet is again routed using the forwarding tables of different routers.

Considering the above network properties, the True Prevention algorithm should conform to the following:

a) Attack traffic can enter the network but should be mitigated away automatically within some constant time. This time can vary and depends upon the network bandwidth of the attacker.
b) Even in the presence of attack traffic, the victim's network should always be safe. In other words, such attack traffic should never reach the victim.

One of the promising philosophies of prevention is to direct all the reflected traffic by the server toward the attacker's network itself or to the individual systems that were part of the attack in case of DDoS. It would require no change in underlying forwarding techniques (i.e., from client to server or the request packet). But it requires modifying reverse forwarding rules (i.e., from server to client or the response packet. The underlying network should forward the packet to where it originated, even if the source IP address is spoofed. We achieve this reverse forwarding by proposing four techniques- **IP-Switching, Port-Mapping, PortMergeIP, and SymSDN**. We categorize these approaches into two categories based on the underlying modification.

a) The basis of the first category is to modify the switch's flow table, not the packet header. SymSDN lies in this category. SymSDN's mechanism is based on symmetric routing. In symmetric routing, the response packet follows the same path (in the reverse direction) as the corresponding request packet, irrespective of the source IP address of the request packet. It will prevent DRDoS attacks, even if IP spoofing is done, as the response packet will go to the attacker rather than the victim, completely bypassing the IP spoofing.
b) The basis of the second category is to modify the packet header. We ensure that the attacker's identity or the request packet's path is somehow embedded within the request

packet itself so that the response packet reaches the source of the attack using that information. IP-Switching, Port-Mapping, and PortMergeIP lie in this category.

i. **Port-Mapping**: In this proposed method, the request packet's path is stored in the options and padding field of the IP header of the packet so that the response packet can follow the same path.
ii. **IP-Switching**: In this method, we propose to change the source IP address in the request packet with another legitimate one within the network.
iii. **PortMergeIP**: Above two techniques have a few limitations (as explained in section 4.4.3). PortMergeIP focuses on the removal of these limitations by merging the two techniques.

All the methods are proven to prevent the victim from the attacks altogether. These techniques also cause the attack to divert back to the attacker, saving the client from the attack and hampering the attacker's ability to launch further attacks. For the techniques presented in this paper, we conform to the following definition of prevention (called True Prevention) as provided in (Saharan & Gupta, 2021): Let,

- **B** be the network bandwidth of the network in which an attacker, or a bot in control of an attacker, resides.
- **V** be the victim of a DDoS attack.
- $\mathbf{I_V}$ be the IP address of victim **V,** and $\mathbf{I_A}$ be the IP address of attacker **A**

True Prevention is defined as a set of techniques embedded into the network routers which prevents the attack traffic from reaching **V**, even though the destination IP address of network packets belonging to the DDoS attack is $\mathbf{I_V}$, and automatically mitigates the attack for some constant time **T** where **T** is directly proportional to **B**.

Because of the flexibility and programmability aspects provided by SDN in terms of controllers, it is used to show and validate the required network intelligence to be induced in the underlying network infrastructure to prevent DRDoS attacks. Also, DNS based DRDoS attack is used to show the effectiveness of the proposed techniques as, attackers still choose DNS architecture to launch DNS amplification attacks, e.g., the attack on Google (Cimpanu, 2020b) and the Spamhaus attack (Prince, 2013b). Such traffic can easily be recognized using the source port number in the IP packet. Henceforth, the discussion is based only on the prevention of DNS amplification attacks. Of course, the proposed prevention algorithms can be applied to other types of DRDoS attacks and traditional network infrastructure.

4.2 Related Work

This section primarily shows the DDoS defense mechanisms close to the definition of True Prevention. (Duan et al., 2018) proposed a defense technique against infrastructure DDoS attacks on specific flows. This technique works for critical links to a server by a proactive routing mutation. The problem with this approach is that the victim can still be affected when the attack traffic is switched to non-critical links. (Keromytis et al., 2004) proposed secure overlay services as a prevention technique. Here, authentication of the packets is done at secure overlay points known as SOAPs, which forward the packets through overlay nodes to beacon nodes. The primary issue with this approach is additional infrastructure requirements, thus making it difficult to scale. (Wu et al., 2013) proposed SAVI, and it complements ingress filtering by adding IP address validity to an individual source. SAVI is defined as network-based so that there is no dependency on a host. SAVI instances may face reliability issues due to a loss of bindings in SAVI devices through a restart of SAVI devices or binding information for a new link not reaching the SAVI device. A new IPv6 address generation algorithm was proposed by (Ying Liu et al., 2015). Firstly, SAVA (Wu et al., 2008) is used to authenticate source addresses. After this, an address is generated using NID (network identity) and time, and this address is assigned to the host. This newly generated IP address will be used for communication. The absence of a built-in security module in the ONOS SDN Controller leaves it vulnerable to DDoS attacks. To address this issue, (Ohri et al.,2024) leverages the widely adopted Suricata (IPS). The proposed structured approach comprises of reconnaissance, detection, and mitigation phases. During reconnaissance, all incoming network packets undergo thorough analysis. In the subsequent detection phase, the system identifies potentially malicious IP packets. Finally, in the mitigation phase, any identified malicious packets are promptly discarded. (S. Kim et al., 2017)have proposed a framework using SDN. A DNS response is only accepted on the client-side when there is a request; otherwise, the packet is dropped. The DNS request information is stored in the switch or the memory of the SDN controller. The problem with this approach is that the attack traffic still reaches the victim's network. (Sahri & Okamura, 2016) have proposed an authentication approach to prevent DNS amplification attacks in SDN as an underlying architecture. In their authentication approach, the DNS server, before sending the response, sends a query back to the client, asking whether the query was sent or not, and if the client responds, only then is the response provided to the client. Because of this, a delay of one extra Round-Trip Time (RTT) is introduced in the response packet. The authentication approach is the algorithm we use to compare our techniques by focusing mainly on the additional RTT needed to get the final DNS response.

4.3 Modifying Switch Flow Table- SymSDN (Symmetric SDN)

In the existing Internet architecture, whenever a router receives a packet, it consults its routing table and forwards the packet. A routing table can be created statically or dynamically using a routing algorithm. In SDN, this process is slightly modified. The control plane of all the routers is placed in a single machine called a controller that controls the data plane of all the routers in the network (Kreutz et al., 2015), (ONF). The router has a local table called the flow table, which the controller usually populates. On encountering a packet, the SDN switch consults its flow table. If there is a match, the router forwards the packet accordingly.

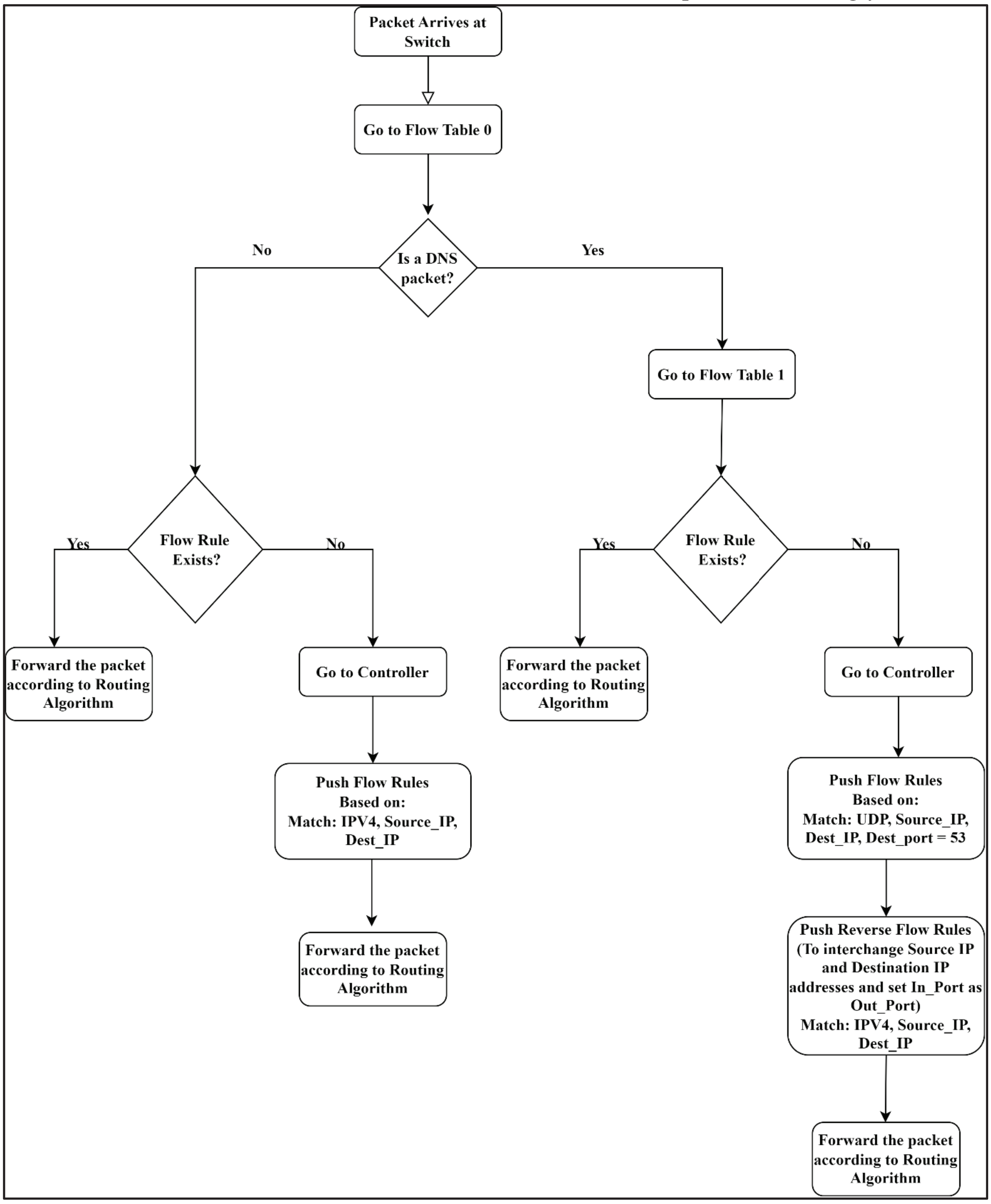

Packet Arrives at Switch
Go to Flow Table 0
Is a DNS packet?
No
Yes
Go to Flow Table 1
Flow Rule Exists?
Yes
No
Forward the packet according to Routing Algorithm
Go to Controller
Push Flow Rules Based on: Match: IPV4, Source_IP, Dest_IP
Forward the packet according to Routing Algorithm
Flow Rule Exists?
Yes
No
Forward the packet according to Routing Algorithm
Go to Controller
Push Flow Rules Based on: Match: UDP, Source_IP, Dest_IP, Dest_port = 53
Push Reverse Flow Rules (To interchange Source IP and Destination IP addresses and set In_Port as Out_Port) Match: IPV4, Source_IP, Dest_IP
Forward the packet according to Routing Algorithm

Figure 4.1 : The architecture of SymSDN

The controller analyzes the packet and decides to forward or drop it based on certain conditions. It also pushes appropriate flow rules in the flow table of the router to match future packets of the same. We push reverse flow rules along with forward flow rules in the SDN switch to implement symmetric routing for DNS packets. (Le Pennec et al., 2014) have proposed a way to implement symmetrical routing in the existing internet architecture. We extend their work to provide defense against DRDoS attacks, by modifying this architecture in SDN.

Figures 4.1 and 4.2 illustrates that two flow tables are used to implement the proposed methodology, i.e., SymSDN. Flow table 0 is responsible for legacy routing, and flow table 1 for symmetric routing.

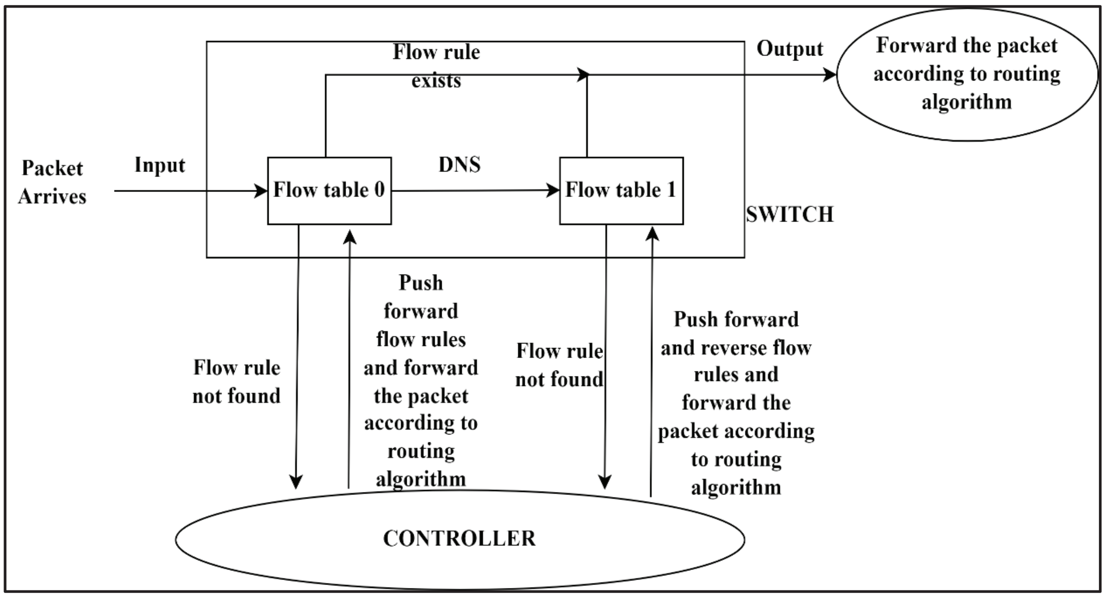

Flow rule exists
Output
Forward the packet according to routing algorithm
Packet Arrives
Input
Flow table 0
DNS
Flow table 1
SWITCH
Flow rule not found
Push forward flow rules and forward the packet according to routing algorithm
Flow rule not found
Push forward and reverse flow rules and forward the packet according to routing algorithm
CONTROLLER

Figure 4.2 : The internal functioning of the SDN switch

As shown in Figure 4.2, flow table 0 is pipelined to table 1 for DNS packets. When the packet first arrives at the SDN switch, it is forwarded to flow table 0 and checked if it is a DNS packet or not (Figure 4.1). If it is a DNS packet, the action is GOTO table 1; otherwise, forward it according to the flow rules. If a flow rule for that packet does not exist in table 1, it is forwarded to the controller, which will push both the forward flow rules and reverse flow rules.

The controller pushes a flow rule (matching a DNS request from the same host to the same server) in the forward direction (i.e., from host to server) and another reverse flow rule in the table (matching the corresponding DNS response sent by the same server to the same host) in the reverse direction (the input interface becomes the output interface). It forces the DNS response packet to take the same path as the DNS request but in the reverse direction, ensuring that the DNS responses go to the same physical machine that placed the corresponding DNS request. This process completely bypasses the IP spoofing and defends against the DNS reflection attack.

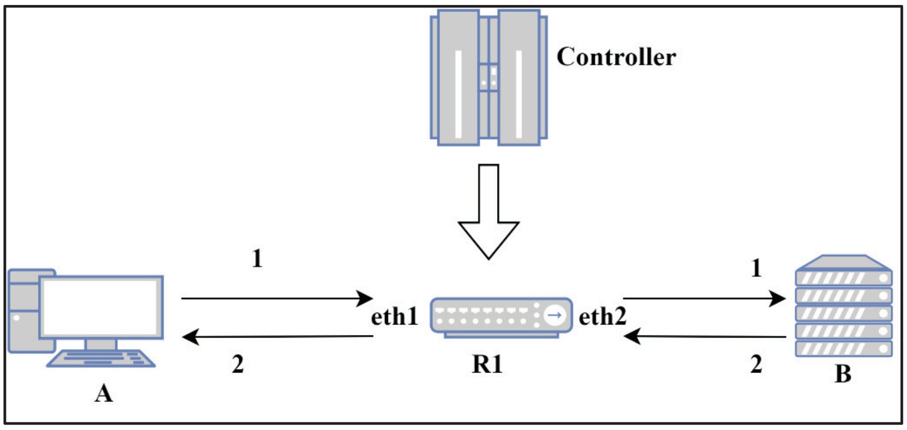


Figure 4.3 : Working of the SDN controller

As shown in Figure 4.3, when the controller encounters a DNS request packet arriving on R1 that originated from A and is bound for B, it pushes a flow rule on R1 to send all DNS request packets originating from A and bound for B to B (through the interface eth2). In our mechanism, however, the controller also pushes a reverse flow rule on R1 to send all DNS response packets of this request packet(A->B) in the reverse direction (B->A), i.e., from interface eth1.

Generally, a flow is defined as a sequence of packets exchanged between a specific source and destination. We discovered that if we use the main parameters defined for a flow rule in SDN, especially the source and destination port numbers, overflow can occur in the flow tables due to DRDoS attacks. At first, we considered five parameters - source IP address, destination IP address, source port, destination port, and protocol for rules entry. However, considering the source ports to define flow rules has a risk. An attacker commencing a DRDoS attack can instruct all its bots to use a different source port for every packet. It leads to an exponential increase in the number of flow rules that need to be pushed to the SDN switch. Eventually, this leads to the flow tables getting over-filled. Hence, this type of attack is called a ***"Flow Table Overflow Attack."*** To make our mechanism resistant to these attacks, we opted not to use the parameter source port for defining the flow rules. Our extensive experimentation proved that this is highly effective in reducing the number of flow rules being pushed to the SDN switch without affecting the mechanism's efficiency.

Another line of defense we have adopted against flow table overflow attacks: is ***"Flow Table Pipelining."*** We keep the primary flow table for general-purpose use and create a new secondary flow table for handling DNS flow rules. It ensures that even if a large number of DNS flow rules are getting pushed to the SDN switch, the switch's primary flow table is unaffected, and only the secondary flow table will overflow, even in the worst-case scenarios.

4.3.1 Experimental Setup and Result Analysis

In this section the experimental setup done to validate SymSDN is explained. To validate SymSDN various parameters such as throughput, packet loss are calculated.

4.3.1.1 Experimental Setup

For experimental purpose, we created a virtual network through mininet (*Mininet*, 2022), whose topology is represented by Figure 4.4.

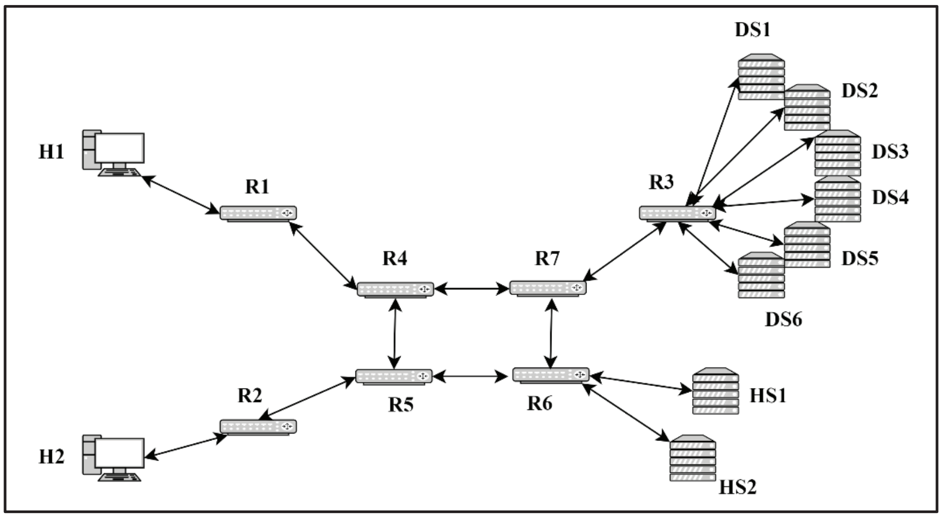


Figure 4.4 : Experimental topology: SymSDN

It consists of 10 hosts and seven routers (SDN switches). Out of the ten hosts, H1 represents the attacker, H2 represents the victim, HS1 and HS2 represent HTTP servers, and DS1 through DS6 represent DNS servers. To simulate a DDoS attack with many attacker hosts, a large number of spoofed requests have been generated by host h1 and amplified response by DNS servers. This topology simulates 4 Local Area Networks (LANs): the attacker's LAN, the victim's LAN, the DNS servers' LAN, and HTTP servers' LAN. All these LANs are connected via intermediate core routers. The DRDoS attack is generated using scapy (*Introduction: About Scapy*, 2023).

4.3.1.2 Results and Analysis

The topology mentioned in Figure 4.4 is connected to the Ryu controller (*RYU A*, 2013), and based on this setup, the following three experiments are performed to validate our prevention approach.

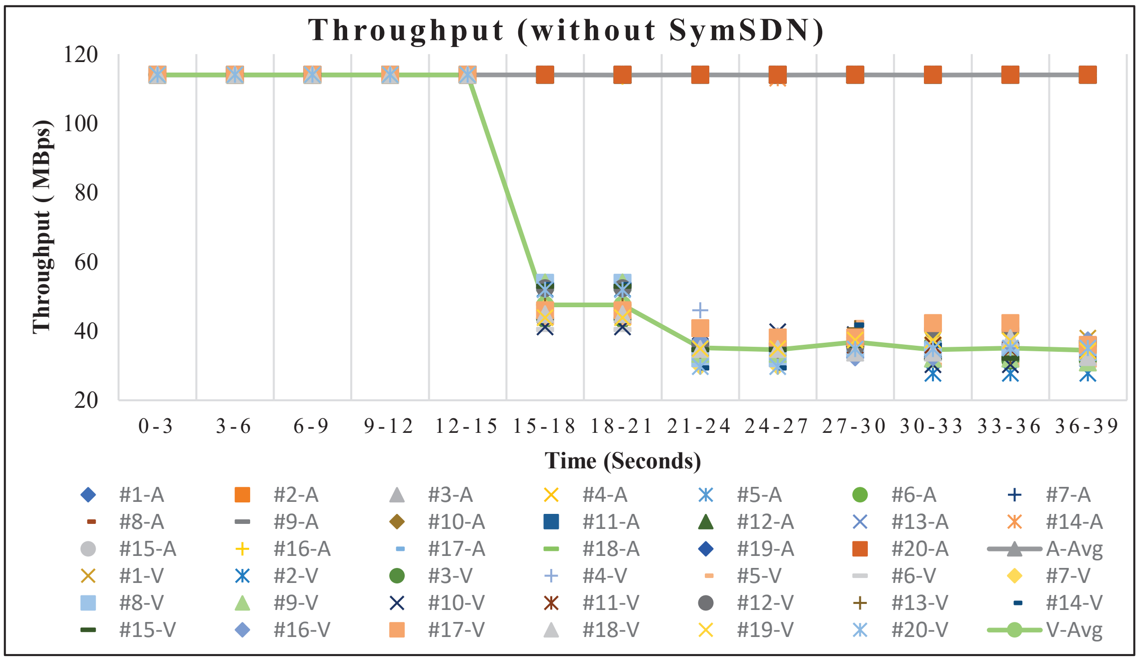


Figure 4.5 : Throughput w.r.t. time (without SymSDN)

a) Throughput is calculated on the victim's and attacker's sides from HTTP servers during an attack of 1 Gbps (by DNS servers) on a link bandwidth of 1 Gbps. Wget is used for the calculation of throughput. Figure 4.5 shows throughput for the time at the attacker's and victim's sides when SymSDN is not in place, and Figure 4.6, shows when SymSDN is in place. Twenty iterations are done for throughput calculation, so #1-A represents throughput at the attacker in the first iteration, #1-V represents throughput at the victim in the first iteration, and so on. The graph's dots represent individual iterations, and the line represents the average at attacker's (A-Avg) and victim's n/w (V-Avg). It is clear from the graphs that when SymSDN is not in place, throughput drops at the victim as soon as the attack starts at the 15th second, and when SymSDN is in pla ce, throughput drops at the attacker, not at the victim. It is because, with SymSDN, the response goes to the originator of the attack (the attacker), irrespective of the spoofed IP address. Hence proving that the proposed approach prevents the attack. The respective data values are also shown in Appendix A.

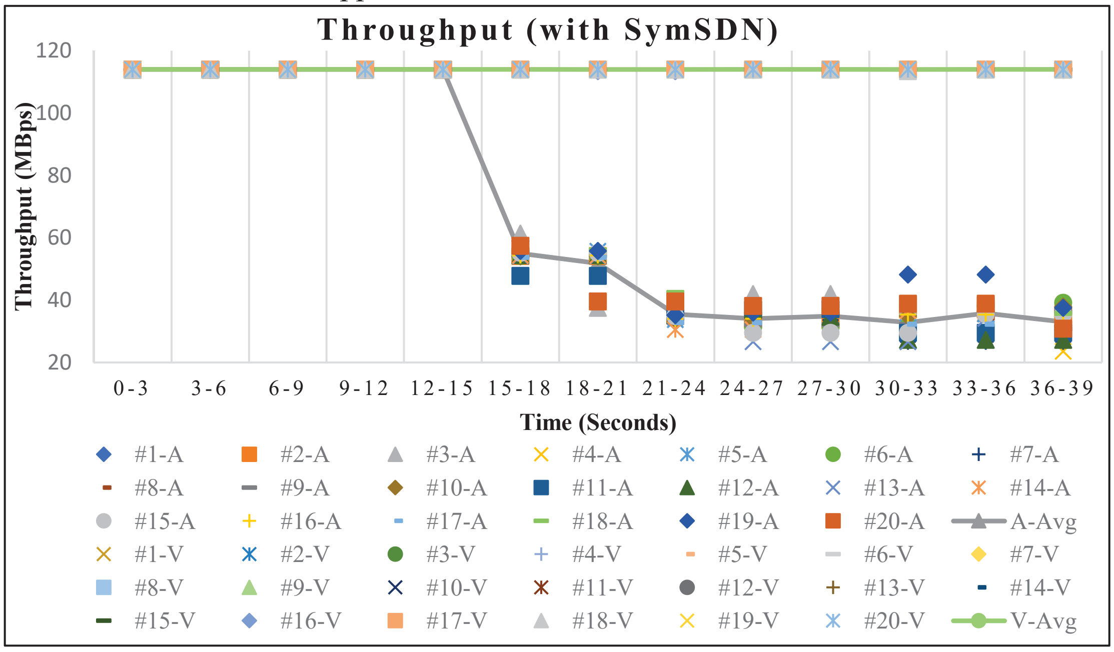


Figure 4.6 : Throughput w.r.t. time (with SymSDN routing)

b) Packet loss due to a DRDoS attack on the victim side without SymSDN (Figure 4.7) and with SymSDN(Figure 4.8) in place is calculated. The link bandwidths of H1-R1 and H2-R2 are set at 500 Mbps with a delay of 1 ms. Ping is started between H1 & HS1 and H2 & HS2 to calculate Packet loss. As soon as the attack starts, packet loss occurs at the victim, while no packet loss occurs at the attacker (Figure 4.7). But when our proposed app roach is in place, the victim remains unaffected, and packet loss occurs at the attacker (Figure 4.8). This experiment is also performed for twenty iterations. The respective data values are also shown in Appendix A

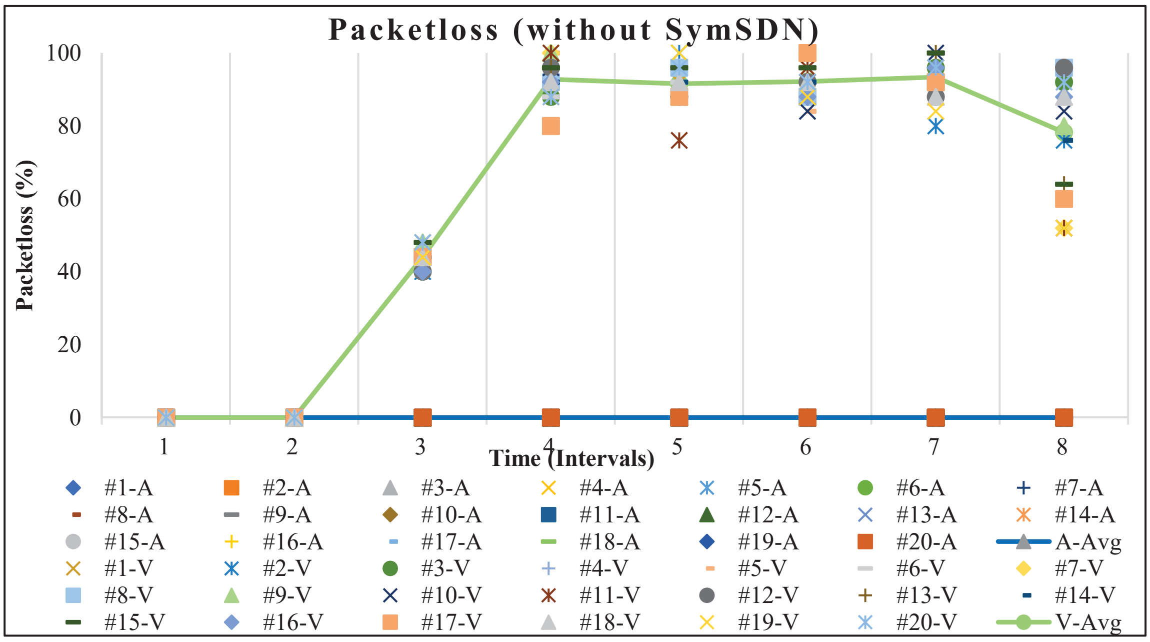


Figure 4.7 : Packet loss % w.r.t. time (without SymSDN)

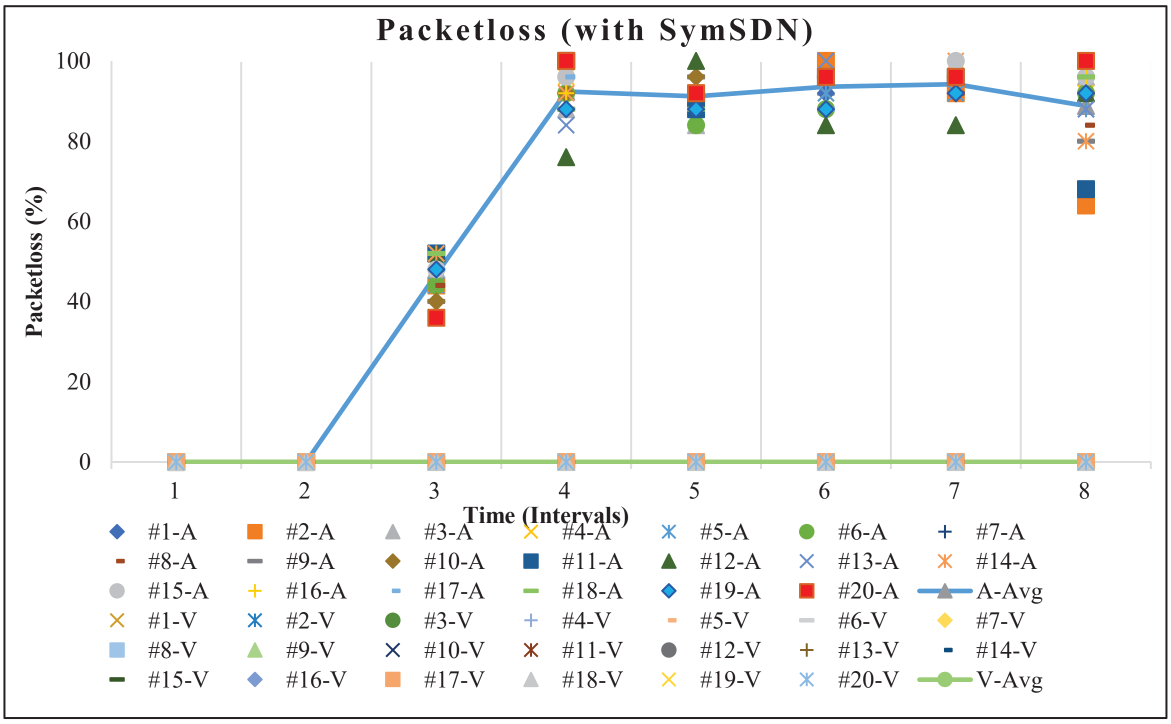


Figure 4.8 : Packet loss % w.r.t. time (with SymSDN routing)

c) We measure the delay caused by DNS request-response due to the SymSDN prevention approach. We calculate this delay and compare it with the authentication approach (Sahri & Okamura, 2016) and the baseline. Figure 4.9 shows this delay. Baseline delay indicates the average delay without any prevention algorithm, which is nearly 0.09 seconds. SymSDN also shows a similar delay of .093 seconds. The authentication algorithm shows the highest delay of .15 seconds as it needs one additional RTT before the server receives the DNS response.

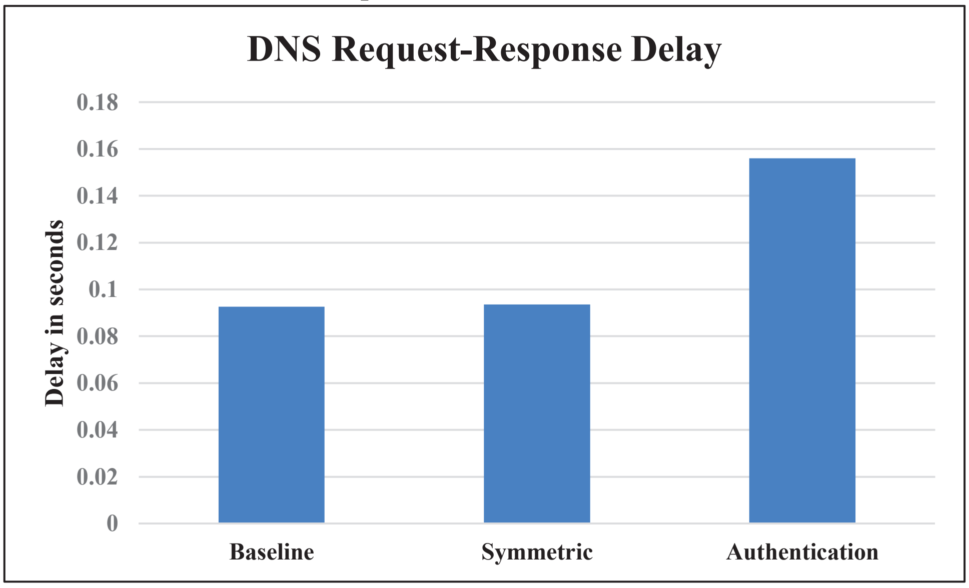


Figure 4.9 : DNS request-response delay

d) As explained in section 4.3, ***"Flow Table Overflow Attack"*** causes an overflow of flow entries in the flow tables at the time of the attack. The prevention is to optimize flow entries by opting out source port numbers. Figure 4.10 showcases this scenario. The attacker (H1) placed many DNS requests with dynamic source ports. The number of flow rules existing inside the flow table is periodically measured at 10-second intervals. The count of flow rules being pushed by the controller is also recorded. The process was repeated three times by setting the flow table capacity as 20000, 30000, and 40000 entries, respectively. Figure 4.10 shows the number of flow entries w.r.t time. When the source port is considered to create the flow rule, the number of flow rules being pushed is observed to be proportional to the number of DNS requests. The actual number of flow rules in the flow table increased along with the number of flow rules pushed by the controller but became constant after reaching the predefined limit, indicating flow-table overflow. When the source port is not considered to create the flow rules, the number of flow entries remain very low (proportional to the number of DNS servers). Hence, Figure 4.10 proves Prevention against the Flow Table Overflow Attack.

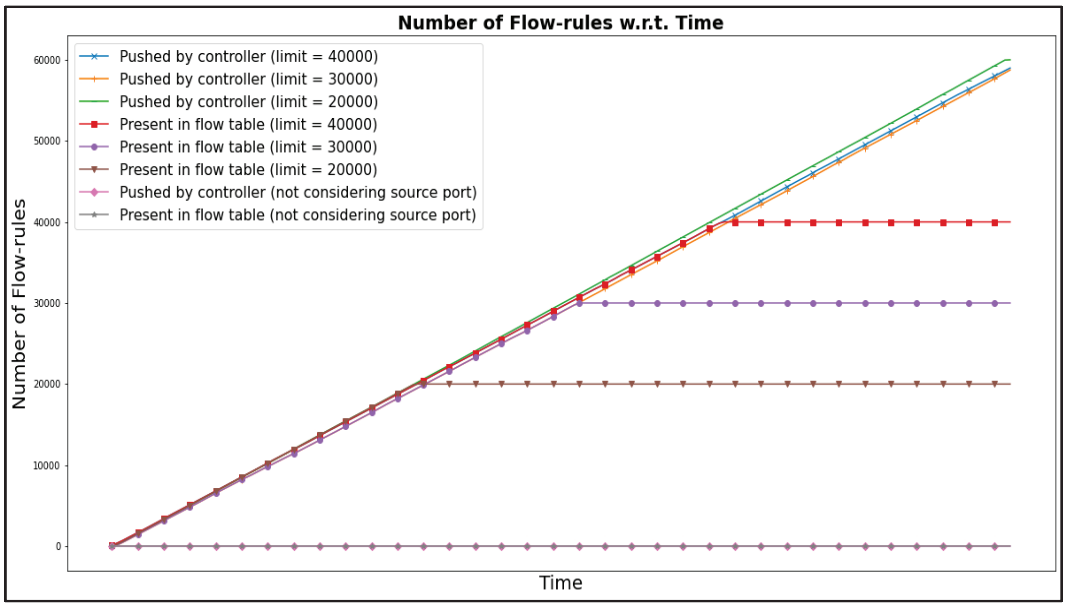
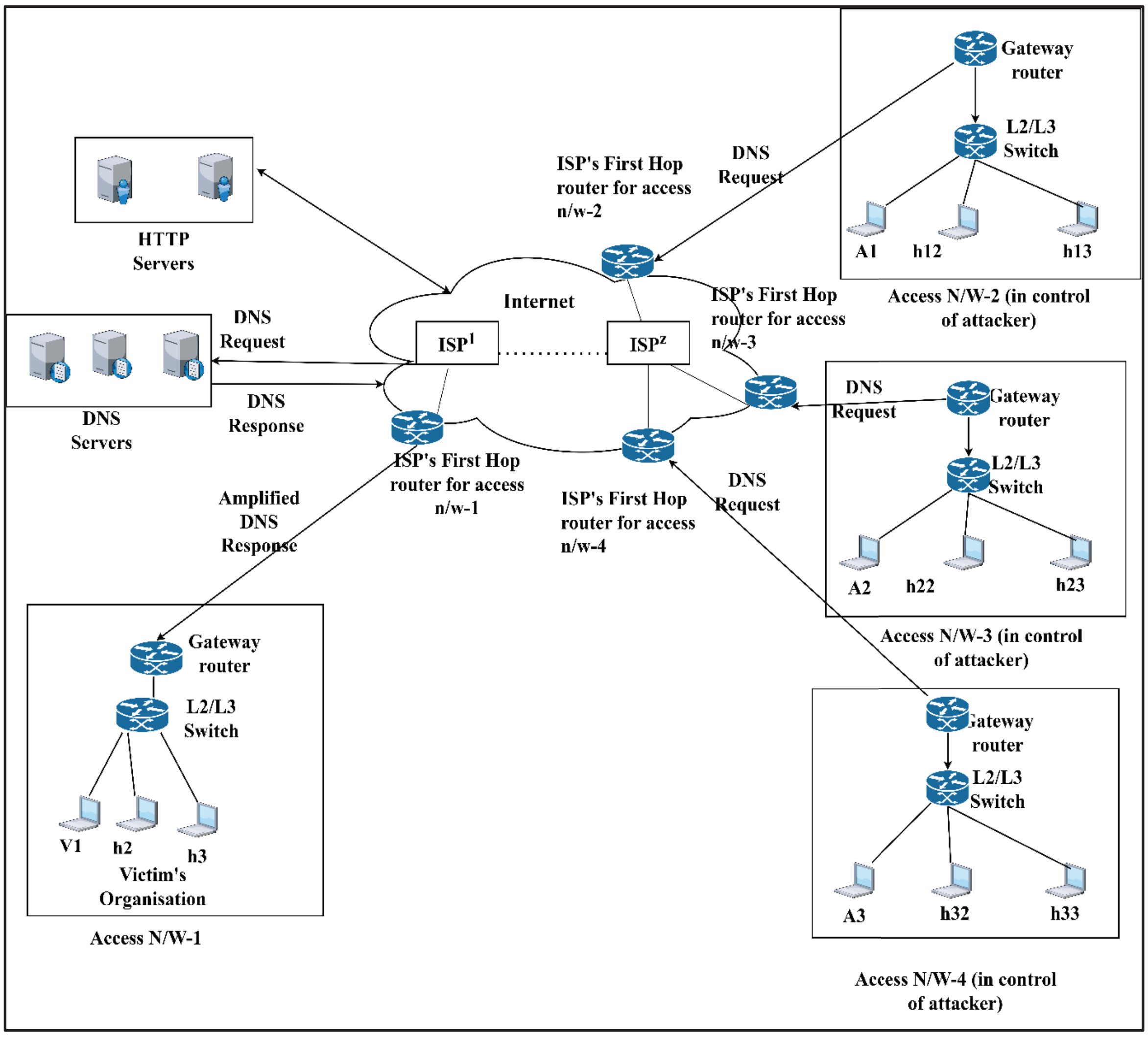


Figure 4.10 : Number of flow-rules w.r.t. time

4.4 Packet Modification to Enforce Reverse Routing

As outlined in section 4.1 above, for True Prevention of DRDoS attacks, the underlying network functionality should be amended. We suggest implementing the change for client-server communication, historically the main culprit for DRDoS attacks. For example, attackers still choose DNS architecture to launch DNS amplification attacks, e.g., the attack on Google (Cimpanu, 2020b) and the Spamhaus attack (Prince, 2013b). Such traffic can easily be recognized using the source port number in the IP packet. Henceforth, the discussion is based only on the prevention of DNS amplification attacks. Of course, the proposed techniques can be extended further for any specific type of amplification attack which exploits the client-server model to launch the attack.



HTTP Servers
DNS Servers
DNS Request
DNS Response
Internet
ISP¹
ISPᶻ
ISP's First Hop router for access n/w-1
ISP's First Hop router for access n/w-2
ISP's First Hop router for access n/w-3
ISP's First Hop router for access n/w-4
DNS Request
DNS Request
DNS Request
Amplified DNS Response
Gateway router
L2/L3 Switch
A1
h12
h13
Access N/W-2 (in control of attacker)
A2
h22
h23
Access N/W-3 (in control of attacker)
A3
h32
h33
Access N/W-4 (in control of attacker)
V1
h2
h3
Victim's Organisation
Access N/W-1

Figure 4.11: Generalized network topology

Any access network (whether an individual or an organization) gets connected to the Internet using the Internet Service Provider (ISP) services. Consider Figure 4.11, which shows how an organization's network gets connected to the Internet. It shows the access network of four organizations (Org1, Org2, Org3, and Org4). Access network 1 belongs to the victim's organization, and the attacker controls access networks 2, 3, and 4. In access network-1, there are three hosts: one is the victim, i.e., V1, and two end-hosts, i.e., h2 and h3. A1, A2, and A3 are attackers in access networks-2,3, and 4, respectively. We have used the terms organization and access network interchangeably in this work. For launching the attack, the attacker will spoof the source IP address field of the packets to that of a host in the victim's network. Each access network is connected to the ISP's first-hop router through its gateway router. These first-hop routers will be responsible for switching IP addresses in the IP-Switching algorithm and PortMergeIP algorithm. DNS servers to generate a DRDoS attack and HTTP servers to generate legitimate traffic and calculate throughput are also connected to the network. We have used three DNS servers to launch the attack.

The algorithms 4.1-4.3, respectively show the proposed algorithms for IP-Switching, Port-Mapping, and PortMergeIP. The notations used by these algorithms are defined in table 4.1.

Table 4.1 : Notations used in the packet modification algorithms

Notation	**Definition**
n	number of routers in-between source and destination
r^i	ith router ($0 < i <= n$)
M	total number of organizations
P	switches or routers between the host machine and ISP's first hop router
T	number of interfaces a router has
r_j^q	jth organization's qth switch or router between host and ISP's first-hop router ($0 < j \leq m$, $0 < q \leq p$)
r_j^h	ISP's jth organization's hth first hop router ($0 < h \leq m$)
r_j^{hk}	ISP's jth organization's hth first hop router's kth incoming interface($0 < k \leq t$)
r_j^{qk}	jth organization's qth router's kth incoming interface
r_j^{ql}	qth router's lth outgoing interface ($0 < l \leq t$)
ipv4-src, *ipv4-dst*	source IP address, destination IP address
src-port, *dst-port,*	source protocol number, destination protocol number
Option	options field of the packet
rtr-id	ID of router
in-port	The interface of router where the packet arrives
out-port	The outgoing interface
SRC-IP-ADDR	Source IP address of the downstream network known by ISP

4.4.1 IP-Switching

In IP-Switching, the source IP address of every DNS request packet (i.e., an IP packet with destination port number 53) is switched after it leaves the organization's network with the organization's downstream address, as shown in algorithm 4.1. ISPs' first-hop routers do this switching. For example, when the packet leaves access network 1, the ISP's first-hop router for access network 1 will do this switching. We assumed that the ISP's network should be SDN-enabled for implementation and testing purposes. Otherwise, a change in the router's functionality would be required. The DNS response traffic would be directed toward the attacker's organization even if the source IP is spoofed. It results in congestion of the traffic in the attacker's organization and will slow down and eventually stop the attack for some time because routers would not be able to handle such a massive surge simultaneously. It would result in the True Prevention of the attack with negligible overhead regarding memory usage and computation. Of course, the time to prevent depends upon the amplification factor, resultant DNS response traffic, and the bandwidth of an attacker. Formally, the algorithm is explained in the IP switching algorithm to prevent DNS-based amplification attacks.

Algorithm 4.1: IP-Switching

```
IP-SWITCHING Algorithm- Rule set to be added in switches by controller

Assumption: There is no intermediate router between the organization's gateway
router and ISP's first hop router.

INPUT- Packet coming at $r^i$

for i= 1 to n do
  while rtr.id = $r_j^h$ and in-port = $r_j^{hk}$ do
     if packet is udp and dst-port==53 then
          ipv4-src = SRC-IP-ADDR
          Forward the packet to out-port
   end while
  Forward the packet to out-port
 end for
```

4.4.2 Port-Mapping

In Port-Mapping, the forward path of the request packet is stored in the packet itself to enable the corresponding response packet to use the same path to return to the source. According to CAIDA's skitter Map, the average-path length of any packet lies between 10 to 30 hops (*CAIDA's Skitter MAP*). This means storing a path of about 30 hops within the packet is required. Also, the fields that can be used to store path information in an IP packet are (Ehrenkranz & Li, 2009): Identification field (16 bits), Time Of Service (TOS) field (8 bits), Flags field (3 bits), Option and padding field. We propose the options and padding field (hereafter called options field) in the IP packet as it is typically not used in a DNS query and can be up to 40 bytes long. The Port-Mapping technique assumes that (a) an intermediate router has 8 to 48 interfaces or physical ports; thus, a router would require 3 to 6 bits to uniquely identify such ports (b) The underlying network is SDN-enabled.

As shown in algorithm 4.2, when a DNS request packet passes through a router, it forwards the packet to the SDN controller. The SDN controller inscribes the packet with the input interface (the in-port of the router) in the options field. Later, it pushes flow entry in the SDN switch to forward the packet through the out-port (outgoing interface) as per IP forwarding rules. When a DNS Response packet arrives at SDN switch, it again is forwarded to the controller. The controller removes the corresponding in-port number engraved earlier and routes the packet to that port. Rather, if the packet is not the DNS packet (i.e., packet.dest-port != 53 or packet.source-port != 53), it is treated as per normal IP forwarding rules. The options field is considered stack memory. So for all UDP packets with a destination port of 53, the incoming interface of the router will be pushed on the options field. Rather, if the source port is 53, the top element will pop out, and the packet will be forwarded to that interface. The steps to be taken by an SDN switch to implement Port-Mapping are explained in the Port-Mapping algorithm.

Algorithm 4.2 : Port-Mapping

```
Port-Mapping Algorithm (Switch-Side Processing)
INPUT- Packet coming at $r^i$
for i= 1 to n do
        if packet is udp and dst-port==53 then
            Send the packet to controller
        else if packet is udp and src-port==53 then
            Send the packet to controller
        else Forward the packet to out-port
        end if
end for
Port-Mapping Algorithm (Controller-Side Processing)
for each packet-in from switch do
    if packet is udp and dst-port==53 then
        if option==none then
            new-option=in-port
        else
            new-option=new-option + in-port
        end if
      // Create a packet with new-option field
        packet=create-new-packet (new-option)
        Forward the packet to out-port
  else if packet is udp and src-port==53 then
      // remove last field from option and put it in outgoing port
        out-port= option [-1]
      // create new option by removing last field
        new-option = option -1
        packet =create-new-packet (new-option)
        Forward the packet to new out-port
    end if
end for
```

4.4.3 PortMergeIP

This technique combines the strength of the Port-Mapping and IP-Switching techniques. The first-hop router of the ISP does IP switching. The information about the path from an end host to the organization's gateway router is stored in the options field of the IP packet in terms of physical port numbers, as proposed in Port-Mapping. Formally, the technique is explained in the PortMergeIP algorithm 4.3. By doing this, the packet does not require to go to the controller every time it hops; thus, the delay is reduced. The packet also reaches the attacker's network without getting lost.

Algorithm 4.3 : PortMergeIP

```
PortMergeIP Algorithm (Switch-Side Processing)
INPUT- Packet coming at $r^i$
for i= 1 to n do
    while rtr.id = $r_j^q$ and in-port = $r_j^{qk}$ or rtr.id = $r_j^q$ and in-port= $r_j^{ql}$ do
        if packet is UDP and src-port==53 then
            Send the packet to Controller
        else if packet is UDP and dst-port==53 then
            Send the packet to Controller
        else
            Forward the packet to out-port
        end if
    end while
    while rtr.id = $r_j^h$ and in-port = $r_j^{hk}$ do
        ipv4-src = SRC-IP-ADDR
        Forward the packet to out-port
    end while
    Forward the packet to out-port
end for
Port-Mapping Algorithm (Controller-Side Processing)
    Same as Port-Mapping
```

The advantages of the IP-Switching technique are (a) It prevents DRDoS attacks and completely saves the victim from attack traffic, as proven in section 4.4.4.2. As proven in section 4.4.4.2(c), it implements True Prevention at a little extra cost concerning the time required to resolve the legitimate query; the only change required in the underlying network infrastructure is in the functionality of the first-hop router of the ISP. Therefore, this router can be SDN-enabled, and the rest of the infrastructure remains the same. On the other hand, there is a limitation to this technique if the organization's gateway router is not Network Address Translation (NAT) enabled; reverse-path traffic will undoubtedly reach the correct network but might not reach the correct source/attacker.

The Port-Mapping technique has the advantage of correctly implementing True Prevention, and no attack traffic reaches the victim network. On the other hand, it also has a few limitations. These are (a) the existing free space in the IP packet is a bottleneck, (b) since the processing time at each intermediate router increases, it is comparatively a bit slower than its counterpart, (c) it requires a change in the functionality of each router along the path, and (d) if some link along the request – response path becomes non-functional while forwarding the response packet, the corresponding packet is lost. On the other hand, the PortMergeIP technique also correctly implements True Prevention; no attack traffic reaches the victim, reduces delay, correctly routes the reverse-path traffic to the attacker's machine or legitimate user, overcomes the bottleneck in terms of free memory requirement for storing physical port numbers, and requires minimal change in ISP network.

4.4.4 Results and Discussion

In this section the experimental setup done to validate IP-Switching, Port-Mapping, and PortMergeIP is explained. To validate these algorithms various parameters such as throughput, packet loss, etc are calculated.

4.4.4.1 Experimental Setup

As shown in Fig. 4.11, a topology was created in an SDN environment using mininet to implement and validate the proposed algorithms. It shows four access networks (access n/w-1 to access n/w-4). Access n/w-1 belongs to the victim's organization, and access n/w's 2, 3, and 4 are in control of an attacker. Any number of hosts in these access networks can launch the DDoS attack; we have used A1, A2, and A3 to launch the attack by spoofing the source IP address field of the packets to that of a host in the victim's network. Each access network is connected to the ISP's first-hop router through its gateway router. The ISPs' first-hop routers are assumed to know the downstream IP addresses. These first-hop routers are responsible for switching the IP addresses in the PortMergeIP and IP switching algorithms. To implement the proposed IP-Switching algorithm in mininet, we connect a node to the organization's gateway router, and the ISPs know the IP address of that node. It works as a NAT router and is responsible for forwarding the packet to the respective host machine. Besides this, DNS servers to generate a DRDoS attack and HTTP servers to generate traffic for analysis are also connected to the network. We used Ryu as the SDN controller and OpenFlow as a protocol for communication between the SDN controller and switch.

DRDoS attack traffic of approx. 1Gbps and 1 Mbps are generated using scapy. The attackers combinedly generate this attack. After attack generation, both algorithms are tested on various parameters described in more detail in section **4.4.4.2.**

4.4.4.2 Result and analysis

The three prevention algorithms, IP-Switching, Port-Mapping, and PortMergeIP, are tested on parameters like throughput and packet loss to check their validity against DDoS attacks. To avoid redundancy, we have shown the parameters graphs for only attacker A1, as the behavior of these parameters is similar for A1, A2, and A3.

A. *Throughput calculation for the victim's and the attacker's network*

Throughput is calculated on the victim's and attacker's sides from HTTP servers during an attack of 1 Gbps for no prevention, IP-Switching, and PortMergeIP. The bandwidth for throughput calculation at the victim's side and attacker's, when there is no prevention in place, is kept at 1 Gbps as an attack of approx. 1Gbps reaches the victim. For IP-Switching and PortMergeIP, the bandwidth is 400Mbps, as each attacker generates an attack of 400 Mbps (combinedly forming an attack of approx. 1Gbps towards the victim). For the Port-Mapping algorithm, as we are using only one controller, and all the packets go to a controller, the attackers generated an attack of approx. 1Mbps (each attacker generating an attack of approx. 450 kbps); hence the bandwidth is kept at 500 kbps. Wget is used for the calculation of throughput.

Figure 4.12 shows throughput for the time at the attacker's and victim's sides when no Prevention approach is in place, and Figures 4.13- 4.15 show when IP-Switching, PortMergeIP, and PortMapping are in place, respectively.

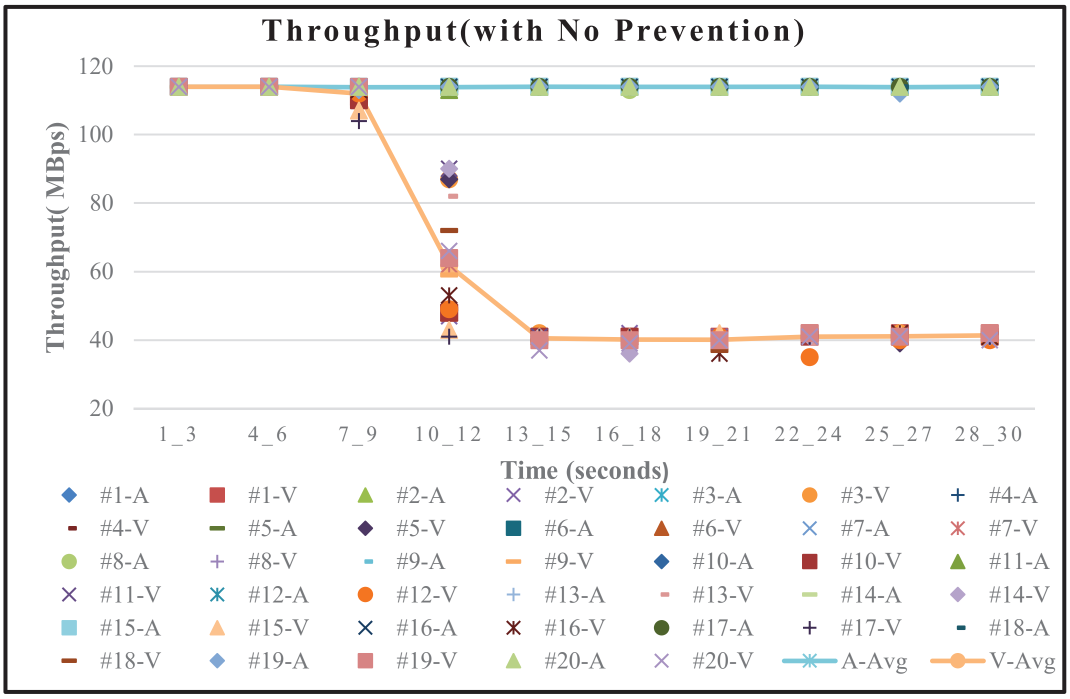


Figure 4.12 : Throughput without any prevention technique

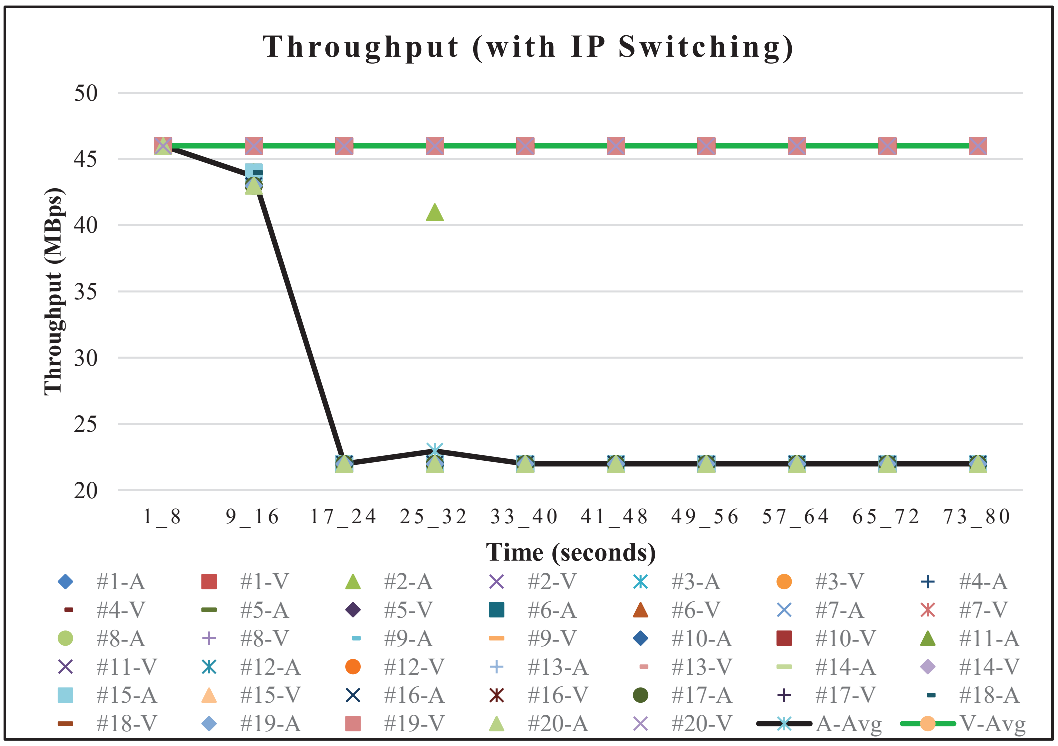


Figure 4.13 : Throughput (with IP-Switching)

Twenty iterations are done for throughput calculation, so #1-A represents throughput at the attacker in the first iteration, #1-V represents throughput at the victim in the first iteration, and so on. The graph's dots represent individual iterations, and the line represents the average.

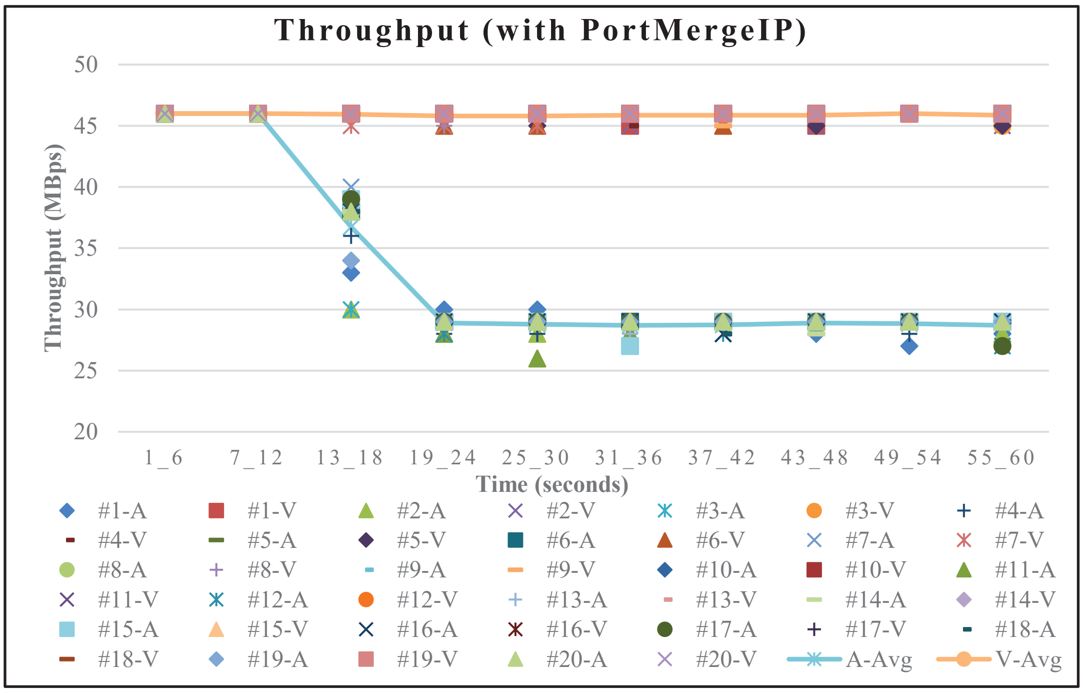


Figure 4.14: Throughput (with PortMergeIP)

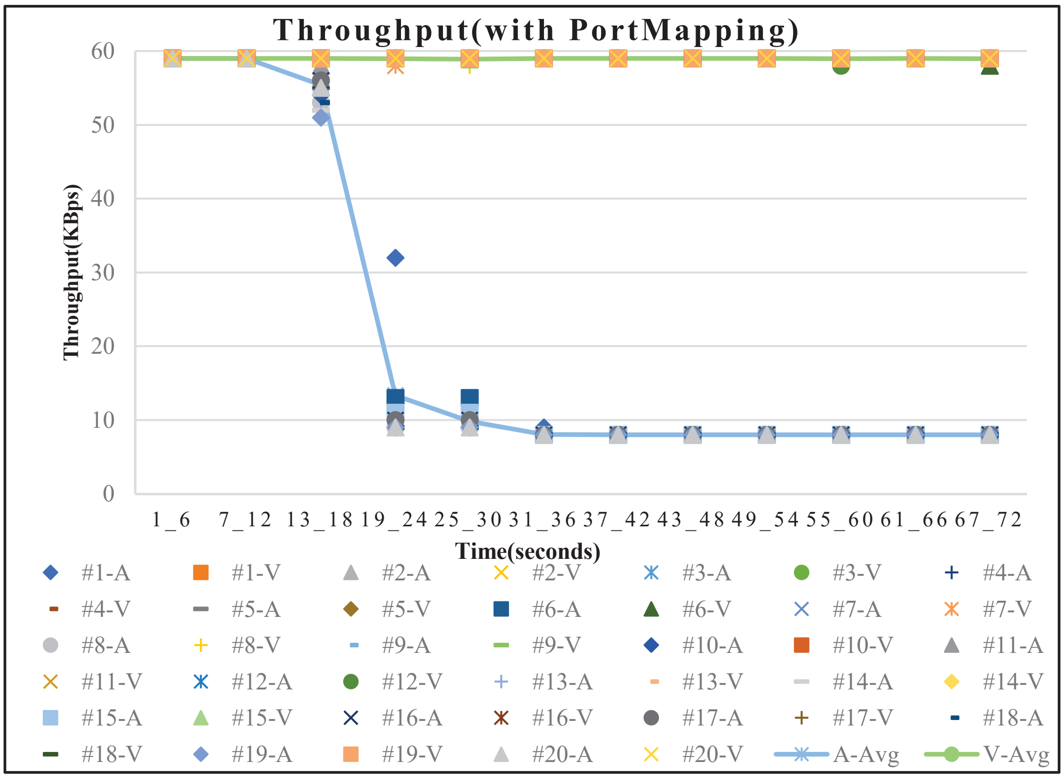


Figure 4.15: Throughput (with PortMapping)

It is clear from the charts that when there is no prevention approach in place, throughput drops at the victim as soon as the attack starts at the 15th second, and when prevention approaches are in place, throughput drops at the attacker, not at the victim. It is because, with prevention approaches, the response goes to the originator of the attack (the attacker), irrespective of the spoofed IP address; hence proving that the proposed approach prevents the attack. The respective data values are also shown in Appendix B.

B. *Packet loss due to attack*

We calculate the loss of packets due to an attack in the network in both cases when the prevention algorithms are in place (Figures 4.17-4.19) and when it is not (Figure 4.16). The intensities of attacks are kept the same as that for throughput calculation. The link bandwidth of the attacker's and victim's network is 500 Mbps for no prevention. For IP-Switching and PortMergeIP, the link bandwidth is 100 Mbps; for Port-Mapping, it is kept at 300 kbps. Ping is started between attacker A1 and the first HTTP server and between V1 and the second HTTP server to calculate packet loss.

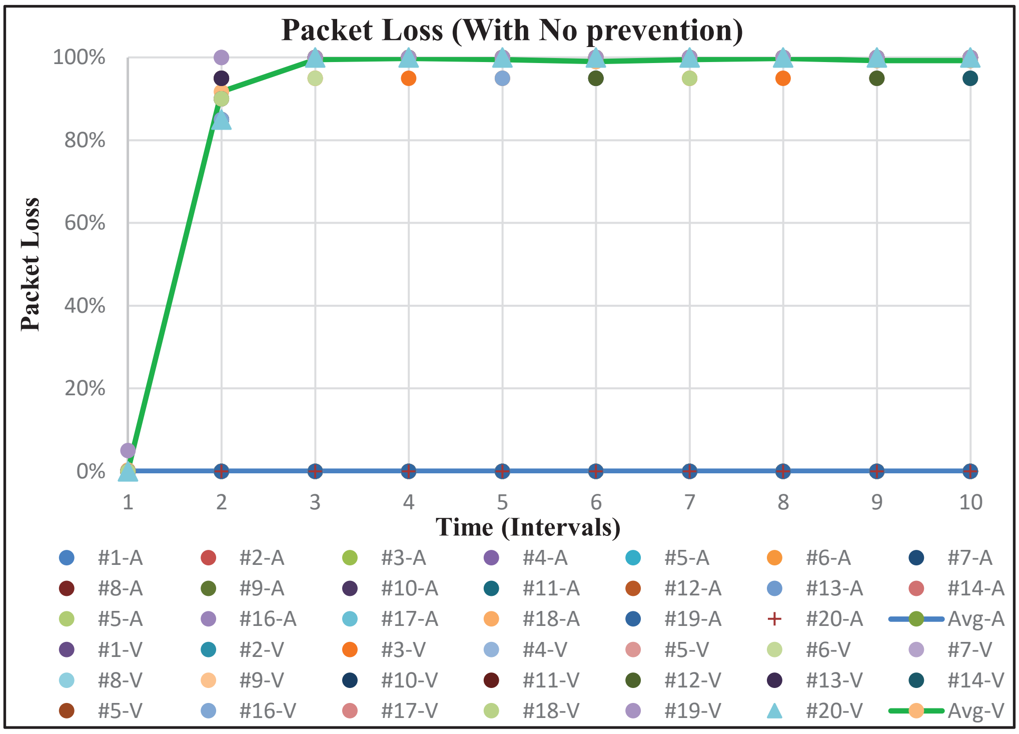


Figure 4.16 : Packet loss (without any prevention technique in place)

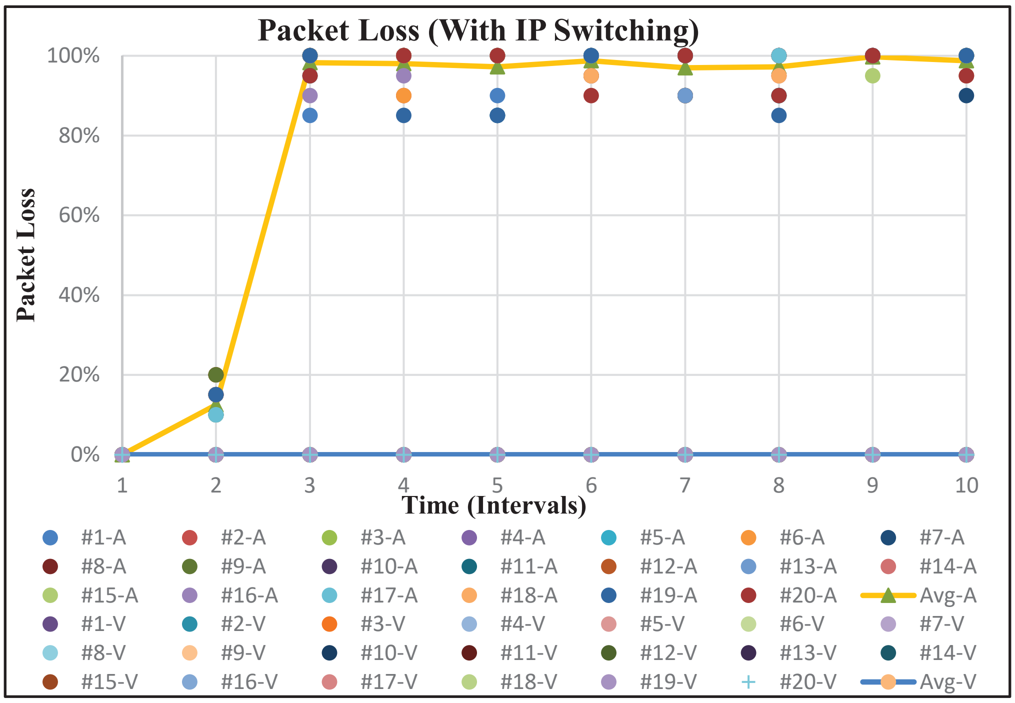


Figure 4.17 : Packet loss w.r.t IP-Switching

As soon as the attack begins, packet loss occurs at the victim, while no packet loss occurs at the attacker (Figure 4.16). But when our proposed approaches are in place, the victim remains unaffected, and packet loss occurs at the attacker (Figures 4.17-4.19). This experiment is also performed for twenty iterations.

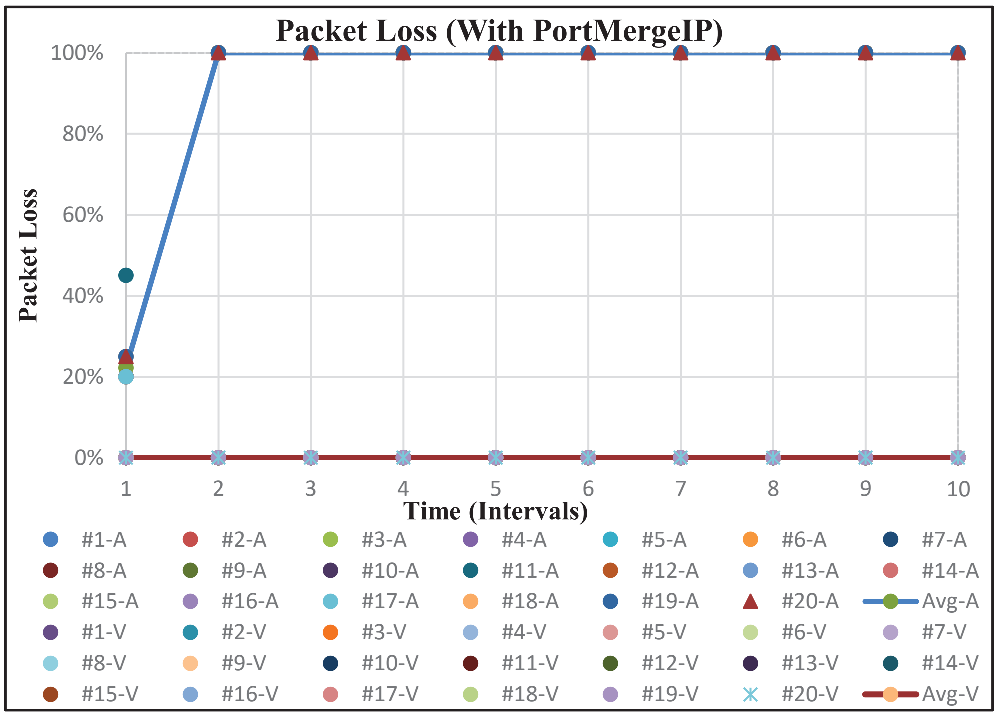


Figure 4.18: Packet loss w.r.t PortMergeIP

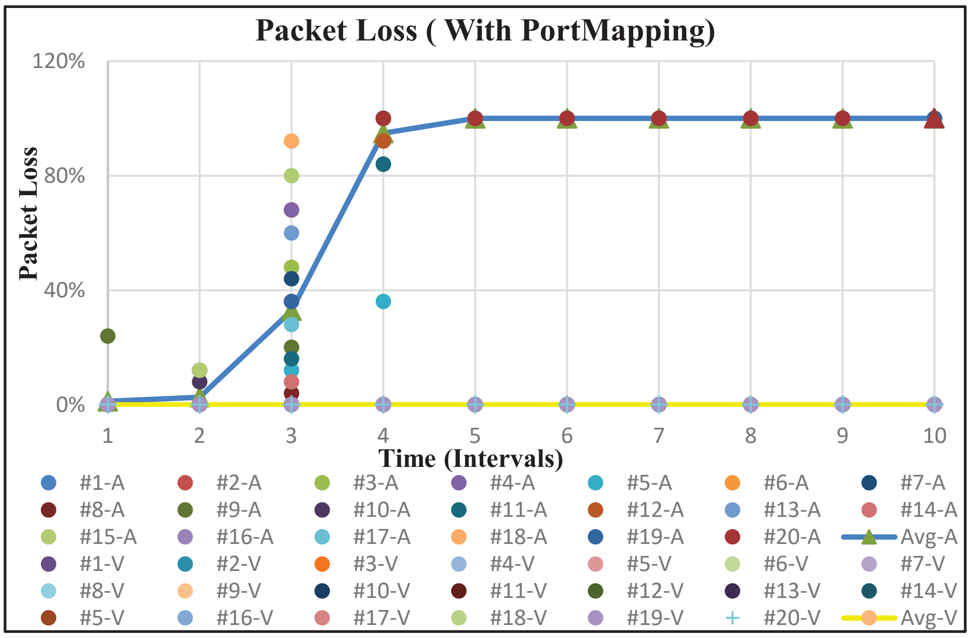


Figure 4.19: Packet loss w.r.t PortMapping

C. *The delay in DNS request response due to prevention algorithms-*

The proposed DDoS prevention algorithms introduce an extra delay in the DNS request-response packets. Therefore, we need to measure this delay. We calculate it for the proposed algorithms and the authentication approach. Figure 4.20 shows the delay between DNS request-response packets when the prevention algorithms are implemented. Baseline delay indicates the average delay without any prevention algorithm, which is nearly 0.09 seconds. After this, PortMergeIP has slightly more delay as the packet goes to the controller after IP is switched.

Furthermore, the IP switching technique has more delay than PortMergeIP as after IP-Switching, we have implemented the NAT router mechanism to route the packet to the actual host by connecting the gateway router to a host mimicking NAT router functionality. It is the limitation of our simulation environment. The Port-Mapping technique delays slightly more as all packets go to the controller at each router before they are forwarded. The highest delay is in the authentication, as it needs one additional RTT before the server receives the DNS response.

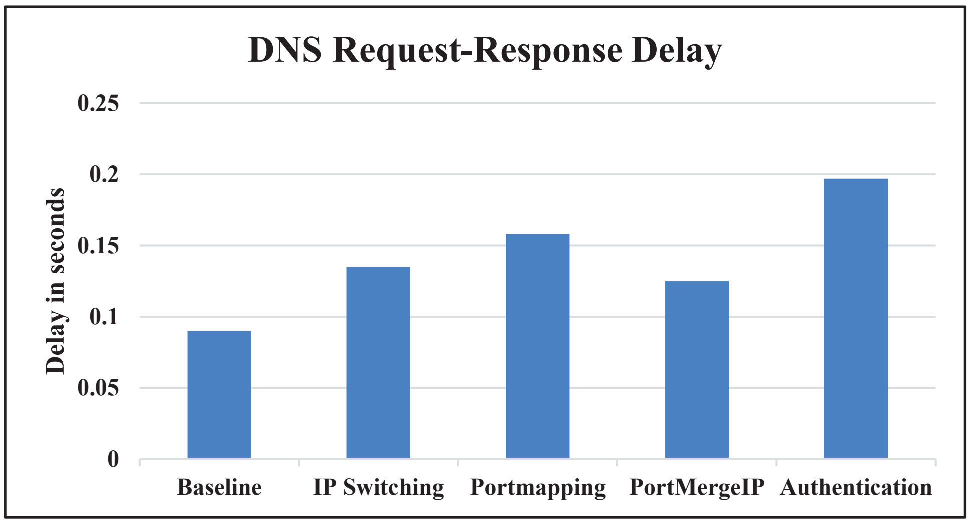


Figure 4.20 : DNS request-response delay due to prevention algorithms

4.4.4.3 Attack mitigation

The throughput and packet loss results show that the targeted victim is always safe as no spoofed traffic reaches the victim. All the attack traffic is sent to the attacker. This leads to constraining the attacker's available bandwidth and temporarily mitigating the attack. This can be shown through Bandwidth Delay Product (BDP).

The congestion in the attacker's network is calculated to prove that the attack will be mitigated for some time using our algorithms. In the results section, we have shown how our algorithms penalize the attacker's network. Now, using the BDP, we show how the congestion in the attacker's network changes. As the name suggests, the BDP is the product of bandwidth and delay in the network (Bandwidth-Delay Product). It tells about the data, which is yet to be acknowledged, present in the link at any given time. It is the in-flight data often referred to as the window size. To put it formally, if c is the data rate of a link ("bandwidth") and RTT is the round-trip time delay, then the BDP defines the window W given by eq (4.1).

$$W = c \times RTT \text{ .... } (4.1)$$

So, through this, we can find the congestion in the network. With no prevention algorithm in place all the attack traffic goes toward the victim's network. The BDP of victim's network increases gradually until the allowed TCP window size of all the links in between is reached, as shown in Figure 4.21. As the attacker constantly sends request packets to the DNS server, which in turn sends responses to the victim, the victim's network is in continuous congestion, and out of 1000 packets sent, only 276 are received.

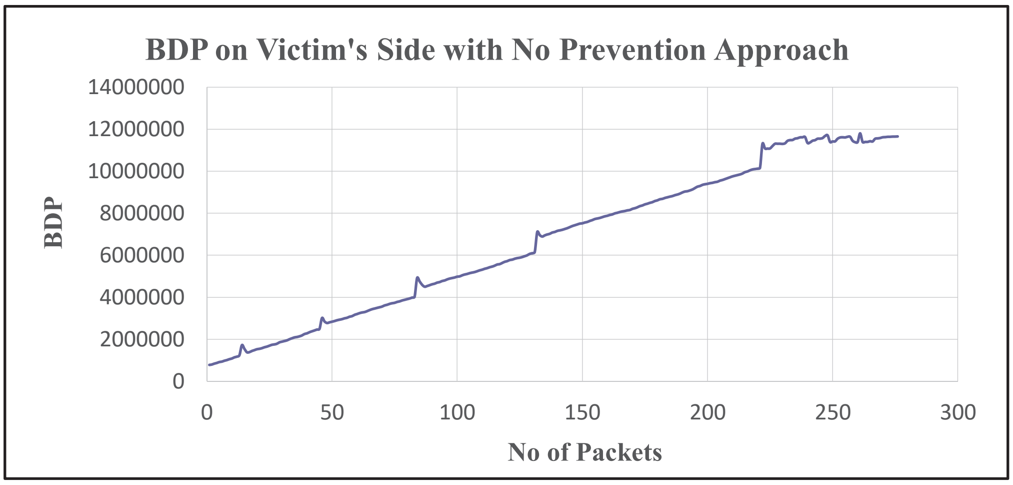


Figure 4.21 : Congestion in victim's network when there is no prevention algorithm

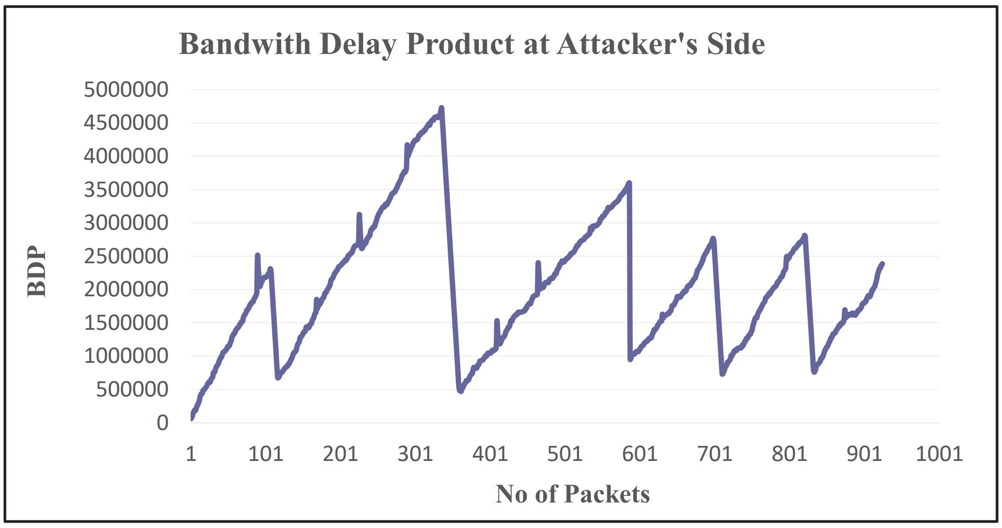


Figure 4.22 : Congestion in attacker's network when prevention algorithm is in place.

A change in the behavior of the BDP line on the attacker's side can be seen in Figure 4.22. As shown in Figure 4.22, the BDP in the network does not increase gradually till it becomes constant; instead, it oscillates. When the attacker's network is congested due to all the DNS responses coming to it, the attacker cannot send any more request packets; as a result, there is no response from the DNS server, and for that duration, the attacker's network becomes congestion free. Now the attacker can send packets again, leading to another congestion, and the same process repeats, i.e., the reason why the B-D product line oscillates. This process repeats till the attacker stops sending the packets. The attack is mitigated for the amount of time the attacker cannot send packets. Hence as per the definition provided of True Prevention, the attack is mitigated automatically for some time. Based on these readings, if the ISP wants to, it can block the attacker, thus mitigating the attack. This time depends majorly upon the bandwidth of an attacker. Moreover, through the proposed techniques, the attacker will always attack itself; hence the techniques will penalize the attacker.

4.5 Summary of the Chapter

In this chapter, we proposed four techniques to prevent DRDoS attacks with implementation and results specific to DNS amplification attacks. All the proposed techniques use the common philosophy to equip underlying network infrastructure with enough rules/intelligence to enforce reverse path forwarding, the same as that of corresponding forward-path forwarding of the client-server communication. The proposed approaches provide prevention against DRDoS attacks focusing on reverse forwarding with two subcategories. With such rules, an attacker would attack itself. The implementation results show that the targeted victim is always safe as no spoofed traffic reaches the victim. With the proposed approaches, the victim remains unaffected while the attack traffic goes to the attacker itself, choking its bandwidth. This leads to attack mitigation for some time T.

The authentication approach (Sahri & Okamura, 2016) is the algorithm we used to compare our techniques by focusing mainly on the additional RTT needed to get the final DNS response. The reason for picking this algorithm is that they also prevent DNS amplification attacks, and its underlying architecture is an SDN environment. SymSDN shows a delay of .093 seconds and the authentication algorithm shows the highest delay of .15 seconds. For IP Switching, PortMapping, and PortMergeIP, the highest delay is shown by the authentication approach only as it needs one additional RTT before the server receives the DNS response.

********** End of Chapter **********

CHAPTER 5- Prevention Of DRDoS Attacks with Reliable-Dynamic Path Identifiers

5.1 Introduction

As mentioned in chapter 1, shifting the defense domain from detection to attack prevention is a more promising strategy. In this chapter, to prevent DRDoS attacks, we use PIDs. PIDs are used as a packet routing mechanism for ICNs (H. Luo et al., 2017), (Hongbin Luo et al., 2014). We propose using these PIDs to prevent DRDoS attacks by ensuring that the response packet path is the same as the corresponding request packet, like in previous chapter. Different authors use PIDs to avoid DRDoS attacks (H. Luo et al., 2017), (Hongbin Luo et al., 2014), (Al-Duwairi et al., 2020). Because of PIDs, DRDoS attacks are not possible even if the source-IP addresses are spoofed because the response packet is not forwarded using regular IP forwarding (H. Luo et al., 2017). Instead, it is based on the PIDs stored in the corresponding request packet. (Hongbin Luo et al., 2014) propose these PIDs as static. Static PIDs prevent DRDoS attacks but make the network vulnerable to flooding attacks. If an attacker sniffs these PIDs (by sniffing the network packet), a DDoS attack can be launched toward the sender of these packets. Hence, DPIDs are proposed by (H. Luo et al., 2017). However, the proposed approaches are prone to packet drops during PID updates.

In the preceding chapter, we discussed storing path information within the packet to ensure that the response path aligns with the request path. In this chapter, we present a novel concept called Reliable-Dynamic PIDs (RDPIDs) as a means of preventing DRDoS attacks. RDPIDs are dynamic in nature, making them effective against DRDoS attacks and flooding attacks (possible because of MITM learning static PIDs). They offer enhanced security against DDoS attacks through the innovative incorporation of Reserved PIDs (RPIDs) and Open PIDs (OPIDs). Additionally, RDPID boasts reduced PID negotiation delay compared to a prior study

(H. Luo et al., 2017). Further details on RDPID are provided in section 5.3, while the subsequent section outlines the existing literature on the utilization of PIDs.

5.2 Related Work

In this section, related work concerning DDoS prevention using PIDs is presented. Instead of using traditional IP forwarding to transmit packets, (Godfrey et al., 2009) have proposed using pathlets, which are inter-domain routing objects. The authors propose to use a standard, wireless path-vector protocol with a pathlet declaration containing forward identifier for pathlets and a sequence of vnode identifiers. CoLoR, an approach proposed by Luo et al. (Hongbin Luo et al., 2014), says that the future Internet will continue to be organized in domains with provider/consumer/peer relationships. Every domain has a logical resource manager, which maintains a registry table that stores content access information identified by source identifiers. CoLoR also uses two local namespaces: domain-based connectors and route identifiers (PIDs). The domain in CoLoR can use different local intra-domain protocols. Inter-domain routing is based on PIDs. In CoLoR, users send GET messages to find their desired content. In the GET messages, PIDs of the inter-domain path are stored in the packet. The routing from the content provider to the content consumer is done based on these PIDs. The PIDs are static, so a flooding attack is still possible. The other approach proposed by (H. Luo et al., 2017) describes how flooding attacks are still possible if the attackers can discover these inter-domain identifiers. The ways to know the PIDs are also described in detail, i.e., GET luring and botnet cooperation. The attackers learn the PIDs between there and the victim's network through these ways. The PIDs are static and do not change, so the attackers can launch a successful attack. That is why (H. Luo et al., 2017) proposed DPIDs, so even if the attackers learn the PIDs and launch the attack, the attack cannot continue as the PIDs will change after some time. We also used DPIDs and compared our approach with the one proposed by (H. Luo et al., 2017). (Al-Duwairi et al., 2020) have also used the concept of DPID with Get-message logging. This approach is mainly for ICNs where users request information using Get messaging. The reason for logging Get messages for DDoS detection and prevention is that normal users correspond to a Get message while an attacker does not. Here, the ICN routers log Get request messages using bloom filters. Bloom filters will help in comprehensive logging, and in return, they don't even take up much space. This approach is claimed to give better results against DDoS Prevention in comparison to DPID.

5.3 Proposed Methodology

The proposed technique for DDoS prevention uses PIDs described in (H. Luo et al., 2017) to identify the inter-domain paths. Here domain refers to an independently maintained network connected to other domains via Border Routers (BRs). A domain might be an AS's or ISP's network in the Internet architecture. We assume that these domains are numbered D_1, D_2… D_n. Each domain has one or more BR through which network traffic is routed to another domain. Assume these BRs in each domain are also numbered B_1, B_2, …, and B_m. In addition, each domain has a Resource Manager (RM) whose task is to share PIDs with the BRs to maintain inter-domain routing. Figure 5.1 shows all these network elements. It shows six domains (D_1 to D_6), BRs in each domain, RMs, and hosts. Host-1 and host-2 are located in domain D_1, and host-3 is in D_2. It also shows a DNS server in D_3. For each link connecting the two BRs, a PID is generated and shared before any communication can start. These PIDs are generated and distributed by a central entity called Network Manager (NM). Although we assume NM to be a centralized entity in this work, it can be maintained as a distributed system.

For more clarity, let’s take an example. Suppose, for inter-domain routing, PID1 is used between the domains D_1 and D_4, PID2 between D_2 and D_4, PID3 between D_4 and D_5, PID4 between D_5 and D_6, and PID5 between D_6 and D_3. These PIDs are negotiated and shared with the BRs before any communication can start. Also, suppose host-1 in D_1 generates a DNS request packet for the DNS server in D_3. This packet will be routed to D_3 using any inter-domain routing protocol, as described in (Avramopoulos & Suchara, 2009). The respective BRs in between will push (i.e., store) the associated PIDs in the options and padding field of the network packet’s IP header. Thus, PID1, PID3, PID4, and PID5 will be stored in the DNS request packet.

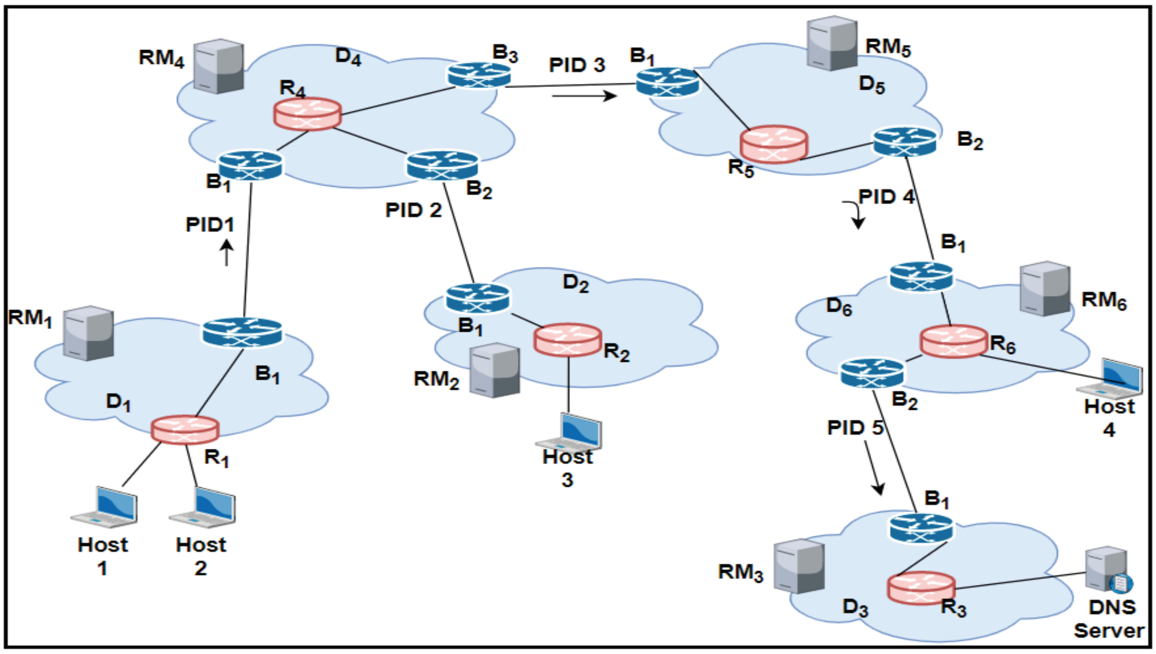


Figure 5.1: Network setup showing Domains, Bord Routers, and Resource Managers.

The corresponding response packet will follow the same path using these PIDs (rather than IP forwarding) to reach back to host-1. With this approach, even if source-IP spoofing is done by host-1 for any potential DRDoS attack, the response packet is assured to go back to the attacker itself. Thus, an attacker will attack itself. There are two following possibilities with PID assignments.

a) As proposed by (Hongbin Luo et al., 2014), PIDs can be assigned once and can be made static. This approach has a disadvantage in that an attacker can learn these PID sequences by sniffing the packets or deploying a honeypot "server" in the network. A sniffed packet (or a packet coming to the honeypot) from some host-H will have a PID sequence. The reverse of this sequence refers to the PID path back to the host-H. An attacker could use this knowledge to launch a flooding attack on H.

b) To overcome the above problem, as proposed in (H. Luo et al., 2017), PIDs can be made dynamic in the sense that they change after every fixed time interval T. Of course, this technique results in an increase in network traffic during PID updates. The value of T being less results in network congestion and T being more makes the source host susceptible to flooding attack, as with static PIDs.

We propose RDPID to tackle the above issues. The proposed methodology is explained using three points: (a) RDPID Generation, (b) Request-Routing, and (c) Response-Routing

5.3.1 Reliable-Dynamic PID (RDPID) Generation

Figure 5.2 shows the RPID and OPID generation for each link interface connecting two BRs. If any two BRs (BR_1 and BR_2 as shown in Figure 5.2) are connected with a link L, the RPID for an interface of BR_1 becomes OPID for an interface of BR_2 and vice versa.

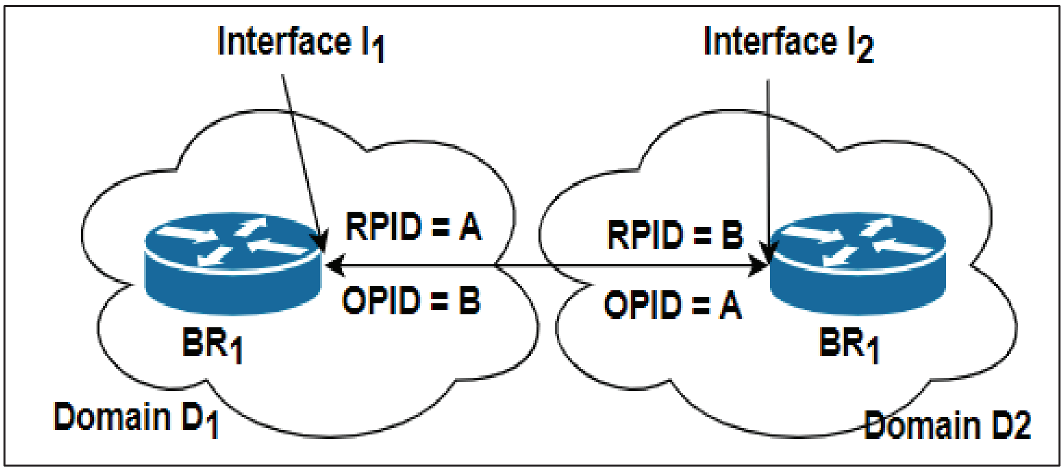


Figure 5.2 : Reserved and Open PIDs

Algorithm 5.1 : PID Generation Algorithm

```
Let
D be the total number of domains, numbered 1... D
$B_D$: Total number of Border Routers (BR) in Dth domain, numbered 1…$B_D$, for all D
$A_{IJ}^{KM}$ : Link_ID between Ith domain's Jth BR and Kth domain's Mth BR ($1 \leq I \leq D$, $1 \leq K \leq D$, $1 \leq J \leq B_I$, $1 \leq M \leq B_K$)
$N_{IJ}^{*}$ : Number of links to which Jth BR of Ith domain is connected to.
RPID ($A_{IJ}^{KM}$): Reserved PID for Link_ID $A_{IJ}^{KM}$
OPID ($A_{IJ}^{KM}$): Open PID for Link_ID $A_{IJ}^{KM}$
[Note that RPID ($A_{IJ}^{KM}$) = OPID ($A_{KM}^{IJ}$)]
PID_Generation (Time interval T)
{
 Input T: time interval for PID update.
 Output: Pair [RPID ($A_{IJ}^{KM}$) , OPID ($A_{KM}^{IJ}$)] for all Link_ID's.
 Repeat after every time interval T
 {
  For d = 1 to D
      Generate_Keys (d)
 }
 Distribute all [RPID ($A_{IJ}^{KM}$) , OPID ($A_{KM}^{IJ}$)] pairs for all Links via RM's.
}
Generate_Keys (d)
{
 Input d: Domain for which key pairs are to be generated.
 Output: Unique RPID and OPID for each link interface.
 For b = 1 to $B_d$
 {
   For p = 1 to $N_{db}^{*}$
      Key = GENERATE_UNIQUE_KEY();
      x = domain to which pth link of $B_d$ is connected to.
      y = BR to which pth link of $B_d$ is connected to.
      RKEY ($A_{db}^{xy}$) = OKEY ($A_{xy}^{db}$) = key
 }
}
```

The procedure PID_Generation(), shown in Algorithm 5.1, runs periodically after every time interval T. For each domain, procedure Generate_Keys () generates RPIDs for each link of each of its BR. The procedure assumes a function GENERATE_UNIQUE_KEY(), which generates a unique key for a specific domain. Finally, the NM communicates all the pairs for a domain with the corresponding RM, which shares the keys with its BRs. Now, these shared PIDs can be used for communication. It should be noted that while each PID is not necessary to be unique in the network, each PID must be unique inside a domain. In other words, these PIDs are used to identify neighboring devices linked to a single device; thus, they do not need to be unique across the network but only within a domain.

5.3.2 Request Routing

While forwarding a request packet, a BR, before sending the packet to another BR, must append the open PID received from that BR within the options field of the IP header of a network packet. The options field can be considered as a stack, and the PIDs are inserted on top of this stack. The BR then consults these "stacked" PIDs to receive the response packets.

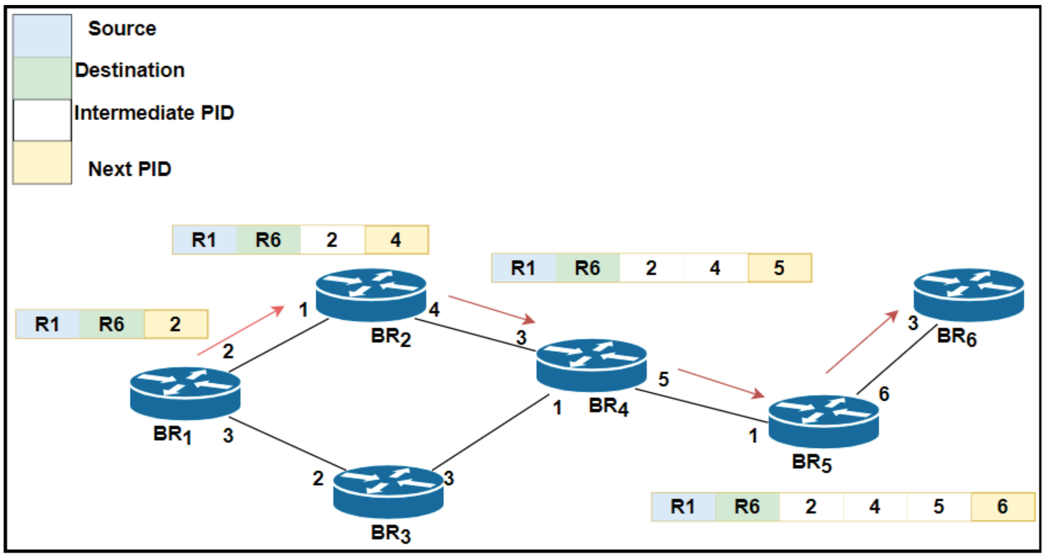


Figure 5.3 : Request packet routing

Figure 5.3 depicts the entire request packet routing. It shows six BRs, and each is assumed to be in a different domain. The link interface of each BR shows the OPID of the next BR on the link. The request packet is to be forwarded from BR_1 to BR_6. Based on the standard

IP routing, BR_1 should first send the packet to BR_2. Hence, BR_1 appends the PID value two within the options field of the packet and sends it. BR_2 verifies this PID for correctness upon receiving the packet; it should be its RPID. If the appended PID is malicious, the packet is dropped.

Similarly, BR_2 now appends the PID value 4 to send the packet to BR_4. In this manner, the entire routing is performed. The packet contains the entire flow indicated through the PIDs at the destination. In this case, it is (2,4,5,6). This PID sequence is then used to trace the packet back to the source.

5.3.3 Response Routing

Once the request packet has been received at the destination, the PID sequence is referred for forwarding the corresponding response packet. Each BR maintains a PID table that stores RPIDs and OPIDs of its interfaces.

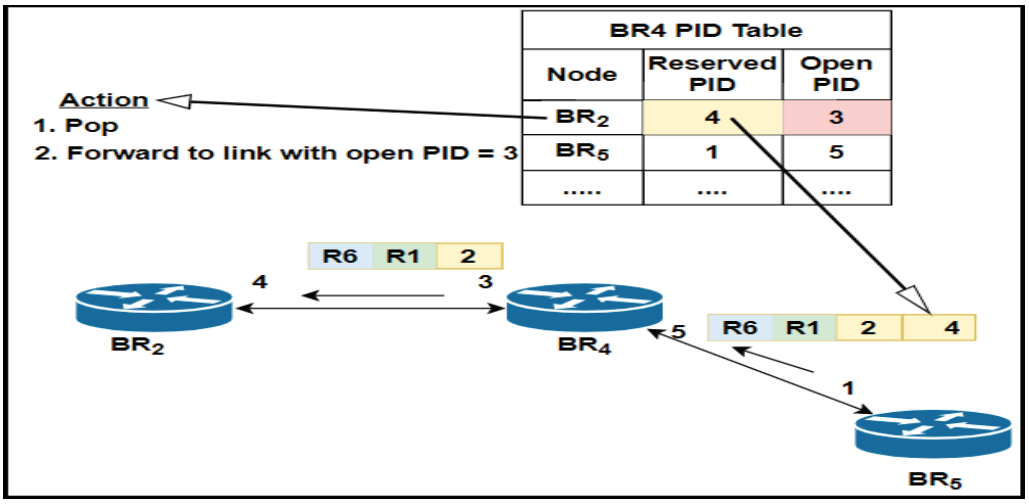


Figure 5.4 : Response packet routing

The last PID in the options field (pointed to by the top of the stack pointer) is referred to determine the subsequent BR to which the packet should be forwarded. Once determined, the PID pointed at the top of the stack is popped, and the remaining PID sequence is sent along with it. Figure 5.4 portrays the flow. It shows the PID table of BR_4. When BR_4 receives the response packet with PID sequence (2, 4), with 4 being the last PID, it refers to its own PID table to determine the next node on which packet would be forwarded. This last PID is then popped, and the packet with the remaining PID sequence is sent further. This exact procedure is then continued further till the packet reaches the destination.

5.4 RDPID v/s DPID

In the past, single PIDs were used to combat DDoS attacks; however, they are vulnerable to other forms of attacks. Instead, using two PIDs (open and reserved) for each link interface helps secure the network further. For example, it secures the network from Syn flood type of DDoS attacks (also called (half-open attacks)) (Imperva). The attacker sends a half-open connection request to the server as a client of this type. Of course, the source IP in this request packet can be spoofed. The server acknowledges the connection and waits for the final acknowledgment. Since the source IP was spoofed or otherwise, this acknowledgment will not arrive; thus, server resources are wasted.

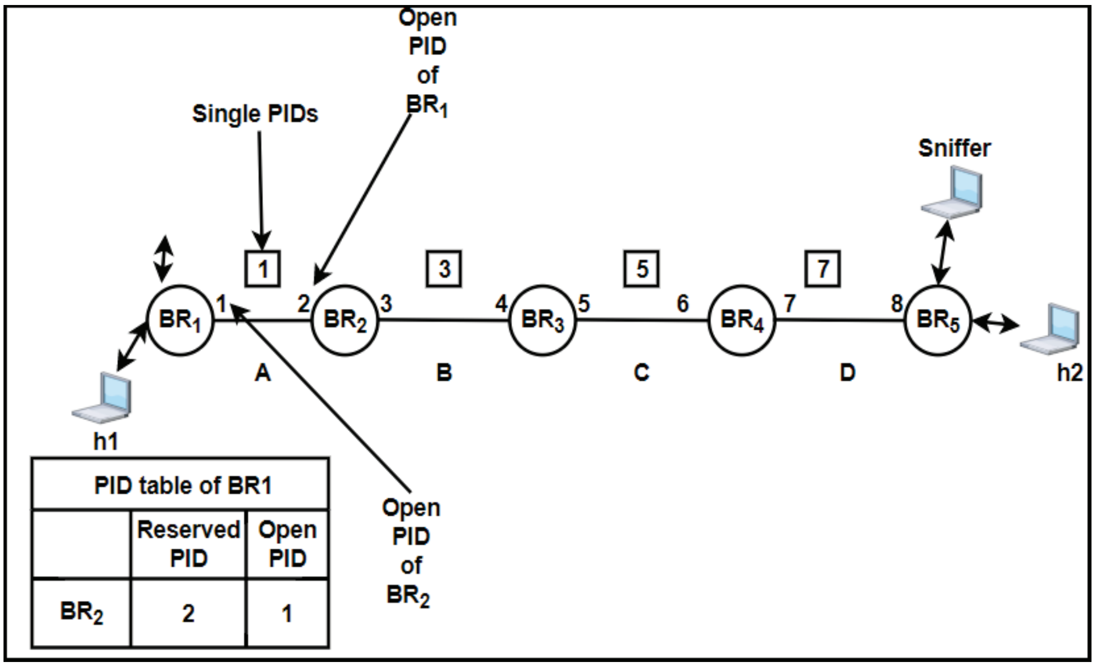


Figure 5.5 : Advantages of using RDPID over DPID

Consider the scenario depicted in Figure 5.5. Assume that host h1 sends a request packet to host h2, and the underlying system uses a single PID for each link (shown within a square box on top of each link). The routers in between would append the PIDs [1, 3, 5, 7] as the packet traverses through the links A, B, C, and D. Now, if the host h2 has to send a request packet to h1, the PIDs would also be [1,3,5,7], but appended in reverse order, i.e. [7, 5, 3, 1]. A sniffer at BR5, trying to sniff incoming packets, would know this PID sequence for communication between h1 and h2. As a result, an attacker can construct a spoofed request packet somewhere between h2 and h1, thus launching a flooding DDoS attack against h1.

With two PIDs per router interface, such attacks are harder to launch. Here, the sequence remains the same for the packet from h1 to h2, i.e., [1,3,5,7]. However, if h2 generates a packet for h1, the PID sequence would be [8,6,4,2]. Therefore, after sniffing the request packet generated from h1 at BR_5 and getting the PID sequence [1, 3, 5, 7], when an attacker generates a spoofed request packet from h2 to h1, the packet will be discarded at BR_4; as for it to forward any request, the PID appended should be 8. But, of course, an attacker can launch a flooding attack toward host h1 using response packets. Its mitigation time depends upon the RDPID update interval.

5.5 Simulation and Results

The topology shown in Figure 5.1 was emulated in mininet, using Click (Kohler et al., 2000) as a router. The purpose is to demonstrate that the proposed technique can prevent DDoS attacks. Click's extensible language for deployment enables the creation of highly customizable router functionalities. This way, routers can be implemented in Linux hardware more efficiently. Also, Click achieves a very high forwarding rate per second. To do this, Click removes the interrupt-driven architecture favoring polling, avoiding expensive context switches and memory accesses. Mininet is a network emulator or, more accurately, a network orchestration system. It works with a set of endpoints, switches, routers, and a single Linux kernel connection.

The proposed technique's effectiveness is shown to prevent the DRDoS attacks from reaching the victim, prevent flooding attacks, and the effect of keys distribution on network functioning. The measured parameters are compared with the DPID technique as presented in (H. Luo et al., 2017).

To validate the prevention of DDoS attacks, in the simulated network (Figure 5.1), host-1, as an attacker, generates DNS request packets with source IP spoofed to that of host-3. To send the request packet from h1 to the DNS server configured in D3, the PID's are updated by the NM only on the three intermediate RM's. It saves the time for updating because it is not needed for border domains.

To test and validate the resilience of RDPID-based routing against preventing DRDoS flooding attacks, host-1 in Figure 5.1 sends the DNS request packets to the DNS server, which is configured as a network honeypot (or otherwise) that allows an attacker to sniff the packets. Thus, an attacker in D_3 learns the PID sequence and launches a flooding attack of 20 Mbps against host-1. This attack rate was chosen to compare the effectiveness of RDPID with that of DPID, as proposed by (H. Luo et al., 2017). As shown in Figure 5.6 four different scenarios were taken. First, when PIDs are static (i.e., no RDPIDs). Second, when the RDPID routing scheme is in place and its update period is 30 seconds. Third, when the RDPID update period is 60 seconds; fourth, reduce the RDPID update period to 10 seconds upon detection of an attack. Three of these four scenarios (i.e., first, second, and fourth) are also considered by (H. Luo et al., 2017) to show resiliency against attacks. As expected, the victim host-1 is subjected to a full 20Mbps attack with static PIDs. The attack gets mitigated in 33.2 seconds with an RDPID update period of 30 seconds. With an update period of 60 seconds, the attack gets mitigated in 70 seconds (as against 115 seconds reported in (H. Luo et al., 2017)). Also, with an update period of 10 seconds after attack detection, it got mitigated in 14 seconds (as against 18 seconds reported in (H. Luo et al., 2017)).

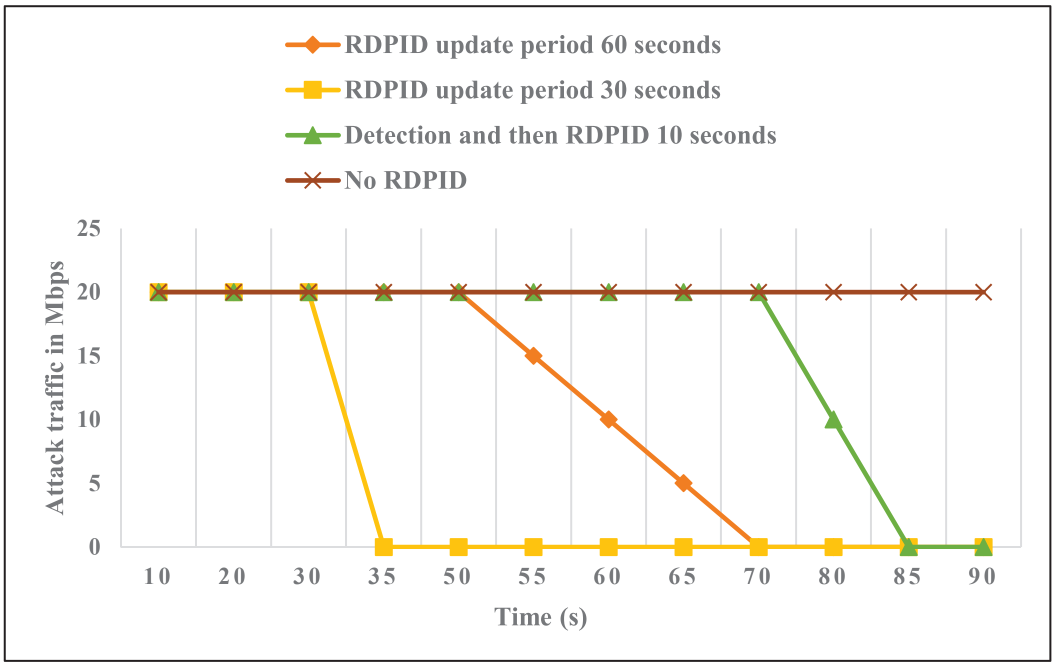


Figure 5.6 : Prevention of DDoS attacks

The performance of RDPID was further evaluated using PID update delay. It is the sum of time NM takes to share the RDPIDs with the intermediate RMs and then RMs to share them with its BRs. 20000 PID update delay samples were recorded by running the proposed RDPID distribution in the topology. As we compare our proposed work with (H. Luo et al., 2017), we only calculate the CDF for OPID update delay. Figure 5.8 shows the CDF for the OPID update delay, which is 10 ms. (H. Luo et al., 2017) calculated the PID negotiation delay and PID distribution delay. The former refers to the interval between when an RM sends the PID update message to its neighbor RM and receiving a corresponding acknowledgment. The latter is when an RM sends the PID distribution message to a BR and receives the corresponding acknowledgment. The sum of these two delays is the PID update delay. (H. Luo et al., 2017) reported an average of 20 ms for PID negotiation and 3 ms for PID distribution, a total of 23 ms. With RDPID, the OPID (which is RPID for opposite BR) update was only 10 ms. Figure 5.8 shows that it was only 6 ms with a 99% probability. This reduction can be attributed to the fact that in our approach, RMs do not negotiate RDPIDs; NM's job is to update all the PIDs to RMs.

The loss of packets caused due to RDPID update is shown in Figure 5.7. As can be seen, with a PID update period of 50 seconds, the loss is 1%, and with that of 100 seconds, it reduces to 0%.

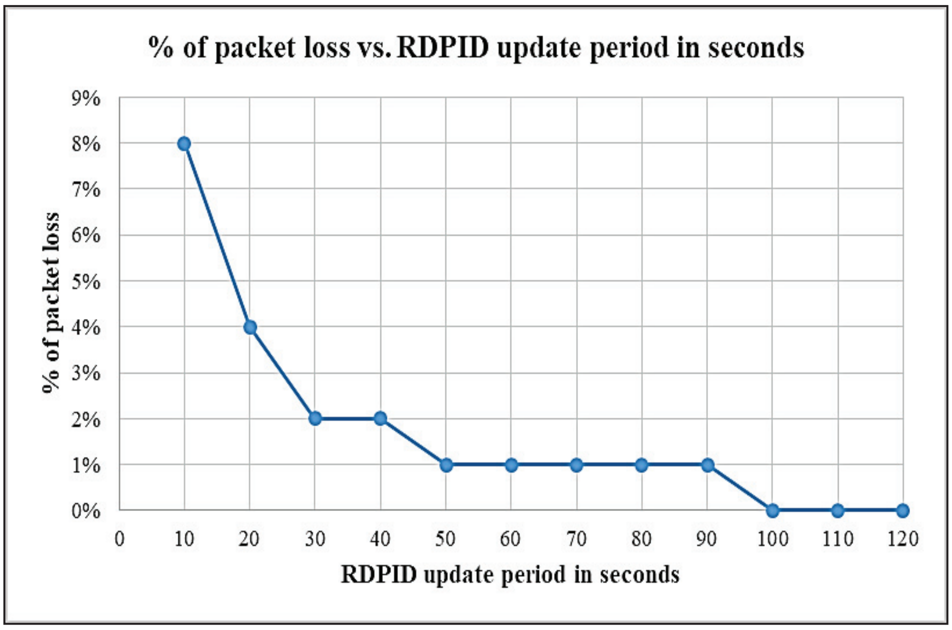


Figure 5.7 : Loss of packets due to PID update

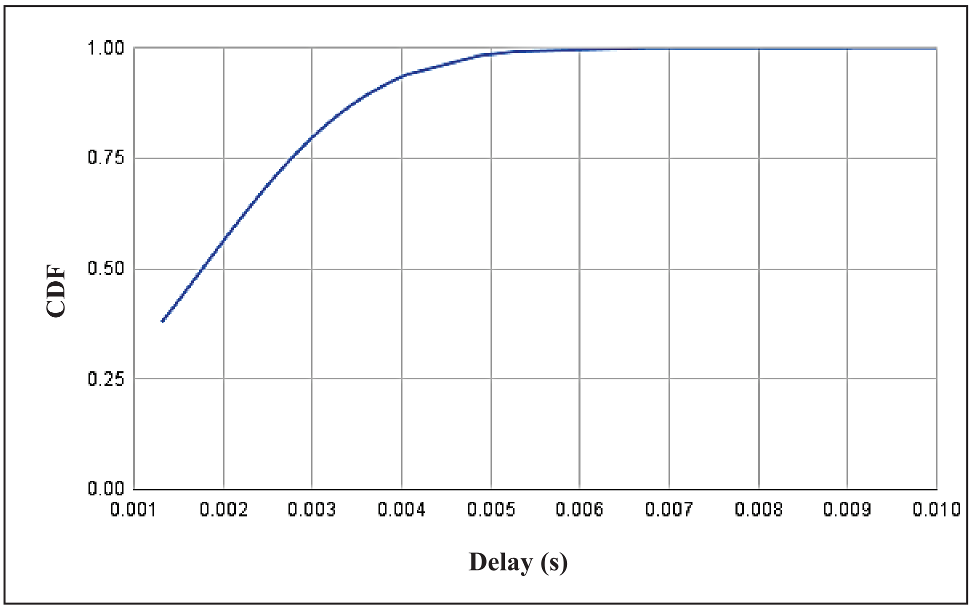


Figure 5.8 : Time taken to update PIDs

Further, the probability P of determining the PID depends on the number of hops (h) and the size of character space $\boldsymbol{S_{ch}}$. It is given by eq (5.1).

$$P = \frac{1}{S_{ch}^{h}} \qquad \ldots\ldots\ldots\ldots\ (5.1)$$

Our experiment has a network with four hops for a packet to reach the DNS server and a character space of 52 characters. Even in this small network, the probability of deducing the correct PID sequence by an attacker is $\approx 10^{-7}$, as it has more than 7 million possibilities. Further, with the PID change interval of 30 seconds, for an attacker to sustain an attack of 1 minute, the probability reduces to $\approx 10^{-14}$, having more than 50 million possibilities of different PID sequences.

5.6 Summary of the Chapter

This chapter proposed a DDoS prevention approach using RDPIDs against DPIDs primarily used for ICNs. The experiments were conducted using DNS-based request-response client-server inter-domain architecture. It is proved that with RDPID based routing mechanism in place, the victim is constantly prevented from DRDoS attacks. Also, a flooding attack is automatically mitigated within some constant time, depending on the PID update period. If the attacker tries to guess the used DPIDs, there are approximately 50 million possible PID sequences to guess, making it a computationally costly business for the attacker. Via different simulations, it is shown that with our proposed approach, the PID negotiation time reduces to 6 ms, as against 23 ms reported in (H. Luo et al., 2017). Also, the attack gets mitigated in 70 seconds with a PID update period of 60 seconds (as against 115 seconds reported in (H. Luo et al., 2017)).

********** End of Chapter **********

CHAPTER 6- PODIBC: Prevention of DRDoS Attacks using Identity -Based Cryptography in Software-Defined Networking Environment

6.1 Introduction

The previous chapters provide prevention against DRDoS attacks using reverse forwarding rules. One more promising approach is source authentication, as it prevents IP spoofing. A way to do this is to use signatures to authenticate the sender's identity (Schridde et al., 2009). Signatures mean digitally signing something from a key. Private keys are used to sign the object, and public keys are used to verify them. Generally, they require a dedicated external service that provides public-key certificates for corresponding private keys. So, private keys remain confidential to the signee, and public keys can be known to all. The complexity of this technique is that we need separate entities, i.e., PKES (Hu et al., 2017), within the network to record all the key pairs. Managing and maintaining these servers to store and distribute all the certificates is a hassle.

The other more promising approach is to use IBC (Long & Xiong, 2020) ("IEEE Standard for Identity-Based Cryptographic Techniques Using Pairings," 2013), as explained in Chapter 3. It is a kind of public-key cryptography in which an identity unique to the participating entity is used as a public key with a corresponding private key. It eliminates the need for separate servers (e.g., PKES servers) to distribute public key certificates.

To prevent DRDoS attacks, in the chapter, we propose using the BLMQ signature scheme (an IBS scheme) with SDN. We name the proposed scheme PoDIBC (Prevention of DRDoS attacks using Identity-Based Cryptography, IBC). The following are the salient features and advantages of PoDIBC:

a) To implement it, only the edge networks which require preventing their resources against DRDoS attacks need to be SDN enabled, and the core network of the Internet architecture remains unaffected.
b) (Schridde et al., 2009) proposed an IBS scheme to prevent IP spoofing. But that scheme has an issue: - if an identity is known, the attacker can also generate the private key, defeating the purpose of signing with a secure private key. We prove this shortcoming in section 6.5.3 and that the proposed PoDIBC technique solves this issue.
c) Since a signature is embedded within the network packet, PoDIBC requires approximately one-sixth of the network packet space as compared to (Schridde et al., 2009).
d) PoDIBC also prevents replay attacks by combining the timestamp with the message to be signed.
e) The packet identity itself is used as a public key, thus eliminating the need for separate servers to provide public keys. The private keys are generated with the help of this identity and are unique to this identity. For verification also, the same identity is used.

In PoDIBC, the SDN controller is responsible for generating and sharing the private keys to all the hosts connected in the edge network based on the respective hosts' identities (IP addresses).

6.2 Related Work

In computer communications, various techniques for source verification/authentication to prevent IP spoofing have been proposed by different authors. One such technique is presented by (Mishra et al., 2023). It improves Datagram Transport Layer Security (DTLS) by using lightweight authentication encryption with Quark. The authors propososed an improved method of over-hearing to prevent various types of attacks such as DOS, Man In The Middle

(MITM), active assaults, and passive attacks. Another proposed technique is by (X. Liu et al., 2008), who suggest adding a new field, called Passport, within the IP header to prevent spoofing. When the packet leaves its originating AS, the border/egress router attaches the message authentication code to the passport header of the packet. A secret key is shared in advance between the source AS and each AS in between the source and the destination. With the help of this key and the message authentication code, the inter-between ASs can verify whether or not the incoming packet belongs to that particular source address. The limitation of this technique is that Diffie-Hellman is used to exchange the keys, which in itself is not secure. A new IPv6 address generation algorithm is proposed by (Y. Liu et al., 2015). The basis of this algorithm is time and NID, and the proposed approach has three steps. First, a scalable structure of NID is designed. The address generation algorithm for IPv6 follows it. Finally, the concatenation of network identity and time is encrypted using the IDEA algorithm ff, generating an address assigned to the host. In this proposed technique, the authenticity of a source address is achieved with the help of SAVA, which involves SAVI devices (Wu et al., 2013). Such network devices complement ingress filtering (Wu et al., 2013) by adding IP address validity to an individual source. This dependency on SAVI devices is a drawback.

(Q. Zhou et al., 2021) propose the integration of SAVI and SDN to meet SAVI requirements in large-scale networks. The authors state that a central controller substantially facilitates the validation process since the SAVI based on SDN can verify the legitimacy of each passing packet and delete illegitimate ones in accordance with rules set by the controller. The authors created a versatile framework for SDN-based networks that utilize the available resources effectively while enabling centralized management. To detect anomalous activities and enable differentiated security management, they introduced a state partitioning and transitioning model for dynamic source address validation and binding relationships in the first level of the flow table. SDN-Ti, an SDN-based solution for identifying and tracing attackers in

IPv6 networks, is presented by (C. Li et al., 2019). It involves translating the source IPv6 address of the packet to a trusted ID-encoded address generated by the SDN controller, enabling effective identification of the attacker by the network administrator. This solution supports multiple IPv6 address assignment scenarios and does not require any modification on the end device, making it easy to deploy. The results suggest that SDN-Ti is a practical solution with the potential to be deployed for a large number of users.

In addition to network packet modification, alternative methods have been suggested that utilize the creation of certificates to sign packets and mitigate the risk of IP spoofing. These certificates can be supported by a third party like a Certifying Authority (CA) (Cooper et al., 2008), (Schridde et al., 2009) or can be self-certified (Andersen et al., 2008). The primary issue with CA is that separate infrastructure is needed to maintain the keys, like in IPSec (Frankel & Krishnan, 2011), (Kent & Atkinson, 1998), and TrueID (Hu et al., 2017).

The technique known as CGA (Aura, 2005) involves generating some of the bits in an IPv6 address by hashing the host's public key. In HIP proposed by (Moskowitz & Nikander, 2006), the public key is used as the host's identity and is created by hashing the corresponding host identifier. However, a drawback of these methods is that public keys must be generated prior to address generation, and for HIP the protocol stack or the entire Internet infrastructure would need to be modified.

To avoid the change in the protocol stack (and/or modifying the Internet architecture), (Schridde et al., 2009) proposed the true-IP technique to prevent DRDoS attacks. TrueIP leverages IBE to prevent IP spoofing by signing packets with the IP address via an IBC system. This eliminates the need for public key distribution and Certificate Authority (CA) infrastructures. The identity private key generator generates the private identity key and stores a set of public parameters, while the sender of the packets signs them using their private key, which is generated based on their IP address and public parameters. However, one drawback of the TrueIP scheme is that if an identity is known, an attacker can generate a private key, thereby compromising the security of the system. We prove this shortcoming in section 6.5.3 and prove that the proposed PoDIBC technique solves this issue. IBS scheme to provide stronger authentication between devices is proposed by (Wei et al., 2020). This scheme protects against Address Resolution Protocol (ARP) spoofing attacks, which android devices are susceptible to when connecting to an insecure Wireless Local Area Network (WLAN). This scheme involves utilizing the MAC address to create a unique identity that can be used to validate the authenticity of an IP address. By employing this method, they ensure a reliable and secure mapping between the identity and the IP address, which helps to establish trust and integrity. This scheme does not require root privileges and can work on low resource consumption.

6.3 BLMQ Signature Scheme using SDN Controller

The proposed use of the BLMQ signature scheme ("IEEE Standard for Identity-Based Cryptographic Techniques Using Pairings," 2013), (Barreto et al., 2005), (Noel Michael McCullagh, 2005) for preventing DRDoS attacks using IBC consists of four phases. The following are the steps taken:

a) *Generation of Master-Secret and Public-Parameters Group:*

   The SDN controller first generates a secret known as Master-Secret, which is only known to the controller. This master-secret is used to generate a group called Public-Parameters Group which would be shared with all the hosts connected to the controller via respective SDN switch.

b) *Private key generation*

   Host (either client or server), which requires to implement BLMQ signature scheme, pings the controller by generating a parameter request packet with the destination IP address as “1.1.1.1”. The SDN switch is configured such that when it receives a packet with this IP address, the packet is simply forwarded to the controller. Upon receiving this packet, the controller generates a private key for the host using its source IP address and Master-Secret. Finally, the controller generates and forwards a request packet embedded with private key and Public-Parameter Group to the host. Figure 6.1 shows step-1 and step-2.

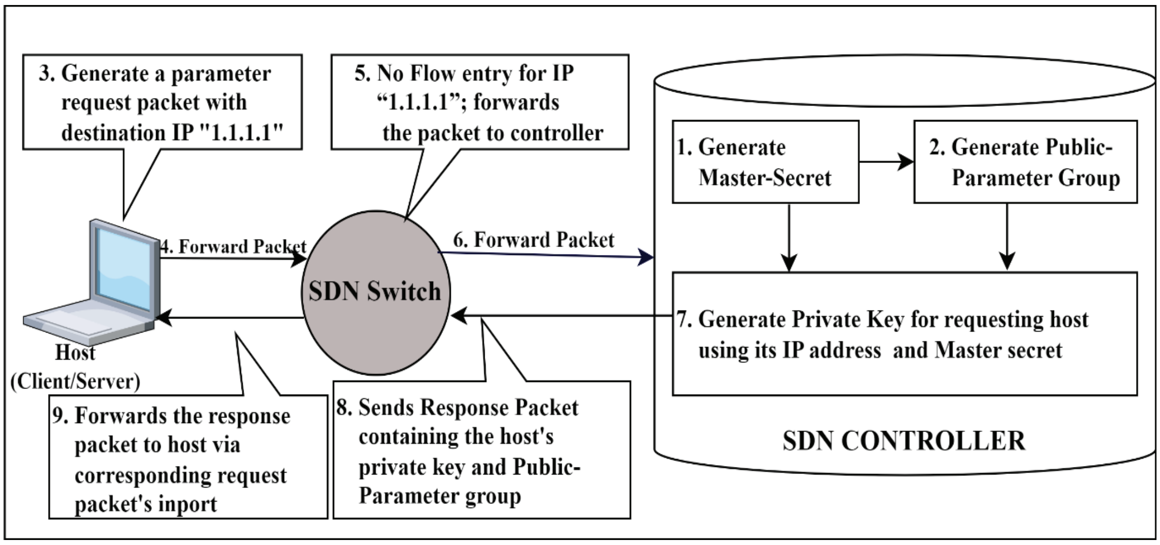

3. Generate a parameter request packet with destination IP "1.1.1.1"
5. No Flow entry for IP “1.1.1.1”; forwards the packet to controller
1. Generate Master-Secret
2. Generate Public-Parameter Group
4. Forward Packet
6. Forward Packet
SDN Switch
7. Generate Private Key for requesting host using its IP address and Master secret
Host (Client/Server)
9. Forwards the response packet to host via corresponding request packet's inport
8. Sends Response Packet containing the host's private key and Public-Parameter group
SDN CONTROLLER

Figure 6.1 : Communication between host and controller to receive private key and Public Parameters Group

c) *Signing the message*

Upon receiving its private key and Public Parameter Group from the controller, the host is now ready to send the message. Upon receiving a message to be sent, a client generates the sign using the BLMQ scheme for [message + timestamp]. Using timestamp for generating the sign prevents replay attacks. It then generates the request packet embedded with the message, timestamp, and the corresponding sign. This packet is forwarded to the server.

d) *Verification by the server*

Upon receiving the request packet, the server extracts its source IP address and the sign. It then recalculates the signature using the source IP and the Public Parameter Group. This newly generated signature is compared with the sign extracted. If they match, and the sum of extracted timestamp (*$\triangle T_h$)* and validation period (*$\triangle V$*) is greater than the current timestamp (*$\triangle T_s$*), the corresponding response packet is generated and forwarded; else, the packet is dropped. Synchronization at both the host and server for timestamp verification is necessary and can be achieved using NTP. Figure 6.2 shows step-3 and step-4.

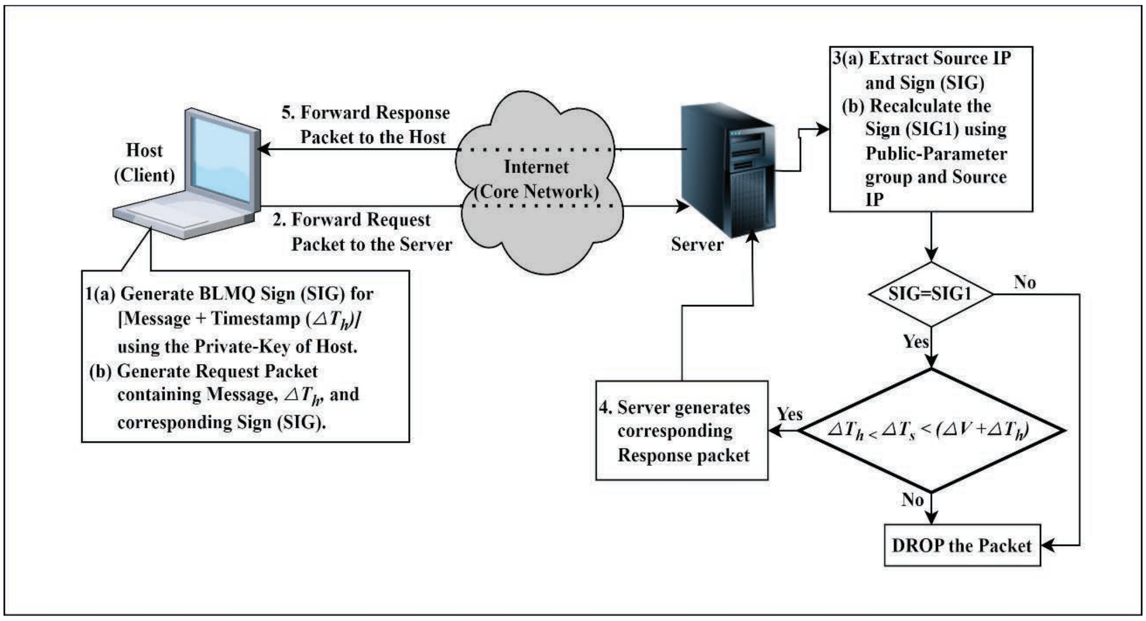

5. Forward Response Packet to the Host
Host (Client)
Internet (Core Network)
2. Forward Request Packet to the Server
Server
3(a) Extract Source IP and Sign (SIG)
(b) Recalculate the Sign (SIG1) using Public-Parameter group and Source IP
1(a) Generate BLMQ Sign (SIG) for [Message + Timestamp ($\triangle T_h$)] using the Private-Key of Host.
(b) Generate Request Packet containing Message, $\triangle T_h$, and corresponding Sign (SIG).
SIG=SIG1
No
Yes
4. Server generates corresponding Response packet
Yes
$\triangle T_h < \triangle T_s < (\triangle V + \triangle T_h)$
No
DROP the Packet

Figure 6.2 : Signing and verifying the packet

Above four steps are explained mathematically in the following subsections in detail. Table 6.1 shows the notations used in the PoDIBC scheme.

6.3.1 Generation Of Master Secret and Public Parameters Group

The controller generates the Master-Secret and the Public- Parameter Group. The steps are as follows:

1) A Master-Secret (M_{sec}) is generated as per eq. (6.1).

$$M_{sec} = Z_s^* \qquad \text{…………} (6.1)$$

where, Z_s is a set of integers with reduction modulo s, and Z_s^* is a multiplicative inverse set of Z_s.

2) Bilinear mapping (or pairing) of groups G_A, G_B, and μ_s of large prime order s is generated as per eq. (6.2). Here G_A and G_B are groups of order s, μ_s is a group of order s, A' is a generator of group G_A, B' is a generator of group G_B, and e is the pairing.

$$e: G_A \times G_B \rightarrow \mu_s \qquad \ldots\ldots\ldots\ldots\ldots (6.2)$$

where, $t = e(A', B')$ and $t \in \mu_s$

3) For verification purposes, a common key *Pub* (which is not identity) is also generated with the help of the controller's Master-Secret, as shown in eq. (6.3).

$$Pub = M_{sec} . B' \qquad \ldots\ldots\ldots\ldots (6.3)$$

4) Two hash functions (H_1 *and* H_2) are also needed for computations. We have used SHA224 for hash functions,

For the BLMQ signature scheme, the Public Parameter Group is (A', B', G_A, G_B, μ_s, Pub, e, t, H_1, H_2). This group is shared with all the hosts connecting to the controller.

Table 6.1 : Notations used in the PoDIBC scheme.

Notation	**Explanation**
M_{sec}	Master-Secret of the SDN controller
Z_s	Set of integers with reduction modulo s
Z_s^*	Multiplicative inverse set of Z_s
G_A ,G_B, μ_s	Groups of order s
A'	Generator of group G_A
B'	Generator of group G_B
e	Bilinear mapping or pairing
H_1 and H_2	SHA224 hash functions
PR_{ID}	Private Key
ΔT_h	Timestamp at host
IP	IP address of the signee (public identity)
Pub	A common key
h_s	Hash of message and a random integer
S_s	Sign generated using private key and h_s

6.3.2 Private Key Generation

The Private key (*PR_{ID}*) is generated using the Master Secret (i.e., *M_{sec}*) and the identity of the host (which, in our case, is the source IP address), as shown in eq. (6.4). This process is similar to ("IEEE Standard for Identity-Based Cryptographic Techniques Using Pairings"), (Barreto et al., 2005), (Noel Michael McCullagh, 2005). Since IP addresses differ for each host, using it as an identity for private key generation guarantees a unique and different private key for each host (Schridde et al., 2009).

$$PR_{ID}= (H_1(IP)+ M_{sec})^{-1}. A' \quad \text{.........} (6.4)$$

6.3.3 Signing the Message

The process of signing the message using the private key PR_{ID} is as described in (("IEEE Standard for Identity-Based Cryptographic Techniques Using Pairings")(Barreto et al., 2005)(Noel Michael McCullagh, 2005)), except that in the proposed scheme, we sign (message+ timestamp). (Schridde et al. 2009) have also signed the timestamp. The reason for doing this is to protect the network against replay and MITM attacks. Its formal proof is shown in section 6.4.

If "Message" denotes the actual message to be sent, the message considered to be signed would be Msg, where Msg = (Message + $\triangle T_h$). Here $\triangle T_h$ is the current timestamp at the host. For l chosen randomly such that $l \in Z_s^*$, eq. (6.6) to (6.8) show the sign (S_s) generation process. Finally, the signature Sig = (h_s, S_s) will be sent within the packet, as shown in Figure 6.2.

$$Msg= (Message+ \triangle T_h) \quad \text{................} (6.5)$$

$$k= t^l \quad \text{................} (6.6)$$

$$h_s= H_2(Msg, k) \quad \text{................} (6.7)$$

$$S_s= (l+h_s)PR_{ID} \quad \text{................} (6.8)$$

6.3.4 Verification by the Server

The server pings the controller and, in turn, receives the Public Parameter Group. Upon receiving the network packet, the server extracts the embedded identity (i.e., the source IP address), Msg, and the corresponding signature (i.e., Sig = (h_s, S_s)). The value of h_{snew} is computed using the public parameters group as shown in eq. (6.9).

$$h_{snew} = H_2(Msg, e(S_s, Pub_{ID})t^{-h_s} \quad \text{............ (6.9)}$$

where, Pub_{ID}= H_1(IP) B' + Pub

If this computed h_{snew} is the same as the received h_s, the source IP is authenticated, and the packet is accepted. Otherwise, if the two are different, it indicates spoofing the source IP, and the packet is dropped. In this case, further action can be taken toward identifying the attacker by traceback.

6.4 Proof against IP Spoofing and Replay Attacks

In this section, we give formal proof that the proposed BLMQ IBS scheme will always prevent the network from IP spoofing, thus preventing DRDoS attacks. In addition, we give mathematical proof that replay attacks are also prevented.

6.4.1 Proof of Prevention against IP Spoofing

Let the valid and original IP of the attacker be IP_1, and the spoofed IP that the attacker will send as the source IP address in the network packets be IP_2. From eq (6.9),

$$Pub_{ID_1} = H_1(IP_1)B' + Pub \quad \text{.......... (6.10)}$$

$$Pub_{ID_2} = H_1(IP_2)B' + Pub \quad \text{.......... (6.11)}$$

The private Key of the attacker will be

$$PR_{ID_1} = (H_1(IP_1) + M_{sec})^{-1}. A' \quad \text{........ (6.12)}$$

The signature (h_s, S_s) is generated as per equations (6.7) and (6.8). Since the attacker spoofs the source IP address, the corresponding field in the network packet contains the source address as IP_2. This packet is sent over the communication channel to the destination.

The process of verification at the server is as follows:

From eq (6.7),

$$h_s = H_2(Msg,k)$$

$$h_s = H_2(Msg,t^l)$$

$$h_s = H_2(Msg,t^{l+hs}.t^{-hs})$$

$$= H_2(Msg, t^{\frac{(H_1(IP_2)+M_{sec})}{(H_1(IP_2)+M_{sec})}.l+h_s}.t^{-h_s})$$

$$= H_2(Msg, e(A', B')^{\frac{(H_1(IP_2)+M_{sec})}{(H_1(IP_2)+M_{sec})}.l+h_s}.t^{-h_s})$$

$$= H_2(Msg, e(A'. \frac{(l+h_s)}{H_1(IP_2+M_{sec})}, B'.(H_1(IP_2)$$
$$+ M_{sec})).t^{-h_s})$$

From eq^n (6.4),

$$= H_2(Msg, e(PR_{ID_2}.(l+h_s), (H_1(IP_2)B'$$
$$+ B'.M_{sec})).t^{-h_s})$$

From eq^n (6.3),

$$= H_2(Msg, e(PR_{ID_2}.(l+h_s), (H_1(IP_2)B'$$
$$+ Pub)).t^{-h_s})$$

From eq^n (6.11),

$$= H_2(Msg, e(PR_{ID_2}.(l+h_s), (Pub_{ID_2})).t^{-h_s})$$

From eq^n (6.8),

$$= H_2(Msg, e(S_{s_2}, (Pub_{ID_2})).t^{-h_s})$$

Received sign on IP address IP1 will be-

$h_s = H_2(Msg,k)$

$h_s = H_2(Msg,t^l)$

$h_s = H_2(Msg,t^{l+hs}.t^{-hs})$

$$= H_2(Msg, t^{\frac{(H_1(IP_1)+M_{sec})}{(H_1(IP_1)+M_{sec})}.l+h_s}.t^{-h_s})$$

$$= H_2(Msg, e(A', B')^{\frac{(H_1(IP_1)+M_{sec})}{(H_1(IP_1)+M_{sec})}.l+h_s}.t^{-h_s})$$

$$= H_2(Msg, e(A'.\ \frac{(l+h_s)}{H_1(IP_1+M_{sec})}, B'.\begin{pmatrix} H_1(IP_1) \\ +M_{sec} \end{pmatrix}).t^{-h_s})$$

From eq^n (6.4),

$$= H_2(Msg, e(PR_{ID_1}.(l+h_s), \begin{pmatrix} H_1(IP_1)B' \\ +B'.M_{sec} \end{pmatrix}).t^{-h_s})$$

From eq^n (6.3),

$$= H_2(Msg, e(PR_{ID_1}.(l+h_s), (H_1(IP_1)B' + Pub)).t^{-h_s})$$

From eq^n (6.10),

$$= H_2(Msg, e(PR_{ID_1}.(l+h_s), (Pub_{ID_1})).t^{-h_s})$$

From eq^n (6.8),

$$= H_2(Msg, e(S_{s_1}, (Pub_{ID_1})).t^{-h_s})$$

$$H_2(Msg, e(S_{s_1}, Pub_{ID_1})t^{-h_s}$$
$$\neq H_2(Msg, e(S_{s_2}, (Pub_{ID_2})).t^{-h_s})$$

Hence, the received signature is not equal to the computed one.

Spoofed IP will not result in the same sign. Hence the server will discard the packet. This proof is based on the BLMQ proof provided by (Noel Michael McCullagh, 2005), except that we have replaced the identity with the source-IP address.

6.4.2 Proof of Protection against Replay Attacks

A replay attack is one form of a network attack wherein an attacker sniffs the legitimate (possibly encrypted) messages and either delays them or retransmits them. In other words, it happens when an attacker captures possibly confidential information on a secure channel and then resends them after a delay, masquerading as the original sender (*What Is a Replay Attack*, 2023).

In the present context of using the BLMQ signature scheme for preventing DRDoS attacks, the attacker can capture the signed packet and reuse the sign to masquerade as the victim to get the packet accepted at the server. For example, suppose a legitimate user U_1 (with source IP address IP_1) sends a DNS query message to the DNS server. Based upon the identity IP_1, this message contains the corresponding signature S_{s_1}. Assume that this DNS query message gets sniffed by a malicious user. Thus, this malicious user now knows the signature $S_{s_1}$and the IP address of U_1. A replay attack is possible if this malicious user sends multiple DNS query messages with source IP IP_1 and signature S_{s_1}. Such replay attacks are not possible with the proposed scheme because of the timestamp field. Formally, in the remaining part of this section, we prove that replay attacks are impossible by considering two scenarios.

Scenario-1: When the attacker uses the same timestamp as that sniffed/captured within the request packet. Let ,

- $\triangle T_h$ be the system's timestamp when the packet is created at the host.

- $\triangle V$ be the validation period of the timestamp to be checked when the packet is received at the server. This validation period depends upon the RTT between the source and the destination.
- $\triangle T_s$ be the timestamp at the destination (or server) when the packet reaches it.

If the timestamp at the server, i.e. $\triangle T_s$ does not fall within the range of $\triangle V + \triangle T_h$, the packet is dropped. Formally, at the destination, if $\triangle T_h, < \triangle T_s < (\triangle V + \triangle T_h)$, the packet is accepted, else the packet is discarded.

Scenario-2: When the attacker uses the current timestamp $\triangle T_h'$, instead of the timestamp of the captured packet $\triangle T_h$. Let,

- The message received at the server be $Msg' = M + \triangle T_h'$
- The actual message on the computed sign be $Msg = M + \triangle T_h$

From eq^n (6.9), h_s computed at the server would be

$h_s = H_2(Msg', e(S_s, Pub_{ID})t^{-h_s}$ where, $Pub_{ID} = H_1(IP)B' + Pub$

The original h_s would be

$h_s = H_2(Msg, e(S_s, Pub_{ID})t^{-h_s}$

where, $Pub_{ID} = H_1(IP)B' + Pub$

$H_2(Msg, e(S_s, Pub_{ID})t^{-h_s} \neq H_2(Msg', e(S_s, Pub_{ID})t^{-h_s}$

Hence, the two signatures will be different, and the packet will be discarded.

6.5 Simulation and Results

This section describes the experimental setup done to validate the proposed approach. In this section, we also show that with PoDIBC in place and with the victim under attack, the attack traffic does not reach the victim, thus preventing its network from DRDoS attack.

6.5.1 Experimental Setup

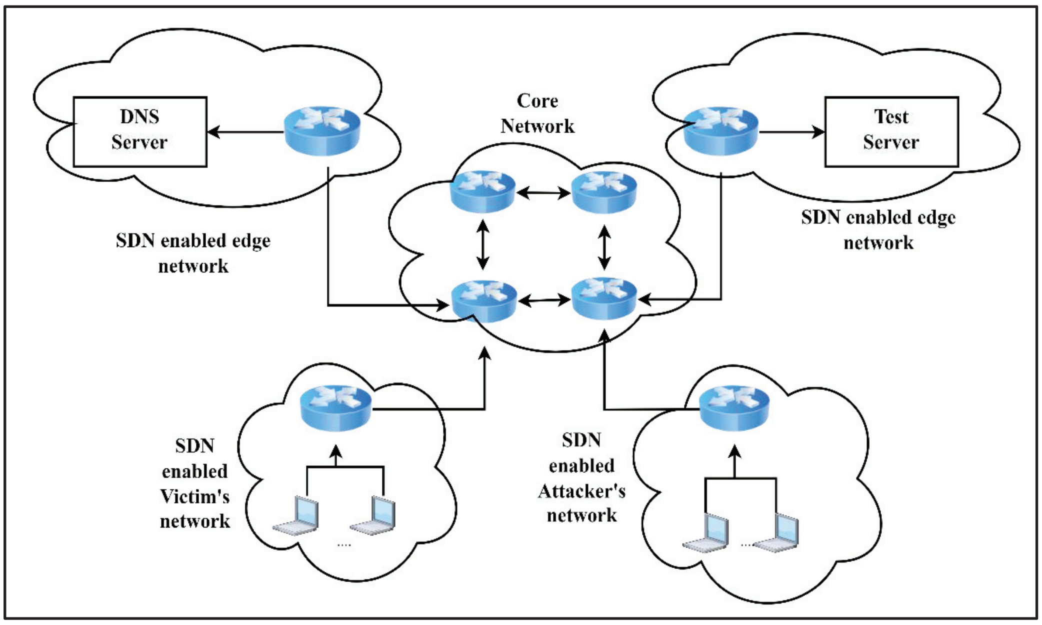


Figure 6.3 : Network architecture for PoDIBC

As shown in Figure 6.3, the experimental topology consists of four subnets: an attacker's network in which the attacker host resides, the victim's network in which the victim host resides, a subnet for the DNS server, and a subnet for a test server. All these four subnets are SDN networks. In addition, Figure 6.4 also shows the core network consisting of four routers. We used Mininet (*Mininet*, 2022) to simulate the topology. This core network does not need to be SDN enabled, but it is for now for experimental purposes. The whole topology is currently connected to a single Controller. Multiple controllers can also be used for larger networks, and these controllers can communicate with each other to pass the Public Parameters Group. A Ryu SDN controller (*RYU A*, 2013) was used to configure the SDN switches and routers and install the flow tables to route the packets. Python libraries scapy (*Introduction: About Scapy*, 2023) and iperf (*What Is IPerf / IPerf3 ?*) were used to generate traffic and bandwidth testing. To implement the proposed technique, we have used the bplib library (Copyright (c) 2014, George Danezis (UCL), and a common group G (Copyright (c) 2014, George Danezis (UCL), Copyright (c) 2016, The OpenSSL Project), with elliptical set of groups G, and pairing e.

Initially, when the controller starts, it creates the Master Secret, and it creates the Public Parameters Group. As the end host starts, it pings the controller. The controller, in turn, creates the corresponding private key and passes this key and public parameters to the host. For simulation purposes, a DRDoS attack is simulated using DNS Servers, towards which the attacker will send source-IP spoofed DNS query request packets. Hence the request is in the form of a DNS query to which the DNS server will respond. Further, to prevent replay attacks, the timestamp was added with the query to be sent. Thus, the total message to be signed is the [DNS-query + timestamp], and this signature (using BLMQ) is added after this message.

Two different sets of experiments were conducted using the experimental setup. The first experiment showed a bandwidth test between the test server and the victim host. During this test, the bandwidth of the network link of the victim was kept at 500 Mbps while conducting a DNS-based DRDoS attack. We used HTTP traffic to generate legitimate traffic in our experimental analysis. The available bandwidth was recorded with both the BLMQ signature scheme and without it. In the second experiment, to measure the overhead due to the proposed method, we measured the RTT of a DNS request-response cycle with both the signature scheme present and not. For this, the victim host sends a legitimate packet to the DNS server, which then decrypts and verifies the signature and replies with the appropriate response packet.

The proposed architecture requires modifying the functionality of the base protocol used for DRDoS attacks. Since, in this paper, we use DNS-based DRDoS attacks for experimentation purposes, DNS functionality needs to change to implement the BLMQ signature scheme.

6.5.2 Results and Discussion

This section shows the validation and effectiveness of the proposed PoDIBC scheme. It proves that with PoDIBC in place and with victim under attack, the attack traffic does not reach the victim, thus preventing its network from DRDoS attack. The second set of experiments shows the overhead of implementing PoDIBC.

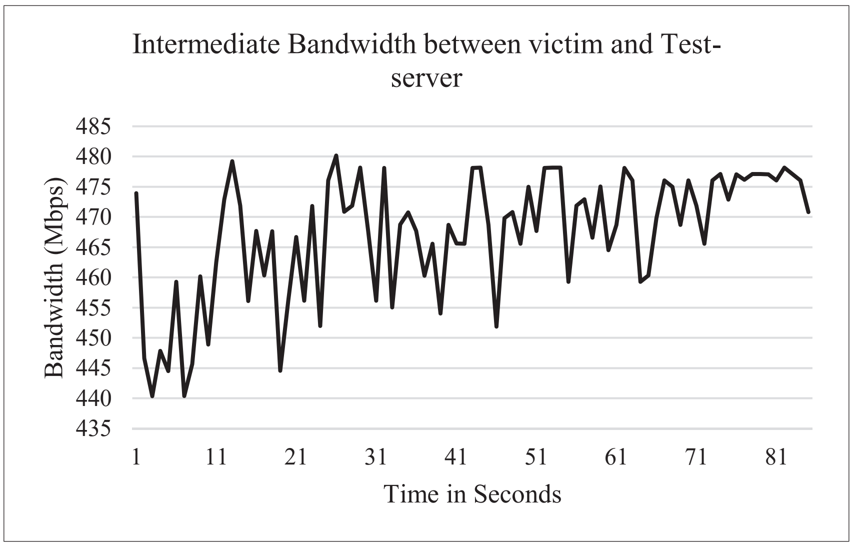


Figure 6.4 : Available intermediate bandwidth between the victim and test server with PoDIBC in place.

Figure 6.4 shows the available TCP test bandwidth during the attack as a function of time between the victim host and the test server. It is when the signature verification scheme is in place. Experiments show that out of the available 500 Mbps link bandwidth, on average, 460 Mbps (or 92%) of effective intermediate bandwidth is available between the victim host and the test server. Thus, even though the attacker is making an attack of 1 Gbps with source-IP spoofed to that of the victim host, there is no effect on the available bandwidth (thus communication) between the victim and servers. Correspondingly, Figure 6.5 shows the available bandwidth without the signature verification scheme in place. In this case, the DNS server reflects the packets toward the victim; thus, it is under attack. Here, an average bandwidth of 0.52 Mbps (or 0.1%) was available, considerably lower than the one achieved in the previous case. It is worth emphasizing that for both Figure 6.4 and 6.5 there was no effect on the bandwidth for first 1-2 seconds of the start of an attack.

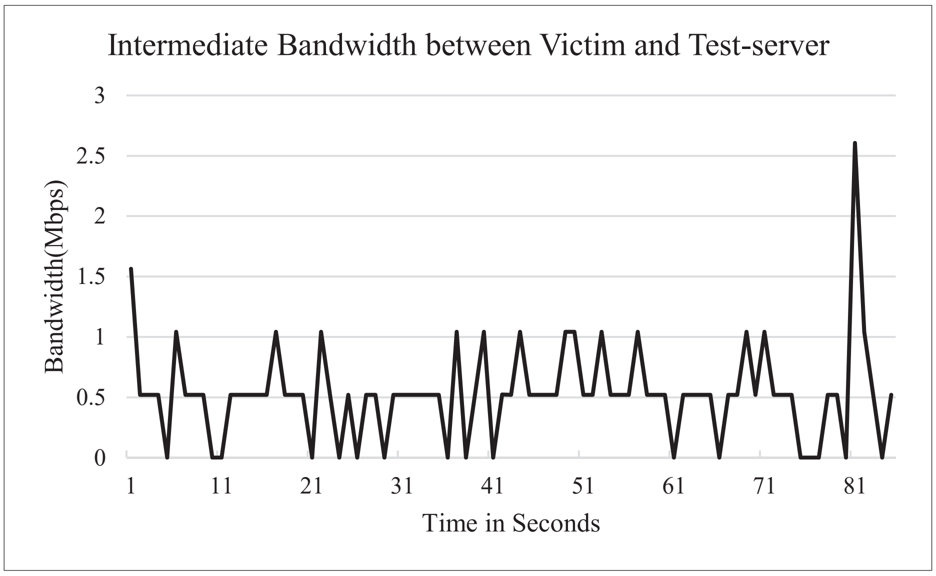


Figure 6.5 : Available intermediate bandwidth between the victim and test server without PoDIBC in place.

Since the BLMQ signature verification scheme requires additional computation, an additional delay is expected in the corresponding client-server communication service. The computational cost would increase because of the following two additional tasks.

Task-1: Getting a private key and Public Parameter Group from the SDN controller.

Task-2: Using the Public Parameter Group and private key, generate the sign using the BLMQ signature scheme, embed the sign into the network packet, and verify it at the receiver end.

To show the delay caused by Task-1, we incorporated fetching the parameters while sending the first request and response packet. The hosts fetch the parameters from the controller using the SDN-enabled OpenFlow switch, as shown in Figure 6.3. Alternatively, a host can be configured to fetch these parameters and a private key when it connects to the network. After that, upon acquiring all the parameters, only Task-2 is required for the subsequent communication. To show these additional delays caused by BLMQ, we calculate the following:

a) The RTT delay of the first request packet- this delay includes fetching the parameters from the controller, creating a request packet with the sign, sending the packet, sign verification at the server after fetching the parameters, and getting the response.

b) Packet creation delay due to signing the packet at the client side- this delay includes generating the sign using the already acquired private key and Public Parameter Group and embedding the sign into the outgoing network packet.

c) Packet processing delay due to verification at the server side- this delay includes overhead because of Signature verification at the receiver using an already acquired Public Parameter Group.

d) The RTT overhead in terms of RTT of the second packet onwards, i.e., once the host has already acquired the private key and Public Parameter Group. It shows the overhead of implementing the proposed PoDIBC scheme for DRDoS prevention.

All four delays are shown respectively in Figures 6.6-6.9. Each graph shows the delay with and without the signature verification approach for twenty iterations of experiments.

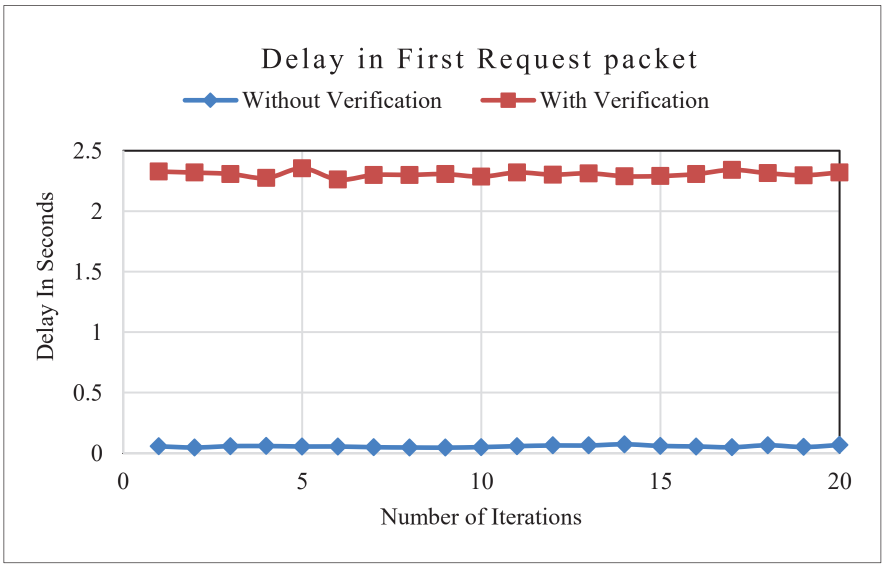


Figure 6.6 : RTT for first request packet

Figure 6.6 shows the RTT delay of the first request packet. The average RTT for the first request packet is 2.3 seconds with the signature verification scheme and 0.05 seconds without the signature verification scheme. It is important to note that this delay is only for the first packet, not subsequent packets, thus no significant impact on overall performance and user experience. Therefore, in the context of preventing DDoS attacks, this delay is an acceptable measure to protect the system from potential threats while minimizing the impact on legitimate traffic.

Figure 6.7 shows the overhead (in terms of delay introduced) in request packet creation at the client side due to sign generation. The request packet creation process with the PoDIBC scheme in place introduces an average delay of 7 ms in request packet creation at the client side, while without it, the delay is only 1 ms. Therefore, the additional delay due to sign generation is about 6 ms on average.

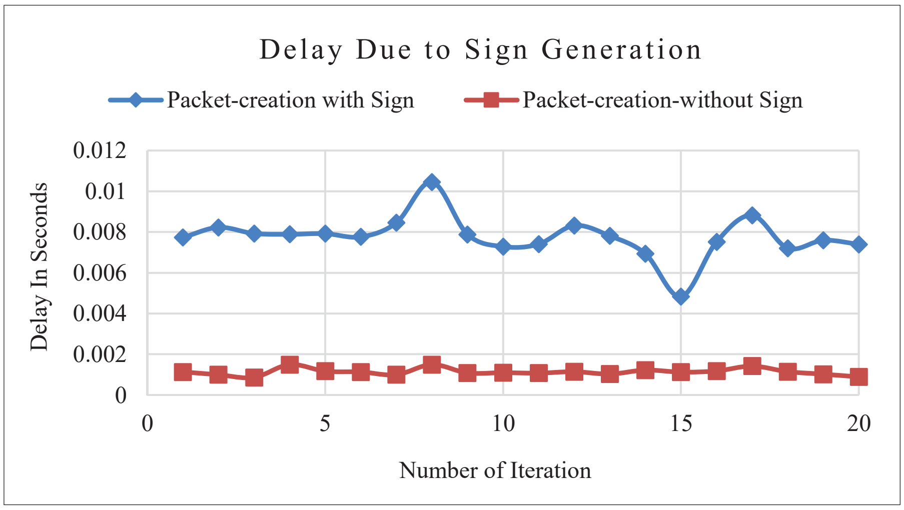


Figure 6.7 : Overhead in packet creation due to sign generation

Figure 6.8 illustrates the processing delay to verify the signature on the server side. Specifically, the delay with sign verification is, on average 16 ms; without it, the delay is only 0.2 ms.

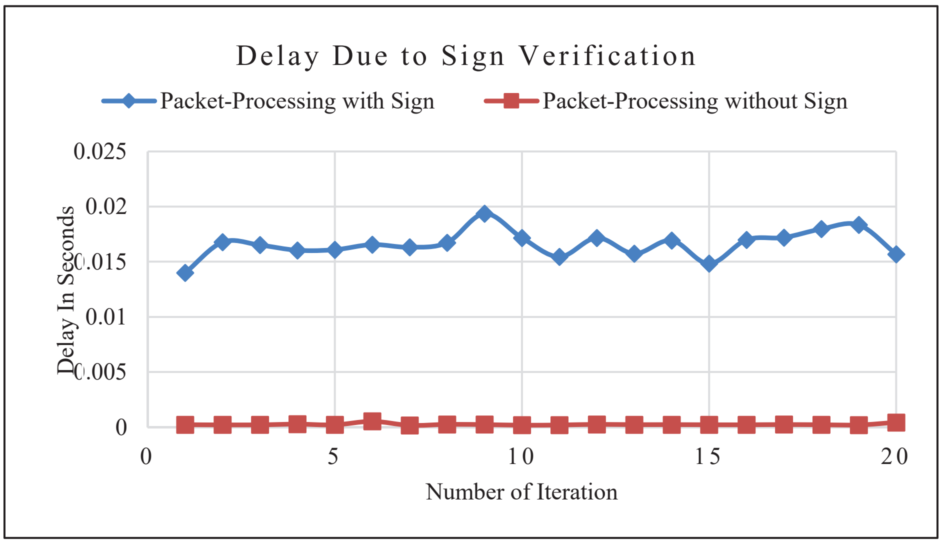


Figure 6.8 : Overhead due to Sign verification

The RTT graph in Figure 6.9 shows the total delay caused by the additional signing and verification. RTT is the time taken to send the signed packet and receive the response after verification. The average RTT when PoDIBC was in place was 76 ms; when PoDIBC was not, it was 55 ms. Thus, the additional average computational delay was 21 ms due to PoDIBC. Figure 6.9 shows the overhead while implementing the proposed DDoS prevention scheme using the BLMQ signature.

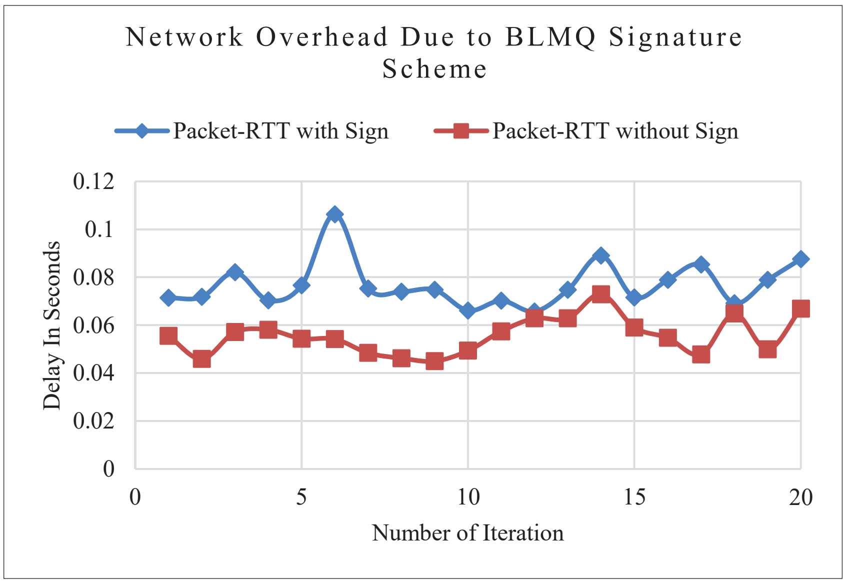


Figure 6.9 : RTT overhead because of BLMQ signature scheme for DDoS prevention

We verified 100 messages with the BLMQ algorithm with 100% accuracy. Accuracy means the number of times the algorithm could authenticate correctly spoofed as spoofed and honest as honest. The BLMQ's implemented runtime for 100 messages was 1.663 seconds.

6.5.3 Comparison and Discussion

The proposed scheme is not susceptible to key revocation problems. Key revocation is a problem for IBC (Schridde et al., 2009). Since the public and private key pairs are generated using an identity unique to a participating entity, if somehow the keys are known by the perpetrator, they can be misused continuously. As the identity unique to a host cannot be changed, the private/public key pair also cannot be changed. The IBC-based scheme proposed using IP addresses as an identity does not pose this threat because IP addresses can be dynamic. It is not hard-coded like a MAC address.

Also, in the proposed scheme, the network overhead in carrying additional bits of information (in the form of a signature) is less than (Schridde et al., 2009). The total bit-length of the sign is- Sig= (h_s, S_s). We have used SHA224 for the hash, so hs= 224 bits. As shown in eq (iv) and eq(viii), Ss are dependent on the order of the field. As mentioned in (*Elliptic Curve Cryptography (ECC)*, 2022) the private keys are in the range of elliptical curve field size, which usually is 256 bits. Hence, Ss can be considered to be 256 bits. We also need to send the query+timestamp to verify. The query is sent with the DNS packet, and the timestamp requires 192 bits. Thus, the total length of sign Sig is = 672 bits (i.e. 224 + 256 + 192). Comparatively, the scheme of TrueIP (Schridde et al., 2009) requires 4128 bits, and that of X.509 certificate requires 6144 bits, as mentioned in (Schridde et al., 2009). Thus, regarding the number of bits required, the proposed scheme in this paper requires approx. one-sixth network packet space as compared to (Schridde et al., 2009) and approx. one-ninth as compared to X.509 (Schridde et al., 2009).

The TrueIP scheme is susceptible to a DRDoS attack, which is not the case in PoDIBC, as shown in section 6.4. In TrueIP, the Private key is calculated as per eq. (6.13):

$$H(ID)^{1/R} \pmod N \text{ ……….. } (6.13)$$

where, ID is public identity, and R and N are public shared parameters.

If the process to generate the private key is also known, then any perpetrator can generate the private key using these public parameters. If an attacker can also generate the private key, it can quickly generate spoofed messages, thus capable of launching DRDoS attacks. Specifically, suppose an attacker A1 knows the IP address of victim V1 (IP_V). It will generate the private key using public shared parameters R and N using eq (6.14).

$$Pr_{key}= H(IP_V)^{1/R} \pmod N \text{ ……… } (6.14)$$

Spoofing can be done using this private key, making a DRDoS attack possible. In the PoDIBC scheme, the private key is generated using a Master Secret only known by the controller, so even if the identity is known, the private key cannot be generated, as shown in eq (6.4).

IPsec is a complete security protocol that provides encryption with authentication using certificates. It involves protocols to negotiate keys and encryption algorithms that will be used before the process of authentication starts. It is a heavy protocol that provides a very secure channel for communication. Only to provide authentication, which our single packet approach can also achieve, renders the need for a heavy protocol like IPsec.

6.6 Summary of the Chapter

In conclusion, our research demonstrates that preventing DRDoS attacks using PoDIBC, a DRDoS prevention technique based on the BLMQ signature scheme of IBC, is a more effective approach than detection and mitigation. Our technique is mathematically and experimentally validated in an SDN environment, and we have shown that it is not susceptible to replay attacks. Moreover, PoDIBC can detect IP spoofing and can be applied to prevent source IP spoofing in general. Our experiments have shown that implementing PoDIBC requires SDN-enabled edge networks, but no change in the core network is necessary. The experiments have also shown that the overhead of PoDIBC in terms of increased RTT and additional space in network packets is reasonable. When PoDIBC is in place, the victim's network is always protected from attack traffic, and the victim's bandwidth remains unaffected.

********** End of Chapter **********

CHAPTER 7- Near Real-Time Detection and Mitigation of DDoS Attacks through Feature Optimization in a Software-Defined Networking Environment

7.1 Introduction

Machine Learning (ML) is a key element in the rapidly expanding discipline of data science (*What Is Machine Learning(ML)?, IBM*). ML algorithms are trained using statistical techniques to produce classifications or predictions about a given dataset. These classifications and predictions provide insightful details for businesses to grow and understand consumers' needs. Following the emergence of ML and AI, scientists began employing ML methods to identify and safeguard against DDoS attacks. It involves analyzing the intrinsic distinctions between malicious and legitimate network traffic. Various factors such as traffic rate, packet quantity and frequency, and the presence of multiple flows directed to a single destination IP address exhibit specific variances between harmful and benign traffic. Thanks to the progress in AI, ML algorithms have evolved to a level of sophistication where they can classify traffic as either malicious or benign by leveraging these distributional dissimilarities.

However, the effectiveness of ML models heavily relies on the quality and comprehensiveness of the training data they receive for detection purposes. Consequently, the subsequent challenge has been to obtain well-defined datasets that accurately represent the various types of attacks. Fortunately, this predicament has been addressed with the contribution of the CICDDoS 2019 attack dataset by the Canadian Institute for Cybersecurity (*DDoS Evaluation Dataset (CIC-DDoS2019)*), (Sharafaldin et al., 2019). This dataset encompasses multiple distinct datasets specifically designed to simulate different types of attacks. The availability of the aforementioned dataset has spurred significant research in this domain, with numerous studies exploring various ML and Deep Learning (DL) approaches for detecting attacks (Samom et al., 2021), (Kshirsagar & Kumar, 2022), (Rajagopal et al., 2021), and others. We have employed the CICDDoS 2019 attack dataset to train our ML model. The rationale behind this choice lies in the dataset's extensive nature, exclusively encompassing diverse DDoS attack types. In contrast, datasets like KDD-cup and UNSW-15 contain a broader spectrum of cybersecurity threats, which includes DDoS attacks but not exclusively. Therefore, we opted for a dataset focusing solely on a variety of DDoS attacks.

In the previous chapters, we study techniques involving amendment in the network layer or routing techniques to provide True Prevention. In this chapter, we delve into the utilization of machine learning models to detect and prevent attacks. The proposed method lies in the category of Partial Prevention. Our primary focus lies on employing the random forest model (BREIMAN, 2001) to identify DDoS attacks such as Portmap, DNS, UDP-lag, UDP, and SYN, which are present in the CICDDoS 2019 attack dataset. Remarkably, we achieve an impressive accuracy rate of 99.9% in detecting DDoS attacks using this model. We deploy the trained model in a topology created in an SDN environment for near real-time attack detection. By near real-time detection, we mean that some attack traffic will reach the victim by the time the attack is detected. In our simulated setup, this attack traffic is approx. 1% of the total attack, and the maximum time to detect the attack is 5.3 seconds; hence near real-time.

Within the simulated environment, an effective defense against DDoS attacks is implemented through the use of an SDN barrier. This barrier is composed of an SDN switch that replicates all incoming network traffic to a dedicated computing device running the trained model. The primary responsibility of this model is to detect attacks and promptly inform the controller regarding the source IP address of the attacker. Subsequently, the controller takes action by discarding packets originating from the identified IP address, effectively blocking the ongoing attack. The key contributions of this study can be summarized as follows:

- We conducted training using the random forest algorithm on the CICDDoS 2019 attack dataset.
- Through meticulous hyperparameter tuning and optimized feature selection, we achieved a significant accuracy for the random-forest model, an impressive 99.9%.
- By reducing the number of features from 88 to 15, we successfully achieved near real-time detection of attacks with an accuracy of 99.99%.
- In the SDN environment, we generated realistic random DDoS attack traffic and legitimate traffic. By leveraging the capabilities of the SDN switch, we seamlessly mirrored this traffic to a specific node running the trained model. Consequently, the model effectively classified the traffic as either an attack or normal.
- Additionally, we computed valuable statistics to determine the proportion of attack traffic versus benign traffic that actually reached the intended victim.
- The accuracy achieved through Random Forest was better than that achieved by the multi-classifier approach proposed by (Rajagopal et al., 2021). We achieved an accuracy of 99.9% for CICDDoS 2019 attack dataset, in comparison to 97% accuracy achieved by (Rajagopal et al., 2021).

7.2 Related Work

There has been promising research in detecting cyberattacks, like DDoS attacks, using ML and DL techniques. Due to their accuracy, ML techniques can be deployed to detect DDoS attacks by extracting a selected set of features from network traffic. The method proposed by (Munivara Prasad et al., 2016) uses ML to achieve fast detection of app-DDoS attacks. The approach focuses on a set of requests over an absolute time interval to detect anomalies in a network. Different metrics are the ratio of packet types, packet count, route context, router chain, context, a ratio of request intervals, etc. The performance is measured by calculating precision, recall, sensitivity, and specificity. Considering that every DDoS attack tool has its signature, (Laskar & Mishra, 2016) have proposed a detection technique. First, a database is created using 14 feature vectors. Two vectors are used to filter out "suspicious" traffic; then, the remaining 12 vectors are used on this suspicious traffic to test it for possible DDoS attacks. Authors claim that this scheme can be used in real time. (Oo et al., 2015) proposed a packet classification approach for DDoS prevention using a hidden semi-markov model based on a selected set of parameters. It involves collecting packets every second, extracting features, applying a classification algorithm, and using the hidden semi-markov model algorithm. The set features include -the number of packets and bytes from a source to a destination IP address, packet rate and byte rate, etc. It then defines a classification algorithm with rigid decision boundaries for each parameter. It is computation and memory intensive since it requires keeping track of packets from each source IP address to each destination IP address and has many parameters to compute. (Yadav & Subramanian, 2016) proposed pattern learning to detect application-layer DDoS attacks. A model based on neural networks like autoencoder was applied for broad learning. The process was divided into training and testing the features extracted from web server logs. The approach (M. E. Ahmed et al., 2017) proposed is based on traffic monitoring and clustering using unsupervised learning. The proposed Dirichlet process mixture model is a Bayesian approach for clustering over nonparametric traffic patterns. Traffic features include- the total number of packets transmitted, connection duration time, and ratio of source and destination bytes. The proposed approach consists of a learning module, a traffic statistics manager, and a network resource manager. The limitation is that as traffic flow increases, the accuracy of detecting an attack traffic connection reduces, and the misclassification rate (i.e., the number of feature vectors assigned to wrong clusters over total features) reaches nearly 50%. (C. C. Chen et al., 2017) have used SDN and ML techniques for implementing a system to detect DRDoS packets and block amplification attacks automatically. Their training model is built upon eight features: forward packet count, backward packet count, flow volume, etc. They have shifted the detection towards the network gate.

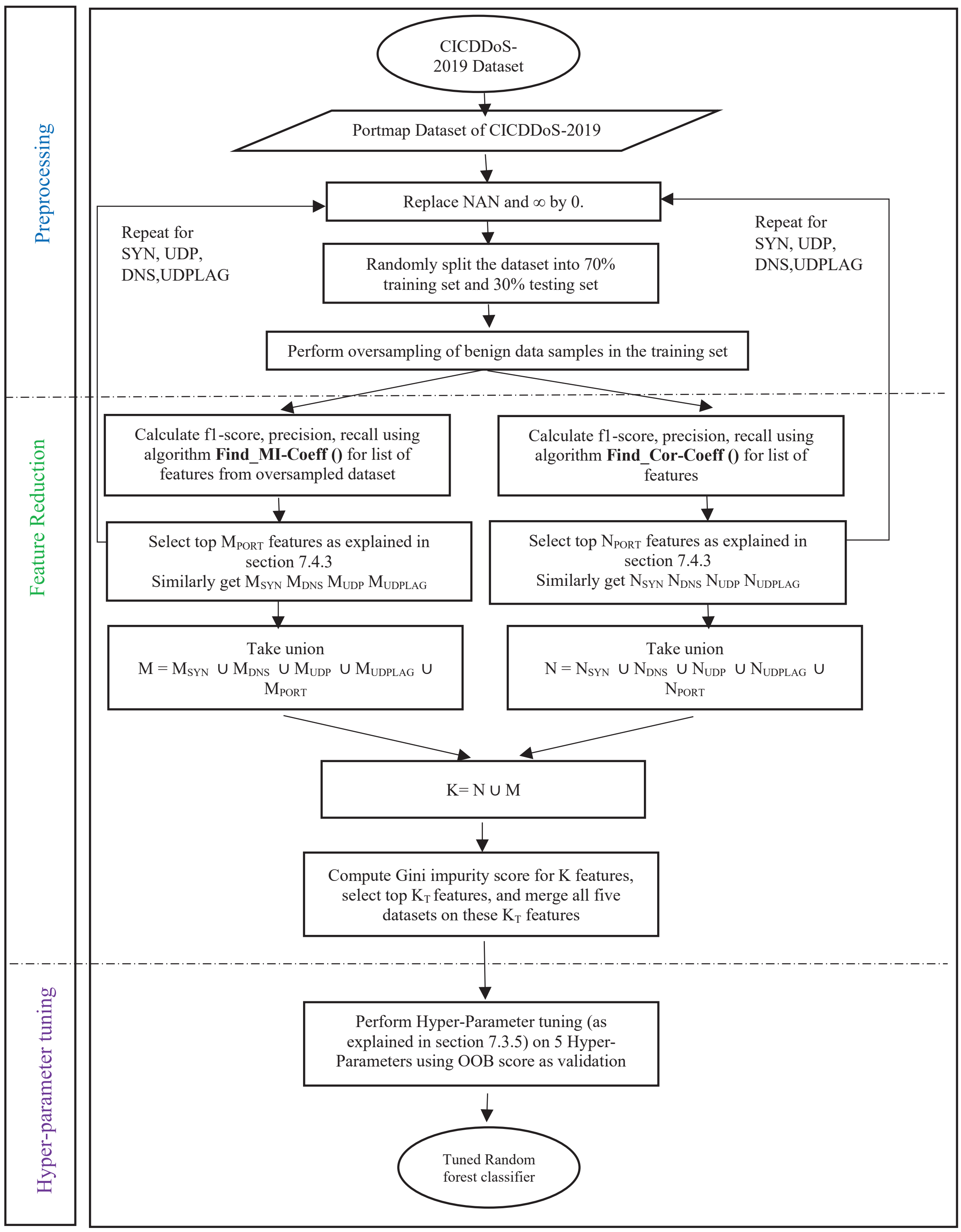
In the study proposed by (Nurwarsito & Nadhif, 2021), a DDoS attack detection and mitigation system was developed within the framework of SDN architecture, employing the random forest algorithm. The random forest algorithm serves to categorize incoming packets as either normal or indicative of an attack, based on their flow entries. A limitation of this study lies in the choice of the attack dataset used for training the random forest model. The dataset employed in this study departs from the conventional standard, as it was generated by the authors within the SDN environment, rather than being sourced from a more diverse and authentic collection of DDoS attacks. Similarly to this (Santos et al., 2020) have also used various ML algorithms like random forest, decision tree, SVM, and multi-layer perceptron to detect and mitigate attacks of SDN environment. The DDoS attacks generated are limited to SDN environment. This study also has the limitation of not using a standard dataset for training the ML models. way to remove the load from controllers.

Using the CICDDoS dataset, we have focused on papers based on the same dataset. (Sharafaldin et al., 2019) explain the CICDDoS-2019 dataset (Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy). The article describes how the data has been collected, the taxonomy of different datasets, and the weights of various features to respective datasets. (Samom et al., 2021) have used the CICDDoS 2019 dataset and tried out different ML models like logistic regression, random forest, naive Bayes, k-nearest neighbor, etc. The results show that random forest gave better results with low latency. K-nearest neighbor also showed better results but took more time to detect the attack. (Rajagopal et al., 2021) has provided a meta-classification approach for network intrusion detection in a cloud environment. Decision jungles, neural networks, and logistic regression are used for multi-class classification. They have verified their approach with UNSW NB-15 (The UNSW-NB15 DATASET), (Moustafa & Slay, 2015), (Moustafa & Slay, 2016), (Moustafa et al., 2017), (Moustafa et al., 2017), (Sarhan et al., 2021), CICIDS 2017 (Sharafaldin et al., 2018), (Intrusion Detection Evaluation Dataset (CIC-IDS2017)), and CICDDOS 2019 attack datasets. (Kshirsagar & Kumar, 2022) have improved attack detection accuracy by implementing a feature reduction method. They have used information gain and correlation feature selection techniques. They also use the CICDDoS 2019 attack dataset with the J48 classifier. (Assis et al., 2021) proposed an SDN defense system against intrusion and DDoS attacks. The proposed approach comprises two main parts, the detection and mitigation modules. The detection module consists of gated recurrent, which is a recurrent DL approach. (Ma et al., 2023) have also used CICDDoS 2019 dataset to train their random forest model in SDN environment. They also reduce the total feature set to only 24 features. We optimize the feature selection process further by reducing it to 15.

7.3 Proposed Methodology

Our proposed approach aims to achieve near real-time DDoS attack detection with minimal impact on the victim. The methodology involves the implementation of a trained ML model within the network, which will be responsible for predicting potential attacks. This model uses the traffic data obtained through port mirroring to make accurate predictions. Consequently, our proposed work can be divided into two main parts.

The first part entails the selection and training of the ML model, while the second part involves setting up an SDN network to thoroughly test and validate the model's performance.



Preprocessing
Feature Reduction
Hyper-parameter tuning
CICDDoS-2019 Dataset
Portmap Dataset of CICDDoS-2019
Replace NAN and ∞ by 0.
Repeat for SYN, UDP, DNS,UDPLAG
Repeat for SYN, UDP, DNS,UDPLAG
Randomly split the dataset into 70% training set and 30% testing set
Perform oversampling of benign data samples in the training set
Calculate f1-score, precision, recall using algorithm **Find_MI-Coeff ()** for list of features from oversampled dataset
Calculate f1-score, precision, recall using algorithm **Find_Cor-Coeff ()** for list of features
Select top M_{PORT} features as explained in section 7.4.3 Similarly get M_{SYN} M_{DNS} M_{UDP} M_{UDPLAG}
Select top N_{PORT} features as explained in section 7.4.3 Similarly get N_{SYN} N_{DNS} N_{UDP} N_{UDPLAG}
Take union $M = M_{SYN} \cup M_{DNS} \cup M_{UDP} \cup M_{UDPLAG} \cup M_{PORT}$
Take union $N = N_{SYN} \cup N_{DNS} \cup N_{UDP} \cup N_{UDPLAG} \cup N_{PORT}$
$K = N \cup M$
Compute Gini impurity score for K features, select top K_T features, and merge all five datasets on these K_T features
Perform Hyper-Parameter tuning (as explained in section 7.3.5) on 5 Hyper-Parameters using OOB score as validation
Tuned Random forest classifier

Figure 7.1 : Proposed ML architecture

7.3.1 Machine learning Architecture

Our model training process involves utilizing the CICDDoS-2019 dataset, specifically selecting datasets such as DNS, SYN, UDP, UDP-lag, and Portmap, encompassing reflection and volumetric attacks. As illustrated in Figure 7.1, we perform preprocessing and feature reduction on the dataset before feeding it into the model. Additionally, we fine-tune the hyperparameters prior to deploying the trained model in the network.

7.3.2 Model Selection

We use random forest, an ML algorithm, to train our model. (Samom et al., 2021) have used the CICDDoS 2019 dataset and tried out different machine learning models like logistic regression, random forest, naive bayes, k-nearest neighbor, etc., The results show that random forest (BREIMAN, 2001) gave better results with low latency. So, we have proceeded with the random forest ML algorithm for our use case.

7.3.3 Preprocessing

As outlined in Section 7.1, we employ the CICDDoS-2019 attack dataset for training the random forest model. Before inputting the dataset into the model, we perform preprocessing steps, which involve replacing any infinity and NaN values with 0. It is important to note that we choose not to drop these values to prevent any loss of information. In order to address the dominance of attack data within the dataset, we employ a data augmentation technique known as oversampling before feeding the dataset into the model. Through oversampling, we increase the representation of benign data to achieve a more balanced distribution with the attack data. This process effectively enlarges the overall dataset size.

Considering memory limitations, during the model training phase, we utilize three million samples for each dataset, encompassing all 78 features, for the purpose of feature selection. Subsequently, as we progressively reduce the number of features through the feature selection approach, we increase the number of samples to enhance the final training of the random forest model.

7.3.4 Feature Selection

To enable near real-time detection of DDoS attacks without compromising accuracy, reducing the number of features is essential. While ensuring the detection accuracy remains at approximately 99%, we reduced the features from 88 to 15. The rationale behind opting for a limited set of features is to efficiently retrieve the minimum necessary information from raw pcap files in real-time, precisely during the occurrence of an attack. Research conducted by (Ma et al., 2023) has demonstrated that reducing the number of features can significantly diminish attack prediction time while maintaining prediction accuracy.

We aim to reduce the dataset's feature count without altering the fundamental feature values. We employ feature subset selection techniques rather than feature reduction to achieve this. As a result, we utilize Correlation and Mutual Information classifiers for feature selection, eschewing feature reduction methods like PCA. PCA, a dimensionality reduction technique, derives principal components grounded in feature covariance. By employing correlation, we get the linear relationship between two variables. When applied to feature selection, it helps identify features with a strong linear correlation with the target variable. MI is a more general measure of the dependence between variables. It can capture non-linear relationships, which correlation may miss. MI makes it suitable for feature selection in cases where the relationship between features and the target is not strictly linear. Hence, we take a union of the features selected from these two algorithms to capture both linear and non-linear relationships.

Algorithm 7.1: Process for feature reduction

```
Input- CICDDoS_2019_dataset- [data1=SYN, data2=PORTMAP, data3=DNS,
data4=UDP, data5=UDPLAG]
/*data is in the form of 2-D matrix i.e. - data[a,b] where a= flow-id number and
 b= feature number
Let, $f_{[n]}$ represents the $n^{th}$ feature, where (n= 1 to 78) */
```

```
Algorithm- Find_Cor-Coeff ()
Repeat for each of the five datasets, i.e. SYN, UDP, UDPLAG, DNS, PORTMAP
{
for ( n = 1; n <= 78; n ++)
        Cor [n] = Correlation-Coefficient of $f_n$ with target feature 'Label'
for ( i = 0.1; i < 1.0; i += 0.05)
{
   for ( n = 1; n <= 78; n ++)
   {
     if ( Cor [n] >i)
     List_cor [i] [n] = $f_n$
    }
            Calculate F1 score, precision, recall, and accuracy of List_cor[i]
      }
}
Algorithm- Find_MI-score ()
Repeat for each of the five datasets, i.e. SYN, UDP, UDPLAG, DNS, PORTMAP
{
for ( n = 1; n <= 78; n ++)
        MI [n] = MI-score of $f_n$ with target feature 'Label'
for ( i =.0001; i  < MI; i += 0.05)
{
   for ( n = 1; n <= 78; n ++)
   {
     if ( MI [n] >i)
     List_MI [i] [n] = $f_n$
    }
            Calculate F1 score, precision, recall, and accuracy of List_MI[i]
      }
}
```

In addition to reducing the overall number of features from 88 to 15, it is important to note that some features of type string (object) were also excluded from consideration to perform data augmentation using oversampling. The feature selection procedure was specifically applied to the remaining 78 features. This methodology involves leveraging the Pearson correlation and mutual info (using the SelectKBest feature selection method) of sklearn library. By employing these methods, we were able to effectively identify the most relevant and informative features for the task at hand. The process can be divided into the following steps:

a) As shown in algorithm 7.1, we separately calculate the correlation coefficient and Mutual Info (MI) of Portmap, SYN, DNS, UDP, and UDP-lag datasets. This calculation was performed using all 78 available features. We determine the range for the correlation coefficient as values ranging from 0.1 to 1, and for the MI score, the range is set from 0.0001 to the maximum MI value until only one feature remains. We calculate various evaluation metrics within these specified ranges, such as precision, recall, and F1 score, for both the benign (0th class) and attack (1st class) datasets. We calculate these metrics specifically for the features falling within the aforementioned range.

b) Precision, recall, and F1 score graphs are plotted with varying correlation coefficients and MI scores, as shown in Figure (7.2-7.6), (The data for these graphs is provided in Appendix C). The x-axis represents the correlation value or MI score in these graphs, while the y-axis represents the corresponding precision, recall, and F1 score. When observing the graphs, if there is a noticeable dip in the value for a particular correlation coefficient or MI score, we consider the features from the dataset that have a value greater than or equal to that specific coefficient or score.

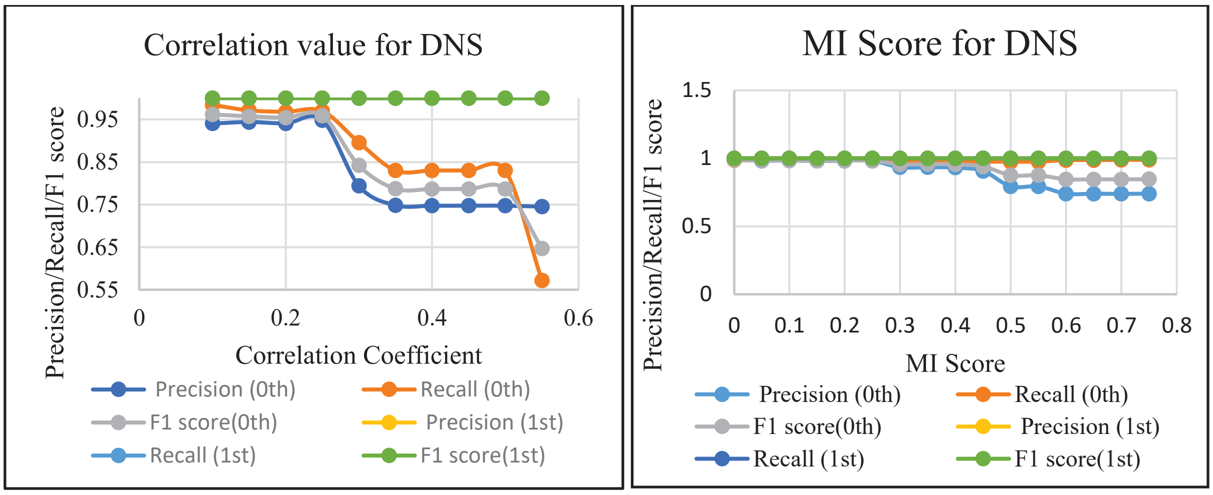


Figure 7.2 : Precision, F1 score and recall for correlation and MI score of DNS dataset

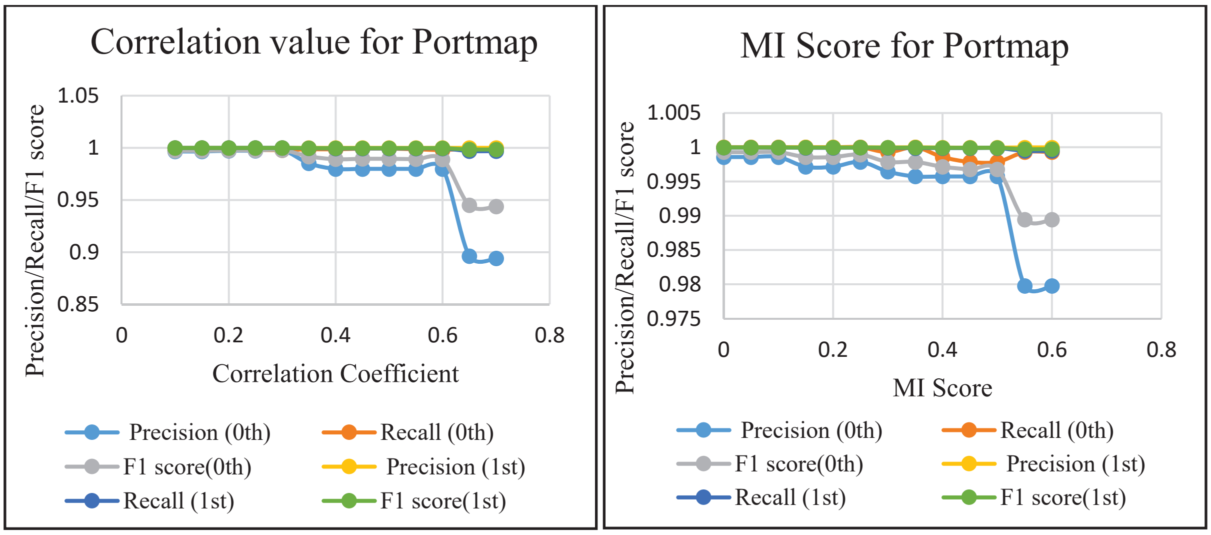


Figure 7.3 : Precision, F1 score and recall for correlation and MI score of Portmap dataset

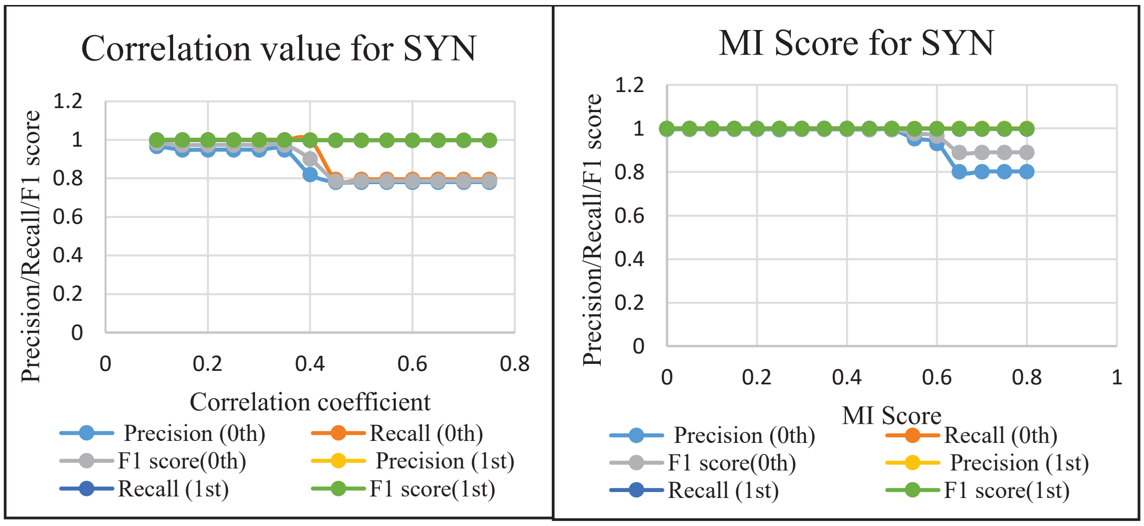


Figure 7.4 : Precision, F1 score and recall for correlation and MI score of SYN dataset

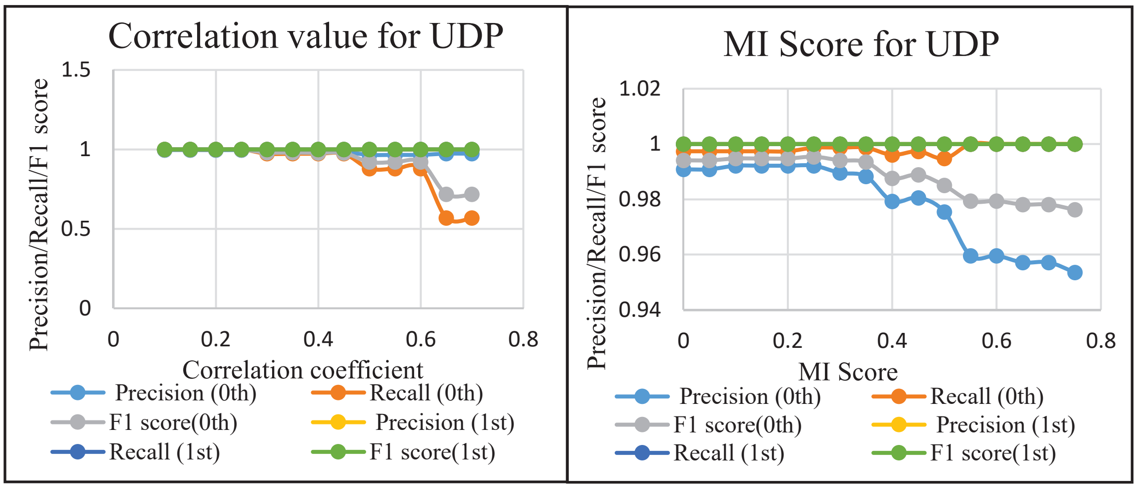


Figure 7.5 : Precision, F1 score and recall for correlation and MI score of UDP dataset

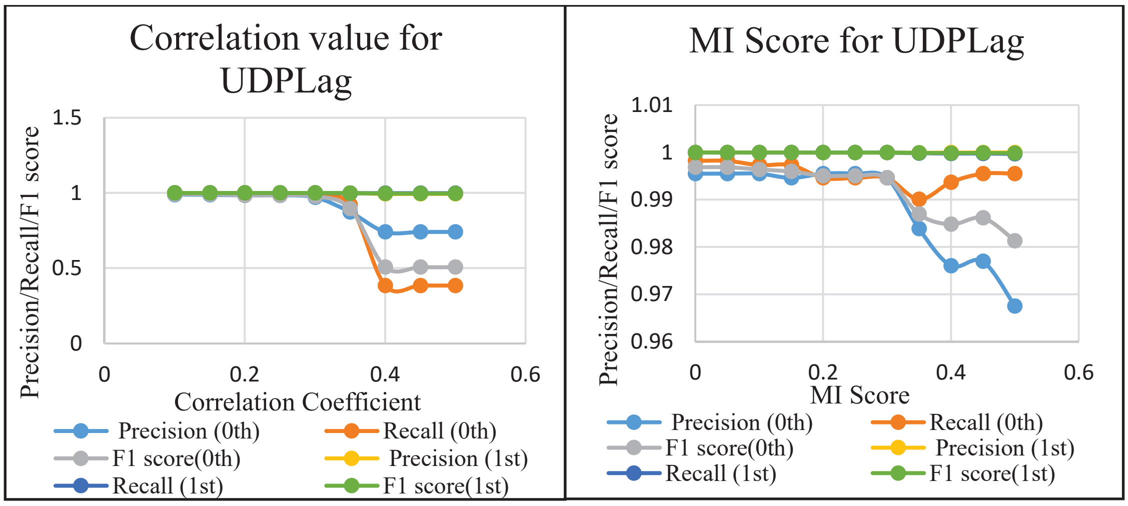


Figure 7.6 : Precision, F1 score and recall for correlation and MI score of UDPLag dataset

c) After this, all the shortlisted features of individual datasets were combined for correlation and MI scores. Using gini-impurity on the whole dataset (combining the five datasets), we shortlisted the top 15 features (feature_importance). These 15 features with description are shown in Table 7.1. For e.g.- the feature ‘ Fwd Packet Length Min’ means minimum size of packet in forward direction, and ‘Bwd IAT Max’ means maximum time between two packets sent in the backward direction, etc.

With these 15 features, we got an accuracy of 99.99% from the Random-Forest model.

Table 7.1 : Selected feature values and their description

S.No.	Feature Name	Description
1	‘ Fwd Packet Length Min’,	Minimum size of packet in forward direction
2	‘URG Flag Count’	Number of packets with URG
3	‘Min Packet Length’	Minimum length of a packet
4	‘Avg Fwd Segment Size’	Average size observed in the forward direction
5	‘ACK Flag Count’	Number of packets with ACK
6	‘Init_Win_bytes_forward’	The total number of bytes sent in initial window in the forward direction
7	‘Bwd IAT Total’	Total time between two packets sent in the backward direction
8	‘Average Packet Size’	Average size of packet
9	‘Fwd Packet Length Mean’	Mean size of packet in forward direction
10	‘Packet Length Mean’	Mean length of a packet
11	‘Packet Length Std’	Standard deviation length of a packet
12	‘Packet Length Variance’	Variance length of a packet
13	‘Bwd IAT Max’	Maximum time between two packets sent in the backward direction
14	‘Flow Duration’	Duration of the flow in Microsecond
15	‘Flow IAT Mean’	Mean time between two packets sent in the flow

The feature importance or relevance in sklearn is calculated by normalizing the decrease in node impurity through the likelihood of reaching that node (Ronaghan, 2018). We computed the node probability by dividing the total number of samples by the number of samples that reach the node. The feature with a higher value is more significant. Feature importance F_t of feature t is calculated as (Ronaghan, 2018):

$$F_t = \frac{\sum_{i=number\ of\ splits\ of\ node\ i} \nabla_i^t}{\sum_{\alpha=all\ nodes} \alpha^t} \dots\dots\dots\dots\dots \text{(7.1)}$$

∇_i^t= node i importance calculated by gini importance.

For each decision tree, scikit-learn calculates node importance (∇_i^t) as

$$\nabla_i^t = Wt_s^i \cdot g^i - Wt_l^i \cdot g_l^i - Wt_r^i \cdot g_r^i \text{...........} (7.2)$$

- ∇_i^t= ith node importance calculated by gini importance
- Wt_s^i = weighted samples reaching node i
- g^i = gini impurity of the ith node
- Wt_l^i= weighted samples reaching the left split of node i
- g_l^i = gini impurity of left split
- Wt_r^i = weighted samples reaching the right split of node i
- g_r^i = gini impurity of right split

To determine the ultimate feature relevance at the random forest level, the process involves calculating the importance of each attribute for every tree and then dividing the total by the number of trees; it is :-

$$F_t^{RF} = \frac{\sum_{k=number\ of\ trees} F_{t_k}^{norm}}{Total\ trees} \dots\dots\dots\dots\dots\dots (7.3)$$

- F_t^{RF}= final feature importance of feature t, through all random forest trees.
- $F_{t_k}^{norm}$= normalized feature importance of feature t

7.3.5 Hyper-Paramter Tuning

Before deploying the trained model in the network, we perform hyper-parameter tuning on the random forest model using the complete dataset.

The hyper-parameters that we tune include n_estimators (number of trees), max_depth, min_samples_leafs, max_features, and max_leaf_nodes. We aim to identify the optimal values for each hyper-parameter, which we achieve by evaluating the OOB scores. The OOB score is computed using data that was not utilized during the model's analysis, and therefore we rely on the OOB scores to select hyper-parameter values. Figure 7.7 illustrates the OOB scores compared to the training scores for various hyper-parameter values.

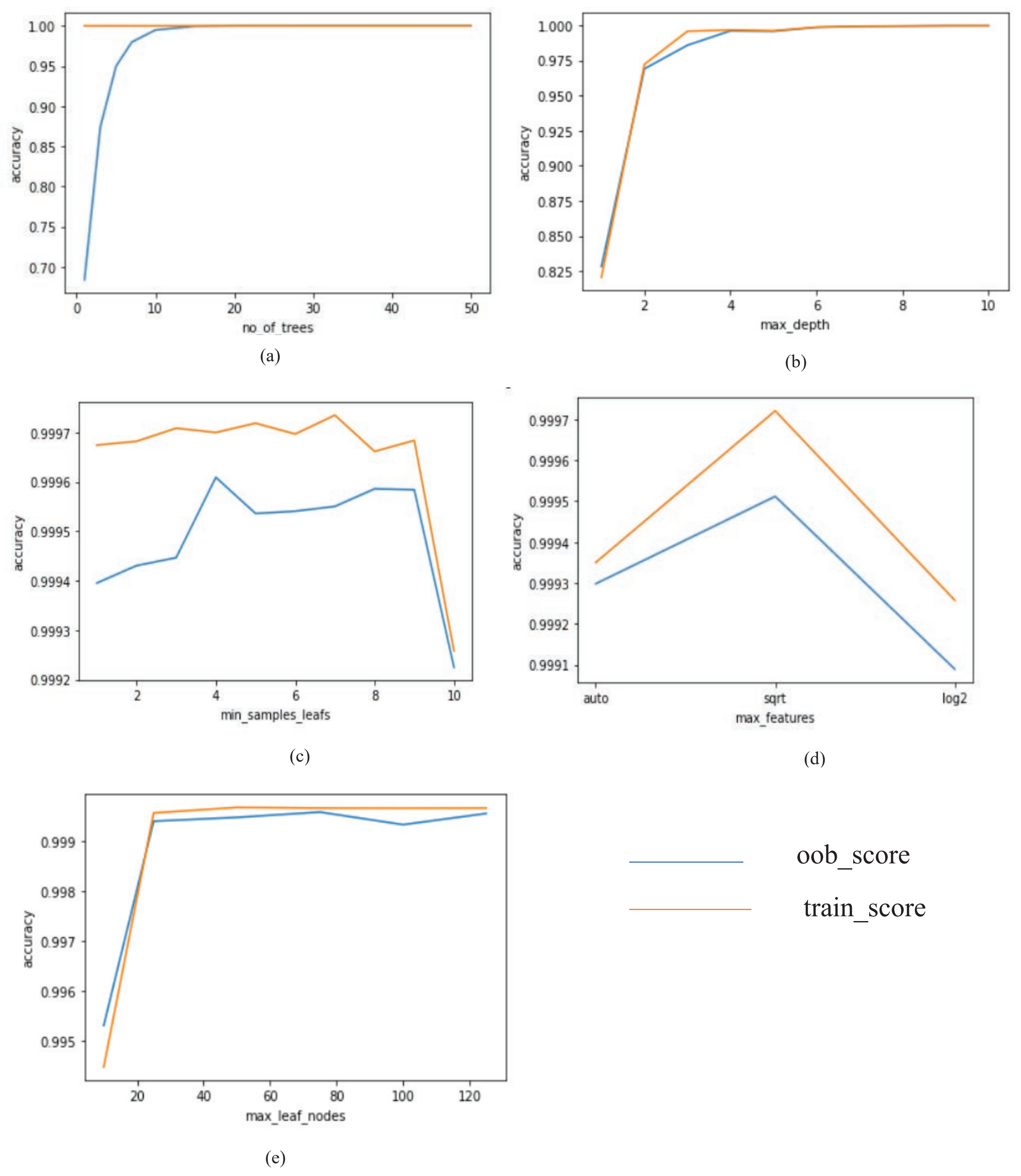


Figure 7.7 : Accuracy and OOB_score of hyper-tuned parameters (a) no_of_trees (b) max_depth (c)min_samples_leaf (d) max_features (e) max_leaf_nodes

With the help of these graphs, the values chosen for hyper-parameters are shown in Table 7.2-

Table 7.2 : Optimal values of the chosen hyper-parameters

S. No.	Hyper-Parameter	Optimal Value
1	n_estimators	20
2	max_depth	8
3	min_samples_leaf	4
4	max_features	sqrt
5	max_leaf_nodes	75

The hyper-parameters min_samples_split, oob_score, and verbose are kept at default values which are two, true, and true, respectively. The accuracy of the trained model on the selected features and the tuned hyper-parameters is 99.94%. This trained random forest model was now applied in our SDN environment to validate the approach.

7.4 The SDN Barrier

As the name suggests, the SDN barrier is a barrier between the attacker and the victim, as shown in Figure 7.8. It is a term used for a setup in an SDN environment to prevent the DDoS attack from reaching the victim. It consists of an SDN controller, the SDN switch responsible for mirroring the traffic, and the detection module (the machine running the ML model for predicting the attack). It is a protective measure that can be applied anywhere between the victim and the attacker organization.

To facilitate the detection of malicious traffic, the traffic directed towards the victim is mirrored to a dedicated node running the trained random forest model. This model assesses the traffic and predicts whether it is benign or an attack.

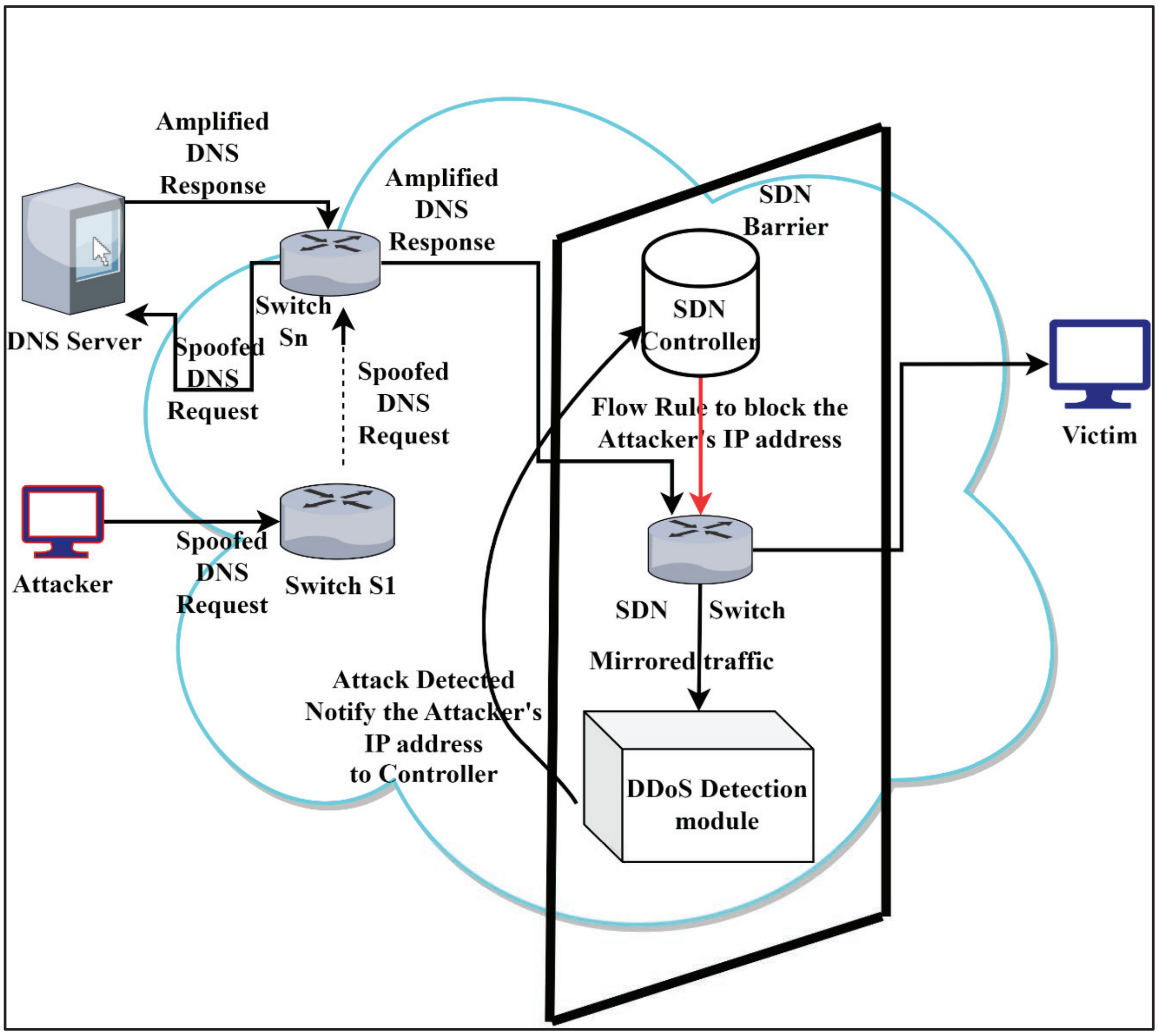


Figure 7.8 : Proposed architecture of SDN barrier

In order to enable this process, the SDN controller configures flow rules for port mirroring on the SDN switch. When the traffic is identified as benign, no further action is taken. However, if the traffic is determined to be malicious, the SDN controller is instructed to block all traffic originating from the corresponding IP address. This is achieved by the controller pushing a flow rule to the switch, thereby dropping any flows from the attacker's IP address.

We simulate a real-case scenario in the SDN environment, where an attacker organization attacks a victim. We have deployed a DNS_DRDoS attack using a DNS Server. The methodology is as follows:

- Initialization: Set up the SDN environment, including the SDN controller, SDN switch, and the node running the trained model.
- DNS_DRDoS attack setup: Configure the DNS server to execute the DNS_DRDoS attack.
- Traffic mirroring: Configure the SDN controller to enable port mirroring on the SDN switch. This ensures that all incoming traffic, including the attack traffic, is duplicated and forwarded to the node running the trained model for analysis.
- Traffic analysis: The trained random forest model on the node analyzes the mirrored traffic in real-time. Based on the learned patterns and features, it classifies the traffic as benign or malicious.
    a. The computing node saves the traffic in the form of .pcap files. These files are saved every 20 seconds or when they exceed 50 MB in size, whichever happens first.
    b. The .pcap files are processed using CICFlowmeter (Cybersecurity). This tool extracts the relevant features from the .pcap files and generates a .csv file.
    c. The generated .csv file is then fed into the pre-trained machine learning classifier. The classifier analyzes the network flows and classifies them as either malicious or benign based on the learned patterns and features.
- Detection of malicious traffic: If the model identifies any traffic as malicious, it notifies the SDN controller about the attacker's IP address associated with the malicious traffic.
- Flow rule deployment: The SDN controller dynamically pushes flow rules to the SDN switch, instructing it to drop any flows originating from the identified attacker's IP address. This action effectively blocks the attack traffic from reaching the victim.
- False positives: If the victim sends a request packet to an IP address previously blocked, the system identifies it as a potential false positive. In such cases, the controller is instructed to unblock the IP address to allow the traffic to flow freely.

By following these steps, the system can effectively monitor and classify network flows in real-time, proactively blocking malicious traffic and mitigating potential false positives for a more efficient and accurate prevention and mitigation system.

7.4.1 Handling False Positives-

There may be instances where benign traffic is erroneously classified as attack traffic by the model, leading to false positives and subsequent blocking of benign IP addresses by the controller. For instance, in the case of a DNS_DRDoS attack, the controller may block the DNS server responsible for generating attack traffic toward the victim. Consequently, if the victim sends a DNS request to this server, the server's response will not reach the victim because it is already blocked. We have implemented a mechanism to unblock IP addresses to address this issue if the victim generates a request directed toward that specific IP address. This ensures that even if the controller mistakenly blocks the DNS server, it will be unblocked once the victim initiates a request to that IP address. By incorporating this concept, we prevent any disruption to legitimate communication between the victim and the blocked IP address, mitigating the impact of false positives.

7.5 Result And Analysis

This section describes the experimental setup done to validate the proposed approach. The time taken to detect the attack through the proposed model is also calculated and shown in this section.

7.5.1 Experimental Setup

As shown in Figure 7.9, to test the proposed approach's effectiveness, we simulated a network using Mininet (Mininet, 2022).

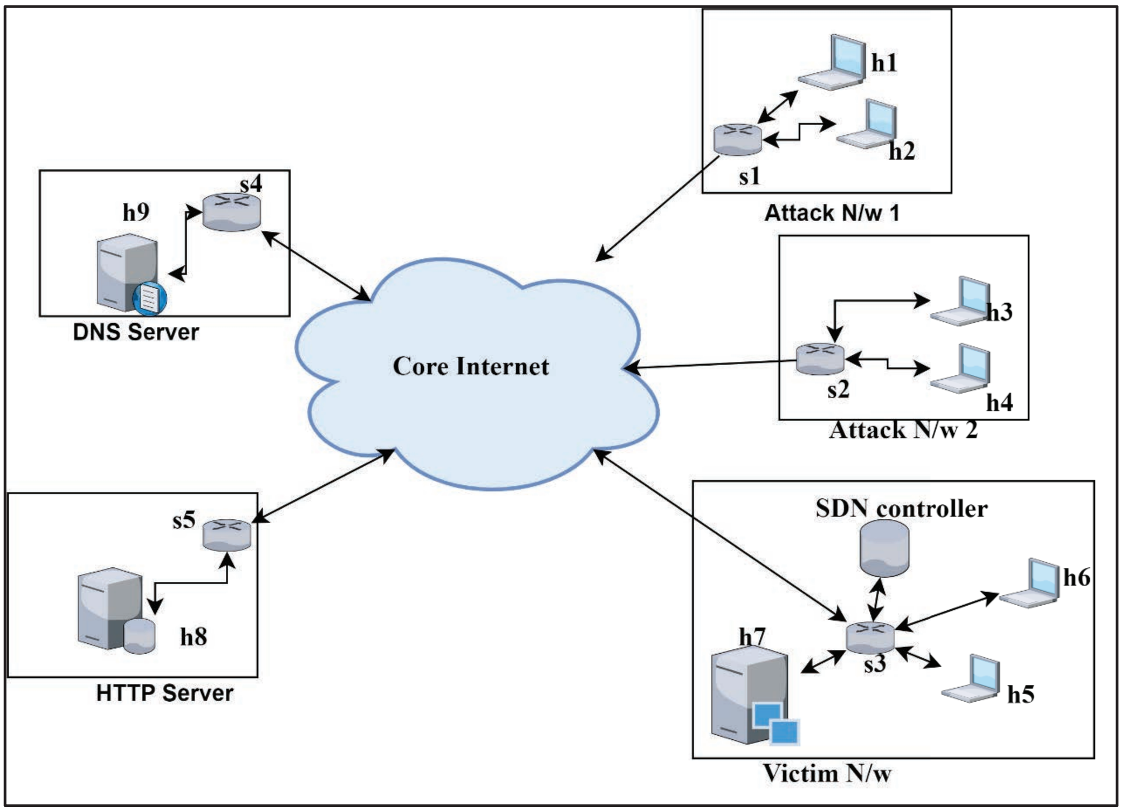


Figure 7.9 : Experimental setup

The topology consists of a victim's network where the model runs on node h7, an attacker's network, a DNS server for generating attack traffic, and a test server for generating benign traffic. The simulations steps are as follows-

- The attackers perform a DNS reflection attack on the victim network, specifically on the node labeled 'h5'. There are two attack networks, each consisting of two attackers, i.e., the nodes 'h1', 'h2', 'h3', and 'h4'. They spoof the source IP address to match that of 'h5' and send DNS request packets to the DNS server 'h9'.
- The nodes 'h5' and 'h6' in the victim network randomly send and receive benign traffic from the TCP test server 'h8'.

In this simulation, the S3 switch acts as an SDN barrier. The controller installs a flow rule to automatically send all traffic that enters and exits the victim network to node 'h7'.

- The trained ML model is running on node ‘h7’. If it detects a malicious flow, it notifies the controller to install a drop rule in ‘S3’ (SDN switch). It corresponds to the malicious flow’s source IP address and blocks it. If the victim network sends traffic towards the blocked IP address even after it was blocked, ‘h7’ notifies the controller to remove the drop rule from ‘S3’ as it is likely the result of a false positive.

We ran the attack simulation for six minutes and six seconds since that is the median length of a DDoS attack (*The Median Duration of DDoS Attacks Was 6.1 Minutes in the First Half of 2021*). To test the algorithm’s efficiency, we calculated the following parameters:- total attack traffic sent, total attack traffic reaching the victim, total benign traffic sent from the test server, total benign traffic received from the test server, time taken to block the attack, percentage of attack traffic reaching the victim, percentage of benign traffic reaching the victim, and avg attack rate.

7.5.1 Results and Observations

Table 7.3 shows the calculated parameters as mentioned in section 7.5 with varying attack rates ranging from 200Mbps to 1Gbps. The time taken to block the attack ranges from 5.3 to 3.8 seconds. The detection time appears to decrease when the avg attack rate increases as the .pcap reach 50MB faster due to the faster rate of incoming traffic.

Table 7.3 : Traffic at the victim with varying attack rates

	200Mb/s	**400Mb/s**	**600Mb/s**	**800Mb/s**	**1Gb/s**
Total attack traffic sent	8.2GB	15.16GB	21.03GB	27.47GB	30.84GB
Total attack traffic reaching the victim	108.51MB	145MB	211.50MB	218MB	241MB
Total benign traffic sent from test server	2.8MB	2.6MB	2.8MB	2.53MB	2.8MB

Total benign traffic received from test server	2.8MB	2.6MB	2.8MB	2.53MB	2.8MB
Time taken to block the attack	5.3 seconds	4.3 seconds	4.1 seconds	3.8 seconds	3.8 seconds
Percentage of attack traffic reaching the victim	1.31%	0.96%	1.01%	0.80%	0.79%
Percentage of benign traffic reaching the victim	99.99%	99.99%	100%	100.00%	99.98%
Avg attack rate	174.79 MBit/s	323.95 MBit/s	451.35 MBit/s	601.37 MBit/s	705.92 MBit/s

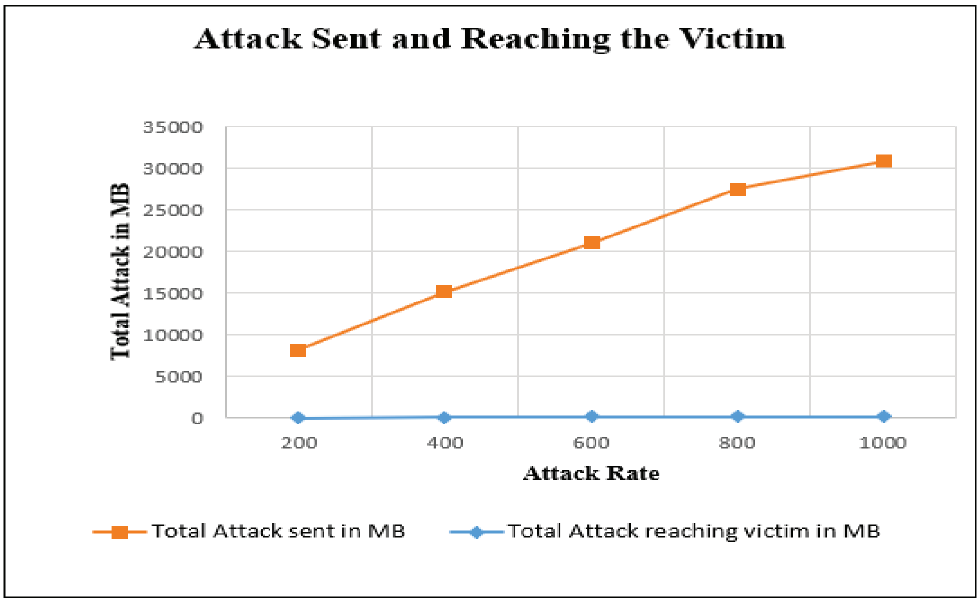


Figure 7.10 : Generated attack vs. attack reaching the victim

As shown in Figure 7.10, the attack traffic reaching the victim is much less than the traffic sent. On the other hand, as shown in Figure 7.11, the benign traffic sent is nearly the same as that received by the victim. On average, 99.27% of the benign traffic is successfully transferred across the network. It implies that the DDoS attack does not hinder benign traffic.

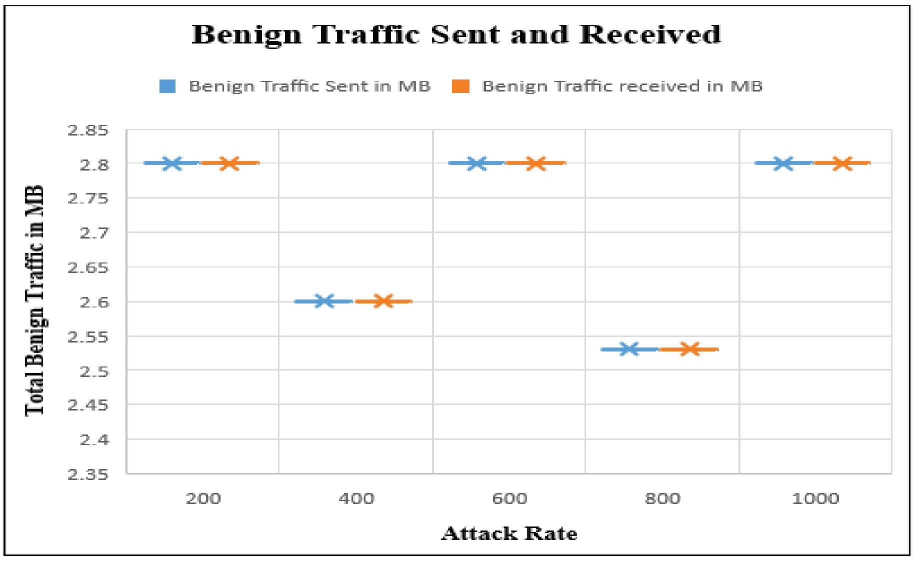


Figure 7.11 : Benign traffic sent vs. reaching the victim

7.6 Summary of the Chapter

In this chapter, we propose a model for near real-time detection of DDoS attacks using random forest. The datasets used for training the model are DNS, SYN, UDP, UDP-lag, and Portmap of the CICDDoS-2019 dataset. We perform feature reduction using correlation and MI. The finalized features are ‘Fwd Packet Length Min,’ ‘URG Flag Count,’ ‘Min Packet Length,’ ‘Avg Fwd Segment Size,’ ‘ACK Flag Count,’ ‘Init_Win_bytes_forward,’ ‘Bwd IAT Total,’ ‘Average Packet Size,’ ‘Fwd Packet Length Mean,’ ‘Packet Length Mean,’ ‘Packet Length Std,’ ‘Packet Length Variance,’ ‘Bwd IAT Max,’ ‘Flow Duration,’ ‘Flow IAT Mean.’

To assess the accuracy of the trained model in a real-time scenario, we construct a SDN topology and generate both attack and legitimate traffic. The attack traffic is simulated to have a speed of approx. 1 Gbps, while legitimate traffic is also included in the simulation. As mentioned earlier, out of the total attack traffic with a speed of 1 Gbps, only 0.79% of the attack traffic reaches the victim, while the remaining 99.21% is effectively blocked. On the other hand, 99.98% of the benign traffic successfully reaches the victim. This demonstrates the effectiveness of the proposed model in mitigating and intercepting the majority of the attack traffic, thus minimizing the impact on the victim.

Furthermore, the trained model exhibits an impressive accuracy rate of 99.9% in detecting the attacks. This high accuracy is achieved through meticulous hyper-parameter tuning and optimized feature selection, reducing the feature set from 88 to 15 relevant features. These improvements significantly enhance the performance of the random-forest model, ensuring robust and accurate detection of attacks within the network.

********** End of Chapter **********

CHAPTER 8- Detection of Distributed Denial of Service Attacks using Entropy on Sliding Window with Dynamic Threshold

8.1 Introduction

In the context of network traffic, entropy refers to the degree of randomness or unpredictability found within the transmitted data. It serves as a metric for evaluating network traffic patterns' complexity and informational content. When discussing network traffic, entropy can be determined through various methods. One common approach involves examining the distribution of data within the network traffic payload. If the values within the payload are evenly spread out, the entropy will be higher. Conversely, if certain values or patterns dominate the payload, the entropy will be lower. A low entropy value indicates that the network traffic exhibits predictable patterns or structures. This can be indicative of specific types of network traffic or anomalies. For instance, network traffic generated by automated bot activity might display low entropy due to the repetitive nature of their actions. Hence; leading to the detection of DDoS attacks. This chapter elaborated on detection of DDoS attacks using entropy, instead of prevention as defined in the previous chapters.

Examining entropy within network traffic holds several benefits, including network monitoring, intrusion detection, anomaly detection, and traffic classification. By analyzing the entropy of network traffic, network administrators and security professionals can gain valuable insights into the characteristics and behavior of the network. This knowledge enables them to identify potential security threats or abnormalities more effectively.

In the literature, many researchers use entropy to detect DDoS attacks. The majority of detection algorithms based on entropy use a static threshold (Bülbül & Fischer, 2020) (Ali et al., 2021). However, such a static threshold would not work efficiently for DDoS attack detection, resulting in many false positives. For instance, in normal traffic, the entropy values can drop due to high traffic from a host and lead to a prediction of an attack, hence false positives. To address this issue, we use a dynamic threshold as proposed by (Wang et al., 2015) for SDN to detect the attack. We used the sliding window concept to get more accurate results on the dataset. For sliding the window, our left and right pointers move by one second to predict the attack in a particular time interval. We identify whether there is a sudden drop in the entropy value of a particular window compared to the average entropy value for some previously encountered windows. Such a drop in entropy value would increase the count of violations. To increase the effectiveness of attack detection, we look for the particular number of such violations in the previously fixed number of windows. It helps us to reduce the number of false positives significantly. The proposed methodology is tested on the (DNS_DRDoS and Portmap) dataset available in the CICDDoS2019 attack dataset, which resembles true real-world data. More on this will be explained in section 8.3.

8.2 Related Work

Entropy measures the randomness in network traffic. One such type of entropy to measure uncertainty is Shannon's entropy (Shannon, 1951). If n is the number of packets in a window and p_i is the probability of occurrence of event x_i, Shannon's entropy H(X) is given by eq. (8.1) and eq. (8.2) –

$$H(X) = -\sum_{i=1}^{n} p_i log p_i \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8.1)$$

$$where, p_i = \frac{x_i}{\sum_{i=1}^{n} x_i} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (8.2)$$

Different authors have used variations of Shannon's entropy to detect DDoS attacks. These include General Entropy(GE) (Sahoo et al., 2018), fast entropy (David & Thomas, 2015), φ entropy (R. Li & Wu, 2020), etc. Sahoo et al. (Sahoo et al., 2018), have proposed the use GE to detect DDoS attacks on SDN controllers. Taking advantage of flow-based traffic in SDN controllers, GE is used to detect low-rate DDoS attacks. Generalized entropy $H_\alpha(X)$ is given by (Sahoo et al., 2018), as shown in eq. (8.3) –

$$H_\alpha(\mathrm{X}) = \frac{1}{1-\alpha} log_2(\textstyle\sum_{i=1}^{n} p_i(\alpha)) \dots\dots\dots\dots\dots\dots \dots (8.3)$$

where α is the order of general entropy, varying which different values of entropy can be calculated. When α = 1, it becomes Shannon's entropy. (R. Li & Wu, 2020) have also proposed the detection of DDoS attacks against SDN controllers using entropy. They have used φ entropy for the detection of DDoS attacks. φ entropy is used to widen the differences between attack traffic and normal traffic, and also adjust parameters according to network conditions. φ entropy, Hφ (X) is given by (R. Li & Wu, 2020), as shown in eq. (8.4) –

$$H_\varphi(X) = -\frac{1}{\sinh(\varphi)} (\textstyle\sum_{i=1}^{n} p_i \sinh(\varphi \, log_2 p_i))\dots\dots\dots\dots\dots\dots\dots\dots (8.4)$$

The parameter φ is used to adjust the sensitivity of measuring the frequency of events, where φ > 0. To reduce computation time in the calculation of entropy, David et al. (David & Thomas, 2015) proposed to use fast entropy to detect DDoS attacks. Their approach is based on flow-based analysis while keeping an adaptive threshold based on traffic patterns. A flow consists of all packets with the same source and destination IP/port pair for a certain amount of time. Let a random variable x(i, t) represent the flow count of a particular connection i over a given time interval t. The fast entropy $H_{(i,t)(}X)$ for a particular interval and particular connection is calculated as shown in eq. (8.5) and eq. (8.6):

$$H_{(i,t)}(X) = -\log \frac{x_{(i,t)}}{\sum_{i=1}^{n} x_{(i,t)}} + \tau_{(i,t)} \dots\dots\dots\dots\dots\dots\dots\dots\dots .. (8.5)$$

$$where,\ \tau_{(i,t)} = (\left(log \frac{x_{(i,t+1)}}{x_{(i,t)}} \right), if\ x_{(i,t)} \geq x_{(i,t+1)},$$

$$\left(log\frac{x_{(i,t)}}{x_{(i,t+1)}}\right), if\ x_{(i,t)} < x_{(i,t+1)}) \dots\dots\dots\dots\dots\dots\dots\dots (8.6)$$

To evaluate the approach proposed in this chapter, the CICDDoS-2019 attack dataset has been used. (R. Zhou et al., 2021), (J. Li et al., 2020), (Bülbül & Fischer, 2020), and (Ali et al., 2021) have also used this dataset to evaluate their approaches based on entropy. To detect anomalies caused in network traffic by botnets and DDoS attacks, (R. Zhou et al., 2021) proposed the use of the Euclidean distance-based multi-scale fuzzy entropy algorithm. The input is taken as a time series X = (x_1, x_2, ... x_n) with a time scale τ, and the output is the entropy value. They analyzed their algorithm on the CICDDoS-2019 dataset and have shown entropy curves on different time scales. In the approach proposed by (J. Li et al., 2020), the primary focus is on volumetric DDoS attacks. For this, they proposed the use of an optimized sliding window for entropy calculation. They used Shannon's entropy only but on a joint pair of (source IP, source port) and (destination IP, destination port), making it a joint entropy. (Bülbül & Fischer, 2020) proposed DDoS detection and mitigation using NFV and SDN environment. For detection of attack, they used entropy. After detecting an attack, traffic is further analyzed to generate attack patterns so that attack traffic can be differentiated from legitimate traffic. (Ali et al., 2021) have proposed the use of entropy and Sequential Probability Ratio Test (SPRT) for DDoS detection. Flows are formed after monitoring the packets, and these flows are gathered in specific window sizes. After this entropy calculation, SPRT is used for the detection of DDoS attacks.

From the approaches mentioned above, we can say that entropy is effective for detection of DDoS attacks. But to detect all types of DDoS attacks, not confined to only volumetric attacks, we need to make our threshold for DDoS detection dynamic so that it can change according to attack traffic and accurately provide detection. The approach proposed in this chapter uses a dynamic threshold with entropy to give better results.

8.3 Proposed Methodology

In the proposed approach, we calculate entropy at fixed time intervals. We use the concept of flows in this approach. A flow is a five-tuple entity, and a flow count is the total number of packets of a flow in a particular time interval. Flow ID, source address, destination address, source port, and destination port are the five tuples of a flow. So, a flow is a uni-directional flow of packets from one source port, source IP address, to another destination port, destination IP address. The entropy value would remain relatively stable without the attack, fluctuating in a specific range. During an attack, the network switch will have multiple packets with the same destination address.

Shannon's entropy would drop significantly when a few flows would dominate others. The algorithm considers entropy for a window, and we keep on sliding this window to consider different time intervals. Table 8.1 describes the notations used in the proposed approach. The algorithm is explained in Algorithm 8.1 (EBDD). The entropy can be calculated as per eq. (8.7) to (8.9), where n is the total number of flows in the interval ΔT, and left and right are pointers pointing to the leftmost and rightmost packets, respectively in the current window. *Received_Packets$_i$(T)* are the packets received till time T for the ith flow.

$$X_i = Received\_Packets_i(left + \Delta T) - Received\_Packets_i(left) \text{ ..... (8.7)}$$

$$p_i = \frac{X_i}{\sum_{i=1}^{n} X_i} \text{ ... ... (8.8)}$$

$$H += -p_i log p_i \text{ ... ... (8.9)}$$

We use a dynamic threshold as proposed by (Wang et al., 2015). We have used the sliding window concept to get more accurate results on the dataset used. For sliding the window, our left and right pointers move by 1 second to predict the attack in a particular time interval. Also, the proposed methodology is designed for real Internet scenarios, where different types of DDoS attacks can take place. Hence, it is tested on DNS_DRDoS and Portmap dataset of CICDDoS-2019 attack dataset, which resembles true real-world data.

Table 8.1 : Notations used in the proposed approach

Notation	**Definition**
n	The total number of flows in an interval
N	The number of nearest normal entropy values (used for calculating average entropy)
D	Dictionary consisting of all the flows along with there flow packets
X_i	Flow packets corresponding to a flow D_i in a given window
H_i	Entropy of flow D_i in a given window
H	Total entropy of a window (calculated using summation of H_i)
E	Weighted average entropy of previous N normal traffic values
M	The minimum times the formulae ($E - H > \delta$) is to be satisfied
s_len	The length of the sliding window
S	Window size to count the number of predictions in the previous history
δ	Dynamic threshold used in prediction of attack
σ	The standard deviation of normal entropy values
λ	Threshold multiplicative factor
α	A constant value for calculation of weighted average (changes with every iteration)
startTime	A constant denoting starting of traffic in seconds
endTime	A constant denoting ending of traffic in seconds

8.3.1 Dynamic Threshold Algorithm and Attack Detection

For the current entropy calculation, if $E - H > \delta$, it would be a violation since the current entropy varies by a range of δ from the average entropy of the previous N traffic values, i.e., E. We then check if there are at least M such violations in previous S such windows, as also proposed in (Wang et al., 2015). If that is the case, a DDoS attack is detected and reported. Checking for M such violations in S ensures that we do not report a normal increase in traffic which is not an attack. If there is no violation, we update δ and the average entropy for the previous N traffic values, i.e., E, and continue the calculations for the next time window.

Algorithm 8.1 : Entropy-based DDoS detection using sliding window and flow entropy

```
Initialize the local parameters E, δ, M, S, H, N, λ, ΔT, σ, left, right, endTime, s_len
left ← startTime
right ← startTime + ΔT
while right ≤ endTime do
        H ← 0
        for i in all flow id between left and right timestamp do
            X_i = Received_Packets_i (left + ΔT) - Received_Packets_i (left)
            $p_i = \frac{x_i}{\sum_{i=1}^{n} x_i}$
            H += -p_i * log p_i
        end for
        $\boldsymbol{H = \frac{H}{\log n}}$
        if E – H > δ then:
          if M times in S then
              Report DDoS attack between window left and left + ΔT
          end if
        else
          $E = \sum_{i=1}^{N} \alpha_i . H_i$
          $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(E - H_i)^2}$
          $\delta = \sigma . \gamma$
        end if
        left += s_len
        right += s_len
end while
```

If there is no violation, we update δ and the weighted average entropy for the previous N traffic values, i.e., E, and continue the calculations for the next time window. A constant term α_i is used to give more weightage to the most recently observed entropy values as compared to the previous ones. In the proposed approach, when there is no attack prediction in a window, that window's traffic is considered normal traffic. The proposed methodology can also be understood from the flow diagram in Fig. 8.1.

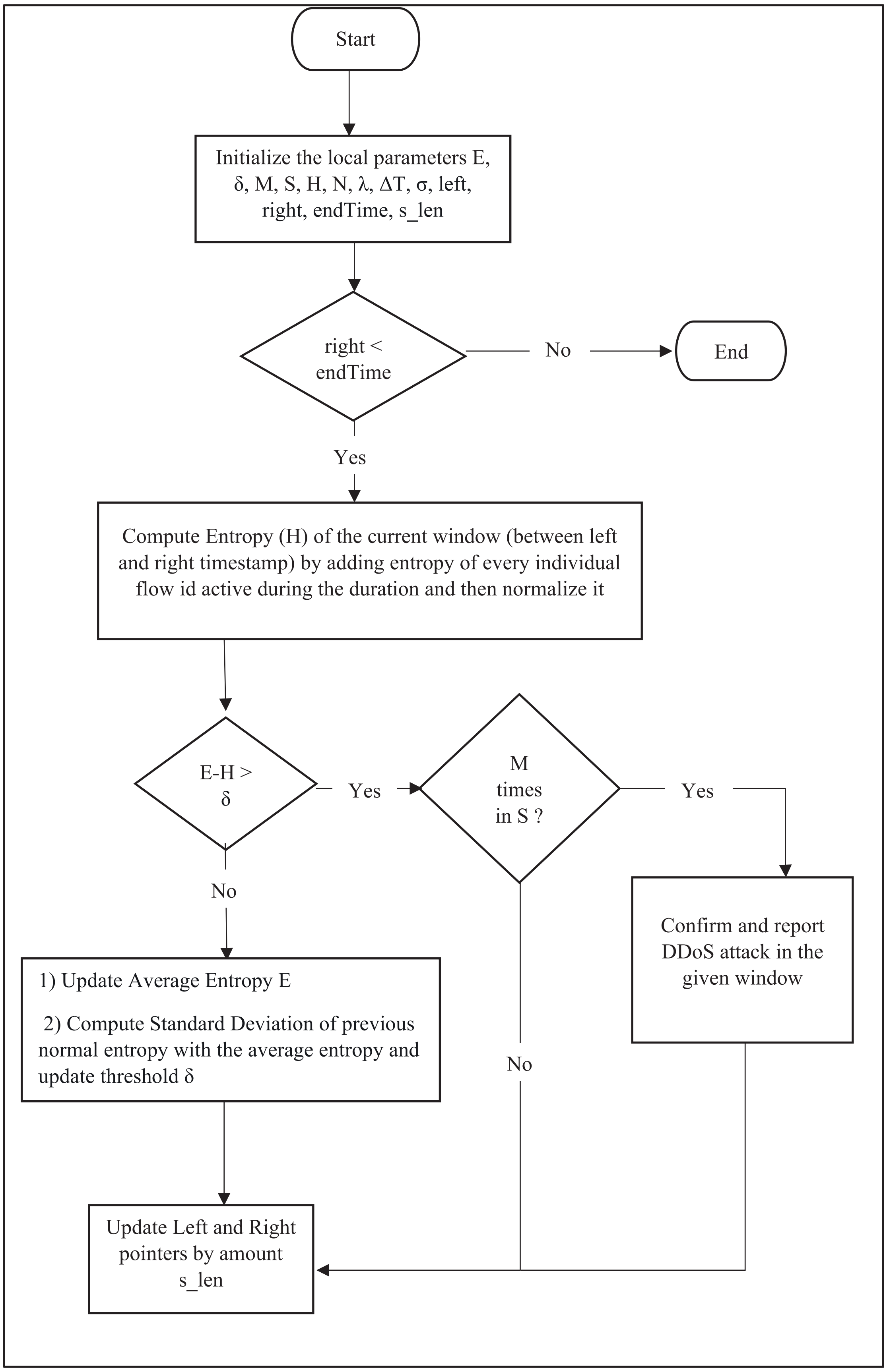

Start
Initialize the local parameters E, δ, M, S, H, N, λ, ΔT, σ, left, right, endTime, s_len
right < endTime
No
End
Yes
Compute Entropy (H) of the current window (between left and right timestamp) by adding entropy of every individual flow id active during the duration and then normalize it
E-H > δ
Yes
M times in S ?
Yes
No
Confirm and report DDoS attack in the given window
1) Update Average Entropy E
2) Compute Standard Deviation of previous normal entropy with the average entropy and update threshold δ
No
Update Left and Right pointers by amount s_len

Figure 8.1 : Flow of the dynamic threshold algorithm

8.4 Results and Discussion

The CICDDoS-2019 dataset contains benign and the most up-to-date common DDoS attacks, resembling actual real-world data. This authentic dataset contains both benign traffic and attack traffic, although the attack traffic is more in comparison to benign traffic. The dataset was pre-processed by removing infinity and NaN values before applying our algorithm. The features taken for entropy calculation are time-stamp, flow ID, and flow-packets/sec. Entropy was calculated by varying ΔT and S. To test the metrics for the prediction of an attack, DNS_DRDoS and Portmap attack datasets of CICDDoS-2019 have been used.

While accuracy is frequently employed as a metric to assess the effectiveness of a classification model, it might not be consistently suitable or enlightening, particularly in situations involving imbalanced datasets. In cases where the distribution of classes is uneven, accuracy can be deceptive. Alternative metrics, such as precision, recall, and F1 score, offer a more detailed perspective on the model's performance. Hence, our proposed algorithm's performance is measured by calculating precision, recall, and F1 score in attack prediction by varying ΔT and S. These metrics depend upon false positives, true positives, false negatives, and true negatives. False positives are legitimate traffic detected as attack traffic; true negatives are the legitimate traffic detected as legitimate traffic; false negatives are attack traffic detected as legitimate traffic, and true positives are attack traffic detected as attack traffic.

Table 8.2 : Analysis of F1 score in prediction by varying S and ΔT for DNS_DRDoS

	S=3	**S=5**	**S=7**
ΔT=10	99.965	99.965	99.965
ΔT=20	99.964	99.964	99.964
ΔT=30	99.964	99.964	99.964

As shown in Table 8.2, the F1 score for the prediction of the attack is 99.96 % for the DNS_DRDoS attack dataset on a time interval of 10 seconds. The algorithm achieves this high value as there is an attack in almost every window, leading to higher prediction and detection of attack. Taking the values ΔT=10 and S=3 from Table 8.2, we plot the graphs in Figures 8.2 and 8.3, respectively.

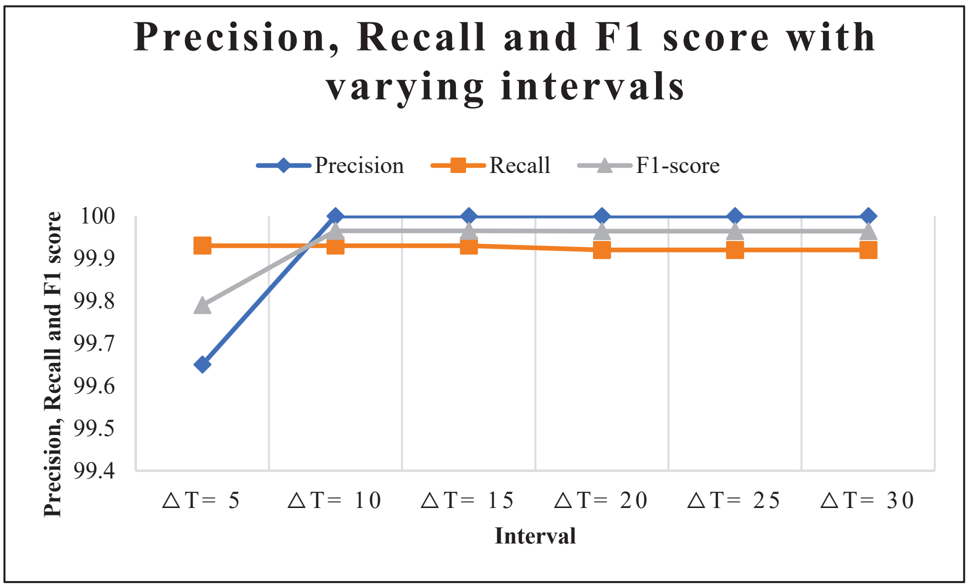


Figure 8.2 : Precision, recall, and F1 score by varying intervals for DNS_DRDoS dataset

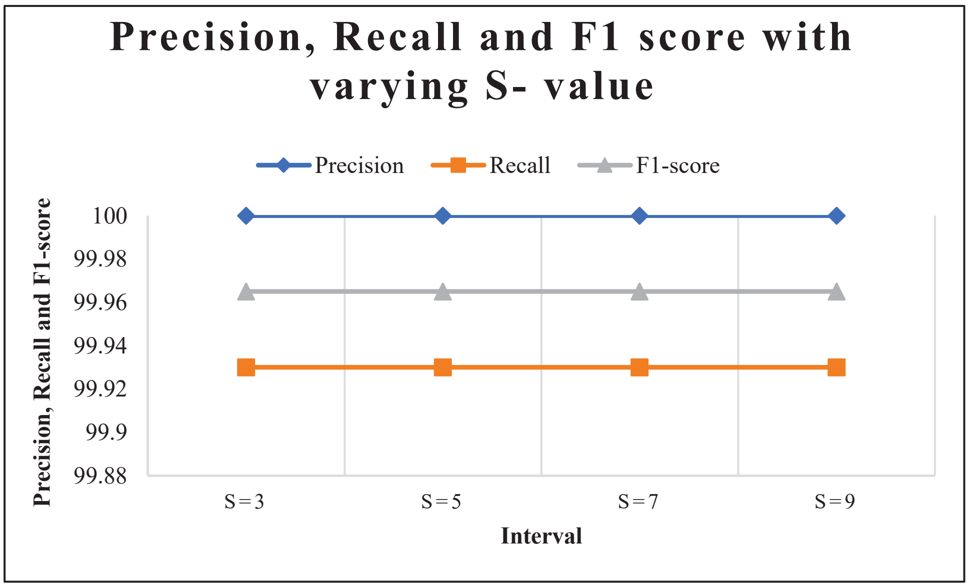


Figure 8.3 : Precision, recall, and F1 score by varying S for DNS_DRDoS dataset

Similarly, we plot the precision, recall, and F1 score v/s ΔT and v/s S graphs for the Portmap dataset in Figures 8.4 and 8.5, respectively. We plot the interval graph by keeping S=3. The time ΔT=30 seconds gave higher performance metrics for prediction compared to its previous windows, as the Portmap attack dataset has chunks of attack data and legitimate data rather than contiguous attack data. Hence, longer intervals gave higher values in predicting the attack. The precision, recall, and F1 score v/s S graph is plotted for ΔT=30. The metric values remain uniform across different S values for this interval.

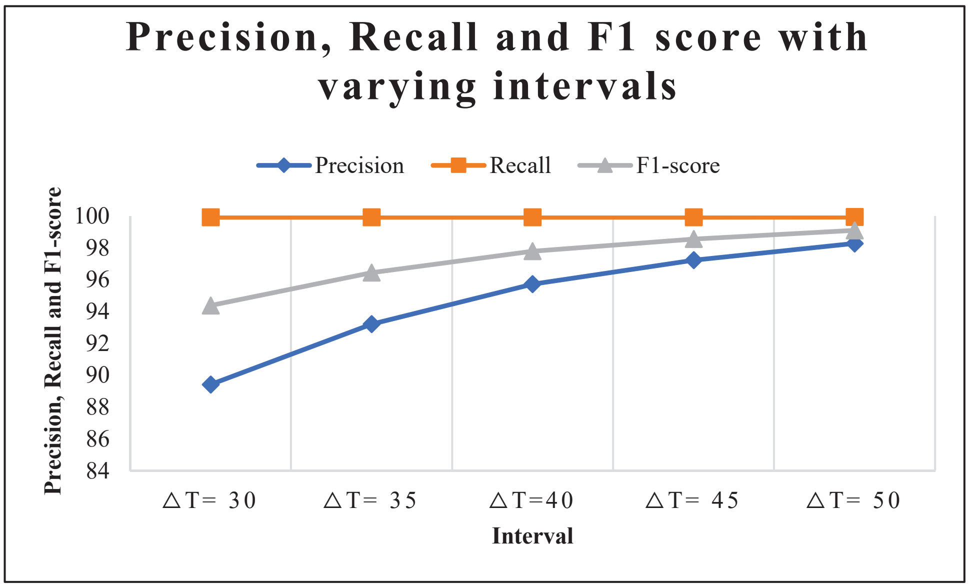


Figure 8.4 : Precision, Recall, and F1 score by varying intervals for Portmap dataset

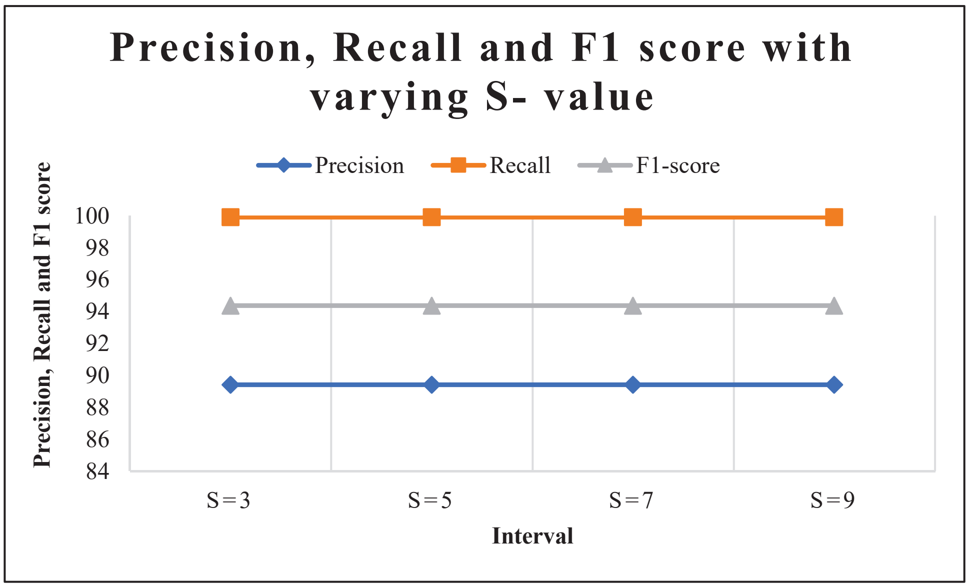


Figure 8.5 : Precision, Recall, and F1 score by varying S for Portmap dataset

8.5 Summary of the Chapter

The focus of this chapter is to detect DDoS attacks using entropy. Entropy is used to measure randomness or uncertainty in a network's traffic. As soon as this randomness decreases, i.e., one type of traffic dominates the network, the entropy value decreases. This abbreviation leads to the possibility of an attack on the network. One benefit of entropy-based algorithms is that entropy calculations require minimal computational effort. The proposed approach employs a dynamic threshold mechanism to effectively distinguish between normal and attack traffic. The entropy of a certain interval is compared to this dynamic threshold for attack prediction. In our evaluation, we investigate the impact of varying the time intervals and the entropy variation in previous windows for the precision of attack prediction. Through this analysis, we achieve a notable F1 score of 99% in predicting DNS_DRDoS attacks and 94% for Portmap attacks. The difference in the values for these predictions is because of the type of attack present in these datasets. The DNS_DRDoS attack has contiguous attack data, resulting in higher accuracy.

********** End of Chapter **********

9.1 Conclusions

This thesis provides defense techniques against DRDoS attacks, primarily focusing on Prevention. More specifically, the prevention techniques presented are IP-Switching, PortMapping, PortMergeIP, SymSDN, RDPID, PoDIBC, and RF-SDN. In addition, we also present a DDoS detection technique using entropy. The techniques provided mainly focus on two types of Prevention- *True and Partial*.

- In True Prevention, the attack traffic can leave an attacker's access network and enter the core Internet network but can never reach the target victim's network.
- In Partial Prevention, some attack traffic reaches the victim's network. An additional prevention layer is proposed to be placed in between the attacker's and the victim's networks. The purpose of this prevention layer is to detect the ongoing attack at the earliest and successively stop it from reaching the victim's network.

The suggested prevention techniques aim to make the underlying network smart enough to counter DrDoS attacks effectively. To demonstrate and validate the efficacy of the proposed methods, the underlying network is considered to be an SDN network. Depending on the specific technique, the entire network is either assumed to be SDN-enabled or only a portion of it requires SDN integration.

Based on the definitions of True Prevention and Partial prevention, the techniques proposed in this thesis can be categorized as described in Table 9.1.

Table 9.1 : Prevention techniques

Prevention Technique	**Requirement of EDGE / CORE network to be SDN enabled**	**Type Of Prevention**
1. IP-Switching	EDGE N/W (ISP works as SDN Barrier)	True Prevention
2. PortMapping	CORE N/W+EDGE N/W	True Prevention
3. PortMergeIP	EDGE N/W (ISP works as SDN Barrier)	True Prevention
4. SymSDN	CORE N/W+EDGE N/W	True Prevention
5. RDPID	CORE N/W	True Prevention
6. PoDIBC	EDGE N/W	True Prevention
7. RF-SDN	EDGE N/W (Partial Prevention Layer as SDN Barrier)	Partial Prevention

IP-Switching is a True Prevention approach, with ISP's first-hop router working as an SDN barrier. It is True Prevention as attack traffic leaves the edge network and enters the core network, but it does not reach the victim due to the IP address being switched to the downstream IP address by ISP.

PortMapping is also a True Prevention technique. However, the entire network must be SDN-enabled to implement this algorithm, as the switches push in-port information in packets for the whole path between the host and the server. It is True Prevention as the request path takes the same path as the response; hence, even when the attack traffic enters the core network, it never reaches the victim.

PortMergeIP is a True Prevention technique where the edge network and the ISP's first-hop router is required to be SDN enabled. It also overcomes the drawback of IP-Switching as the response traffic not only reaches the attacker's organization but also towards the attacker responsible for the attack. The victim remains unaffected by the attack.

SymSDN is also a True Prevention approach requiring the entire network to be SDN enabled as the OpenFlow switches store path information in its tables to implement symmetric routing. It leads to attack traffic diverting back to the attacker, even with a spoofed source IP, keeping the victim's network unaffected by the attack.

RDPID is also a True Prevention approach, and it requires modifying the entire underlying network to incorporate appropriate rules for forwarding packets based on PIDs. The response reaches back to the attacker as the forwarding is based on PIDs, not the IP address; hence the victim is unaffected.

PoDIBC is a True Prevention approach with the edge network as an SDN barrier. The client and server do the signing and verification process, with the controller providing parameters for the encryption process. It is True Prevention as the server drops the attack request packets on unsuccessful sign verification; hence, True Prevention as no response packet reaches the victim.

RF-SDN is a Partial Prevention approach with the DDoS detection module and SDN controller working as an SDN barrier. The detection module can detect the attacks in near real-time, and if the attack is detected, the controller is notified to block the malicious IP address. It is Partial Prevention as a small part of attack traffic (experimentally proved to be approximately 0.79% for a 1-Gbps attack) reaches the victim.

Finally, chapter 8 explains a DDoS detection approach using entropy. The proposed approach employs a dynamic threshold mechanism to effectively distinguish between normal and attack traffic. Although we did achieve good precision in detecting the attacks, the time interval for detection of attacks with good accuracy was more (experimentally proved to be approx. 30 seconds) for non-contiguous attack data like Portmap.

9.2 Future Research Directions

The works proposed in this thesis can be extended in the following ways-

- It is established in the thesis that prevention is a better approach than detection and mitigation of DDoS attacks. In this work, we have focused on the prevention of DRDoS attacks and provided techniques for only these attacks. This work can be extended to provide prevention from more categories of DDoS attacks like volumetric attacks and zero-day attacks.
- To cover larger networks, multiple SDN controllers will be needed, and these controllers need to communicate with each other to pass shared information. For e.g., in Chapter 6, for the PoDIBC technique, we currently connect the whole topology to a single controller, but for larger networks, multiple controllers will be needed, and these controllers need to communicate with each other to pass the public parameters. Further research is required to get more insights into the effectiveness of the proposed techniques in such cases.
- The techniques presented in the thesis require a change in the functionality of the underlying network. We have proven the techniques assuming SDN as the under lying network; however, the same can be achieved for the traditional network, which can be researched in the future.

********** End of Chapter **********

2016 Dyn cyberattack. Retrieved June 10, 2021, from https://en.wikipedia.org/wiki/2016_Dyn_cyberattack

Ahmed, M. E., Kim, H., & Park, M. (2017). Mitigating DNS query-based DDoS attacks with machine learning on software-defined networking. *Proceedings - IEEE Military Communications Conference MILCOM*, *2017-Octob*, 11–16.

Ahmed, Z., Afaqui, N., & Humayun, O. (2019). Detection and Prevention of DDoS attacks on Software Defined Networks Controllers for Smart Grid. *International Journal of Computer Applications*, *181*(45), 16–21.

Al-Duwairi, B., Özkasap, Ö., Uysal, A., Kocaoğullar, C., & Yildirim, K. (2020b). LogDoS: A Novel logging-based DDoS prevention mechanism in path identifier-Based information centric networks. *Computers and Security*, *99*, 102071.

Ali, B. H., Sulaiman, N., Al-Haddad, S. A. R., Atan, R., & Hassan, S. L. M., Alghrairi, M. (2021). Identification of distributed denial of services anomalies by using combination of entropy and sequential probabilities ratio test methods. *Sensors*, *21(19)*.

Andersen, D. G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D., & Shenker, S. (2008). Accountable internet protocol (AIP). *ACM SIGCOMM 2008 Conference on Data Communication*, 339–350.

Assis, M. V. O., Carvalho, L. F., Lloret, J., & Proença, M. L. (2021). A GRU deep learning system against attacks in software defined networks. *Journal of Network and Computer Applications*, *177*(November 2020), 102942.

Aura, T. (2005). *Cryptographically Generated Addresses (CGA), RFC 3972*. https://www.rfc-editor.org/rfc/rfc3972

Avramopoulos, I., & Suchara, M. (2009). Protecting the DNS from routing attacks: Two alternative anycast implementations. *IEEE Security and Privacy*, *7*(5), 14–20.

Bagnulo, M., & Garcia-Martinez, A. (2014). *SEcure Neighbor Discovery (SEND) Source Address Validation Improvement (SAVI)*. available: https://rfc-editor.org/rfc/rfc7219.txt

Baker, F., & Savola, P. (2004). *Ingress Filtering for Multihomed Networks* (pp. 1–16). IETF. https://www.hjp.at/doc/rfc/rfc3704.html

Barreto, P. S. L. M., Libert, B., McCullagh, N., & Quisquater, J. J. (2005). Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3788 LNCS*, 515–532.

Bawany, N. Z., Shamsi, J. A., & Salah, K. (2017). DDoS attack detection and mitigation using SDN: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, *42*(2), 425–441. https://doi.org/DOI 10.1007/s13369-017-2414-5

Bhardwaj, K., Miranda, J. C., & Gavrilovska, A. (2018). Towards IoT-DDoS prevention using edge computing. *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*.

Bhatia, S., Behal, S. and A. I. (2018). Distributed Denial of Service Attacks and Defense Mechanisms: Current Landscape and Future Directions. In *Verstile Cybersecurity* (pp. 55–97). Springer, Cham.

Bi, J., Wu, J., Yao, G., & Baker, F. (2015). *Source Address Validation Improvement (SAVI) Solution for DHCP*. https://rfc-editor.org/rfc/rfc7513.txt

Bi, J., Yao, G., Halpern, J. M., & Levy-Abegnoli, E. (2017). *Source Address Validation Improvement (SAVI) for Mixed Address Assignment Methods Scenario*. https://rfceditor.org/rfc/rfc8074.txt

Booters, Stressers and DDoSers. Retreived June 10, 2021, from, https://www.imperva.com/learn/ddos/booters-stressers-ddosers/

Breiman, L. (1996). Bagging predictors. *Machine Learning, Springer*, *24*.

BREIMAN, L. (2001). Random Forests. *Machine Learning, Springer*, *45*.

Bülbül, N. S., & Fischer, M. (2020). SDN/NFV-based DDoS Mitigation via Pushback. *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, 1–6.

CAIDA's Skitter MAP. Retrieved October 1, 2021, from https://www.caida.org/~bhuffake/papers/skitviz/

Chen, C. C., Chen, Y. R., Lu, W. C., Tsai, S. C., & Yang, M. C. (2017). Detecting amplification attacks with Software Defined Networking. *2017 IEEE Conference on Dependable and Secure Computing*, 195–201.

Chen, X., Xiao, L., Feng, W., Ge, N., & Wang, X. (2021). DDoS Defense for IoT: A Stackelberg Game Model Enabled Collaborative Framework. *IEEE Internet of Things Journal*, *4662*(c), 1–16.

Cimpanu, C. (2020a). *AWS said it mitigated a 2.3 Tbps DDoS attack, the largest ever*. Retreived June 11, 2021, from, https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever/

Cimpanu, C. (2020b). *Google says it mitigated a 2.54 Tbps DDoS attack in 2017, largest known to date*. Retreived June 11, 2021, from, https://www.zdnet.com/article/google-says-it-mitigated-a-2-54-tbps-ddos-attack-in-2017-largest-known-to-date/

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280*. https://www.rfc-editor.org/rfc/rfc5280

Cybersecurity, C. I. for. *CICFlowMeter (formerly ISCXFlowMeter)*. Retrieved March 28, 2023, from https://www.unb.ca/cic/research/applications.html#CICFlowMeter

Daemen, J., Govaerts, R., & Vandewalle, J. (1994). Weak keys for IDEA. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *773 LNCS*, 224–231.

Dao, N. N., V. Phan, T., Saad, U., Kim, J., Bauschert, T., Do, D. T., & Cho, S. (2021). Securing Heterogeneous IoT With Intelligent DDoS Attack Behavior Learning. *IEEE Systems Journal*, 1–10.

Dao, N. N., Vu, D. N., Lee, Y., Park, M., & Cho, S. (2018). MAEC-X: DDoS prevention leveraging multi-access edge computing. *International Conference on Information Networking*, *2018-Janua*, 245–248.

David, J., & Thomas, C. (2015). DDoS attack detection using fast entropy approach on flowbased network traffic. *Procedia Computer Science,50,* 30–36.

DDOS attacks and the GitHub case. (2018). Institute of Research on Internet and Society. Retreived June 11, 2021, from, https://irisbh.com.br/en/ddos-attacks-and-the-github-case/

DDoS Evaluation Dataset (CIC-DDoS2019),. University of Brunswick. Retrieved June 11, 2021, from https://www.unb.ca/cic/datasets/ddos-2019.html

DDoS quick guide. (2020). Retreived June 12, 2021, from, https://us-cert.cisa.gov/sites/default/files/publications/DDoS Quick Guide.pdf

Deadlocks. Retrieved June 13, 2021, from https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/7_Deadlocks.html

Dhanapal, A., & Nithyanandam, P. (2019). The slow http ddos attacks: Detection, mitigation and prevention in the cloud environment. *Scalable Computing*, *20*(4), 669–685.

Duan, Q., Al-Shaer, E., Chatterjee, S., Halappanavar, M., & Oehmen, C. (2018). Proactive routing mutation against stealthy Distributed Denial of Service attacks: metrics, modeling, and analysis. *The Journal of Defense Modeling and Simulation*, *15*(2), 219–230.

Ehrenkranz, T., & Li, J. (2009). On the state of IP spoofing defense. *ACM Transactions on Internet Technology*, *9*(2).

Elliptic Curve Cryptography (ECC). (2022). Retreived April 30, 2023, from, https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc

Famous DDoS attacks | The largest DDoS attacks of all time. (2021). Retreived June 10, 2021, from, https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/

François, J., Aib, I., & Boutaba, R. (2012). FireCol: A collaborative protection network for the detection of flooding DDoS attacks. *IEEE/ACM Transactions on Networking*, *20*(6), 1828–1841.

Frankel, S., & Krishnan, S. (2011). *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap RFC 6071*. https://datatracker.ietf.org/doc/html/rfc6071

Freiling, F. C., Holz, T., & Wicherski, G. (2005). Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. *European Symposium on Research in Computer Security*, 319–335.

Ghoshal, A. (2018). *How GitHub braved the world's largest DDoS attack*. Retreived June 11, 2021, from, https://thenextweb.com/news/how-github-braved-the-worlds-largest-ddos-attack

Godfrey, P. B., Ganichev, I., Shenker, S., & Stoica, I. (2009). Pathlet routing. *Computer Communication Review*, *39*(4), 111–122.

Goncalves, J. A., Faria, V. S., Vieira, G. B., Silva, C. A., & Mascarenhas, D. M. (2017). WIDIP: Wireless distributed IPS for DDoS attacks. *2017 1st Cyber Security in Networking Conference, CSNet 2017*, *2017-Janua*, 1–3.

Greg, L. (2006). *Geeking with Greg*. Retreived June 10, 2021, from, http://glinden.blogspot.com/2006/04/early-amazon-shopping-cart.html

Grover, S., & Mittal, P. (2016). A novel model based on group controlled observation for ddos attack detection and prevention in vanet. *Indian Journal of Science and Technology*, *9*(36).

Gupta, B. B., Joshi, R. C., & Misra, M. (2010). Distributed Denial of Service Prevention Techniques. *International Journal of Computer and Electrical Engineering*, *2*(2), 268–276.

Harikrishna, P., & Amuthan, A. (2021). Rival-Model Penalized Self-Organizing Map enforced DDoS attack prevention mechanism for software defined network-based cloud computing environment. *Journal of Parallel and Distributed Computing*, *154*, 142–152.

Hu, G., Chen, W., Li, Q., Jiang, Y., & Xu, K. (2017). TrueID: A practical solution to enhance Internet accountability by assigning packets with creditable user identity code. *Future Generation Computer Systems*, *72*, 219–226.

Huong, T. T., & Thanh, N. H. (2017). Software defined networking-based One-packet DDoS mitigation architecture. *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication, IMCOM 2017*, 2–8.

IEEE Standard for Identity-Based Cryptographic Techniques using Pairings. (2013). *IEEE Std 1363.3-2013*, 1–151.

Imperva. *TCP SYN Flood*. Retrieved April 4, 2023, from, https://www.imperva.com/learn/ddos/syn-flood/

Introduction: About Scapy. (2023). Retreived July 3, 2022, from, https://scapy.readthedocs.io/en/latest/introduction.html

Intrusion Detection Evaluation Dataset (CIC-IDS2017). Retrieved March 7, 2023, from https://www.unb.ca/cic/datasets/ids-2017.html

Islam, M., Chowdhury, M., Li, H., & Hu, H. (2018). Cybersecurity attacks in vehicle-to-infrastructure applications and their prevention. *Transportation Research Record*, *2672*(19), 66–78.

Jaber, A. N., Zolkipli, M. F., Shakir, H. A., & Jassim, M. R. (2018). Host based intrusion detection and prevention model against ddos attack in cloud computing. *Lecture Notes on Data Engineering and Communications Technologies*, *13*, 241–252.

Jingle, I. D. J., & Rajsingh, E. B. (2014). ColShield: an effective and collaborative protection shield for the detection and prevention of collaborative flooding of DDoS attacks in wireless mesh networks. *Human-Centric Computing and Information Sciences*, *4*(1), 1–19.

Kalkan, K., & Alagöz, F. (2016). A distributed filtering mechanism against DDoS attacks: ScoreForCore. *Computer Networks*, *108*, 199–209.

Kaushal, K., & Sahni, V. (2016). Early Detection of DDoS Attack in WSN. *International Journal of Computer Applications*, *134*(13), 975–8887.

Kent, S., & Atkinson, R. (1998). *Security Architecture for the Internet Protocol, RFC 2401*. https://datatracker.ietf.org/doc/html/rfc2401

Keromytis, A. D., Misra, V., & Rubenstein, D. (2004). SOS: An architecture for mitigating

DDoS attacks. *Journal on Selected Areas in Communications*, *22*(1), 176–188.

Kim, S., Lee, S., Cho, G., Ahmed, M. E., Jeong, J. P., & Kim, H. (2017). Preventing DNS amplification attacks using the history of DNS queries with SDN. *In European Symposium on Research in Computer Security*, 135–152.

Kim, Y., Lau, W. C., Chuah, M. C., & Chao, H. . (2006). PacketScore: a statistics-based packet filtering scheme against distributed denial-of-service attacks. *IEEE Transactions on Dependable and Secure Computin*, *3*(2), 141–155.

Kohler, E., Morris, R., Chen, B., Jannotti, J., & Kaashoek, M. F. (2000). The click modular router. *ACM Transactions on Computer Systems*, *18*(3), 263–297.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., & Uhlig, S. (2015). *Software-Defined Networking : A Comprehensive Survey*. *103*(1).

Kshirsagar, D., & Kumar, S. (2022). A feature reduction based reflected and exploited DDoS attacks detection system. *Journal of Ambient Intelligence and Humanized Computing*, *13*(1), 393–405.

Kupreev Oleg, B. E., & Alexander, G. (2019). *DDoS attacks in Q3 2019*. Retreived June 11, 2021, from, https://securelist.com/ddos-report-q3-2019/94958/

Kupreev Oleg, B., Ekaterina, G., & Alexander. (2021). *DDoS attacks in Q4 2020*. Retreived June 10, 2021, from, https://securelist.com/ddos-attacks-in-q4-2020/100650/

Lad, N., Prof, A., & Baria, J. (2014). *DDoS Prevention on Rest Based Web Services* in) International Journal of Computer Science and Information Technologies, *5*(6), 7314–7317.

Lai, X., & Massey, J. L. (1991). A proposal for a new block encryption standard. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *473 LNCS*, 389–404.

Laskar, S., & Mishra, D. (2016). Qualified Vector Match and Merge Algorithm (QVMMA) for DDoS Prevention and Mitigation. *Procedia Computer Science*, *79*, 41–52.

Le Pennec, J. F., Bruno, A., & Grisi, N. (2014). *Method and system for symmetric routing* (Patent No. U.S. Patent No. 8,634,428.). Washington, DC: U.S. Patent and Trademark Office.

Li, C., Wu, Q., Li, H., & Zhou, J. (2019). SDN-Ti: A General Solution Based on SDN to Attacker Traceback and Identification in IPv6 Networks. *IEEE International Conference on Communications*, *2019-May*.

Li, J., Liu, M., Xue, Z., Fan, X., & He, X. (2020). A real-time volumetric detection scheme for ddos in the internet of things. *IEEE Access*, *8*.

Li, J., Mirkovic, J., Wang, M., Reiher, P., & Zhang, L. (2002). SAVE: Source address validity enforcement protocol. *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, 1557–1566.

Li, R., & Wu, B. (2020). Early detection of DDoS based on ϕ-entropy in SDN networks. *In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 731–735.

Liu, X., Li, A., Yang, X., & Wetherall, D. (2008). Passport: Secure and adoptable source authentication. *NSDI*, *8*, 365–378.

Liu, Y., Ren, G., Wu, J., Zhang, S., He, L., & Jia, Y. (2015). Building an IPv6 address generation and traceback system with NIDTGA. *Address Driven Network*, *58*(12), 1–14.

Liu, Z., Cao, Y., Zhu, M., & Ge, W. (2018). Umbrella: Enabling ISPs to offer readily deployable and privacy-preserving DDoS prevention services. *IEEE Transactions on Information Forensics and Security*, *14*(4), 1098–1108.

Liu, Zhuotao, Jin, H., Hu, Y. C., & Bailey, M. (2018). Practical Proactive DDoS-Attack Mitigation via Endpoint-Driven In-Network Traffic Control. *IEEE/ACM Transactions on Networking*, *26*(4), 1948–1961.

Long, Y., & Xiong, F. (2020). Secret sharing based BLMQ signature generation. *ACM International Conference Proceeding Series*, 6–12.

Luo, H., Chen, Z., Li, J., & Vasilakos, A. V. (2017). Preventing distributed denial-of-service flooding attacks with dynamic path identifiers. *IEEE Transactions on Information Forensics and Security*, *12*(8), 1801–1815.

Luo, H., Lin, Y., Zhang, H., & Zukerman, M. (2013). Preventing DDoS attacks by identifier/locator separation. *IEEE Network*, *27*(6), 60–65.

Luo, Hongbin, Chen, Z., Cui, J., Zhang, H., Zukerman, M., & Qiao, C. (2014). *CoLoR: An Information-Centric Internet Architecture for Innovations*. *June*, 4–10.

Ma, R., Wang, Q., Bu, X., & Chen, X. (2023). Real-Time Detection of DDoS Attacks Based on Random Forest in SDN. Applied Sciences (Switzerland), 13(13).

Malhi, A. K., & Batra, S. (2016). Genetic-based framework for prevention of masquerade and DDoS attacks in vehicular ad-hocnetworks. *Security and Communication Networks*, *9*(15), 2612–2626.

Menscher, D. (2020). *Exponential growth in DDoS attack volumes*. Retreived June 11, 2021, from, https://cloud.google.com/blog/products/identity-security/identifying-and-protecting-against-the-largest-ddos-attacks

Mininet. (2022). Retreived July 3, 2022, from, http://mininet.org/

Mirković, J., Prier, G., & Reiher, P. (2003). Source-end DDoS defense. *Proceedings - 2nd IEEE International Symposium on Network Computing and Applications, NCA 2003*, 171–178.

Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, *34*(2), 39–53.

Mishra, S., Phuc, V. M., & Tanh, N. V. (2023). Lightweight Authentication Encryption to Improve DTLS, Quark Combined with Overhearing to Prevent DoS and MITM on Low-Resource IoT Devices. *Internet of Things – ICIOT 2022*.

Misra, S., Venkata Krishna, P., Agarwal, H., Saxena, A., & Obaidat, M. S. (2011). A learning automata based solution for preventing distributed denial of service in internet of things. *Proceedings - 2011 IEEE International Conferences on Internet of Things and Cyber, Physical and Social Computing, IThings/CPSCom 2011*, 114–122.

Moskowitz, R., & Nikander, P. (2006). *Host Identity Protocol (HIP) Architecture, RFC 4423*. https://www.rfc-editor.org/info/rfc4423

Moustafa, N., Creech, G., & Slay, J. (2017). Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models. In *Data Analytics and Decision Support for Cybersecurity*. Springer.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 Military Communications and Information Systems Conference (MilCIS)*.

Moustafa, N., & Slay, J. (2016). The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set. *Information Security Journal*, *25*(1–3), 18–31.

Moustafa, N., Slay, J., & Creech, G. (2017). Novel Geometric Area Analysis Technique for Anomaly Detection Using Trapezoidal Area Estimation on Large-Scale Networks. *IEEE Transactions on Big Data*, *5*(4), 481–494.

Munivara Prasad, K., Rama Mohan Reddy, A., & Venugopal Rao, K. (2016). Anomaly based real time prevention of under rated app-DDOS attacks on web: An experiential metrics based machine learning approach. *Indian Journal of Science and Technology*, *9*(27).

Nadeem, M., Arshad, A., Riaz, S., Band, S. S., & Mosavi, A. (2021). Intercept the cloud network from brute force and ddos attacks via intrusion detection and prevention system. *IEEE Access*, *9*, 152300–152309.

Nagar, S., Rajput, S. S., Gupta, A. K., & Trivedi, M. C. (2017). Secure routing against DDoS attack in wireless sensor network. *3rd IEEE International Conference On* , 1–6.

Navaz, A. S., Sangeetha, V., & Prabhadevi, C. (2013). Entropy based Anomaly Detection System to Prevent DDoS Attacks in Cloud. *International Journal of Computer Applications*, *62*(15), 42–47.

Ndibwile, J. D., Govardhan, A., Okada, K., & Kadobayashi, Y. (2015). Web server protection against application layer DDoS attacks using machine learning and traffic authentication. *Proceedings - International Computer Software and Applications Conference*, *3*, 261–267.

Netscout. (2019). *CLOUD IN THE CROSSHAIRS*. Retreived June 10, 2021, from, https://www.netscout.com/report/

Newman, L. H. (2018). *GitHub Survived the Biggest DDoS Attack Ever Recorded*. Retreived June 11, 2021, from, https://www.wired.com/story/github-ddos-memcached/

Noel Michael McCullagh. (2005). *Cryptographic Applications of Bilinear Maps*. Dublin City University.

Nordmark, E., Bagnulo, M., & Levy-Abegnoli, E. (2012). *FCFS SAVI: FirstCome, First-Served Source Address Validation Improvement for Locally Assigned IPv6 Addresses*.

Nurwarsito, H., & Nadhif, M. F. (2021). DDoS Attack Early Detection and Mitigation System on SDN using Random Forest Algorithm and Ryu Framework. Proceedings of the 8th International Conference on Computer and Communication Engineering, ICCCE 2021, 178–183.

Ohri, P., Arockiam, D., Neogi, S. G., & Muttoo, S. K. (2024). Intrusion Detection and Prevention System for Early Detection and Mitigation of DDoS Attacks in SDN Environment. *2024 IEEE International Students' Conference on Electrical, Electronics and*

Computer Science, SCEECS 2024, 1–6. https://doi.org/10.1109/SCEECS61402.2024.10481906

ONF. *Software-Defined Networking (SDN) Definition*. Retrieved April 10, 2023, from, https://opennetworking.org/sdn-definition/

Oo, K. K., Ye, K. Z., Tun, H., Lin, K. Z., & M., E. P. (2015). Enhancement of Preventing Application Layer Based on DDOS Attacks by Using Hidden Semi-Markov Model. In *Genetic and Evolutionary Computing* (pp. 125–135).

OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06). (2015). In *Open Networking Foundation*. https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

Osanaiye, O. A. (2015). Short Paper: IP spoofing detection for preventing DDoS attack in Cloud Computing. *18th International Conference on Intelligence in Next Generation Networks*, 139–141.

Pappas, C., Reischuk, R. M., & Perrig, A. (2016). FAIR: Forwarding accountability for Internet reputability. *IEEE 23rd International Conference on Network Protocols (ICNP)*, 189–200.

Park, K., & Lee, H. (2001). On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. *ACM SIGCOMM Computer Communication Review*, *31*(4), 15–26.

Paul, N. (2020). *Five Most Famous DDoS Attacks and Then Some*. Retrieved June 10, 2021, from, https://www.a10networks.com/blog/5-most-famous-ddos-attacks/

Poongodi, M., Vijayakumar, V., Al-Turjman, F., Hamdi, M., & Ma, M. (2019). Intrusion prevention system for DDoS attack on VANET with reCAPTCHA controller using information based metrics. *IEEE Access*, *7*, 158481–158491.

prevent. Merriam-Webster. Retrieved June 12, 2021, from, https://www.merriam-webster.com/dictionary/prevent

Prince, M. (2013a). *The DDoS That Almost Broke the Internet*. Retrieved June 11, 2021, from, https://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet/

Prince, M. (2013b). *The DDoS That Knocked Spamhaus Offline (And How We Mitigated It)*. Retrieved June 11, 2021, from, https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/

Rajagopal, S., Kundapur, P. P., & Hareesha, K. S. (2021). Towards Effective Network Intrusion Detection: From Concept to Creation on Azure Cloud. *IEEE Access*, *9*, 19723–19742.

Random Forest. IBM Cloud Education. Retrieved March 6, 2023, from https://www.ibm.com/in-en/topics/random-forest#:~:text=Random forest is a commonly,both classification and regression problems.

Ronaghan, S. (2018). *The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark*. Towards Data Science. https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

Rowshanrad, S., Namvarasl, S., Abdi, V., Hajizadeh, M., Keshtgary, M., Ieee, F., Rothenberg, C. E., Ieee, M., Azodolmolky, S., Ieee, S. M., Uhlig, S., & Ieee, M. (2014). A survey on SDN, the future of networking. *Journal of Advanced Computer Science & Technology*, *3*(2), 232.

RYU A. COMPONENT-BASED SOFTWARE DEFINED NETWORKING FRAMEWORK. Retrieved July 3, 2022, from, https://ryu-sdn.org/

Saharan, S., & Gupta, V. (2021). DDoS Prevention: Review and Issues. In *Advances in Machine Learning and Computational Intelligence* (pp. 579-586.).

Sahi, A., Lai, D., Li, Y. A. N., & Diykh, M. (2017). An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Environment. *IEEE Access*, *5*, 6036–6048.

Sahoo, K. S., Puthal, D., Tiwary, M., Rodrigues, J. J., Sahoo, B., & Dash, R. (2018). An early detection of low rate DDoS attack to SDN based data center networks using information distance metrics. *Future Generation Computer Systems*, 685–697.

Sahri, N. M., & Okamura, K. (2016). Protecting DNS services from IP spoofing-SDN collaborative authentication approach. *ACM International Conference Proceeding Series*, *15-17-June*, 83–89.

Samom, P. S., Taggu, A., & Taggu E-Mail:, A. (2021). Distributed denial of service (Ddos) attacks detection: A machine learning approach. *Lecture Notes in Networks and Systems*, *187*, 75–87.

Santos, R., Souza, D., Santo, W., Ribeiro, A., & Moreno, E. (2020). Machine learning algorithms to detect DDoS attacks in SDN. Concurrency and Computation: Practice and Experience, 32(16), 1–14.

Sarhan, M., Layeghy, S., Moustafa, N., & Portmann, M. (2021). NetFlow Datasets for Machine Learning-Based Network Intrusion Detection Systems. In *Big Data Technologies and Applications*. Springer, Cham.

Saxena, R., & Dey, S. (2015). Cloud shield: Effective solution for ddos in cloud. *International Conference on Internet and Distributed Computing Systems*, 3–10.

Schridde, C., Smith, M., & Freisleben, B. (2009). TrueIP: Prevention of IP spoofing attacks using identity-based cryptography. *SIN'09 - Proceedings of the 2nd International Conference on Security of Information and Networks*, 128–137.

Shafi, Q., & Basit, A. (2019). DDoS Botnet Prevention using Blockchain in Software Defined Internet of Things. *Proceedings of 2019 16th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2019*, 624–628.

Shannon, C. E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, *30*(1), 50–64.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, *2018-Janua*(Cic), 108–116.

Sharafaldin, I., Lashkari, A. H., Hakak, S., & Ghorbani, A. A. (2019). Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. *Proceedings - International Carnahan Conference on Security Technology*, *2019-Octob*(Cic).

Shridhar, K., & Gautam, N. (2014). A Prevention of DDos Attacks in Cloud Using Honeypot. *International Journal of Science and Research (IJSR)*, *3*(11), 2378–2383.

Siaterlis, C., & Maglaris, B. (2003). A novel approach for a Distributed Denial of Service Detection Engine. *National Technical University of Athens. Athens, Greece*, 1–16.

Singh, A. K., Jaiswal, R. K., Abdukodir, K., & Muthanna, A. (2020). ARDefense: DDoS detection and prevention using NFV and SDN. *12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 236–241.

Singh, J., & Behal, S. (2020). Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions. In *Computer Science Review* (Vol. 37, p. 100279). Elsevier Ireland Ltd.

Software-Defined Networking: The New Norm for Networks. (2012). In *Open Networking Foundation*. http://opennetworking.wpengine.com/wp-content/uploads/2011/09/wp-sdn-newnorm.pdf

Somasundaram, A., & Meenakshi, V. S. (2021). A novel three layer filtering (3L-F) framework for prevention of DDoS attack in cloud environment. *International Journal of Computer Networks and Applications*, *8*(4), 334–345.

Subbulakshmi, T., Parameswaran, P., Parthiban, C., Mariselvi, M., Anusha, J. A., & Mahalakshmi, G. (2013). A unified approach for detection and prevention of DDoS attacks using enhanced support vector machines and filtering mechanisms. *ICTACT Journal on Communication Technology*, *4*(2), 737–743.

Swami, R., Dave, M., & Ranga, V. (2019). Software-defined networking-based ddos defense mechanisms. *ACM Computing Surveys (CSUR), 52*(2), 1–36.

The median duration of DDoS attacks was 6.1 minutes in the first half of 2021. Retrieved March 23, 2023, from https://venturebeat.com/2021/09/04/the-median-duration-of-ddos-attacks-was-6-1-%0Aminutes-in-the-first-half-of-2021/

The UNSW-NB15 DATASET. Retrieved March 7, 2023, from https://research.unsw.edu.au/projects/unsw-nb15-dataset

Threat Landscape Report – Q1 2020. (2020). https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf

Timcenko, V. V. (2014). An approach for DDoS attack prevention in mobile ad hoc networks. *Elektronika Ir Elektrotechnika*, *20*(6), 150–153.

Wang, R., Jia Z., & Ju, L. (2015). An entropy-based distributed DDoS detection mechanism in software-defined networking. *IEEE Trustcom/BigDataSE/ISPA*, 310–317.

Wei, S., Wang, X., & Xu, K. (2020). NoPTPeer:Protecting android devices from stealthy spoofing and stealing in WLANs without privilege. *Proceedings - 2020 16th International Conference on Mobility, Sensing and Networking, MSN 2020*, 576–583.

What Is a Replay Attack. (2023). Kaspersky. Retreived July 1, 2022, from, https://www.kaspersky.com/resource-center/definitions/replay-attack

What is iPerf / iPerf3 ?. Retrieved July 3, 2022, from https://iperf.fr/

What is machine learning(ML)? IBM. Retrieved March 6, 2023, from https://www.ibm.com/in-en/topics/machine-learning

Wu, J., Bi, J., Bagnulo, M., Baker, F., & Vogt, C. (2013). *Source address validation improvement (SAVI) framework*. IETF. https://www.hjp.at/doc/rfc/rfc7039.html

Wu, J., Bi, J., Li, X., Ren, G., Xu, K., & Williams, M. (2008). *A Source Address Validation Architecture ,RFC 5210*. https://www.rfc-editor.org/rfc/rfc5210

Yadav, S., & Subramanian, S. (2016). Detection of Application Layer DDoS attack by feature learning using Stacked AutoEncoder. *2016 International Conference on Computational Techniques in Information and Communication Technologies, ICCTICT 2016 - Proceedings*, 361–366.

Yan, Q., Yu, F. R., Gong, Q., & Li, J. (2015). Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials*, *18*(1), 602–622.

Yiu, T. (2019). *Understanding Random Forest*. Towards Data Science. Retreived March 6, 2023, from, https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Zeebaree, S. R., Sharif, K. H., & Amin, R. M. M. (2018). Application layer distributed denial of service attacks defense techniques: a review. *Academic Journal of Nawroz University*, *7*(4), 113–117.

Zhou, Q., Yu, J., & Li, D. (2021). A dynamic and lightweight framework to secure source addresses in the SDN-based networks. *Computer Networks*, *193*(March), 108075.

Zhou, R., Wang, X., Yang, J., Zhang, W., & Zhang, S. (2021). Characterizing Network Anomaly Traffic with Euclidean Distance-Based Multiscale Fuzzy Entropy. *Security and Communication Networks*, 1–9.

Appendix A

Throughput (without SymSDN)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	114	114	114	114	114	114	114	114	114	114	114	114	114	114
18-21	114	114	114	114	114	114	114	114	114	114	114	114	114	114
21-24	114	114	114	114	114	114	114	114	114	114	114	114	114	114
24-27	114	114	114	114	114	114	114	114	114	114	114	114	114	113
27-30	114	114	114	114	114	114	114	114	114	114	114	114	114	114
30-33	114	114	114	114	114	114	114	114	114	114	114	114	114	114
33-36	114	114	114	114	114	114	114	114	114	114	114	114	114	114
36-39	114	114	114	114	114	114	114	114	114	114	114	114	114	114

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	A-Avg	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	114	114	114	114	114	114	114	42.3	52.2	43.5	51.8	42.4	40.4	46.6
18-21	114	113.5	114	114	114	114	113.975	42.3	52.2	43.5	51.8	42.4	40.4	46.6
21-24	114	114	114	114	114	114	114	33.7	34.4	36.4	46	34.7	35.7	29.9
24-27	114	114	114	114	114	114	113.95	33.7	34.4	36.4	38.7	34.7	35.7	29.9
27-30	114	114	114	114	114	114	114	34.3	36.3	36.5	38.7	42.5	33.7	38
30-33	114	114	114	113.5	114	114	113.975	34.3	27.8	36.5	36.8	30.2	33.7	38
33-36	114	114	114	114	114	114	114	37.9	27.8	31.5	36.8	30.2	36.5	39.4
36-39	114	114	114	114	114	114	114	37.9	27.8	31.5	33.2	30.2	36.5	35.9

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	V-Avg
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	53.9	54	41.2	44.6	52.4	47.3	53.4	52.9	45.9	46	44.9	43.9	51.9	47.575
18-21	53.9	54	41.2	44.6	52.4	47.3	53.4	52.9	45.9	46	44.9	43.9	51.9	47.575
21-24	32.1	33.4	39.8	35.1	36.3	34.3	29.3	34.6	37.1	40.8	34.8	34.9	29.7	35.15
24-27	32.1	33.4	39.8	35.1	35.4	34.3	29.3	34.6	37.1	38.1	34.8	34.9	29.7	34.605
27-30	34.4	37.9	39	35.9	35.4	40.8	41.7	35.9	32.4	38.1	33.8	37.4	34.8	36.875
30-33	34.4	32	30.3	35.9	40.4	32.7	35.9	32.1	32.4	42.3	33.8	37.4	34.8	34.585
33-36	34	32	30.3	34.6	40.4	32.7	35.9	32.1	36.2	42.3	37.9	37.2	35.1	35.04
36-39	34	30.9	34.2	34.6	35.7	37	37.7	36.8	37.3	36	32.5	34.2	35.1	34.45

Throughput (with SymSDN)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	59	54.1	61.3	54.1	55.7	54.1	54.1	54.1	55.3	54	47.8	54.4	54.1	54.3
18-21	37.9	54.1	37.5	54.1	55.7	54.1	54.4	54.1	55.3	54	47.8	54.4	54.1	54.3
21-24	37.9	34.9	37.5	34.8	33.7	34.7	36.1	35.4	34.2	33.1	36.9	35.2	34.5	30.5
24-27	34.9	33.6	41.9	32.9	33.7	31.4	33.6	35.9	32.2	33.1	36.9	36.9	26.6	30.5
27-30	34.9	33.6	41.9	32.9	33.7	31.4	33.6	35.9	32.2	34.8	37.1	34.6	26.6	38.4
30-33	37.7	37	37.9	36.7	31.3	31.4	26.6	6	32.2	34.8	29.5	27.2	26.6	36.6
33-36	37.7	37	37.9	36.7	35.6	38.2	26.6	34.3	36	36.3	29.5	27.2	34.1	36.6
36-39	32	35.3	34.1	23.6	35.7	39.2	26.6	30.7	36.3	37.9	29.5	27.2	34.1	32.6

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	A-Avg	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	55.2	54.3	53.7	56.1	55.8	57.4	54.945	114	114	114	114	114	114	114
18-21	55.2	54.3	53.7	56.1	55.8	39.6	51.825	114	114	114	114	114	114	114
21-24	34.8	34	32.8	42.4	35.2	39.6	35.41	114	114	114	114	114	114	114
24-27	29.5	34	32.8	36.2	36	38.1	34.035	114	114	114	114	114	114	114
27-30	29.5	37.1	38.2	36.2	36	38.1	34.835	114	114	114	114	114	114	114
30-33	29.5	35.4	32.3	40.4	48.2	38.8	32.805	114	114	114	114	114	114	114
33-36	35.5	35.4	32.3	40.4	48.2	38.8	35.715	114	114	114	114	114	114	114
36-39	35.5	33.1	32.5	35.8	37.5	30.8	33	114	114	114	114	114	114	114

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	V-Avg
0-3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
3-6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
6-9	114	114	114	114	114	114	114	114	114	114	114	114	114	114
9-12	114	114	114	114	114	114	114	114	114	114	114	114	114	114
12-15	114	114	114	114	114	114	114	114	114	114	114	114	114	114
15-18	114	114	114	114	114	114	114	114	114	114	114	114	114	114
18-21	114	114	114	114	114	114	114	114	113.5	114	114	114	114	113.975
21-24	114	114	114	114	114	114	114	114	113.5	114	114	114	114	113.975
24-27	114	114	114	114	114	114	114	114	114	114	114	114	114	114
27-30	114	114	114	114	114	114	114	114	114	114	114	114	114	114
30-33	114	114	114	114	114	114	114	114	114	114	113.5	114	114	113.975
33-36	114	114	114	114	114	114	114	114	114	114	114	114	114	114
36-39	114	114	114	114	114	114	114	114	114	114	114	114	114	114

Packet loss (without SymSDN)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	A-Avg	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	48	40	40	48	48	44	40
4	0	0	0	0	0	0	0	92	92	88	88	100	88	100
5	0	0	0	0	0	0	0	92	100	88	96	92	92	96
6	0	0	0	0	0	0	0	96	92	88	96	84	100	100
7	0	0	0	0	0	0	0	96	80	96	96	100	92	92
8	0	0	0	0	0	0	0	96	76	92	96	64	96	52

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	V-Avg
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	44	48	44	44	40	40	44	48	40	44	44	44	48	44
4	92	100	96	100	96	96	92	96	92	80	92	88	88	92.8
5	96	88	88	76	88	88	92	96	88	88	92	100	96	91.6
6	88	92	84	96	92	88	92	96	88	100	92	88	92	92.2
7	92	92	100	88	88	100	100	100	96	92	88	84	96	93.4
8	96	80	84	52	96	64	76	64	88	60	88	52	92	78.2

Packet loss (with SymSDN)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	52	44	48	44	44	44	48	44	40	40	52	52	52	52
4	92	100	88	96	100	92	92	96	88	92	100	76	84	92
5	88	92	84	88	92	84	92	96	96	96	88	100	92	92
6	92	100	96	92	92	88	96	96	96	96	96	84	100	96
7	92	92	96	96	92	96	92	92	96	92	96	84	96	100
8	92	64	96	92	88	92	92	84	80	92	68	92	88	80

#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	A-Avg	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V	#8-V
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	48	48	52	48	36	46.8	0	0	0	0	0	0	0	0
96	92	96	88	88	100	92.4	0	0	0	0	0	0	0	0
92	92	92	88	88	92	91.2	0	0	0	0	0	0	0	0
96	88	88	96	88	96	93.6	0	0	0	0	0	0	0	0
100	92	96	96	92	96	94.2	0	0	0	0	0	0	0	0
96	96	96	96	92	100	88.8	0	0	0	0	0	0	0	0

#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	V-Avg
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

Appendix B

Throughput (without any Prevention scheme)

	#1-A	#1-V	#2-A	#2-V	#3-A	#3-V	#4-A	#4-V	#5-A	#5-V	#6-A	#6-V	#7-A	#7-V
1_3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
4_6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
7_9	113	111	113	111	114	114	114	110	114	114	114	111	114	114
10_12	114	48	114	47	114	87	114	49	113	87	114	48	114	62
13_15	114	41	114	41	114	42	114	41	114	41	114	41	114	41
16_18	114	41	114	42	114	41	114	41	114	40	114	41	114	40
19_21	114	41	114	41	114	41	114	41	114	41	114	41	114	40
22_24	114	42	114	42	114	41	114	42	114	41	114	42	114	42
25_27	114	42	114	42	114	42	114	42	114	39	114	41	114	41
28_30	114	42	114	42	114	42	114	42	114	42	114	42	114	41

	#8-A	#8-V	#9-A	#9-V	#10-A	#10-V	#11-A	#11-V	#12-A	#12-V	#13-A	#13-V	#14-A	#14-V
1_3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
4_6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
7_9	114	111	114	114	114	110	114	114	114	112	114	114	114	114
10_12	114	50	114	59	114	48	113	90	114	49	114	82	114	90
13_15	114	40	114	40	114	41	114	41	114	40	114	41	114	41
16_18	113	41	114	40	114	41	114	40	114	40	114	40	114	36
19_21	114	41	114	40	114	40	114	40	114	41	114	40	113	39
22_24	114	42	114	40	114	41	114	41	114	35	114	39	114	41
25_27	114	42	114	40	114	41	114	41	114	40	114	40	114	41
28_30	114	41	114	41	114	41	114	41	114	40	114	41	114	41

	#15-A	#15-V	#16-A	#16-V	#17-A	#17-V	#18-A	#18-V	#19-A	#19-V	#20-A	#20-V	A-Avg	V-Avg
1_3	114	114	114	114	114	114	114	114	114	114	114	114	114	114
4_6	114	114	114	114	114	114	114	114	114	114	114	114	114	114
7_9	114	107	114	113	114	104	114	114	113	114	114	114	113.85	112
10_12	114	43	114	53	114	41	114	72	114	64	114	66	113.9	61.75
13_15	114	41	114	40	114	41	114	40	114	40	114	37	114	40.55
16_18	114	41	114	40	114	41	114	39	114	40	114	39	113.95	40.2
19_21	114	42	114	36	114	40	114	37	114	40	114	40	113.95	40.1
22_24	114	42	114	41	114	41	114	41	114	42	114	41	114	40.95
25_27	114	42	114	42	114	41	114	41	112	41	114	41	113.9	41.1
28_30	114	42	114	41	114	41	114	42	114	42	114	40	114	41.35

Throughput (with IP-Switching)

	#1-A	#1-V	#2-A	#2-V	#3-A	#3-V	#4-A	#4-V	#5-A	#5-V	#6-A	#6-V	#7-A	#7-V
1_8	46	46	46	46	46	46	46	46	46	46	46	46	46	46
9_16	46	46	44	46	44	46	43	46	43	46	44	46	44	46
17_24	22	46	22	46	22	46	22	46	22	46	22	46	22	46
25_32	22	46	41	46	22	46	22	46	22	46	22	46	22	46
33_40	22	46	22	46	22	46	22	46	22	46	22	46	22	46
41_48	22	46	22	46	22	46	22	46	22	46	22	46	22	46
49_56	22	46	22	46	22	46	22	46	22	46	22	46	22	46
57_64	22	46	22	46	22	46	22	46	22	46	22	46	22	46
65_72	22	46	22	46	22	46	22	46	22	46	22	46	22	46
73_80	22	46	22	46	22	46	22	46	22	46	22	46	22	46

	#8-A	#8-V	#9-A	#9-V	#10-A	#10-V	#11-A	#11-V	#12-A	#12-V	#13-A	#13-V	#14-A	#14-V
1_8	46	46	46	46	46	46	46	46	46	46	46	46	46	46
9_16	44	46	44	46	43	46	44	46	44	46	43	46	44	46
17_24	22	46	22	46	22	46	22	46	22	46	22	46	22	46
25_32	22	46	22	46	22	46	22	46	22	46	22	46	22	46
33_40	22	46	22	46	22	46	22	46	22	46	22	46	22	46
41_48	22	46	22	46	22	46	22	46	22	46	22	46	22	46
49_56	22	46	22	46	22	46	22	46	22	46	22	46	22	46
57_64	22	46	22	46	22	46	22	46	22	46	22	46	22	46
65_72	22	46	22	46	22	46	22	46	22	46	22	46	22	46
73_80	22	46	22	46	22	46	22	46	22	46	22	46	22	46

	#15-A	#15-V	#16-A	#16-V	#17-A	#17-V	#18-A	#18-V	#19-A	#19-V	#20-A	#20-V	A-Avg	V-Avg
1_8	46	46	46	46	46	46	46	46	46	46	46	46	46	46
9_16	44	46	43	46	43	46	44	46	43	46	43	46	43.7	46
17_24	22	46	22	46	22	46	22	46	22	46	22	46	22	46
25_32	22	46	22	46	22	46	22	46	22	46	22	46	22.95	46
33_40	22	46	22	46	22	46	22	46	22	46	22	46	22	46
41_48	22	46	22	46	22	46	22	46	22	46	22	46	22	46
49_56	22	46	22	46	22	46	22	46	22	46	22	46	22	46
57_64	22	46	22	46	22	46	22	46	22	46	22	46	22	46
65_72	22	46	22	46	22	46	22	46	22	46	22	46	22	46
73_80	22	46	22	46	22	46	22	46	22	46	22	46	22	46

Throughput (with PortMergeIP)

	#1-A	#1-V	#2-A	#2-V	#3-A	#3-V	#4-A	#4-V	#5-A	#5-V	#6-A	#6-V	#7-A	#7-V
1_6	46	46	46	46	46	46	46	46	46	46	46	46	46	46
7_12	46	46	46	46	46	46	46	46	46	46	46	46	46	46
13_18	33	46	30	46	30	46	36	46	38	46	38	46	40	45
19_24	30	46	29	45	29	46	28	46	29	46	29	45	29	45
25_30	30	46	28	45	29	46	28	46	29	45	29	45	29	45
31_36	29	45	28	45	28	46	29	45	29	46	29	46	29	46
37_42	29	46	29	45	29	46	29	46	29	46	29	45	28	46
43_48	28	45	29	45	29	46	29	46	29	45	29	46	29	46
49_54	27	46	29	46	29	46	28	46	29	46	29	46	29	46
55_60	28	46	28	45	29	45	29	46	29	45	29	46	29	46

	#8-A	#8-V	#9-A	#9-V	#10-A	#10-V	#11-A	#11-V	#12-A	#12-V	#13-A	#13-V	#14-A	#14-V
1_6	46	46	46	46	46	46	46	46	46	46	46	46	46	46
7_12	46	46	46	46	46	46	46	46	46	46	46	46	46	46
13_18	38	46	39	46	38	46	38	46	39	46	34	46	38	46
19_24	29	45	29	46	29	46	28	46	28	46	29	46	29	46
25_30	29	46	29	46	29	46	26	46	29	46	29	46	29	46
31_36	29	46	29	46	29	46	28	46	29	46	28	46	29	46
37_42	29	46	28	45	29	46	29	46	28	46	29	46	29	46
43_48	29	46	29	46	29	46	29	46	29	46	29	46	28	46
49_54	29	46	29	46	29	46	29	46	29	46	29	46	29	46
55_60	29	46	29	46	29	46	29	46	27	46	29	46	29	46

	#15-A	#15-V	#16-A	#16-V	#17-A	#17-V	#18-A	#18-V	#19-A	#19-V	#20-A	#20-V	A-Avg	V-Avg
1_6	46	46	46	46	46	46	46	46	46	46	46	46	46	46
7_12	46	46	46	46	46	46	46	46	46	46	46	46	46	46
13_18	39	46	38	46	39	46	38	46	34	46	38	46	36.75	45.95
19_24	29	46	29	46	29	46	29	46	29	46	29	46	28.9	45.8
25_30	29	46	29	46	29	46	29	46	29	46	29	46	28.8	45.8
31_36	27	46	29	46	29	46	29	46	29	46	29	46	28.7	45.85
37_42	29	46	28	46	29	46	28	46	29	46	29	46	28.75	45.85
43_48	29	46	29	46	29	46	29	46	29	46	29	46	28.9	45.85
49_54	29	46	29	46	29	46	29	46	29	46	29	46	28.85	46
55_60	29	46	29	46	27	46	29	46	29	46	29	46	28.7	45.85

Throughput (with PortMapping)

	#1-A	#1-V	#2-A	#2-V	#3-A	#3-V	#4-A	#4-V	#5-A	#5-V	#6-A	#6-V	#7-A	#7-V
1_6	59	59	59	59	59	59	59	59	59	59	59	59	59	59
7_12	59	59	59	59	59	59	59	59	59	59	59	59	59	59
13_18	59	59	59	59	56	59	59	59	55	59	59	59	55	59
19_24	32	59	59	59	10	59	12	59	9	59	13	59	10	58
25_30	9	59	11	59	10	59	12	59	9	59	13	59	10	59
31_36	9	59	8	59	8	59	8	59	8	59	8	59	8	59
37_42	8	59	8	59	8	59	8	59	8	59	8	59	8	59
43_48	8	59	8	59	8	59	8	59	8	59	8	59	8	59
49_54	8	59	8	59	8	59	8	59	8	59	8	59	8	59
55_60	8	59	8	59	8	59	8	59	8	59	8	59	8	59
61_66	8	59	8	59	8	59	8	59	8	59	8	59	8	59
67_72	8	59	8	59	8	59	8	59	8	59	8	58	8	59

	#8-A	#8-V	#9-A	#9-V	#10-A	#10-V	#11-A	#11-V	#12-A	#12-V	#13-A	#13-V	#14-A	#14-V
1_6	59	59	59	59	59	59	59	59	59	59	59	59	59	59
7_12	59	59	59	59	59	59	59	59	59	59	59	59	59	59
13_18	53	59	52	59	54	59	58	59	55	59	52	59	52	59
19_24	9	59	9	59	9	59	10	59	9	59	9	59	9	59
25_30	9	58	9	58	9	59	10	59	9	59	9	59	9	59
31_36	8	59	8	59	8	59	8	59	8	59	8	59	8	59
37_42	8	59	8	59	8	59	8	59	8	59	8	59	8	59
43_48	8	59	8	59	8	59	8	59	8	59	8	59	8	59
49_54	8	59	8	59	8	59	8	59	8	59	8	59	8	59
55_60	8	59	8	59	8	59	8	59	8	58	8	59	8	59
61_66	8	59	8	59	8	59	8	59	8	59	8	59	8	59
67_72	8	59	8	59	8	59	8	59	8	59	8	59	8	59

	#15-A	#15-V	#16-A	#16-V	#17-A	#17-V	#18-A	#18-V	#19-A	#19-V	#20-A	#20-V	A-Avg	V-Avg
1_6	59	59	59	59	59	59	59	59	59	59	59	59	59	59
7_12	59	59	59	59	59	59	59	59	59	59	59	59	59	59
13_18	59	59	56	59	56	59	53	59	51	59	55	59	55.4	59
19_24	11	59	10	59	10	59	9	59	9	59	9	59	13.35	58.95
25_30	11	59	10	59	10	59	9	59	9	59	9	59	9.8	58.9
31_36	8	59	8	59	8	59	8	59	8	59	8	59	8.05	59
37_42	8	59	8	59	8	59	8	59	8	59	8	59	8	59
43_48	8	59	8	59	8	59	8	59	8	59	8	59	8	59
49_54	8	59	8	59	8	59	8	59	8	59	8	59	8	59
55_60	8	59	8	59	8	59	8	59	8	59	8	59	8	58.95
61_66	8	59	8	59	8	59	8	59	8	59	8	59	8	59
67_72	8	59	8	59	8	59	8	59	8	59	8	59	8	58.95

Packet loss (without any Prevention scheme)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

	#5-A	#16-A	#17-A	#18-A	#19-A	#20-A	Avg-A	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	0%	0%	95%	85%	95%	95%	95%	95%	100%
3	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	95%	100%
4	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	100%	100%
5	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	100%	100%
6	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	100%	100%
7	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	100%	100%
8	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	100%	100%	100%	100%
9	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	95%	100%	100%	100%
10	0%	0%	0%	0%	0%	0%	0%	100%	100%	95%	95%	100%	100%	100%

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#5-V	#16-V	#17-V	#18-V	#19-V	#20-V	Avg-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%	0%	0%
2	95%	90%	85%	95%	90%	95%	90%	85%	85%	90%	90%	100%	85%	92%
3	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
4	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	100%	100%	100%	100%	100%	100%	100%	100%	95%	100%	100%	100%	100%	100%
6	95%	100%	95%	100%	95%	100%	100%	100%	100%	100%	100%	100%	100%	99%
7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	95%	100%	100%	100%
8	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
9	100%	100%	100%	100%	95%	100%	100%	100%	100%	100%	100%	100%	100%	99%
10	100%	100%	100%	100%	100%	100%	95%	100%	100%	100%	100%	100%	100%	99%

Packet loss (with IP-Switching)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	15%	0%	20%	15%	15%	20%	10%	20%	20%	15%	10%	15%	10%	15%
3	85%	100%	100%	100%	100%	100%	100%	100%	100%	100%	95%	100%	100%	100%
4	100%	90%	100%	100%	100%	90%	100%	100%	100%	100%	100%	100%	100%	100%
5	90%	100%	100%	100%	85%	100%	85%	100%	100%	100%	100%	100%	100%	100%
6	100%	100%	100%	100%	100%	95%	100%	95%	100%	100%	100%	100%	100%	100%
7	90%	100%	100%	90%	100%	100%	90%	100%	90%	100%	100%	90%	90%	100%
8	100%	95%	100%	100%	90%	100%	100%	100%	100%	100%	90%	100%	100%	100%
9	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%	90%	100%	100%	100%	100%	100%	100%	100%

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	Avg-A	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	10%	0%	10%	15%	15%	0%	13%	0%	0%	0%	0%	0%	0%	0%
3	100%	90%	100%	100%	100%	95%	98%	0%	0%	0%	0%	0%	0%	0%
4	100%	95%	100%	100%	85%	100%	98%	0%	0%	0%	0%	0%	0%	0%
5	100%	100%	100%	100%	85%	100%	97%	0%	0%	0%	0%	0%	0%	0%
6	100%	100%	100%	95%	100%	90%	99%	0%	0%	0%	0%	0%	0%	0%
7	100%	100%	100%	100%	100%	100%	97%	0%	0%	0%	0%	0%	0%	0%
8	100%	100%	100%	95%	85%	90%	97%	0%	0%	0%	0%	0%	0%	0%
9	95%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
10	100%	95%	100%	95%	100%	95%	99%	0%	0%	0%	0%	0%	0%	0%

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	Avg-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Packet loss (with PortMergeIP)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	25%	20%	20%	20%	20%	20%	20%	20%	20%	20%	45%	20%	20%	20%
2	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
3	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
4	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
6	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
8	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
9	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	Avg-A	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
1	20%	20%	20%	25%	25%	25%	22%	0%	0%	0%	0%	0%	0%	0%
2	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
3	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
4	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
5	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
6	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
7	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
8	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
9	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
10	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#15-V	#16-V	#17-V	#18-V	#19-V	#20-V	Avg-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Packet loss (with PortMapping)

	#1-A	#2-A	#3-A	#4-A	#5-A	#6-A	#7-A	#8-A	#9-A	#10-A	#11-A	#12-A	#13-A	#14-A
1	0%	0%	0%	0%	0%	0%	0%	0%	24%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	8%	0%	0%	0%	8%	12%	0%	0%	12%
3	0%	68%	48%	68%	12%	0%	44%	4%	20%	0%	16%	36%	60%	8%
4	100%	100%	100%	100%	36%	100%	84%	100%	100%	100%	84%	92%	100%	100%
5	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
6	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
7	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
8	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
9	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
10	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

	#15-A	#16-A	#17-A	#18-A	#19-A	#20-A	Avg-A	#1-V	#2-V	#3-V	#4-V	#5-V	#6-V	#7-V
1	0%	0%	0%	0%	0%	0%	1%	0%	0%	0%	0%	0%	0%	0%
2	12%	0%	0%	0%	0%	0%	3%	0%	0%	0%	0%	0%	0%	0%
3	80%	36%	28%	92%	36%	0%	33%	0%	0%	0%	0%	0%	0%	0%
4	100%	100%	100%	100%	100%	100%	95%	0%	0%	0%	0%	0%	0%	0%
5	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
6	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
7	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
8	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
9	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%
10	100%	100%	100%	100%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%

	#8-V	#9-V	#10-V	#11-V	#12-V	#13-V	#14-V	#5-V	#16-V	#17-V	#18-V	#19-V	#20-V	Avg-V
1	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
2	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
3	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
4	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
5	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
6	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
7	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
8	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
9	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
10	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Appendix C

Precision, F1 score and Recall for Correlation value of DNS attack Dataset

	DNS						
corelation value	**Features name**	**Precision (0th)**	**Recall (0th)**	**F1 score(0th)**	**Precision (1st)**	**Recall (1st)**	**F1 score(1st)**
0.1	Flow Duration 0.283253 Total Backward Packets 0.120505 Fwd Packet Length Max 0.107764 Fwd Packet Length Min 0.117172 Fwd Packet Length Mean 0.115600 Fwd Packet Length Std 0.172959 Bwd Packet Length Max 0.239816 Bwd Packet Length Min 0.525153 Bwd Packet Length Mean 0.341763 Bwd Packet Length Std 0.238461 Flow IAT Mean 0.198413 Flow IAT Std 0.251703 Flow IAT Max 0.248864 Fwd IAT Total 0.274338 Fwd IAT Mean 0.216968 Fwd IAT Std 0.241460 Fwd IAT Max 0.239073 Bwd IAT Total 0.240343 Bwd IAT Mean 0.207499 Bwd IAT Std 0.212781 Bwd IAT Max 0.219618 Bwd IAT Min 0.188097 Fwd PSH Flags 0.355401 Min Packet Length 0.117275 Packet Length Mean 0.113725 Packet Length Std 0.262631 Packet Length Variance 0.157347 RST Flag Count 0.355401 ACK Flag Count 0.297723 URG Flag Count 0.538700 CWE Flag Count 0.306429 Down/Up Ratio 0.570168 Average Packet Size 0.114442 Avg Fwd Segment Size 0.115600 Avg Bwd Segment Size 0.341763 Subflow Bwd Packets 0.120505 Init_Win_bytes_forward 0.276853	0.940276	0.983974	0.961629	0.999989	0.999957	0.999973

	Init_Win_bytes_backward 0.283531 Idle Mean 0.235001 Idle Max 0.235850 Idle Min 0.230191 Label 1.000000						
0.15	Flow Duration 0.283253 Fwd Packet Length Std 0.172959 Bwd Packet Length Max 0.239816 Bwd Packet Length Min 0.525153 Bwd Packet Length Mean 0.341763 Bwd Packet Length Std 0.238461 Flow IAT Mean 0.198413 Flow IAT Std 0.251703 Flow IAT Max 0.248864 Fwd IAT Total 0.274338 Fwd IAT Mean 0.216968 Fwd IAT Std 0.241460 Fwd IAT Max 0.239073 Bwd IAT Total 0.240343 Bwd IAT Mean 0.207499 Bwd IAT Std 0.212781 Bwd IAT Max 0.219618 Bwd IAT Min 0.188097 Fwd PSH Flags 0.355401 Packet Length Std 0.262631 Packet Length Variance 0.157347 RST Flag Count 0.355401 ACK Flag Count 0.297723 URG Flag Count 0.538700 CWE Flag Count 0.306429 Down/Up Ratio 0.570168 Avg Bwd Segment Size 0.341763 Init_Win_bytes_forward 0.276853 Init_Win_bytes_backward 0.283531 Idle Mean 0.235001 Idle Max 0.235850 Idle Min 0.230191 Label 1.000000	0.943925	0.971154	0.957346	0.99998	0.99996	0.99997
0.2	Flow Duration 0.283253 Bwd Packet Length Max 0.239816 Bwd Packet Length Min 0.525153 Bwd Packet Length Mean 0.341763 Bwd Packet Length Std 0.238461 Flow IAT Std 0.251703 Flow IAT Max 0.248864 Fwd IAT Total 0.274338 Fwd IAT Mean 0.216968 Fwd IAT Std 0.241460 Fwd IAT Max 0.239073 Bwd IAT Total 0.240343 Bwd IAT Mean 0.207499 Bwd IAT Std 0.212781 Bwd IAT Max 0.219618 Fwd PSH Flags 0.355401 Packet Length Std 0.262631 RST Flag Count 0.355401 ACK Flag Count 0.297723 URG Flag Count 0.538700 CWE Flag Count 0.306429 Down/Up Ratio 0.570168 Avg Bwd Segment Size 0.341763 Init_Win_bytes_forward 0.276853 Init_Win_bytes_backward 0.283531 Idle Mean 0.235001 Idle Max 0.235850 Idle Min 0.230191 Label 1.000000	0.94081	0.967949	0.954186	0.999978	0.999958	0.999968
0.25	Flow Duration 0.283253 Bwd Packet Length Min 0.525153 Bwd Packet Length Mean 0.341763 Flow IAT Std 0.251703 Fwd IAT Total 0.274338 Fwd PSH Flags 0.355401 Packet Length Std 0.262631 RST Flag Count 0.355401 ACK Flag Count 0.297723 URG Flag Count 0.538700 CWE Flag Count 0.306429 Down/Up Ratio 0.570168 Avg Bwd Segment Size 0.341763 Init_Win_bytes_forward 0.276853 Init_Win_bytes_backward 0.283531 Label 1.000000	0.948276	0.969551	0.958796	0.999979	0.999963	0.999971
0.3	Bwd Packet Length Min 0.525153 Bwd Packet Length Mean 0.341763 Fwd PSH Flags 0.355401 RST Flag Count 0.355401 URG Flag Count 0.538700 CWE Flag Count 0.306429 Down/Up Ratio 0.570168 Avg Bwd Segment Size 0.341763 Label 1.000000	0.794034	0.895833	0.841867	0.999928	0.999839	0.999883
0.35	Bwd Packet Length Min 0.525153 Fwd PSH Flags 0.355401 RST Flag Count 0.355401 URG Flag Count 0.538700 Down/Up Ratio 0.570168 Label 1.000000	0.748555	0.830128	0.787234	0.999882	0.999807	0.999844

0.4	Bwd Packet Length Min 0.525153 URG Flag Count 0.538700 Down/Up Ratio 0.570168 Label 1.000000	0.747475	0.830128	0.786636	0.999882	0.999805	0.999844
0.45	Bwd Packet Length Min 0.525153 URG Flag Count 0.538700 Down/Up Ratio 0.570168 Label 1.000000	0.747475	0.830128	0.786636	0.999882	0.999805	0.999844
0.5	Bwd Packet Length Min 0.525153 URG Flag Count 0.538700 Down/Up Ratio 0.570168 Label 1.000000	0.747475	0.830128	0.786636	0.999882	0.999805	0.999844
0.55	Down/Up Ratio 0.570168 Label 1.000000	0.745303	0.572115	0.647325	0.999703	0.999864	0.999784

Precision, F1 score and Recall for MI-score of DNS attack Dataset

Mi score value	Features name	Precision (0th)	Recall (0th)	F1 score(0th)	Precision (1st)	Recall (1st)	F1 score(1st)
0.0001	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', 'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count', ' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Fwd Avg Packets/Bulk', ' Fwd Avg Bulk Rate', ' Bwd Avg Bytes/Bulk', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min'	0.990132	0.98527	0.987695	0.99999	0.999993	0.999992
0.05	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' RST Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.982055	0.98527	0.98366	0.99999	0.999988	0.999989
0.1	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.98366	0.98527	0.984464	0.99999	0.999989	0.999989
0.15	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.980424	0.983633	0.982026	0.999989	0.999987	0.999988

0.2	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.982085	0.986907	0.98449	0.999991	0.999988	0.999989
0.25	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.982026	0.983633	0.982829	0.999989	0.999988	0.999988
0.3	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.934375	0.978723	0.956035	0.999986	0.999953	0.999969
0.35	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.934579	0.981997	0.957702	0.999988	0.999953	0.999971
0.4	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' min_seg_size_forward'	0.931571	0.98036	0.955343	0.999987	0.999951	0.999969
0.45	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes'	0.907154	0.97545	0.940063	0.999983	0.999932	0.999958
0.5	' Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.793609	0.97545	0.875184	0.999983	0.999828	0.999905
0.55	' Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.793609	0.97545	0.875184	0.999983	0.999828	0.999905
0.6	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.739877	0.986907	0.845722	0.999991	0.999764	0.999878
0.65	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s',	0.739877	0.986907	0.845722	0.999991	0.999764	0.999878

	' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'						
0.7	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.738386	0.988543	0.845346	0.999992	0.999762	0.999877
0.75	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Avg Fwd Segment Size'	0.740196	0.988543	0.846531	0.999992	0.999764	0.999878

Precision, F1 score and Recall for Correlation value of Portmap attack Dataset

corelation value	Features name	Precision (0th)	Recall (0th)	F1 score(0th)	Precision (1st)	Recall (1st)	F1 score(1st)
0.1	Flow Duration 0.342514 Fwd Packet Length Max 0.138812 Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Fwd Packet Length Std 0.324970 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd Packet Length Std 0.334866 Flow Bytes/s 0.303703 Flow Packets/s 0.270935 Flow IAT Mean 0.190359 Flow IAT Std 0.304207 Flow IAT Max 0.317377 Fwd IAT Total 0.334536 Fwd IAT Mean 0.234730 Fwd IAT Std 0.300181 Fwd IAT Max 0.306861 Bwd IAT Total 0.308199 Bwd IAT Mean 0.281647 Bwd IAT Std 0.287126 Bwd IAT Max 0.286565 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Fwd Packets/s 0.271683 Min Packet Length 0.729168 Max Packet Length 0.131981 Packet Length Mean 0.530715 Packet Length Std 0.407571 Packet Length Variance 0.293627 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Init_Win_bytes_forward 0.342721 Init_Win_bytes_backward 0.286310 Active Mean 0.154199 Active Max 0.127805 Active Min 0.134215 Idle Mean 0.301292 Idle Std 0.115712 Idle Max 0.304278 Idle Min 0.291854 Label 1.000000	0.996582	0.997947	0.997264	0.999946	0.999911	0.999929
0.15	Flow Duration 0.342514 Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Fwd Packet Length Std 0.324970 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd Packet Length Std 0.334866 Flow Bytes/s 0.303703 Flow Packets/s 0.270935 Flow IAT Mean 0.190359 Flow IAT Std 0.304207 Flow IAT Max 0.317377 Fwd IAT Total 0.334536 Fwd IAT Mean 0.234730 Fwd IAT Std 0.300181 Fwd IAT Max 0.306861 Bwd IAT Total 0.308199 Bwd IAT Mean 0.281647 Bwd IAT Std 0.287126 Bwd IAT Max 0.286565 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Fwd Packets/s 0.271683 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 Packet Length Variance 0.293627 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081	0.996582	0.997947	0.997264	0.999946	0.999911	0.999929

	CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Init_Win_bytes_forward 0.342721 Init_Win_bytes_backward 0.286310 Active Mean 0.154199 Idle Mean 0.301292 Idle Max 0.304278 Idle Min 0.291854 Label 1.000000						
0.2	Flow Duration 0.342514 Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Fwd Packet Length Std 0.324970 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd Packet Length Std 0.334866 Flow Bytes/s 0.303703 Flow Packets/s 0.270935 Flow IAT Std 0.304207 Flow IAT Max 0.317377 Fwd IAT Total 0.334536 Fwd IAT Mean 0.234730 Fwd IAT Std 0.300181 Fwd IAT Max 0.306861 Bwd IAT Total 0.308199 Bwd IAT Mean 0.281647 Bwd IAT Std 0.287126 Bwd IAT Max 0.286565 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Fwd Packets/s 0.271683 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 Packet Length Variance 0.293627 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Init_Win_bytes_forward 0.342721 Init_Win_bytes_backward 0.286310 Idle Mean 0.301292 Idle Max 0.304278 Idle Min 0.291854 Label 1.000000	0.997264	0.997947	0.997605	0.999946	0.999929	0.999938
0.25	Flow Duration 0.342514 Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Fwd Packet Length Std 0.324970 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd Packet Length Std 0.334866 Flow Bytes/s 0.303703 Flow Packets/s 0.270935 Flow IAT Std 0.304207 Flow IAT Max 0.317377 Fwd IAT Total 0.334536 Fwd IAT Std 0.300181 Fwd IAT Max 0.306861 Bwd IAT Total 0.308199 Bwd IAT Mean 0.281647 Bwd IAT Std 0.287126 Bwd IAT Max 0.286565 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Fwd Packets/s 0.271683 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 Packet Length Variance 0.293627 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Init_Win_bytes_forward 0.342721 Init_Win_bytes_backward 0.286310 Idle Mean 0.301292 Idle Max 0.304278 Idle Min 0.291854 Label 1.000000	0.997264	0.997947	0.997605	0.999946	0.999929	0.999938
0.3	Flow Duration 0.342514 Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Fwd Packet Length Std 0.324970 Bwd Packet Length Max 0.376745	0.997947	0.997947	0.997947	0.999946	0.999946	0.999946

	Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd Packet Length Std 0.334866 Flow Bytes/s 0.303703 Flow IAT Std 0.304207 Flow IAT Max 0.317377 Fwd IAT Total 0.334536 Fwd IAT Std 0.300181 Fwd IAT Max 0.306861 Bwd IAT Total 0.308199 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Init_Win_bytes_forward 0.342721 Idle Mean 0.301292 Idle Max 0.304278 Label 1.000000						
0.35	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Label 1.000000	0.985145	0.998631	0.991842	0.999964	0.999607	0.999786
0.4	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Bwd Packet Length Max 0.376745 Bwd Packet Length Min 0.550527 Bwd Packet Length Mean 0.419155 Bwd IAT Min 0.360017 Fwd PSH Flags 0.372674 Min Packet Length 0.729168 Packet Length Mean 0.530715 Packet Length Std 0.407571 RST Flag Count 0.372674 ACK Flag Count 0.351102 URG Flag Count 0.615081 CWE Flag Count 0.420886 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Avg Bwd Segment Size 0.419155 Label 1.000000	0.979852	0.998631	0.989153	0.999964	0.999465	0.999714
0.45	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Bwd Packet Length Min 0.550527 Min Packet Length 0.729168 Packet Length Mean 0.530715 URG Flag Count 0.615081 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Label 1.000000	0.979866	0.999316	0.989495	0.999982	0.999465	0.999723
0.5	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Bwd Packet Length Min 0.550527 Min Packet Length 0.729168 Packet Length Mean 0.530715 URG Flag Count 0.615081 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Label 1.000000	0.979866	0.999316	0.989495	0.999982	0.999465	0.999723
0.55	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Bwd Packet Length Min 0.550527 Min Packet Length 0.729168 URG Flag Count 0.615081 Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Label 1.000000	0.979852	0.998631	0.989153	0.999964	0.999465	0.999714
0.6	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Min Packet Length 0.729168 URG Flag Count 0.615081	0.979839	0.997947	0.98881	0.999946	0.999465	0.999706

	Down/Up Ratio 0.648523 Average Packet Size 0.614206 Avg Fwd Segment Size 0.661613 Label 1.000000						
0.65	Fwd Packet Length Min 0.729103 Fwd Packet Length Mean 0.661613 Min Packet Length 0.729168 Avg Fwd Segment Size 0.661613 Label 1.00000	0.896255	0.999316	0.944984	0.999982	0.996985	0.998481
0.7	Fwd Packet Length Min 0.729103 Min Packet Length 0.729168 Label 1.000000	0.89406	0.999316	0.943762	0.999982	0.996913	0.998445
0.75	Label 1.0						

Precision, F1 score and Recall for MI-score value of Portmap attack Dataset

Mi score value	Features name	Precision (0th)	Recall (0th)	F1 score(0th)	Precision (1st)	Recall (1st)	F1 score(1st)
0.0001	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean' , ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Bwd URG Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' RST Flag Count', ' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Fwd Avg Packets/Bulk', ' Bwd Avg Bytes/Bulk', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd' , ' min_seg_size_forward', 'Active Mean', ' Active Std' , ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min'	0.99858	1	0.999289	1	0.999964	0.999982
0.05	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max',	0.99858	1	0.999289	1	0.999964	0.999982

	' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean' , ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Max', ' Idle Min'						
0.1	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.99858	1	0.999289	1	0.999964	0.999982
0.15	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow	0.997163	1	0.99858	1	0.999929	0.999964

	Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'						
0.2	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s',' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.997163	1	0.99858	1	0.999929	0.999964
0.25	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s',' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' min_seg_size_forward'	0.997871	1	0.998934	1	0.999947	0.999973
0.3	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Down/Up Ratio' , ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1' , ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes'	0.996454	0.999289	0.997869	0.999982	0.999911	0.999947
0.35	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max',	0.995751	1	0.997871	1	0.999893	0.999947

	' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max' , ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s' , ' Bwd Packets/s', ' Min Packet Length' , ' Max Packet Length', ' Packet Length Mean' , ' Packet Length Std', ' Packet Length Variance', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets'						
0.4	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total' ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length' , ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes'	0.995745	0.998578	0.997159	0.999964	0.999893	0.999929
0.45	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.995742	0.997866	0.996803	0.999947	0.999893	0.99992
0.5	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max' ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max' , 'Fwd Packets/s', ' Min Packet Length' , ' Max Packet Length', ' Packet Length Mean' , ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.995742	0.997866	0.996803	0.999947	0.999893	0.99992
0.55	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Min Packet Length', ' Max Packet Length' , ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.979777	0.999289	0.989437	0.999982	0.999483	0.999733
0.6	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.979777	0.999289	0.989437	0.999982	0.999483	0.999733

Precision, F1 score and Recall for Correlation value of SYN attack Dataset

SYN								
corelation value	**Features name**	**Precision (0th)**	**Recall (0th)**	**F1 score(0th)**	**Support(0th)**	**Precision (1st)**	**Recall (1st)**	**F1 score(1st)**
0.1	Total Backward Packets 0.120493	0.967376	0.999191	0.983026	7419	0.999993	0.99972	0.999857
	Total Length of Fwd Packets 0.208247							
	Fwd Packet Length Max 0.322236							
	Fwd Packet Length Mean 0.175248							
	Fwd Packet Length Std 0.339199							
	Bwd Packet Length Max 0.383470							
	Bwd Packet Length Min 0.381006							
	Bwd Packet Length Mean 0.393921							
	Bwd Packet Length Std 0.359309							
	Fwd PSH Flags 0.411012							
	Fwd Header Length 0.142923							
	Bwd Header Length 0.128348							
	Max Packet Length 0.395465							
	Packet Length Mean 0.290465							
	Packet Length Std 0.416720							
	Packet Length Variance 0.315024							
	RST Flag Count 0.411012							
	ACK Flag Count 0.784367							
	URG Flag Count 0.639606							
	CWE Flag Count 0.427593							
	Average Packet Size 0.240398							
	Avg Fwd Segment Size 0.175248							
	Avg Bwd Segment Size 0.393921							
	Fwd Header Length.1 0.142923							
	Subflow Fwd Bytes 0.208247							
	Subflow Bwd Packets 0.120493							
	Init_Win_bytes_backward 0.270303							
	min_seg_size_forward 0.423877							
	Label 1.000000							
0.15	Total Length of Fwd Packets 0.208247	0.949194	0.99973	0.973807	7419	0.999998	0.999555	0.999776
	Fwd Packet Length Max 0.322236							
	Fwd Packet Length Mean 0.175248							
	Fwd Packet Length Std 0.339199							
	Bwd Packet Length Max 0.383470							
	Bwd Packet Length Min 0.381006							
	Bwd Packet Length Mean 0.393921							
	Bwd Packet Length Std 0.359309							
	Fwd PSH Flags 0.411012							
	Max Packet Length 0.395465							
	Packet Length Mean 0.290465							
	Packet Length Std 0.416720							
	Packet Length Variance 0.315024							
	RST Flag Count 0.411012							
	ACK Flag Count 0.784367							
	URG Flag Count 0.639606							
	CWE Flag Count 0.427593							
	Average Packet Size 0.240398							
	Avg Fwd Segment Size 0.175248							
	Avg Bwd Segment Size 0.393921							
	Subflow Fwd Bytes 0.208247							
	Init_Win_bytes_backward 0.270303							
	min_seg_size_forward 0.423877							
	Label 1.000000							
0.2	Total Length of Fwd Packets 0.208247	0.949315	0.99973	0.973871	7419	0.999998	0.999556	0.999777
	Fwd Packet Length Max 0.322236							
	Fwd Packet Length Std 0.339199							
	Bwd Packet Length Max 0.383470							
	Bwd Packet Length Min 0.381006							
	Bwd Packet Length Mean 0.393921							
	Bwd Packet Length Std 0.359309							
	Fwd PSH Flags 0.411012							
	Max Packet Length 0.395465							
	Packet Length Mean 0.290465							
	Packet Length Std 0.416720							
	Packet Length Variance 0.315024							
	RST Flag Count 0.411012							
	ACK Flag Count 0.784367							
	URG Flag Count 0.639606							
	CWE Flag Count 0.427593							
	Average Packet Size 0.240398							
	Avg Bwd Segment Size 0.393921							
	Subflow Fwd Bytes 0.208247							
	Init_Win_bytes_backward 0.270303							
	min_seg_size_forward 0.423877							
	Label 1.000000							
0.25	Fwd Packet Length Max 0.322236	0.949322	0.999865	0.973938	7419	0.999999	0.999556	0.999778
	Fwd Packet Length Std 0.339199							
	Bwd Packet Length Max 0.383470							
	Bwd Packet Length Min 0.381006							
	Bwd Packet Length Mean 0.393921							
	Bwd Packet Length Std 0.359309							
	Fwd PSH Flags 0.411012							
	Max Packet Length 0.395465							
	Packet Length Mean 0.290465							
	Packet Length Std 0.416720							
	Packet Length Variance 0.315024							
	RST Flag Count 0.411012							
	ACK Flag Count 0.784367							
	URG Flag Count 0.639606							
	CWE Flag Count 0.427593							
	Avg Bwd Segment Size 0.393921							
	Init_Win_bytes_backward 0.270303							

	min_seg_size_forward 0.423877 Label 1.000000							
0.3	Fwd Packet Length Max 0.322236 Fwd Packet Length Std 0.339199 Bwd Packet Length Max 0.383470 Bwd Packet Length Min 0.381006 Bwd Packet Length Mean 0.393921 Bwd Packet Length Std 0.359309 Fwd PSH Flags 0.411012 Max Packet Length 0.395465 Packet Length Std 0.416720 Packet Length Variance 0.315024 RST Flag Count 0.411012 ACK Flag Count 0.784367 URG Flag Count 0.639606 CWE Flag Count 0.427593 Avg Bwd Segment Size 0.393921 min_seg_size_forward 0.423877 Label 1.000000	0.9492	0.999865	0.973874	7419	0.999999	0.999555	0.999777
0.35	Bwd Packet Length Max 0.383470 Bwd Packet Length Min 0.381006 Bwd Packet Length Mean 0.393921 Bwd Packet Length Std 0.359309 Fwd PSH Flags 0.411012 Max Packet Length 0.395465 Packet Length Std 0.416720 RST Flag Count 0.411012 ACK Flag Count 0.784367 URG Flag Count 0.639606 CWE Flag Count 0.427593 Avg Bwd Segment Size 0.393921 min_seg_size_forward 0.423877 Label 1.000000	0.9492	0.999865	0.973874	7419	0.999999	0.999555	0.999777
0.4	Fwd PSH Flags 0.411012 Packet Length Std 0.416720 RST Flag Count 0.411012 ACK Flag Count 0.784367 URG Flag Count 0.639606 CWE Flag Count 0.427593 min_seg_size_forward 0.423877 Label 1.000000	0.821425	0.999461	0.901739	7419	0.999996	0.998194	0.999094
0.45	ACK Flag Count 0.784367 URG Flag Count 0.639606 Label 1.000000	0.780659	0.79539	0.787956	7419	0.998299	0.998142	0.998221
0.5	ACK Flag Count 0.784367 URG Flag Count 0.639606 Label 1.000000	0.780659	0.79539	0.787956	7419	0.998299	0.998142	0.998221
0.55	ACK Flag Count 0.784367 URG Flag Count 0.639606 Label 1.000000	0.780659	0.79539	0.787956	7419	0.998299	0.998142	0.998221
0.6	ACK Flag Count 0.784367 URG Flag Count 0.639606 Label 1.000000	0.780659	0.79539	0.787956	7419	0.998299	0.998142	0.998221
0.65	ACK Flag Count 0.784367 Label 1.000000	0.781279	0.79539	0.788271	7419	0.998299	0.998149	0.998224
0.7	ACK Flag Count 0.784367 Label 1.000000	0.781279	0.79539	0.788271	7419	0.998299	0.998149	0.998224
0.75	ACK Flag Count 0.784367 Label 1.000000	0.781279	0.79539	0.788271	7419	0.998299	0.998149	0.998224

Precision, F1 score and Recall for MI-score value of SYN attack Dataset

Mi score value	Features name	Precision (0th)	Recall (0th)	F1 score(0th)	Precision (1st)	Recall (1st)	F1 score(1st)
0.0001	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', 'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Bwd Avg Packets/Bulk', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min']	0.99601	0.999199	0.997602	0.999993	0.999966	0.99998
0.05	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length	0.995878	0.999333	0.997602	0.999994	0.999965	0.99998

	Variance', ' RST Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min']						
0.1	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward']	0.996143	0.999333	0.997735	0.999994	0.999968	0.999981
0.15	Label', ' Flow Duration', ' Total Fwd Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward']	0.995878	0.999333	0.997602	0.999994	0.999965	0.99998
0.2	Label', ' Flow Duration', ' Total Fwd Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward']	0.995745	0.999333	0.997536	0.999994	0.999964	0.999979
0.25	Label', ' Flow Duration', ' Total Fwd Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward']	0.995481	0.999333	0.997403	0.999994	0.999962	0.999978
0.3	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward', ' min_seg_size_forward']	0.995613	0.999466	0.997536	0.999996	0.999963	0.999979
0.35	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' ACK Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward', ' min_seg_size_forward']	0.995613	0.999466	0.997536	0.999996	0.999963	0.999979
0.4	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max', ' Fwd Header Length', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' ACK Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward', ' min_seg_size_forward']	0.995746	0.9996	0.997669	0.999997	0.999964	0.99998
0.45	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max', 'Fwd Packets/s', ' Min Packet Length', ' Packet Length Mean', ' ACK Flag Count', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward']	0.994029	0.9996	0.996806	0.999997	0.99995	0.999973

0.5	Label', 'Total Length of Fwd Packets', 'Flow Bytes/s', ' Flow Packets/s', 'Fwd Packets/s', ' ACK Flag Count', ' Average Packet Size', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward',	0.994161	0.9996	0.996873	0.999997	0.999951	0.999974
0.55	Label', 'Total Length of Fwd Packets', 'Flow Bytes/s', ' ACK Flag Count', ' Average Packet Size', 'Init_Win_bytes_forward']	0.954158	0.999867	0.976477	0.999999	0.999597	0.999798
0.6	Label', 'Flow Bytes/s', 'Init_Win_bytes_forward']	0.931957	0.999733	0.964656	0.999998	0.999387	0.999692
0.65	Label', 'Init_Win_bytes_forward'	0.802699	1	0.890553	1	0.997936	0.998967
0.7	Label', 'Init_Win_bytes_forward'	0.802699	1	0.890553	1	0.997936	0.998967
0.75	Label', 'Init_Win_bytes_forward'	0.802699	1	0.890553	1	0.997936	0.998967
0.8	' Label', 'Init_Win_bytes_forward'	0.802699	1	0.890553	1	0.997936	0.998967

Precision, F1 score and Recall for Correlation value of UDP attack Dataset

	UDP-features						
corelation value	**Features name**	**Precision (0th)**	**Recall (0th)**	**F1 score(0th)**	**Precision (1st)**	**Recall (1st)**	**F1 score(1st)**
0.1	Flow Duration 0.264399 Total Backward Packets 0.178735 Fwd Packet Length Max 0.216441 Fwd Packet Length Min 0.232204 Fwd Packet Length Mean 0.272626 Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Bwd Packet Length Std 0.334964 Flow IAT Mean 0.125424 Flow IAT Std 0.187512 Flow IAT Max 0.251776 Fwd IAT Total 0.256064 Fwd IAT Mean 0.179106 Fwd IAT Std 0.210681 Fwd IAT Max 0.240887 Bwd IAT Total 0.241923 Bwd IAT Mean 0.271974 Bwd IAT Std 0.267322 Bwd IAT Max 0.238984 Bwd IAT Min 0.264667 Fwd PSH Flags 0.470833 Bwd Packets/s 0.128154 Min Packet Length 0.233385 Packet Length Mean 0.242002 Packet Length Std 0.129118 Packet Length Variance 0.286102 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Average Packet Size 0.132722 Avg Fwd Segment Size 0.272626 Avg Bwd Segment Size 0.380976 Subflow Bwd Packets 0.178735 Init_Win_bytes_forward 0.464147 Init_Win_bytes_backward 0.239698 Active Mean 0.146077 Active Max 0.151150 Active Min 0.124369 Idle Mean 0.235977 Idle Max 0.236119 Idle Min 0.229957 Label 1.000000	0.996109	1	0.998051	1	0.999997	0.999998
0.15	Flow Duration 0.264399 Total Backward Packets 0.178735 Fwd Packet Length Max 0.216441 Fwd Packet Length Min 0.232204 Fwd Packet Length Mean 0.272626 Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Bwd Packet Length Std 0.334964 Flow IAT Std 0.187512 Flow IAT Max 0.251776 Fwd IAT Total 0.256064 Fwd IAT Mean 0.179106 Fwd IAT Std 0.210681 Fwd IAT Max 0.240887 Bwd IAT Total 0.241923 Bwd IAT Mean 0.271974 Bwd IAT Std 0.267322 Bwd IAT Max 0.238984 Bwd IAT Min 0.264667 Fwd PSH Flags 0.470833 Min Packet Length 0.233385 Packet Length Mean 0.242002	0.996104	0.998698	0.997399	0.999999	0.999997	0.999998

	Packet Length Variance 0.286102 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Avg Fwd Segment Size 0.272626 Avg Bwd Segment Size 0.380976 Subflow Bwd Packets 0.178735 Init_Win_bytes_forward 0.464147 Init_Win_bytes_backward 0.239698 Active Max 0.151150 Idle Mean 0.235977 Idle Max 0.236119 Idle Min 0.229957 Label 1.000000						
0.2	Flow Duration 0.264399 Fwd Packet Length Max 0.216441 Fwd Packet Length Min 0.232204 Fwd Packet Length Mean 0.272626 Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Bwd Packet Length Std 0.334964 Flow IAT Max 0.251776 Fwd IAT Total 0.256064 Fwd IAT Std 0.210681 Fwd IAT Max 0.240887 Bwd IAT Total 0.241923 Bwd IAT Mean 0.271974 Bwd IAT Std 0.267322 Bwd IAT Max 0.238984 Bwd IAT Min 0.264667 Fwd PSH Flags 0.470833 Min Packet Length 0.233385 Packet Length Mean 0.242002 Packet Length Variance 0.286102 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Avg Fwd Segment Size 0.272626 Avg Bwd Segment Size 0.380976 Init_Win_bytes_forward 0.464147 Init_Win_bytes_backward 0.239698 Idle Mean 0.235977 Idle Max 0.236119 Idle Min 0.229957 Label 1.000000	0.994812	0.998698	0.996751	0.999999	0.999996	0.999997
0.25	Flow Duration 0.264399 Fwd Packet Length Mean 0.272626 Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Bwd Packet Length Std 0.334964 Flow IAT Max 0.251776 Fwd IAT Total 0.256064 Bwd IAT Mean 0.271974 Bwd IAT Std 0.267322 Bwd IAT Min 0.264667 Fwd PSH Flags 0.470833 Packet Length Variance 0.286102 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Avg Fwd Segment Size 0.272626 Avg Bwd Segment Size 0.380976 Init_Win_bytes_forward 0.464147 Label 1.000000	0.996109	1	0.998051	1	0.999997	0.999998
0.3	Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Bwd Packet Length Std 0.334964 Fwd PSH Flags 0.470833 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Avg Bwd Segment Size 0.380976 Init_Win_bytes_forward 0.464147 Label 1.000000	0.988095	0.972656	0.980315	0.999977	0.99999	0.999983
0.35	Bwd Packet Length Max 0.350890 Bwd Packet Length Min 0.483829 Bwd Packet Length Mean 0.380976 Fwd PSH Flags 0.470833 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Avg Bwd Segment Size 0.380976 Init_Win_bytes_forward 0.464147 Label 1.000000	0.988095	0.972656	0.980315	0.999977	0.99999	0.999983
0.4	Bwd Packet Length Min 0.483829	0.982895	0.972656	0.977749	0.999977	0.999986	0.999981

	Fwd PSH Flags 0.470833 RST Flag Count 0.470833 ACK Flag Count 0.401689 URG Flag Count 0.727944 CWE Flag Count 0.430615 Down/Up Ratio 0.618930 Init_Win_bytes_forward 0.464147 Label 1.000000						
0.45	Bwd Packet Length Min 0.483829 Fwd PSH Flags 0.470833 RST Flag Count 0.470833 URG Flag Count 0.727944 Down/Up Ratio 0.618930 Init_Win_bytes_forward 0.464147 Label 1.000000	0.982895	0.972656	0.977749	0.999977	0.999986	0.999981
0.5	URG Flag Count 0.727944 Down/Up Ratio 0.618930 Label 1.000000	0.965665	0.878906	0.920245	0.999897	0.999973	0.999935
0.55	URG Flag Count 0.727944 Down/Up Ratio 0.618930 Label 1.000000	0.965665	0.878906	0.920245	0.999897	0.999973	0.999935
0.6	URG Flag Count 0.727944 Down/Up Ratio 0.618930 Label 1.000000	0.965665	0.878906	0.920245	0.999897	0.999973	0.999935
0.65	URG Flag Count 0.727944 Label 1.000000	0.973274	0.56901	0.718159	0.999632	0.999987	0.999809
0.7	URG Flag Count 0.727944 Label 1.000000	0.973274	0.56901	0.718159	0.999632	0.999987	0.999809

Precision, F1 score and Recall for MI-score value of UDP attack Dataset

Mi score value	Features name	Precision (0th)	Recall (0th)	F1 score(0th)	Precision (1st)	Recall (1st)	F1 score(1st)
0.0001	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Bwd URG Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' RST Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' ECE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Fwd Avg Bytes/Bulk', ' Fwd Avg Bulk Rate', 'Bwd Avg Bulk Rate', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min'	0.99085	0.997368	0.994098	0.999998	0.999992	0.999995
0.05	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' RST Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.99085	0.997368	0.994098	0.999998	0.999992	0.999995
0.1	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.992147	0.997368	0.994751	0.999998	0.999993	0.999996

0.15	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.992147	0.997368	0.994751	0.999998	0.999993	0.999996
0.2	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.992147	0.997368	0.994751	0.999998	0.999993	0.999996
0.25	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.992157	0.998684	0.99541	0.999999	0.999993	0.999996
0.3	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', 'Init_Win_bytes_forward', ' act_data_pkt_fwd', ' min_seg_size_forward'	0.98957	0.998684	0.994106	0.999999	0.999991	0.999995
0.35	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', 'Init_Win_bytes_forward'	0.988281	0.998684	0.993455	0.999999	0.99999	0.999994
0.4	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.979301	0.996053	0.987606	0.999997	0.999982	0.999989
0.45	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Max', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.980595	0.997368	0.988911	0.999998	0.999983	0.999991

0.5	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', 'Fwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.975484	0.994737	0.985016	0.999996	0.999979	0.999987
0.55	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.959596	1	0.979381	1	0.999964	0.999982
0.6	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.959596	1	0.979381	1	0.999964	0.999982
0.65	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.957179	1	0.978121	1	0.999962	0.999981
0.7	Label', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', ' Subflow Fwd Bytes'	0.957179	1	0.978121	1	0.999962	0.999981
0.75	Label', ' Fwd Packet Length Min', ' Min Packet Length', ' Average Packet Size'	0.953576	1	0.976236	1	0.999959	0.999979

Precision, F1 score and Recall for Correlation value of UDPLag attack Dataset

	udplag						
corelation value	**Features name**	**Precision (0th)**	**Recall (0th)**	**F1 score(0th)**	**Precision (1st)**	**Recall (1st)**	**F1 score(1st)**
0.1	Total Backward Packets 0.257960 Total Length of Bwd Packets 0.144216 Fwd Packet Length Std 0.267577 Bwd Packet Length Max 0.352576 Bwd Packet Length Min 0.262471 Bwd Packet Length Mean 0.350310 Bwd Packet Length Std 0.293907 Bwd IAT Total 0.261406 Bwd IAT Std 0.100520 Bwd IAT Max 0.181043 Fwd PSH Flags 0.337964 Bwd Header Length 0.297883 Max Packet Length 0.128425 Packet Length Std 0.317316 Packet Length Variance 0.256579 RST Flag Count 0.337964 ACK Flag Count 0.191164 URG Flag Count 0.512530 CWE Flag Count 0.357120 Down/Up Ratio 0.322534 Avg Bwd Segment Size 0.350310 Subflow Bwd Packets 0.257960 Subflow Bwd Bytes 0.144216 Init_Win_bytes_backward 0.311125 act_data_pkt_fwd 0.181445 Label 1.000000	0.988656	0.994732	0.991685	0.999945	0.999882	0.999914
0.15	Total Backward Packets 0.257960 Fwd Packet Length Std 0.267577 Bwd Packet Length Max 0.352576 Bwd Packet Length Min 0.262471 Bwd Packet Length Mean 0.350310 Bwd Packet Length Std 0.293907 Bwd IAT Total 0.261406 Bwd IAT Max 0.181043 Fwd PSH Flags 0.337964 Bwd Header Length 0.297883 Packet Length Std 0.317316 Packet Length Variance 0.256579 RST Flag Count 0.337964 ACK Flag Count 0.191164 URG Flag Count 0.512530 CWE Flag Count 0.357120 Down/Up Ratio 0.322534 Avg Bwd Segment Size 0.350310 Subflow Bwd Packets 0.257960 Init_Win_bytes_backward 0.311125 act_data_pkt_fwd 0.181445 Label 1.000000	0.987794	0.994732	0.991251	0.999945	0.999873	0.999909
0.2	Total Backward Packets 0.257960 Fwd Packet Length Std 0.267577 Bwd Packet Length Max 0.352576 Bwd Packet Length Min 0.262471 Bwd Packet Length Mean 0.350310 Bwd Packet Length Std 0.293907 Bwd IAT Total 0.261406 Fwd PSH Flags 0.337964 Bwd Header Length 0.297883 Packet Length Std 0.317316 Packet Length Variance 0.256579 RST Flag Count 0.337964 URG Flag Count 0.512530 CWE Flag Count 0.357120	0.984238	0.986831	0.985533	0.999864	0.999836	0.99985

	Down/Up Ratio 0.322534 Avg Bwd Segment Size 0.350310 Subflow Bwd Packets 0.257960 Init_Win_bytes_backward 0.311125 Label 1.000000						
0.25	Total Backward Packets 0.257960 Fwd Packet Length Std 0.267577 Bwd Packet Length Max 0.352576 Bwd Packet Length Min 0.262471 Bwd Packet Length Mean 0.350310 Bwd Packet Length Std 0.293907 Bwd IAT Total 0.261406 Fwd PSH Flags 0.337964 Bwd Header Length 0.297883 Packet Length Std 0.317316 Packet Length Variance 0.256579 RST Flag Count 0.337964 URG Flag Count 0.512530 CWE Flag Count 0.357120 Down/Up Ratio 0.322534 Avg Bwd Segment Size 0.350310 Subflow Bwd Packets 0.257960 Init_Win_bytes_backward 0.311125 Label 1.000000	0.984238	0.986831	0.985533	0.999864	0.999836	0.99985
0.3	Bwd Packet Length Max 0.352576 Bwd Packet Length Mean 0.350310 Fwd PSH Flags 0.337964 Packet Length Std 0.317316 RST Flag Count 0.337964 URG Flag Count 0.512530 CWE Flag Count 0.357120 Down/Up Ratio 0.322534 Avg Bwd Segment Size 0.350310 Init_Win_bytes_backward 0.311125 Label 1.000000	0.971429	0.985075	0.978204	0.999845	0.9997	0.999773
0.35	Bwd Packet Length Max 0.352576 Bwd Packet Length Mean 0.350310 URG Flag Count 0.512530 CWE Flag Count 0.357120 Avg Bwd Segment Size 0.350310 Label 1.000000	0.873439	0.920983	0.896581	0.999182	0.998619	0.9989
0.4	URG Flag Count 0.51253 Label 1.00000	0.741117	0.384548	0.506358	0.993661	0.99861	0.996129
0.45	URG Flag Count 0.51253 Label 1.00000	0.741117	0.384548	0.506358	0.993661	0.99861	0.996129
0.5	URG Flag Count 0.51253 Label 1.00000	0.741117	0.384548	0.506358	0.993661	0.99861	0.996129

Precision, F1 score and Recall for MI-score value of UDPLag attack Dataset

Mi score value	**Features name**	**Precision (0th)**	**Recall (0th)**	**F1 score(0th)**	**Precision (1st)**	**Recall (1st)**	**F1 score(1st)**
0.0001	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Bwd PSH Flags', ' Bwd URG Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', 'FIN Flag Count', ' SYN Flag Count', ' RST Flag Count', ' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1' , ' Fwd Avg Packets/Bulk', ' Bwd Avg Bytes/Bulk', 'Subflow Fwd Packets', ' Subflow Fwd Bytes' , ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward' , ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Std', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Std', ' Idle Max', ' Idle Min']	0.995495	0.998193	0.996843	0.999982	0.999955	0.999968
0.05	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length',	0.995495	0.998193	0.996843	0.999982	0.999955	0.999968

	'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' CWE Flag Count' , ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', ' Active Max', ' Active Min', 'Idle Mean', ' Idle Max', ' Idle Min']						
0.1	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s' , ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd']	0.995491	0.99729	0.99639	0.999973	0.999955	0.999964
0.15	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std' , ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size' , ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes' , 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd']	0.994595	0.99729	0.99594	0.999973	0.999945	0.999959
0.2	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward']	0.995479	0.99458	0.995029	0.999945	0.999955	0.99995
0.25	Label', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length Mean', 'Bwd Packet Length Max', ' Bwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min Packet Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' ACK Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' Avg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bwd Bytes', 'Init_Win_bytes_forward']	0.995479	0.99458	0.995029	0.999945	0.999955	0.99995

0.3	Label', ' Flow Duration', ' Total Backward Packets', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Mean', 'Flow Bytes/s', ' Flow Packets/s' ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' Average Packet Size', ' Avg Fwd Segment Size', ' Fwd Header Length.1', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', 'Init_Win_bytes_forward',	0.99458	0.99458	0.99458	0.999945	0.999945	0.999945
0.35	Label', ' Flow Duration', 'Total Length of Fwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Mean', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', ' Bwd IAT Min', 'Fwd Packets/s', ' Bwd Packets/s', ' Max Packet Length', ' Packet Length Mean', ' Average Packet Size', ' Avg Fwd Segment Size', 'Init_Win_bytes_forward']	0.983842	0.990063	0.986943	0.9999	0.999836	0.999868
0.4	Label', ' Flow Duration', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd IAT Total', ' Fwd IAT Mean', ' Fwd IAT Max', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Max', 'Fwd Packets/s', ' Bwd Packets/s', 'Init_Win_bytes_forward']	0.976043	0.993677	0.984781	0.999936	0.999755	0.999846
0.45	Label', ' Flow Duration', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', 'Fwd Packets/s', ' Bwd Packets/s', 'Init_Win_bytes_forward']	0.97695	0.995483	0.98613	0.999955	0.999764	0.999859
0.5	Label', ' Flow Duration', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Max', 'Fwd Packets/s', 'Init_Win_bytes_forward']	0.967515	0.995483	0.9813	0.999955	0.999664	0.999809

Papers Published

- Saharan, S., and Gupta, V., (2019). Prevention and Mitigation of DNS based DDoS attacks in SDN Environment. In 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Bengaluru, India, pp. 571-573, doi: 10.1109/COMSNETS.2019.8711258. [Core National]
- Gupta, V., Kochar, A., Saharan, S., and Kulshrestha, R. (2019). DNS Amplification Based DDoS Attacks in SDN Environment: Detection and Mitigation. In 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), Singapore, pp. 473-478, doi: 10.1109/CCOMS.2019.8821716.
- Saharan, S., Gupta, V. (2021). DDoS Prevention: Review and Issues. In: Patnaik, S., Yang, XS., Sethi, I. (eds) Advances in Machine Learning and Computational Intelligence. Algorithms for Intelligent Systems. Springer, Singapore. https://doi.org/10.1007/978-981-15-5243-4_53
- Saharan, S., Gupta, V. (2022). Prevention of DrDoS Amplification Attacks by Penalizing the Attackers in SDN Environment. In: Barolli, L., Hussain, F., Enokido, T. (eds) Advanced Information Networking and Applications. AINA 2022. Lecture Notes in Networks and Systems, vol 450. Springer, Cham. https://doi.org/10.1007/978-3-030-99587-4_58 [Core B]
- Gupta, V., Saharan, S., Parida, S. (2022). Prevention of DDoS Attacks with Reliable-Dynamic Path Identifiers. In: Barolli, L., Hussain, F., Enokido, T. (eds) Advanced Information Networking and Applications. AINA 2022. Lecture Notes in Networks and

Systems, vol 449. Springer, Cham. https://doi.org/10.1007/978-3-030-99584-3_39 [Core B]

- Saharan, S., Gupta, V., Vora, N., Maheshwari, M. (2022). Detection of Distributed Denial of Service Attacks Using Entropy on Sliding Window with Dynamic Threshold. In: Barolli, L., Hussain, F., Enokido, T. (eds) Advanced Information Networking and Applications. AINA 2022. Lecture Notes in Networks and Systems, vol 449. Springer, Cham. https://doi.org/10.1007/978-3-030-99584-3_37 [Core B]
- Gupta, V., Saharan, S., and Raje S. (2023). SymSDN: A DRDoS Attack Prevention Approach. In 2023 IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, United Kingdom, pp. 1-6,
doi: 10.1109/WCNC55385.2023.10119119. [Core B]

Under-review/Working Papers

- Saharan, S., Gupta, "Prevention of DDoS attacks: A comprehensive review and future directions", Information Security Journal: A Global Perspective, Taylor and Francis, doi: 10.1080/19393555.2024.2347243 (Accepted)
- Saharan, S., Gupta, V., Sharma, H., Chaitanya, A.S.K., (2023). Optimizing Feature Selection for Near-Real Time Detection and Mitigation of DDoS Attacks in Software-Defined Networking Environment [under review]

Vishal Gupta is working as an Associate Professor in the Department of Computer Science and Information Systems of Birla Institute of Technology and Science, Pilani, India (BITS Pilani). He did his Ph.D. from BITS Pilani in the year 2014. Since then, primarily, he has been working in the area related to different aspects of Computer Networks. This includes Network Security (specifically concerning DDoS attacks), Software Defined Networking, and Wireless Networks (specifically IEEE 802.11). Other than Computer Networks, his research interests include 3G-WLAN Interworking, Multi-Criteria Decision Making, Ranking algorithms, Link Structure of the Web, and Search Engine ranking techniques.

Brief Biography of Candidate

Shail Saharan is currently pursuing her Ph.D. in Computer Science from the Birla Institute of Technology and Science, Pilani, India. Her Ph.D. is in preventing DDoS attacks, and her research area includes Network Security, Software Defined Networking, and DDoS attacks. She has published research papers in core ranking national and international conferences during her Ph.D. She also acted as a teaching assistant for subjects such as Computer programming, Theory of computation, Object Oriented Programming, Network Security. She completed her Master's in 2016 from IGDTUW, Delhi, and her Bachelors' from MEC in 2014.