DESIGN AND ANALYSIS OF ANOMALY DETECTION AND MITIGATION SCHEMES FOR DISTRIBUTED DENIAL OF SERVICE ATTACKS IN SOFTWARE DEFINED NETWORK

A.O. SANGODOYIN

Ph.D

2019

DESIGN AND ANALYSIS OF ANOMALY DETECTION AND MITIGATION SCHEMES FOR DISTRIBUTED DENIAL OF SERVICE ATTACKS IN SOFTWARE DEFINED NETWORK

An Investigation into the Security Vulnerabilities of Software Defined Network and the Design of Efficient Detection and Mitigation Techniques for DDoS Attack using Machine Learning Techniques

Abimbola Oladimeji SANGODOYIN

Submitted for the degree of

Doctor of Philosophy

Faculty of Engineering and Informatics

University of Bradford

2019

Abstract

Abimbola Oladimeji Sangodoyin
Design and Analysis of Anomaly Detection and Mitigation Schemes for Distributed Denial of Service Attacks in Software Defined Network.
An Investigation into the Security Vulnerabilities of Software Defined Network and the Design of Efficient Detection and Mitigation Techniques for DDoS Attack using Machine Learning Techniques.

Keywords: Software Defined Network, DDoS, Network security, Attack detection, Attack mitigation, Controller.

Software Defined Networks (SDN) has created great potential and hope to overcome the need for secure, reliable and well managed next generation networks to drive effective service delivery on the go and meet the demand for high data rate and seamless connectivity expected by users. Thus, it is a network technology that is set to enhance our day-to-day activities. As network usage and reliance on computer technology are increasing and popular, users with bad intentions exploit the inherent weakness of this technology to render targeted services unavailable to legitimate users. Among the security weaknesses of SDN is Distributed Denial of Service (DDoS) attacks.

Even though DDoS attack strategy is known, the number of successful DDoS attacks launched has seen an increment at an alarming rate over the last decade. Existing detection mechanisms depend on signatures of known attacks which has not been successful in detecting unknown or different shades of DDoS attacks. Therefore, a novel detection mechanism that relies on deviation from confidence interval obtained from the normal distribution of throughput polled without attack from the server.

Furthermore, sensitivity analysis to determine which of the network metrics (jitter, throughput and response time) is more sensitive to attack by introducing white Gaussian noise and evaluating the local sensitivity using feed-forward artificial neural network is evaluated. All metrics are sensitive in detecting DDoS attacks. However, jitter appears to be the most sensitive to attack. As a result, the developed framework provides an avenue to make the SDN technology more robust and secure to DDoS attacks.

Declaration

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Abimbola Sangodoyin

Acknowledgements

To the El-SHADDAI, the true source of inspiration be all the glory for the strength to conduct and carry on with this research. If I have seen further, it is by standing on the shoulder of giants – Sir. Isaac Newton. Even though this project is as a result of hard work and perseverance, I take no credit for it is the cumulative effort of all (Academia, family, friends . . .) that has led to this. Nonetheless, I will like to express my profound gratitude to the following people: **Team of Supervisors**: Prof Irfan Awan who took me on as a research student despite his many academic and professional commitments. His calm and thorough approach to research is laudable. I still remember his words "Abimbola, you have to switch on your turbo mode" when I was lagging in the experiment stage of this research. I deeply appreciate the efforts of Prof Fun Hu and Prof Prashant Pillai for shaping my research direction at the initial stage. **Family**: Words will fail me in quantifying how amazing you guys are. From financial to moral and emotional support, I am grateful. Prof & Mrs. AY Sangodoyin, Dr. Kulu Ayeni, Dr. Fagbola, Bantulahi, Akule bobo, Adefare. I deeply appreciate the support and advice of Prof Babagana Zulum before and throughout this programme. Deborah & Grace Adekanye, thanks for your level of understanding. You all have shown me that anything is possible once passion and determination to succeed is my watchword. **Friends**: RCCG Chapel of His Glory Bradford Youth wing, Dr. Rolake Fasae, Mobayode Akinsolu, Temitope Laniran, Helsy baby and others numerous to list here, I appreciate you all. **Colleagues**: K.Maiyama, B.Modu, Aristotle,Umar, Bashir, George thank you all for the late night gists and encouragement. It was nice collaborating with some of you and I will continue to reach out to you all until we leave indelible footprints in the research community. **Posterity**: I have realised that what you are scared of is also scared of you.

Only time and perseverance will determine who wins the race. NEVER GIVE UP! Above all, I am because we are – UBUNTU!!!

Publications and Presentations

Books

- **Sangodoyin Abimbola**, Tshiamo Sigwele, Prashant Pillai, Yim Fun Hu, Irfan Awan, and Jules Disso. ”DoS Attack Impact Assessment on Software Defined Networks.” In International Conference on Wireless and Satellite Systems, pp. 11-22. Springer, Cham, 2017.
- Sigwele, Tshiamo, Prashant Pillai, **Sangodoyin Abimbola,** and Yim Fun Hu. ”Security Aware Virtual Base Station Placement in 5G Cloud Radio Access Networks.” In International Conference on Wireless and Satellite Systems, pp. 3-10. Springer, Cham, 2017.

Journals

- **Sangodoyin Abimbola,** Mobayode Akinsolu, and Irfan Awan. ”A Deductive Approach for the Sensitivity Analysis of Software Defined Network Parameters” In Simulation Modelling Practice and Theory, Elsevier.
- **Sangodoyin Abimbola,** Bashir Mohammed,and Irfan Awan,” Data driven machine learning approach to detect DDoS attack in software defined network” in Concurrency and Computation: Practice and Experience, Wiley 2020.(under review)

Conference proceedings

- **Sangodoyin Abimbola,** Bashir Mohammed,Sibusiso Moyo, Irfan Awan, and Jules Pagna Disso. ”A Framework for Distributed Denial of Service Attack Detection and Reactive Countermeasure in Software Defined Network.” In 2019 IEEE 7th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 80-87. IEEE, 2019.

- Omololu Makinde, **Sangodoyin Abimbola,** Bashir Mohammed,Daniel Neagu, Umaru Adamu. ”Distributed Network Behaviour Prediction Using Machine Learning and Agent-based Micro Simulation” In 2019 IEEE 7th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 182-188. IEEE, 2019.
- **Sangodoyin Abimbola,** Babagana Modu, Irfan Awan, and Jules Pagna Disso. ”An approach to detecting distributed denial of service attacks in software defined networks.” In 2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 436-443. IEEE, 2018.

Contents

List of Figures

List of Tables

Glossary

ACK	Acknowledgement
ANN	Artificial Neural Network
API	Application Program Interface
AWGN	Additive White Gaussian Noise
CART	Classification and Regression Tree
CI	Confidence Interval
CIA	Confidentiality Integrity and Availability
CPU	Central Processing Unit
CV SS	Common Vulnerability Scoring System
DARPA	Defense Advanced Research Projects Agency
DDoS	Distributed Denial of Service
FN	False Negative
FP	False Positive
GSA	Global Sensitivity Analysis
GSMP	Global Standards Management Process

HTTP	HyperText Transfer Protocol
HULK	HTTP Unbearable Load King
ICMP	Internet Control Messaging Protocol
IDS	Intrusion Detection System
IOS	Internet Operating System
ITU − T	International Telecommunications Unit
KNN	K Nearest Neighbour
LDA	Linear Discriminant Analysis
LOIC	Low Orbit Ion Canon
LR	Logistic Regression
LSA	Local Sensitivity Analysis
MSE	Mean Squared Error
NB	Naive Bayes
ODL	OpenDaylight
ONF	Open Network Foundation
OS	Operating System
PDF	Probability Density Function
QoS	Quality of Service
RFC	Request For Comment
RTT	Round Trip Time
SDN	Software Defined Networking
SNMP	Simple Network Management Protocol

SOM	Self Organising Map
SSL	Secure Sockets Layer
SV M	Support Vector Machine
SY N	Synchronise
TCAM	Ternary Content Addressable Memory
TCB	Transmission Control Block
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TP	True Positive
TTL	Time-To-Live
UDP	User Datagram Protocol
WAN	Wide Area Network

Chapter 1

INTRODUCTION

1.1 Introduction

Computer networks have become part of our everyday lives from the government to commercial enterprises to individuals [88]. These networks are built from a large number of devices such as routers, switches and middleboxes with complex protocols running on them. Operating and maintaining a computer network infrastructure for secure and seamless connectivity is not an easy task and remains a challenging task as advancement and demand for ubiquitous connections from end users is increasing. The race to keep up with network administration tasks is further heightened due to the integration of software to our daily routine and wide adoption of smart mobile devices and lately internet-of-things. To accommodate the continually changing demand of the network environment, network administrators are saddled with the responsibility of configuring vendor-specific devices and setting configuration policies for effective and reliable operation. As a result, network management and dynamic response to events and applications are arduous and prone to error. Similarly, in the face of growing traffic and demand for more data rates from consumers, the service providers need to keep up with the pace through investments in bigger and faster links and edge routers, even though revenues are growing quite slowly [32] [118].

In view of the pressing challenges network administrators and vendors are faced with, there is a need to optimise and bring innovation to existing network design and architecture. The innovation is expected to bear in mind cost, programmability, and

how robust the network is, in meeting the increasing demand of users. Thus, the emergence of Software Defined Network (SDN).

As a revolutionary concept, SDN has created great potential and hope to overcome the need for flexible, secure, reliable, and well managed next generation networks [3]. SDN separates the control and data plane in networks such that switches become simple data forwarding devices and network management is controlled by logically centralised controller. This remarkable feature of SDN provides a programmable, vendor-agnostic, cost effective and innovative network architecture.

The future of SDN lies in its acceptance and deployment. Even though technology and its deployment take years before it can be available to end users due to standardisation process and Request For Comments (RFC). However, speculations remain as to whether the same should be expected for SDN or not. So far, the need for researchers to run experiments for campus networks gave birth to the deployment of SDN on a small scale which subsequently led to the proposal of new network architecture, ETHANE, for enterprise network [88], [23]. Figure 1.1 shows the conventional development trend of technology[111].

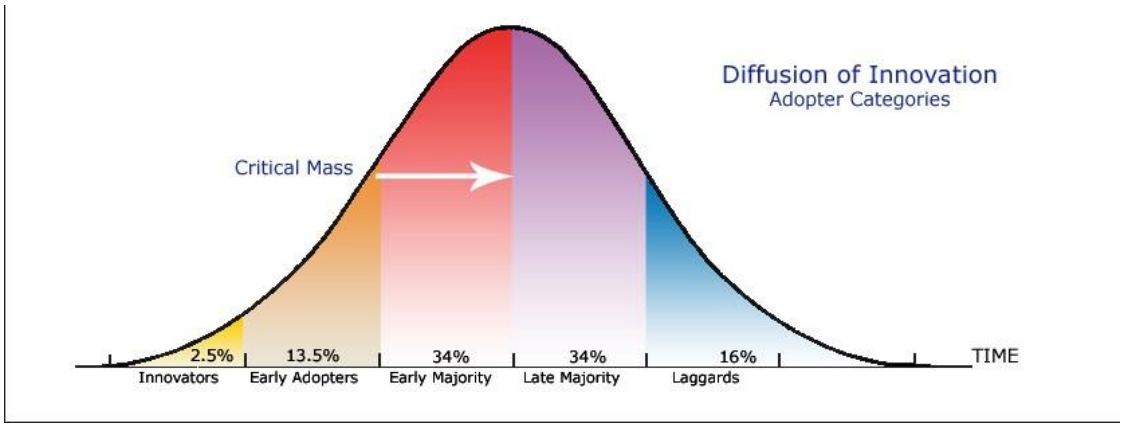

Figure 1.1: Diffusion of innovation

The trend of SDN might differ slightly from this conventional trend as network equipment manufacturers and vendors are interested in the promising nature of the SDN technology and its implementation, as seen from Google B4 deployment in WAN data centre [68], VMware NSX virtualisation platform [132], and Linux foundation collaborative project, OpenDaylight[109].

In spite of the programmability, flexibility, universal connectivity and centralised

control, which are critical to the success of SDN, these features are at odds with making it more secure. The SDN platform can bring with it several security breaches which include an increased potential for Distributed Denial-of-Service (DDoS) attacks due to controller centralisation and flow table limitations in network devices[120]. Furthermore, the abstraction of flows and underlying hardware resources make it easier for harvesting of intelligence which can be used effortlessly for further exploitation and reprogramming the entire network by malicious user [78] [118].

1.2 Aim and Objectives

In today's world, many networks are connected. Security breach spreads at an alarming rate, sometimes spanning the globe within hours or days. Even enterprises that have secure perimeters often find themselves with significant internal security breaches. As a result, security-focused SDN architecture is designed to keep malicious users at bay. Consequently, this study is aimed at:

- Effectively detecting and mitigating distributed denial of service attack in software defined networks using statistical analysis and machine learning approach to achieve better system performance and availability to legitimate users.

To achieve the stated aim, the following objectives are defined:

- Identification of SDN vulnerabilities and investigation of existing SDN security solutions related to DDoS in SDN
- Design and implementation of network topology to extract real time SDN traffic data before and after DDoS attack
- Model a statistical and machine learning techniques that are simple, less computationally intense and offers the best performance in detecting DDoS attacks
- Perform sensitivity analysis on selected network traffic attributes for attack traffic classification

1.3 Research Questions

Research Question 1 Although the deployment of SDN promises to significantly improve network applicability and efficiency, the programmability aspect also makes it more vulnerable to a number of attacks and configuration errors which may pose more serious consequences than in traditional networks. Hence, ***How vulnerable is SDN to DDoS attack?***

Research Question 2 Technologies that are not developed with security and privacy in mind will eventually constitute the weakest link in a network. With high-speed data transfer and signalling, attack on a very large scale can generate large signalling storm that can take a network down in no time. This gives rise to Research question 2: ***What is the impact of DDoS attack on SDN?*** knowing if DDoS attack has significant impact on the SDN architecture create an avenue to build robust security measures to mitigate DDoS attacks in SDN controller.

Research Question 3 Even though DDoS attack strategy is known,***Why is it difficult to handle DDoS attacks?*** In addition, ***are there deployment points suitable to mitigate DDoS attack?***

Research Question 4 Several detection techniques are available in the public domain, yet, DDoS attack is on the increase. Hence, the need for effective detection techniques. ***Which defense mechanism is lightweight and at what interval can network traffic be polled?*** Also, given a known detection mechanism, ***How would they perform if an unknown attack variant occurred?***

Research Question 5 A lot of DDoS datasets exists in the research community with several metrics utilised in detecting DDoS attack. This gives rise to the question ***How sensitive are DDoS attack impact metrics in detecting and mitigating attacks in SDN***. Having only the sensitive metrics may reduce complex computation time and demand for high-performance computing hardware in detecting and mitigating attacks.

Security should be considered when the architecture is designed, not an afterthought or a patch. Hence, the need to build robust security solutions around SDN architecture to combat shades of attack.

1.4 Scope of the Research

This research focuses on the use of machine learning techniques and statistical analysis to effectively detect DDoS attacks in SDN. It investigates the vulnerabilities of the SDN network using Kali Linux, and DDoS flooding attacks launched on the network using open source Low Orbit Ion Canon (LOIC) attack tool. Custom network topologies were modelled using Mininet and all simulations performed in an isolated network using Oracle virtual box and Linux operating system (Ubuntu).

Real SDN traffic datasets for DDoS flooding and attack free network traffic were extracted and cleaned using KNIME as discussed in Chapter 3 and Chapter 4.

It is worth noting that all DDoS flooding attacks launched were orchestrated from internal network. Hence, DDoS attacks launched through external networks are not covered and are outside the scope of this work.

1.5 Thesis Contribution

This thesis centres on effectively detecting and mitigating distributed denial of service attacks in software defined networks. The core contributions of the research encompass five areas:

- 1. **Analysis of the impact of DDoS flooding attack in SDN**

  Despite the concerted efforts by researchers for detecting and mitigating the menace of DDoS attacks, it is continuously growing in volume and severity. SDN as a revolutionary concept alters existing network architecture and decouples data plane from control plane for interoperability and network programmability. Hence, the need to analyse the impact of DDoS attacks on the new architecture. Using throughput and jitter as impact metrics, this study reveals that for an SDN network, a DDoS attack on the infrastructure plane will highly degrade network performance.

- 2. **A lightweight approach to detect DDoS attacks using statistical analysis**
  Various DDoS detection mechanisms rely on pattern recognition to identify attacks. Understanding network characteristics such as protocols, CPU utilisation,

delay, throughput, packet header, and packet size will help in determining the type of detection mechanism to be deployed. Hence, most techniques rely on data collection, filtering, and processing for anomaly detection using statistics and machine learning techniques [10]. In this research, a statistical approach to detect DDoS flooding attacks in SDN is presented. It offers unique complementary advantages compared to existing methods by employing a lightweight approach to detect DDoS flooding attacks. Our detection mechanism relies on deviation from the confidence interval obtained from the normal distribution of throughput polled without attack from the server. Similarly, data collected effectively with less overhead from victim server using 'iperf' i.e. heavy communication between controller and switches is reduced.

- 3. **Vulnerability assessment of SDN to spoofing attack and mitigation of DDoS attack using reactive flow rule insertion**

  Information gathering is an essential step needed to gain access to a network. It involves knowing which information is useful for launching an attack and how to extract it through reconnaissance. In this work, the feasibility of spoofing and flooding DDoS attacks on data plane devices in SDN using Mininet emulator, floodlight controller, and network performance testing tools is demonstrated. Furthermore, these attacks are mitigated by pushing reactive flow through the controller to the attacking switch port.

- 4. **Implementation of machine learning algorithms for the detection of UDP, TCP and ICMP flooding attack in SDN**

  Machine learning is a wide interdisciplinary area of research that involves learning patterns from datasets for the computer to perform a specific task without explicit instructions and it has been applied in research to detect DDoS attacks. All DDoS detection techniques leverage on identifying key features (traffic flow metrics) relevant to identify malicious traffic from benign traffic. Our approach is simpler and mimics network architecture obtainable in a mid-sized enterprise network. A new dataset that contains three types of DDoS attacks namely: UDP flood, TCP flood, and HTTP flood are simulated in a controlled environment. The TCP, UDP, and ICMP flood attacks are considered because the pro-

tocols represent a large portion of web application traffic usage and characteristics. Also, six different machine learning algorithm is applied to the generated datasets for the detection of UDP, TCP, and HTTP flooding attacks.

- 5. **Sensitivity analysis of DDoS attack detection metrics using Artificial Neural Network**

  Sensitivity analysis offers an efficient approach to assess the extent to which output is affected by changes in input variables. One of the key advantages of sensitivity analysis is that it identifies critical variables that may be given less consideration when designing a robust detection model. This research is a novel attempt to identify network parameters that are more sensitive in detecting DDoS attacks in SDN by implementing local sensitivity analysis using artificial neural networks to identify key network metrics that mainly influence the prediction of whether an SDN is under attack or secure.

Figure 1.2 outlines the relationship between the thesis contribution, research questions answered and how it has been addressed in the thesis chapters. The research outcome has led to a number of publications listed under publications and presentation section.

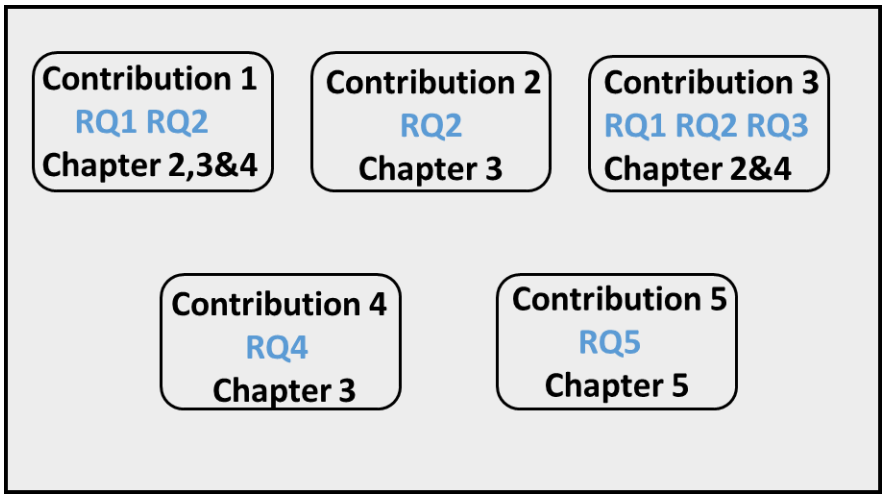

Figure 1.2: Relationship between chapters and research questions

1.6 Organisation of the Thesis

This thesis is divided into six chapters. The first chapter introduces the concept of SDN and highlights the aim and objectives of the research together with research questions and scope of the research. The remainder of this thesis is organised as follows:

- **Chapter2** reviews the roadmap to SDN and the internal architecture of SDN. This chapter also examines the security threats and vulnerabilities in the architecture. Also, the research trend in SDN security and analysis coupled with available security solutions platform in the SDN planes according to ITU-T specifications is presented.

- **Chapter3** presents an overview of DDoS attack. Available DDOS detection mechanism in both SDN and the traditional network is explored. Simulation scenarios are setup with DDoS flooding attack launched against TCP and UDP server. Statistical and Machine learning approach to detect DDoS attacks were looked at. Some of the algorithms that were considered include Support Vector Machine, Classification and Regression Trees and Linear Discriminant Analysis.

- **Chapter4** presents an information gathering session to explore vulnerabilities in SDN using Nmap scanner. The results obtained are used to spoof source IP address to launch DDoS attacks on the server. The attack launched is mitigated by pushing reactive flow to the attacking switch port via the controller.

- **Chapter5** presents the sensitivity analysis of DDoS attack impact metrics considered in the previous chapter. This chapter examines which of the impact metrics(Jitter, throughput and response time) is more sensitive to DDoS attacks. Feed forward Artificial Neural Network (ANN) is used to generate input data and local sensitivity analysis is performed on the normalised data

- **Chapter6** This chapter presents the conclusion of the thesis and recommends a future research direction.

Chapter 2

Background on SDN Architecture and Security

2.1 Introduction

This chapter, which is made up of two major sections, describes the SDN architecture and SDN security. Firstly, a brief history of SDN followed by a general overview of SDN architecture and comparison of security between traditional network and SDN is presented. The security issues and vulnerabilities of SDN are then examined. One of the security issues SDN is prone to is Distributed Denial of Service attacks. This led to Overview of DDoS attacks and classification.

2.2 Roadmap to SDN

Networking devices have been successfully developed and deployed over the past few years [54]. This has led to migration functionality grown into hardware as shown in Figure. 2.1. Each device is designed to be autonomous to the greatest extent possible due to relatively small fixed networks and shared domain. Hence, the introduction of intelligence resident in every device and the birth of vendor-agnostic devices [54]. However, the traditional network architecture is at a point where its ability to adapt to changing user demands, like those enabled by virtualisation technologies has become a hindrance [123]

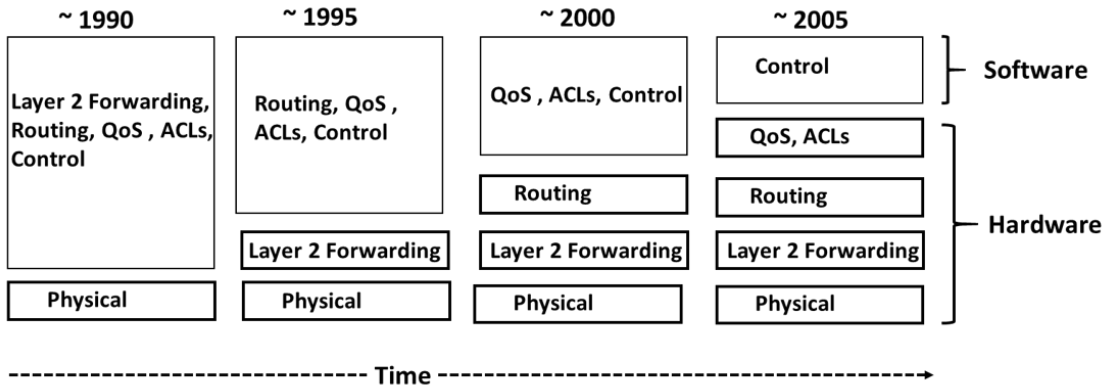

Figure 2.1: Migration of network functionality to hardware [54]

While SDN is seen as an evolving networking paradigm, it is worth noting that the concept of programmable networks that matures into SDN has been around for many years [102]. Precursors to SDN began with Open Signalling (OpenSig) with the intent of making internet and many mobile networks more programmable [21]. This further progressed to General Switch Management Protocol (GSMP) by using a controller to perform resource allocation on a multicast connection [40]. Similarly, active networking [129] [137] proposed the concept of programmable network infrastructure to support customised services. However, this idea received little attention due to security and performance concerns [92].

Another concept of decoupling control and management functions is raised in the 4D project [55] [20]. The authors proposed clean slate design for separation between routing decision logic and governing protocols between network devices. In line with this approach, a management protocol responsible for the modification of network devices is proposed by NETCONF [46]. The proposed management protocol is saddled with the responsibility of overcoming the security limitations of SNMP.

SANE [24] and ETHANE [23] defined a new network architecture for an enterprise network. The research work laid the foundation for SDN. Consequently, ForCES [41] and OpenFlow [88] standardise information exchange between SDN planes. Implementation of globally deployed software defined WAN has also been witnessed in the datacentre network [68]. Figure 2.2 shows the timeline of SDN technology maturity. With the formation of a consortium called Open Networking Foundation (ONF)

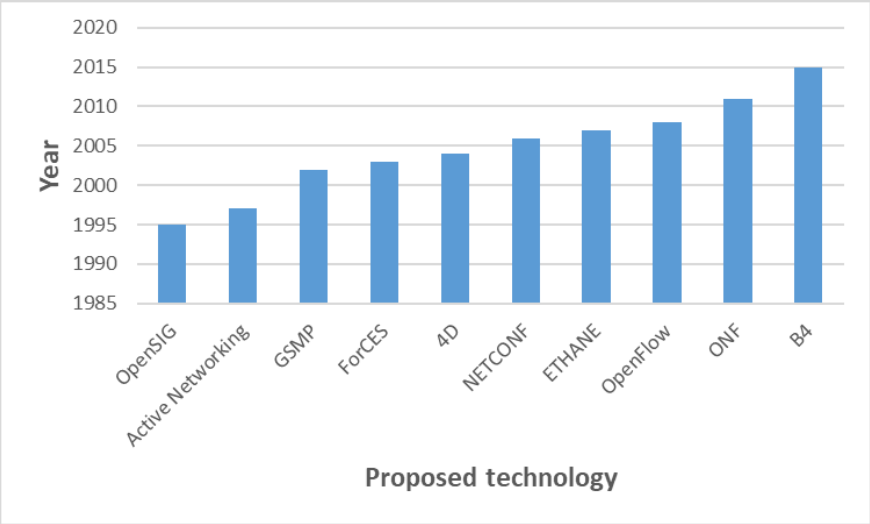

Figure 2.2: SDN technology development

to promote networking through SDN and the standardisation of OpenFlow protocol and related technologies, acceptance and implementation of SDN technology is not far from reach.

2.3 Comparison of SDN and Traditional Networks

The traditional approach to networking involves assigning a complete computing unit to a single task. This approach is effective and works fine in a small scale network. However, as networks grow in size, they become much more complex to manage and maintain. On the other hand, SDN seems to address some of the limitations inherent in the traditional network. Table 2.1 [18] presents a comparison between traditional network and SDN in handling security issues.

Table 2.1: Security in Traditional network vs SDN

Security Challenges	Traditional approach	SDN based approach	Benefits of SDN
Perimeter security	.perimeter defined through physical objects	.perimeter defined through application layer	.Policy is decoupled from physical perimeter
	.Each device statically configured	.Different policies can be applied to external and internal traffic	.Policies applied based on application-layer attributes
	.Each device operates autonomously	single configuration is possible for all security devices in each domain	.Security complexity does not increase to changes in logical perimeter
New security threat	.security signature is identified	.End-to-end network visibility is derived from centralised controller	.Operations staff can react to threats from controller
	.User is located with available tools within the system	-	.Significant reduction in network devices required for security processing, thereby reducing capital expenditure
	.user is denied network access	.Fine-grained counter- - measure policy in real-time	
	.malicious user moves to another vulnerable port to launch an attack	-	-
High Scalability	.Requires proportional increase in hardware to meet user demands	.virtualised process reduce hardware demand	.Improves resource allocation and utilisation
Proactive patch management	.Difficult to achieve in a consistent manner due to the availability of finite resources in embedded device	.Centralised patch management to respond rapidly to new threats	.Simpler to introduce enhanced features

2.4 OpenFlow basics

Most switches and routers available in the market are vendor specifics, running their own IOS and do not typically provide an open software platform. This creates the need to virtualise their hardware or software next to nothing. This closed source IOS does not give room for researchers to experiment with new ideas and network vendors are understandably nervous about disclosing their extensive and fragile distributed protocols and algorithms [88]. The reason for this is not far-fetched; it simply lowers the barrier for new competitors.

The OpenFlow started as an academic experiment and rapidly gained significance which subsequently led to the consortium of industry giants to form Open Networking Foundation (ONF) [104].

OpenFlow brings to the fore the concept known as Application Programming Interface (API) which allows direct access to manipulation of the forwarding plane of network devices such as routers and switches, both physical and virtual [3].

An OpenFlow switch typically consists of a flow table as shown in Figure. 2.3, which performs packet lookup and forwarding. Each table in an open flow switch holds a set of entries that consist of the following:

- Header fields or match fields, with information found in packet header, ingress port, and metadata used to match incoming packets.
- Counters, used to collect statistics of particular flow, such as number of received packets, number of bytes, and duration of the flow.
- A set of instructions to be applied after a match rule that dictates how to handle matching packets [29].

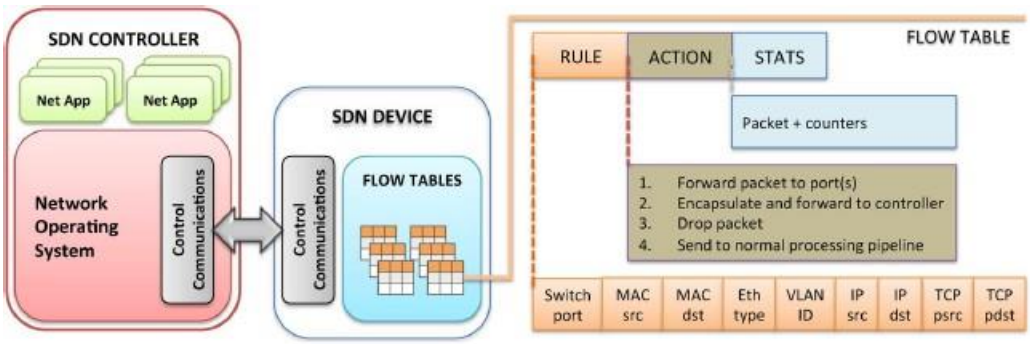

Figure 2.3: OpenFlow enabled SDN device [78]

Inside an OpenFlow device, there is a set of algorithm that defines what goes where when a packet arrives based on the matching flow rule. A flow rule is a combination of different matching rule. If there is no default matching rule, then, the packet will be discarded. The priority of rules follow the natural sequence number of the table and the row order. It is worthy of note that the throughput of commercial OpenFlow switch is relatively low (500-1000 flow-mod per second) which is a limiting factor [15].

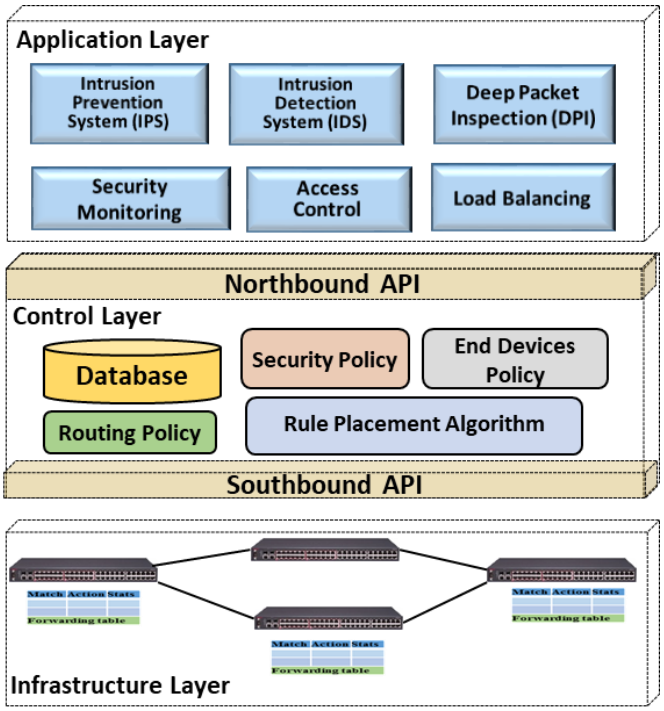

Figure 2.4: SDN architecture illustrating the infrastructure, control and application layers. [118]

2.5 SDN Architecture

SDN architecture encompasses the complete network platform. It is a modular approach that defines chain of command and interoperability within the network [118]. Unlike traditional network, the intelligence of data plane devices is removed to a logically centralised control system [54]. Figure 2.4 presents the SDN architecture showing the data/infrastructure, control and application layer.

In Figure 2.4, there are two main elements: the controllers and the forwarding devices. A forwarding device is a hardware or software element specialised in packet forwarding and based on a pipeline of flow tables where each entry of a flow table has: a matching rule, action to be executed on matching packets and counters that keep statistics of matching packets [78]. The controller serves as the brain of the network and it deals with the management of network state. Below is a description of various layers [118]:

Infrastructure layer: This layer is also known as data plane. It consists of simple

forwarding elements without embedded control or software to make autonomous decisions. It is accessible through the southbound interface and allows packet switching and forwarding.

Control layer: This layer consists of SDN controllers providing a consolidated control functionalities through Application Programming Interfaces (APIs). The crucial value of the controller is to provide abstractions, essential services, and common APIs to developers. Three communication interfaces allow the controller to interact: northbound, southbound and the east/westbound interfaces.

- Southbound Interfaces: Southbound interface allows the controller and forwarding elements to interact in the infrastructure layer, thus being the crucial instrument for clearly separating the control and data plane functionality.
- Northbound Interfaces: This interface is the connecting bridge between the application layer and control layer. It enables the programmability of the controllers by exposing the data models and other functionalities within the controllers for use by applications at the application layer. The northbound interface is mostly a software ecosystem, hence, a common northbound interface is still an open issue.
- East/Westbound Interfaces: This interface is a special communication interface provisioned for distributed controllers to synchronise state for high availability. Its function includes import/export data between controllers and monitoring/notification capabilities to check if a controller is up or notify a takeover on a set of forwarding elements.

Application Layer: The application layer consists of end-user business applications and network services. Example of application that runs here is network virtualisation. Network policy is also defined here.

2.5.1 Fundamental Characteristics of SDN

SDN is characterised by five fundamental traits namely: plane separation, network automation and virtualisation, centralised control, openness and simplified device [29].

- **plane separation**: with the data plane separated from the control plane, a level of packet forwarding intelligence has been added to the controller. Consequently, this led to a reduction in long convergence time for changes made in SDN network.

- **network virtualisation**: Network virtualisation creates an avenue for multiple virtual networks to run over a shared infrastructure [47]. Each virtual network created can have simpler topology than the underlying hardware. With virtualisation, it is easier to spin up multiple virtual devices more portable, scalable and cost-effective mid-sized and small scale office network.

- **centralised control**: Since the control plane resides in the controller, the controller sees and know where each host connects to the network and the type of topology that connects the network. Centralisation allows network engineers to implement unique, flexible forwarding policies and monitoring/management protocols only limited by the ability of software running on it.

- **openness**: Openness in SDN means that the four compass point of Northbound, Southbound, Eastbound and Westbound interfaces should remain standard and not proprietary. This would make communication simple, robust and perhaps most important, extensible. In addition, openness in the SDN controller helps network providers and customers add value to the platform with their innovation.

- **simplified device**: with the separation of data from the control plane, thousands of lines of complicated control plane software to enable autonomous behaviour has been removed. Hence, fast decisions based on forwarding instructions from the controller can be executed by data plane devices with less memory requirement.

2.5.2 SDN Development Platforms

With simulators, network administrators are better equipped to analyse the complexity of their network. There are many platforms that can be used to simulate or emulate SDN projects. Table 2.2 shows the list of SDN development platforms available today [5]

Table 2.2: SDN development platforms

Platform	Mininet	Estinet	NS-3	Trema
Last version	2.2.1	9.0	3.26	0.10.1
Vendor	Stanford University, ON. Lab	Estinet technologies Inc.	NS-3 project	NEC Corporation
Website	www.mininet.org	www.estinet.com	www.nsnam.org	trema.github.io/trema/
Operating System	Ubuntu, Fedora	Linux, Fedora	GNU/Linux, Windows, Mac	GNU/Debian, Ubuntu, Fedora
Emulation mode	Yes	Yes	No	Yes
Simulation mode	No	Yes	Yes	No
Free/Proprietary	Free	Proprietary	Free	Free

2.6 Security Issues and Vulnerabilities in SDN

Although SDN promises more robust security features than traditional networks, SDN itself is not fully immune to attacks. The separation of the control plane from the data plane expose the network to a range of new attacks if identified loopholes are not addressed before deploying.

2.6.1 Uncovering Security flaws using STRIDE Approach

Network and computer security are built on three pillars commonly referred to as Confidentiality, Integrity and Availability. Figure 2.5 presents a simple but widely applicable CIA triad with their mitigation techniques. A data is said to be confidential if it can only be assessed by those authorised to use it and no one else. Integrity comes in when the information remains the same or identical to its state when the last authorised user assessed it. Data is available when it is accessible only to authorised users in a convenient format within a reasonable time.

Three key parameters that often come up in computer security issues are: 1. Vulnerability 2. Threat and 3. Countermeasures.

The state where a system is susceptible to attack is the vulnerability state, while threat is a potential violation of system security. Countermeasures are the techniques employed for protecting the system.

In SDN, there are two main properties which can serve as attractive honeypots for malicious users and headache for the less prepared enterprise. First, the ability to control networks by means of software. Second, the centralisation of network intelligence in the controller. A malicious user with access to the controller can modify/control the entire network. Common attack types present in today's network can be summarised

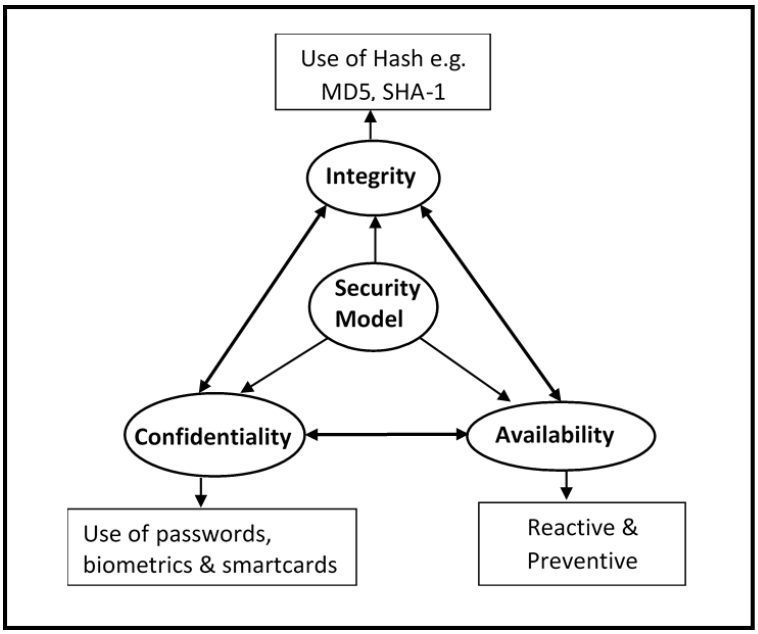

Figure 2.5: CIA Triad

under STRIDE approach as shown in Table 2.3. The STRIDE approach seems to give a broad view of available attack types [44]. However, grey hat hackers keep evolving and attack types keep increasing by the day.

Table 2.3: Microsoft STRIDE attack types and security properties [48]

Attack type	Security property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privileges	Authorisation

- Spoofing: In this approach, the application or a malicious user masquerades with the purpose of concealing their identity and impersonate to gain unauthorised access to a network.
- Tampering: Tampering refers to the intentional modification of data so as to compromise the integrity of data sent or received.

- Repudiation: Repudiation attack happens when a system does not track and log users' actions properly, thus permitting malicious manipulation of new actions.
- Information disclosure: This is often targeted at breaching the confidentiality reposed by end users' on enterprise.
- Denial of service: Denial-of-service (DoS) attack can be very serious in an enterprise network because it prevents intended users from accessing temporarily or indefinitely resources they would normally expect to have.
- Elevation of privileges: The main aim of a malicious attacker is to have access to privilege information or access contents reserved for higher privileged users.

Although attacks have different level of severity, it is a worthy practice to have appropriate security measures in place.

2.6.2 Major Security Threats in the SDN Planes

According to [77] seven threat vectors were identified and three are more specific to SDN namely:

- Threat vector 1 – forged or fake traffic flows: this traffic can be used to attack switches and controllers. This threat can overwhelm the switch and controller flow tables and inject latency into the network. Possible solution: the use of Intrusion Detection System (IDS) with support for run-time root-cause analysis could help identify abnormal flows in the network.
- Threat vector 2 – attacks on vulnerabilities in switches: A susceptible switch may drop, slow down or redirect packets to overload neighbouring switches or controllers. Possible solution: deploying autonomic trust management solutions for network devices operating system.
- Threat vector 3 – attacks on control plane communications: this attack explores vulnerabilities in the protocol such as TLS/SSL that comprises the controller-device link. Possible Solution: securing communication with threshold cryptography across controller replicas.

- Threat vector 4 – attacks on vulnerabilities in controllers: the controller is the honeypot of the network. This attack is specific to SDN and is probably the most severe threat the network can experience. The use of common IDS may not be enough to mitigate this problem. Possible solutions: securing all sensitive elements such as crypto keys and secrets in the controller. Another solution is employing a diversity of controllers, programming languages and protocols used in communications.

- Threat vector 5 – Lack of mechanisms to ensure trust between the controller and management applications: the techniques used to certify network devices are different from those used for applications. Multi-vendor applications and controllers lack the ability to establish trusted relationships. Possible solution: a robust autonomic trust management certificate could help guarantee a trusted application during its lifetime.

- Threat vector 6 – vulnerabilities in administrative stations: administrative stations are potential exploitable target in the current network. The impact can be severe in SDN because it makes programming and launching of coordinated attacks from a single location. Possible solution: employing the use of protocols requiring double credential verifications e.g. requiring two different users to access a control server.

- Threat vector 7 – lack of trusted resources for remediation: good understanding of the cause of network failure and secure recovery mode will help prevent future network downtime. Possible solution: good logging and tracing mechanisms in the data and control plane will help address this problem.

The identified threats provide an avenue for adequate understanding and good risk analysis which will make SDN more secure and dependable than the current network.

Threat vectors 3, 4 and 5 are specific to SDN as shown in Table 2.4. These vectors arise as a result of the separation of control plane and data plane and the introduction of controller.

Table 2.4: SDN specific vs non-specific threats [77]

Threats	Specific to SDN?	Consequences in SDN
Vector 1	No	Open door for DDoS attacks
Vector 2	No	Potential attack inflation
Vector 3	Yes	Exploiting logically centralised controllers
Vector 4	Yes	Network compromise
Vector 5	Yes	Easy development and deployment of malicious applications on controllers
Vector 6	No	Potential attack inflation
Vector 7	No	Negative impact on fast recovery and fault diagnosis.

2.6.3 Security Vulnerabilities in the SDN Planes

A number of security vulnerabilities have been identified both in OpenFlow protocol and SDN architecture [12]. Most of the vulnerabilities are due to the powerful authority granted to SDN and third-party applications. Table 2.5 shows the attack types SDN planes are prone to with respect to the severity of attacks. It can be seen from Table 2.5 and 2.6 that DDoS attack poses a high risk to the continuous operation of SDN controller when under attack. At present, there is no standardised universal classification or methodology for vulnerability analysis. Although the use of Common Vulnerability Scoring System (CVSS) has been reported in [146].

Table 2.5: Security vulnerabilities in SDN according to planes

SDN Plane	Attack types	Level of Severity	Possible Countermeasures
Application	Information leakage	Low	Use of strong encryption
	Application abuse	Medium	Update security patches regularly
	API abuse	Low	Use of strong encryption
	Communication hijacking	Low	Use of strong encryption
	Masquerading	Low	Use of strong encryption
	DDoS attack	High	Islanding, Rate limiting,Packet dropping techniques
Control	Network manipulation	High	Having distributed controllers with different vendors
	DDoS Attack	High	Islanding, Rate limiting,Packet dropping techniques
Infrastructure	DDoS attack	High	Islanding, Rate limiting, Packet dropping techniques
	Compromised network	Medium	Use of strong encryption
	Compromised system	low	Change vendor default password
	Communication hijacking	Medium	Use of strong encryption

Table 2.6: DDoS attacks on SDN layers [33]

Plane	Possible attacks
Data Plane	TCAM exhaustion, switch DDoS, ICMP flood, TCP flood, TCP-SYN flood
Control plane	Resource depletion, OpenFlow bandwidth exhaustion, amplification attacks
Application plane	Exhausting northbound API, application layer DDoS (HTTP flooding, slowloris)

2.6.4 Security Solutions Platform in the SDN Planes according to ITU-T Specifications

According to ITU, a secure network should be protected against malicious attack and should have appropriate response time, reliability and high availability [65]. ITU-T recommendation identifies eight of such sets that protects against all major security threats. The identified security dimensions are: access control, authentication, non-repudiation, data confidentiality, communication security, data integrity, availability and privacy. The security dimensions provide an end-to-end solution when applied to the security layers. Table 2.7 presents SDN solutions according to security dimensions recommended by ITU-T. The remark reflects the current state of stable solutions available both in research and practice.

Table 2.7: Security solutions according to ITU-T security recommendations [2]

	Security property	Solution name	Mechanism used	Remark
S	Authentication	FortNox [108]	Role-based authentication and authorization	Open Challenge
		FSL [61]	Controls authentication policies (admission control)	
T	Data Integrity	OFHIP [95]	IPSec encapsulated security payload (ESP)	Identity management systems are lacking
		Others [72]	VeriFlow, FortNOX , ensure integrity through flow rule legitimacy	
		Isolation [57]	Traffic isolation-based integrity	
R	Non-Repudiation	Others [145]	Uses permanent user identities (LISP)	Open Challenge
		OFHIP [95]	Uses HIP for permanent identities	
		VAVE [141]	Source address validation of incoming packets	
I	Data Confidentiality	OF-RHM [67]	Random host mutation	No specific security systems or applications
		FortNOX [108]	Data confidentiality through flow rules-legitimacy	
		IBC [119]	Identity-based cryptography	
D	Availability	DISCO [107]	Distribute SDN control plane	Less research efforts to increase availability through higher security. App. plane and data plane availability is still a challenge
		McNettle [133]	Extended processing capabilities	
		MAESTRO [99]	Parallelism in multi-core processors	
		Others	Control-data plane functionality trade-off and optimal controller placement strategies	
E	Access Control	PermOF [136]	Impose access control on OF apps	Access control also needed for application plane and multiple controllers
		FRESCO [121]	Enables security architectures for ACL	
		FSL [61]	Access control policy enforcement framework	
		Resonance [97]	Enables dynamic access control policies	
	Privacy	OF-RHM [119]	OpenFlow random host mutation	Systematic user privacy enforcement mechanisms are lacking
		Isolation [57]	Traffic-isolation-based privacy	
		ident++ [96]	User-selected security procedures	
	Communication Security	TLS [36]	Ensure controller-switch communication security	Complex Configurations

2.6.5 Recent Studies on DDoS Attacks Detection and Mitigation

DDoS attacks are catastrophic and can be a major security issue for overall network availability. Different approaches to secure network from DDoS have been proposed [91]. Handling DDoS attack includes two steps: detection and mitigation. The defence and mitigation can operate in a centralised or distributed mode depending on the deployment of its modules. Approaches to detection and mitigation may be based on statistical analysis, policy based (predefined rules), machine learning and data mining or a hybrid of the approach.

A large number of DDoS detection and mitigation methods have been documented and categorised in [34], [10], [42]. In [93], the authors proposed an entropy based approach to detect DDoS attack. A threshold is set and if entropy is lower than the threshold and it persists for five consecutive windows, presence of attack is signified. This solution is effective for identifying volumetric traffic but not so appropriate in detecting slow rate DDoS attack. Another entropy based detection scheme is presented in [135]. This method extends a counter copy of flow entry in OpenFlow table, propose a low calculation overload and a level of intelligence at the OpenFlow edge switch. In [71], a joint entropy that relies on a statistical model used to detect and mitigate DDoS attack in SDN environment is presented. This method combines nominal, preparatory and active mitigation stage to detect and stop DDoS attack. A profile is generated in an attack-free period which is used in comparison with attack period to detect traffic anomalies and if detected, the controller determines suspicious pair and informs the switch to drop attack packets. This method gives high success rate in determining known and unknown attacks, however, the simulation is performed using a single topology and the latency introduced as a result of the multistage approach is not quantified.

A statistical approach to detecting DDoS attack in SDN is also presented in [117]. This approach utilises deviation from confidence interval to signify the presence of attack using throughput as impact metric. Similarly, analyses of average number of connection per user is presented in [31]. The associated traffic values per user are used to classify regular and attack traffic. If the statistics counter of IP address is less than the minimum number of packets per connection, the traffic is said to be malicious and the controller sends a drop rule for the IP address to the switch.

In [122], a policy based framework to improve security against DDoS attack is implemented by adding two new modules: connection migration (CM) and actuating triggers by extension to the OpenFlow data plane. The connection migration module adds intelligence to the data plane to monitor sources that will complete the TCP connection or not. The CM module only authorises useful TCP sessions to be established while the actuating trigger is used to activate flow rule under predefined conditions to manage network flows without delays. This method is good in addressing TCP SYN Flood attack but no other forms of DDoS attack. The intelligence at the data plane also introduces a level of delay which is a function of performance trade-off.

Another policy based scheme is proposed in [134]. The scheme incorporates three layered architecture to detect and mitigate attack using two modules: DaMask–D module and DaMask–M module. DaMask-D is an anomaly based attack detection system that can be trained in offline and online mode. If an attack is detected, alert is issued alongside packet information and forwarded to the DaMask-M module. The DaMask-M module match received alert to pre-existing policies of drop, forward or modify countermeasure and log result in a database. This policy based scheme introduces more communication overhead and increased latency in the network.

In [74], machine learning approach to detect DDoS attack in early stage is highlighted. Several techniques were suggested for classification of normal traffic from malicious traffic. These techniques have self-learning ability to adapt to network changes and Support Vector Machine (SVM) provides higher accuracy in classification than others. In addition, the work in [16] presents the use of Self Organising Map (SOM) with 6 tuples of attributes to detect DDoS attack. The proposed lightweight method considers median values in training the SOM and shows high rate of true positives and low rate of false alarm. The drawback of this method is that false negatives will be reported when the attack parameter is set to a low value.

Hybrid approach has also been used in the detection and mitigation of DDoS. In [35], SPHINX is used to detect known and potential attacks on SDN network topology. This model monitors and judges the legitimacy of the infrastructure plane devices and ensures only legitimate messages are executed. However, SPHINX introduces minimal overheads in the mitigating DDoS attack.

2.6.6 Recent Studies Using Sensitivity Analysis for DDoS Attacks Detection

Application of sensitivity analysis to computer networks has become a popular research topic in recent years, especially in relation to security [25][4]. In [63], the authors examined using small sample computer models to make decisions and judgement in the face of uncertainty for models associated with risk assessment of disposal of radioactive waste. The work was further extended in [64] to determine the applicability of three widely used techniques to computer models having large uncertainties and varying degrees of complexity. Sensitivity analysis has also been applied to address computer networks availability in [70]. The authors implement parametric sensitivity analysis to compute the effect of changes in the rate of constants of a Markov model on system dependability. Authors in [60] present a method for computing network output sensitivity with respect to variations in the inputs for multilayer feed-forward artificial neural network with different activation functions.

In terms of security, the authors in [100] performed a sensitivity analysis on DARPA intrusion detection datasets and reported that 33 out of 41 features of the network traffic characteristics can be removed without causing great harm to the classification accuracy of DDoS attacks and normal network traffic. Similarly, sensitivity analysis has been applied to attack pattern discovery in trusted routing scheme [69]. Using packet delivery ratio, normalised routing overhead, distrust threshold and trust update interval as performance metrics in different network conditions, the work carried out in [69] revealed that distrust threshold is more sensitive as compared to other metrics in optimising the detection rate of schemes employed. The authors in [4] explored the detection of bots in a compromised machine using dendritic cell algorithm (DCA). Their proposed algorithm and sensitivity analysis showed that incorporation of MAC value has a significant effect on the detection of bot using DCA algorithm.

All the works mentioned above highlights the application of sensitivity analysis in identifying input parameters that significantly affect system response in designing attack detection algorithms. However, our approach differs from the ones mentioned in the following aspects. Firstly, the features of interest are extracted from emulated SDN environment with normal and DDoS attack traffic. Secondly, the implementation of local sensitivity analysis using artificial neural network to identify key network metrics

that mainly influence the prediction of whether an SDN is under attack or secure. Full details of this approach can be found in Chapter 5.

2.6.7 Overview and Classification of DDoS Attacks

Availability requires computer systems to function normally without loss of resources to valid users at any point in time. DDoS attacks remain one of the most challenging issues to availability and constitute a major threat to security problems in today's internet.DDoS attacks can be classified under the availability section of the CIA Triad presented in section 2.6.1. This form of attack has been a menace in the network security world and the end is still far from sight as it poses high severity risk to the SDN planes. A brief overview and taxonomy is presented in the subsection below.

2.6.7.1 Overview of DDoS Attacks

The objective of a DDoS attack is to bring down the services of a target using distributed multiple sources with or without their consent. It's history dates back to over three decades ago. However, the large scale DDoS attack occurred in the nineties when a malicious user used Trinoo to disable University's computer network for more than two days [37]. Since the Trinoo attack, the motivation and growing prevalence of DDoS attacks show that legacy defence mechanisms are only partially effective. In DDoS attack, malicious users focus on tearing down network infrastructure rather than gaining access to the end users, this makes it difficult for intrusion detection mechanism in traditional defence mechanism inefficient [43] [14]. DDoS attacks are inevitable due to inherent weakness in internet architecture to keep the intermediate network as simple as possible to optimise packet forwarding [56]. In addition, susceptibility to DDoS attack is dependent on the position of security in the rest of the global internet [91].

2.6.7.2 Classification of DDoS Attacks

A comprehensive taxonomy of DDoS attack based on the degree of automation, exploited weaknesses, source address validity and attack rate dynamics has been presented in [91]. The distributed nature of a DDoS attack makes it significantly more

powerful, harder to identify and block its source. DDoS attacks can be divided into five categories based on attacked protocol level as shown in figure 2.6[43].

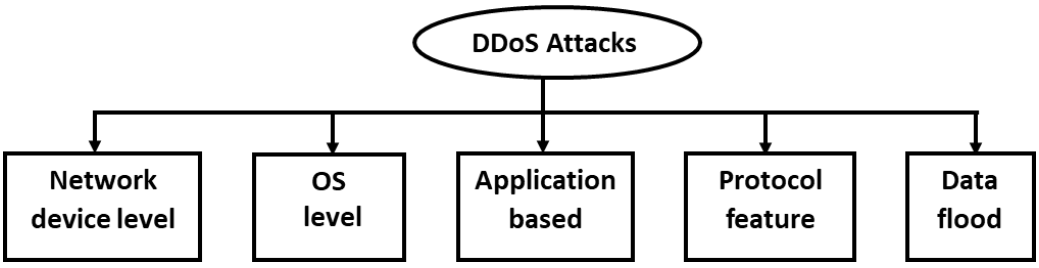

Figure 2.6: Classification of DDoS attacks

Network device/host level attack: Attack at this level explore bugs or weaknesses in device software or by exhausting the hardware resources. This attack renders the target machine unavailable or disables the communication mechanism making the host crash, freeze or reboot. This form of attack can be detected due to high volume traffic but its mitigation cannot be done at the host level alone. It requires help from an intermediate device such as a firewall. Example of this is TCP SYN attack which exploits the weakness of the three-way handshake in the TCP connection set-up. A server receives an initial SYN request from a client, responds with a SYN/ACK packets and waits for the final ACK of the client. A large number of SYN packets are sent without acknowledgement while the server is waiting for non-existent ACKs. This process results in a server with a full buffer queue that is unable to process legitimate connections.

Operating System level attack: operating system DDoS attacks take advantage of the ways protocols are implemented. Example of this is the ping-of-death attack caused by malicious user deliberately sending echo requests greater than Internet Protocol (IP) standard size. This attack can cause the victim's machine to crash.

Application level attack: this attack targets a given application on victim host, thus restricting legitimate clients use of that application and possibly tying up resources of host machine. Other applications can still be accessed by clients if the host resource is not completely consumed. For example, a signature attack on an authentication server ties up signature verification authentication, but the affected server will still respond to ICMP ECHO and other applications that are independent of authentication services. Detection of this form of attack is challenging because other applications on the vic-

tim's host will still operate undisturbed. The attack volume is usually small enough to appear abnormal and transmitted malicious packets are virtually indistinguishable from legitimate packets. A robust defence system would have to be modelled and monitored for each application to account for possible detection against small volume attacks.

Protocol feature attack: attack at this level takes advantage of certain standard protocol features. For example, attack launched on domain name system cache on name servers.

Data flood: this attack aims for the victim's bandwidth. It usually thrives by sending voluminous traffic to process. Example of this is a UDP flood attack and ICMP flood attack. This attack can be detected statistically and can be stopped from edge host devices.

2.7 Summary

In this chapter, SDN architecture and SDN security have been discussed. The discussion starts with the technologies that led to SDN followed by the comparison of security approach in traditional network and SDN. OpenFlow basics are also part of the discussion. A comprehensive review of the SDN architecture and interface coupled with SDN development platform is discussed. Security issues and vulnerabilities SDN is exposed to is also examined. A brief overview and classification of DDoS attack that SDN is prone to is also presented. From the review, it is easier to spot the vulnerability of SDN architecture to DDoS attacks and effort has been made to device a means of detecting and mitigating this form of attack to make SDN more secure. The next chapter delves into DDoS attack strategy and detection techniques employed to identify normal from abnormal traffic.

Chapter 3

Detecting DDoS attacks in Software Defined Networks

3.1 Introduction

One of the known forms of network attack that threatens SDN architecture is DDoS attack. DDoS attack is aimed at consuming available resources of network devices; hence, making it impossible to access by legitimate users. Also, it can be launched to consume network bandwidth by compromising network traffic.

The number of successful DDoS attacks launched has seen an increment at an alarming rate over the last decade [98]. Shade of attacks seem to be evolving per launch and several detection and mitigation techniques are been offered by different companies. Although these companies claim to protect enterprise networks, government and other critical sectors, their progress is not as impressive as claimed. This setback is due to lack of attack information on the frequency, duration, number of agents machines, attempted response, effectiveness and damages suffered as a result of the DDoS attack. Figure 3.1. shows the rising scale of DDoS attack in the last 10 years. It can be seen that DDoS attack magnitude on networks increased in the last five years as compared to attack magnitude witnessed in previous years. This attack strength may be attributed to exploitation of vulnerabilities found in Internet of Things (IoT) devices and continued use of amplification techniques to maximise the scale of attack [75].

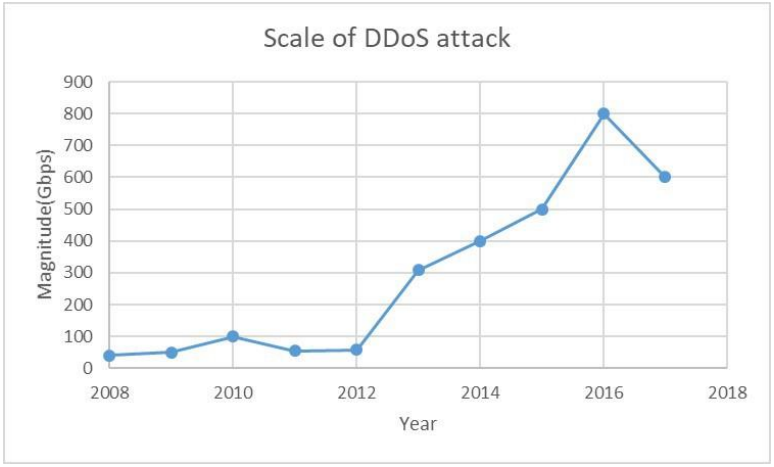

Figure 3.1: Rising scale of DDoS attack experienced in the last 12 years [98]

Due to the global view and monitoring of network provided by SDN, anomaly in network traffic can be detected in real time and mitigated from a central controller. Currently, service providers are in the process of deploying SDN trial versions or investigating how this technology can be leveraged on to mitigate DDoS attacks [98].

3.2 DDoS Attack Detection in Traditional Network

Since the emergence of DDoS attacks, variety of both attacks and defence mechanism is overwhelming. A comprehensive DDoS attack overview and defence mechanism is presented in [91]. The authors classify defence mechanism based on activity level and by the degree of cooperation.

The use of dynamic resource allocation can also help in accurately detecting DDoS attack [144]. The authors present a dynamic way of utilising reserved cloud resources to cloud customers under DDoS attack using queueing theory based model to ensure the availability of cloud services to benign users.

Another popular detection method is the use of machine learning [50]. In [50], Radial-basis-function neural network model is proposed. A small number of statistical features is employed to describe attack behaviour and classify attack traffic. These features implement a passive monitoring system and report high detection rates. Similarly, the authors in [80] propose a clustering based detection method. This method

employ entropy value on select attributes and then clustered to recognise the phases of attack.

A detection approach that involves the use of correlation analysis in data centre network is presented in [140]. The model predicts flow classes based on the k closest training in the feature space and evaluates the influence of correlation analysis.

3.3 DDoS Attack Detection in SDN

Different techniques have been proposed in the detection of DDoS attack in SDN. In [38], flow events are collected from switch interface and a sequential probability ratio test which has bounded false positive and false negative error rates threshold is applied to make a decision and locate compromised interfaces.

The use of entropy in identifying anomaly from normal flows have been explored in [135][93][89]. Entropy can measure the randomness of benign and malicious traffic and good detection accuracy can be obtained using suitable window size and appropriate threshold.

Rate limiting has also been applied in DDoS attack detection and mitigation in SDN [79] [83]. Rate limiting can shield the network from complete outage during DDoS attack. However, all flows are affected in this approach and response time increases for legitimate traffic.

3.4 DDoS Attack Strategy

Numerous DDoS attack methods are being used to degrade the performance or availability of targeted network equipment. These attacks can be classified broadly as network or application level attack. A successful DDoS attack generally follows the following steps:

- An attacker scans the network for vulnerable active host.
- The vulnerable hosts are then compromised for exploitation.
- Finally, compromised hosts are used to an launch attack on victim.

Figure 3.2. Shows a typical DDoS attack strategy. The attack strategy and how it affects SDN is presented in the vulnerability section

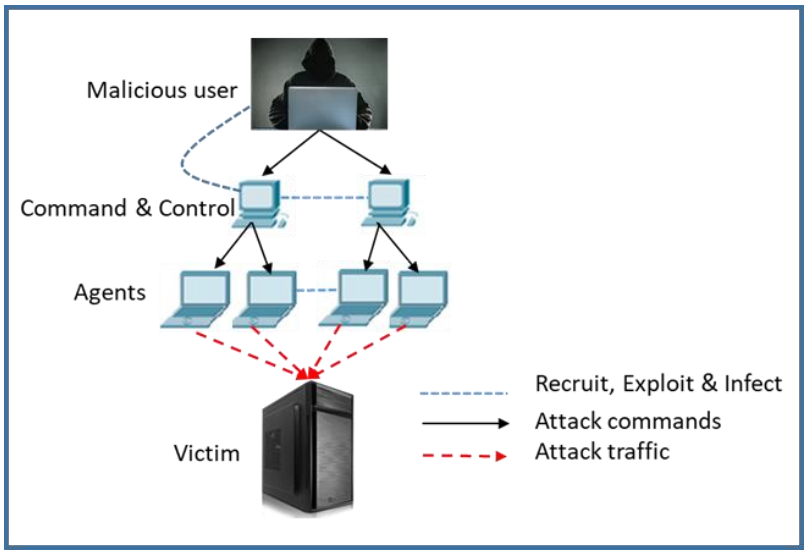

Figure 3.2: DDoS Attack Strategy

3.4.1 Vulnerability of SDN to DDoS Attack

Seven threat vectors have been identified in [77]. Three of the identified vectors namely: (1) Attack on control plane communications (2) Attack on vulnerabilities in controller and (3) Lack of trust mechanisms to ensure communication between the controller and management applications are specific to SDN. Hijacking the communication channel between OpenFlow switches and controller could be a potential launchpad for DDoS reflection attack or traffic redirection for malicious gains.

In the case of a vulnerable controller, the use of common IDS may not be sufficient as it may be hard to determine the exact combination of events that trigger abnormal traffic behaviour [77]. To further clarify this vulnerability, a DDoS attack process in SDN is presented in Figure 3.3.

In Figure 3.3, the malicious user crafts a packet and send to the victim. In a situation where there is no match entry rule in the table of the OpenFlow switch, the OpenFlow switch encapsulates the header of the packet and sends a packet_in message to the controller for instruction. The controller decrypts the packet in message and

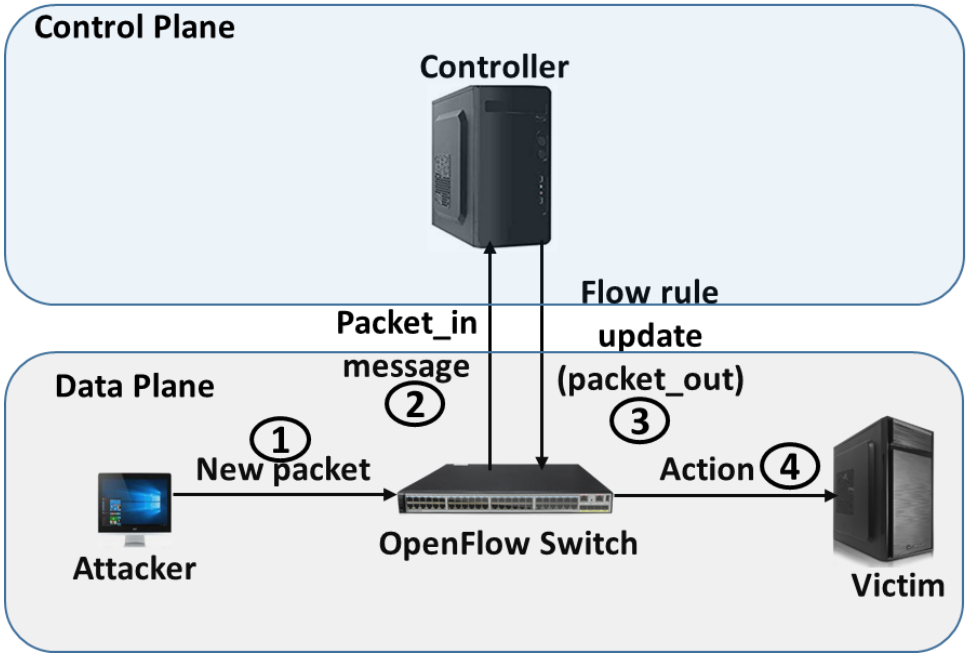

Figure 3.3: DDoS attack process in SDN

calculate the appropriate route to the destination address based on the installed rule (drop, forward packet to port, send) on the controller. Packet_out message is generated and a new flow rule is installed on the OpenFlow switch table. Then, the OpenFlow switch forwards the packet to the destination address.

It is worthy of note that if a malicious user spoof source IP address and generates spurious packet_in messages at specified intervals, more packet_in messages will be sent to the controller making the data and control plane resources vulnerable to flooding attack depending on attack strategy deployed [35]. Hence, the need to build a robust flooding attack detection algorithm in SDN controller.

Associated risks DDoS attack pose on SDN can be summarised as follows:

- Network devices at data and control plane buffer capacity is limited
- DDoS attacks change rate of flow of network traffic dynamically and employ multistage attack depending on the nature and strength of the attack
- DDoS attacks change rate of flow of network traffic dynamically and employ multistage attack depending on the nature and strength of the attack

- For every new flow, the switch sends packet_in message to the controller. As a result, DDoS attack can exhaust the communication channel between switch and controller.
- The controller represents the brain of the network and it controls the underlying data plane devices by installing flow rules. Hence, single point management failure can be attained by compromising the availability of the SDN controller.

The aim of identifying potential security flaws in SDN is not to project it as a weak innovation in network architecture but rather to build a robust security measure around the controller to detect and mitigate both known and unknown security breach.

3.5 Types of DDoS Attacks in SDN

In this section, a brief description of DDoS attacks launched during this simulation is presented.

3.5.1 ACK Flood

A large number of ACK packets, usually not related to open connection is sent to the victim server. As a result, available system resources to evaluate legitimate incoming packets are exhausted. ACK flood attack can be used as a smokescreen for more advanced attacks as the packets usually go through routers, firewalls and other intrusion prevention/detection system [117].

3.5.2 SYN Flood

SYN flooding attacks exploit known vulnerabilities in a 3-way handshake that begins a TCP connection. The goal of SYN flooding is to deplete the backlog of Transmission Control Block (TCB) that holds all information about a connection as shown in Figure 3.4. This is done by sending a large number of SYN request to the server. The server replies to the request by sending SYN + ACK packet and waits for the ACK response from client. The malicious user does not send ACK packets and the server waits for non-existent ACK message. Hence, the buffer queue of the server becomes full and incoming valid requests are dropped [117].

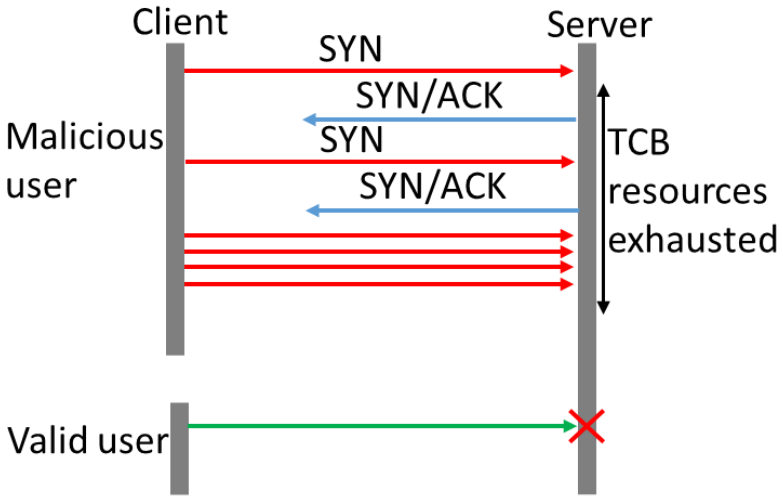

Figure 3.4: progression of SYN flood attack
[110]

3.5.3 Slowloris

Slowloris attack opens multiple connections to the victim's web server and keeps them open for as long as possible. In this attack, partial HTTP requests are sent and subsequent headers for each request is sent to keep the connection open, but the requests are never completed. Ultimately, the victim's maximum concurrent connection pool is filled and legitimate connection attempts are denied afterwards [117].

3.6 Experimental Approach

In this section, a brief description of the experimental setup is presented. And then we investigate the feasibility of DDoS detection mechanism by several TCP SYN Flood, ACK flood and Slowloris attacks performed.

3.6.1 Experimental Setup

Mininet software [90] was used to create the tree network topology shown in Figure. 3.5. The mininet emulator is an open source network emulator capable of creating a realistic virtual network, running real linux kernel, switch, application code and

devoted entirely to OpenFlow architecture and SDN implementation. Also, we use OpenDaylight [109] controller (Nitrogen) to deliver SDN platform to make network adaptive and programmable. For the topology in Figure. 3.5, tree topology is used which is created by python API of Mininet. For the two experiments carried out, the bandwidths were set to 10MbPS and 100Mbps to represent Ethernet and Fast Ethernet connections respectively. These bandwidths were picked to mimic what is obtainable in the real day-to-day network activities to analyse and detect DDoS attacks [117].

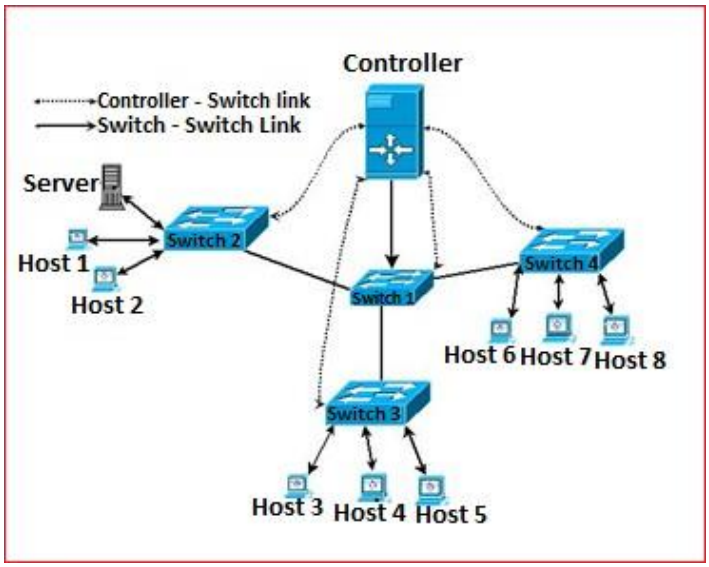

Figure 3.5: Network topology

3.6.2 System Implementation

Depending on the nature of the attack, volumetric attacks which exhaust device throughput can be detected by monitoring the rate of change of throughput before and during the attack to detect presence of an attack. As shown in Figure. 3.5, the attacker in the SDN environment could be a host or a compromised switch. In this work, hosts in the network have been used to perform attack in SDN. ACK, SYN and Slowloris attacks are launched on victim's server connected to switch 2 in Figure. 3.5 to identify their impact on SDN planes. These attacks have been launched independently by using conventional attack methods and tools [53]. Table 3.1 provides a summary on the description of tools used in implementing these attack.

Table 3.1: Simulation parameters description

Simulation	Descriptions
Mininet	Used for emulating the network topology
Host machine	Intel Core i7, 16G RAM
Open Daylight Controller	Ubuntu server as the base operating system for Open Daylight Nitrogen release controller
Oracle virtual box	Virtual environment for simulation
Other tools	Pentmenu for launching (Slowloris: using netcat; TCP ACK Flood: using hping3 and TCP SYN Flood: using hping3) iperf tool for monitoring network throughout before and during an attack

The same attack was launched for both Ethernet and Fast Ethernet link. The steps followed are explained as follows:

Step 1: Create mininet topology and set bandwidth of 100Mbps for Fast-Ethernet link and 10Mbps for Ethernet link

```
Sudo mn --controller=remote,ip=192.168.1.4

--topo=tree,depth=2,fanout=3--link tc,bw=10
```

Step 2: Confirm connectivity between hosts, switches and ODL controller

```
>net

>pingall
```

Step 3: Setup Server in the network to listen at a specified port number and interval

```
iperf {s {p 80 {i1
```

In the topology, Hosts 1 was used to generate iperf requests from the server and Host 2 was used as ping probe to monitor packet loss.

Step 4: Launch attack using Pentmenu

```
  Cd Pentmenu
./pentmenu  select attack and launch
```

Host 3 to host 8 in the tree topology were the compromised hosts made to launch DDoS attack traffic on victim's server at 25%, 50% and 75% attack rate respectively. Once the DDoS attack commenced on the victim, the arrival rate of packets at the server port becomes higher than the server capacity within a short period. Consequently, the server could not respond to burst of open connections from either valid or malicious users. Hence a drop in throughput. Table 3.2 shows the parametric values of attack traffic launched.

Table 3.2: Parameter used for simulating the attack traffic

Experiment No	Attack rate	Hosts involved	TCP SYN-FLOOD	ACK-FLOOD	Slowloris
Experiment 1	25%	H5, H7			
Experiment 2	50%	H5, H6, H7, H8	2.5×10^5 data-bytes sent	2×10^5 data-bytes sent	2.5×10^5 data-bytes sent
Experiment 3	75%	H4, H5, H6, H7, H8, H9			

3.7 Performance Evaluation

In this study, an experiment for investigating three different attack rates: 75%, 50% and 25% was performed on Fast-Ethernet and Ethernet. The severity response due to the two media is simulated and presented in Figure. 3.9 and 3.10. Time series data were generated, and the observations recorded at a regular interval of 1 second. This process was continuous using the same experimental condition, the stream of measurements were recorded until the simulation clocks-out at 360 seconds.

Let us suppose the series $X = \{x(t)|1 \leq t \leq 360\}$ denotes the throughputs recorded at equally spaced discrete time interval of 1 second each. We then present mathematically the throughputs vector for each of Fast-Ethernet and Ethernet at different attack rates as:

$$x(t) = (x_1(t), x_2(t), x_3(t), \cdots, x_{360}(t)) \tag{3.1}$$

However, we are interested to study the efficacy of the attack rates severity for Fast-Ethernet and Ethernet. The stream of the simulated throughputs were split into three components. Where the first component is 1 minute, which has 6 different levels within the possible outcomes of the experimental time. The second and third components are 2 and 3 minutes, and their levels are 3 and 2 respectively.

We established the 95% confidence band (see Table 3.5 and 3.6) [39] to enable us make a robust inference on the basis of probability. Taking α = 0.05, to indicate level of error tolerance to the experiments. Therefore, we use the method of data condensation to study the properties of the throughputs distribution by looking at the mean and standard deviation. We deployed Kolmogorov-Smirnov tests [51], at α = 0.05, the p_{value} = 0.0001 and reject the null hypothesis that the throughputs for Fast-Ethernet and Ethernet at different attack rates are normally distributed. Since the throughputs are normally distributed at the indicated level of significance. The confidence band for detecting attack severity is evaluated in Eqn. 3.2 as follows:

$$\bar{x}(t) \pm Z_{\alpha}\{\frac{\sigma}{\sqrt{n}}\} \tag{3.2}$$

where: $\bar{x}(t)$ denotes mean of the throughputs; $Z_{\alpha}(\pm 1.645)$ denotes the theoretical distribution of the throughputs at α level of significance; σ denotes the dispersion of the throughputs from the mean and n is the number of instances.

3.8 Normality Test

To test the data obtained for statistical error, normality test using Doornik Hansen, Shapiro Wilk, Lilliefor and JarqueBera is performed. Figure. 3.6 3.7 3.8 presents the frequency distribution of the observed FastEthernet values (throughput) against their frequency. It can be seen that the distribution of the three scenarios without an attack (SYN, ACK and Slowloris) is bell shaped and well modelled by a normal distribution [51]. In both frequency distribution for FastEthernet and p-values obtained in Table 3.3 for ethernet, both data follows a normal distribution. Hence, a justification for comparison with attack scenario for skewness.

Table 3.3: Normality test for Ethernet data without attack

Normal	Doornik-Hansen test			Shapiro-Wilk			Lilliefors test			Jarque-Bera test		
	TS	P-Value	Remark	TS	P-Value	Remark	TS	P-Value	Remark	TS	P-Value	Remark
EACK	24.25	5.41e-006	Normal	0.85	5.04e-018	Normal	0.295	0	Normal	15.17	0.0005	Normal
ESYN	26.23	2.0048e-006	Normal	0.857	1.139e-017	Normal	0.283	0	Normal	61.27	4.95e-014	Normal
ESLOW	50.38	1.148e-011	Normal	0.85	9.42e-018	Normal	0.272	0	Normal	101.106	1.109e-022	Normal

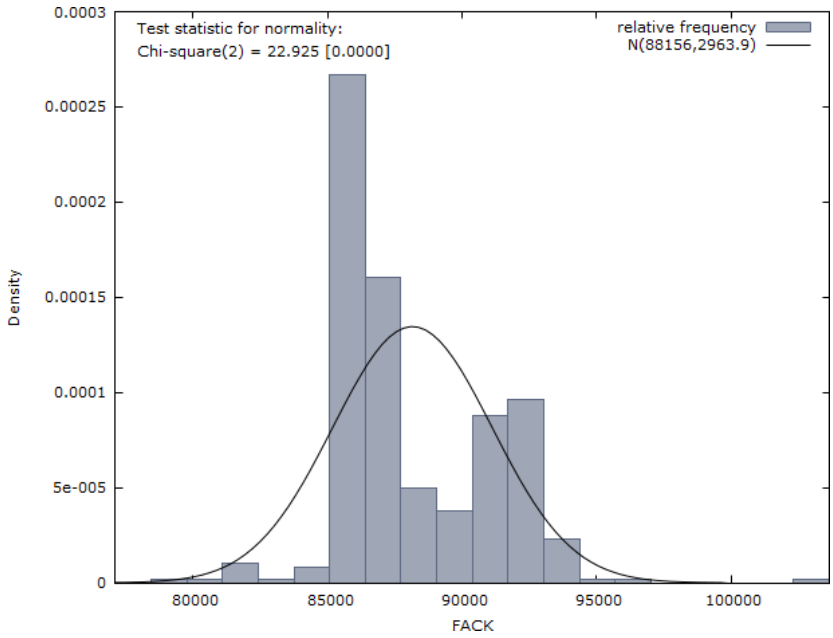

Figure 3.6: Frequency distribution of Fast-Ethernet Link without ACK attack

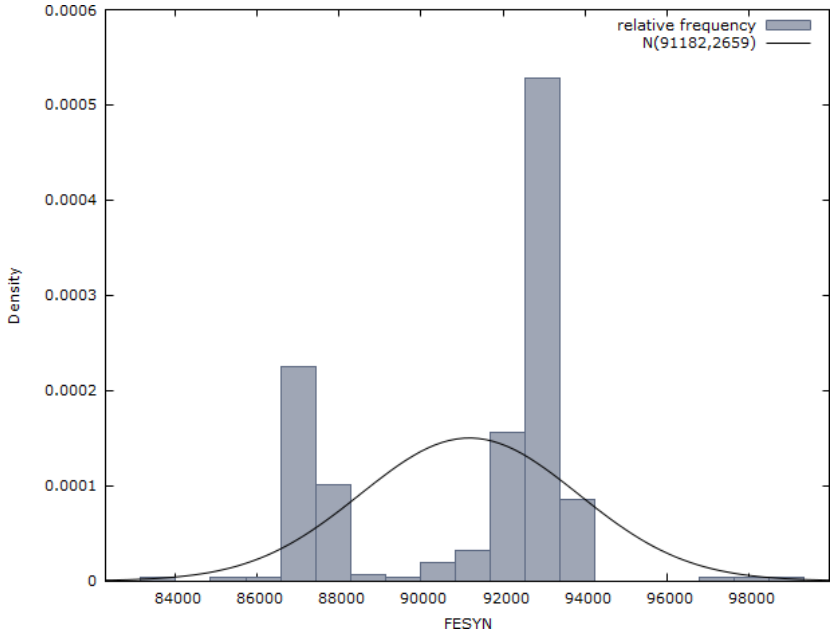

Figure 3.7: Frequency distribution of Fast-Ethernet Link without SYN attack

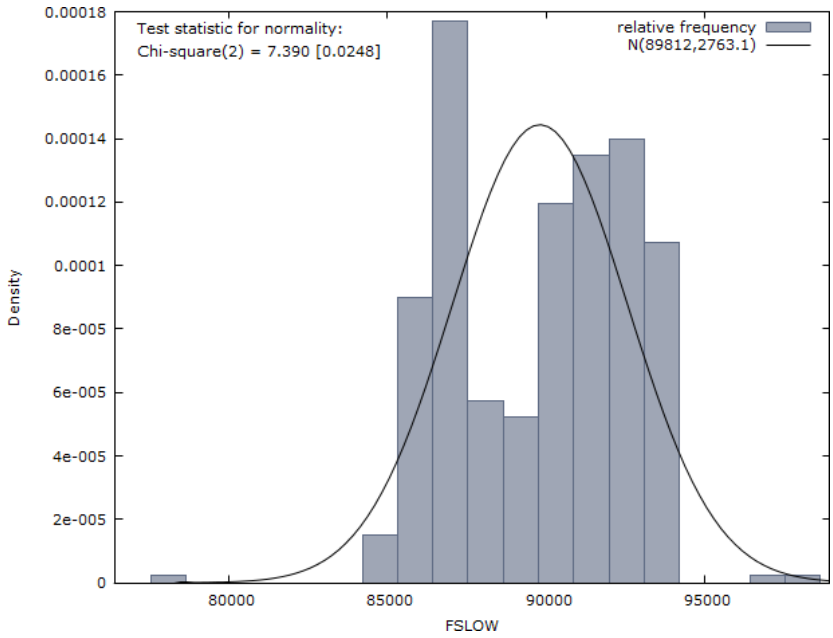

Figure 3.8: Frequency distribution of Fast-Ethernet Link without slowloris attack

3.9 Result and Discussions

Two experiments were completed using the network topology presented in Figure. 3.5. In the first experiment, we use bandwidth of 10Mbps for the Ethernet link and Fast-Ethernet bandwidth of 100Mbps for the second experiment. In both cases, ACK, SYN and slowloris attacks were launched and the data collated.

3.9.1 Effect of DDoS Attack on the Server

As shown in Figure. 3.9-3.10, the performance of the server appears as a decreasing function of the server capacity. FACK represents throughput without attack for a duration of 360 seconds while FA75, FS75 and FSL75 represent SYN, ACK and Slowloris throughput values at 75% attack rate respectively. The impact of the attack launched by compromised hosts 3-8 is observed after 75 seconds of transmission with a bandwidth utilisation degradation (a sharp dip as shown in Figure. 3.9. For FA75, A spike can be seen at 81seconds indicating a sign of recovery from the attack. However, the server was rendered completely unavailable for the rest of the transmission. The observed trend is similar for ACK, SYN and slowloris attack launched on the Ethernet network

shown in Figure. 3.10.

The simulation result in Figure. 3.9-3.10 shows that ACK, SYN and Slowloris all have a negative effect on the server and is capable of rendering it unavailable within minutes of successful attack launch.

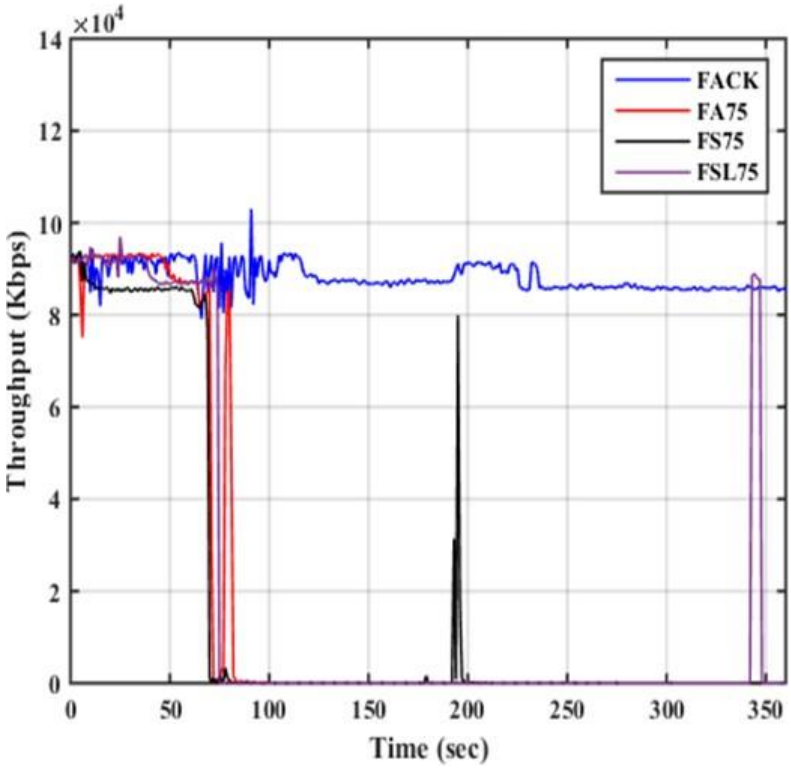

Figure 3.9: Effect of DDoS attack on server throughput for Fast-Ethernet

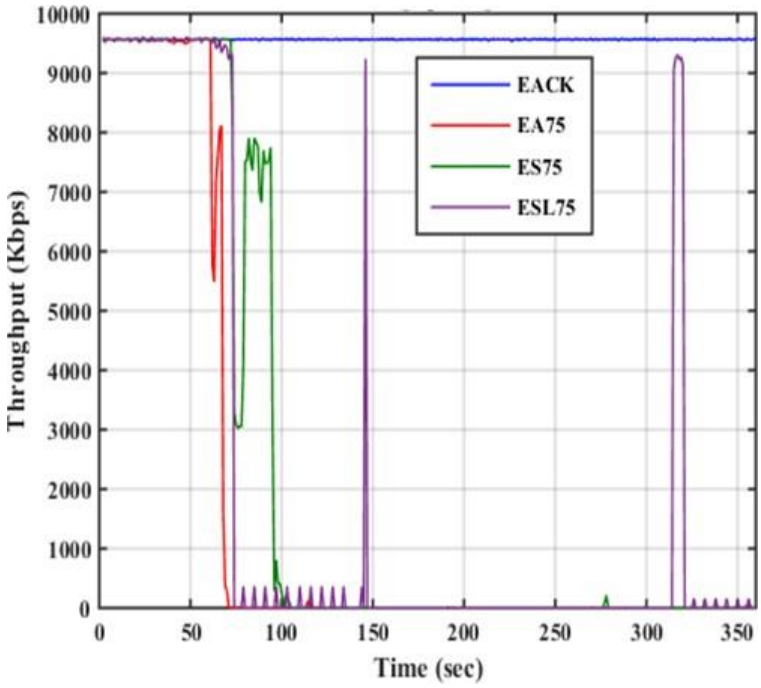

Figure 3.10: Effect of DDoS attack on server throughput for Ethernet

3.9.2 Severity of DDoS Attacks on Server

Figure. 3.11 and Figure. 3.12 shows the visual distribution of traffic data before and during the attack. In Figure. 3.11, the 75% attack shows a large variation from the central tendency and quite a number of packets are skewed towards the lower end of the range which is similar for the SYN and ACK attack. Similarly, the 50% and 25% attack scenarios show mild variation. For the fast Ethernet, the server was made completely unavailable in the three attack scenarios. The plot without attack is more condensed which means it varies less and more consistent without outliers. Hence, prediction is more dependable and shows the severity of the 75%, 50% and 25% attack scenarios. In Figure. 3.12, the 75%attack is severe as it is the only one with outliers that made the server completely unavailable for the ACK and SYN attack. It can be observed that the 25% attack scenario for ACK and SYN had a better upper quartile than 50% attack scenario even though they all exhibit variations based on severity for without attack scenario. In both cases, the slowloris attack open large connections and com-

pletely overwhelmed the server resources, which consequently makes it unavailable. The variation in upper quartile shows the severity of attack in the network when 50% and 25% is compared with 75% attack rate.

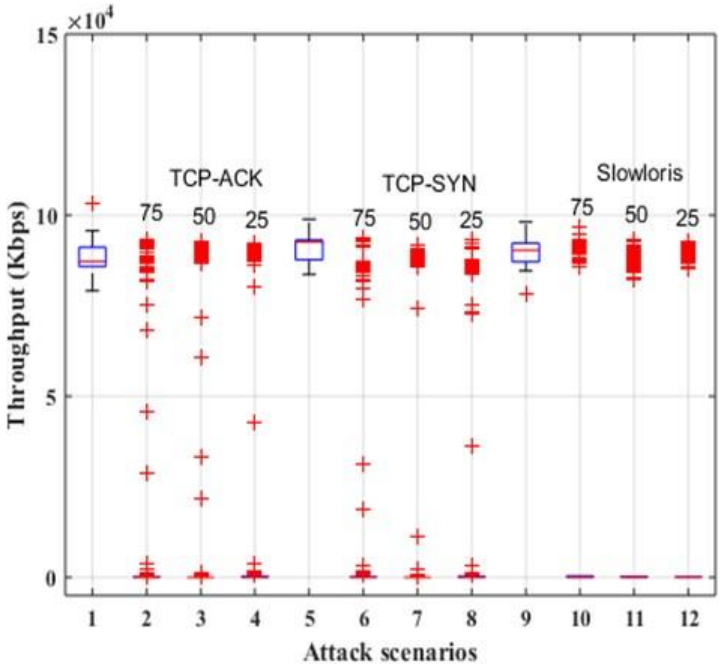

Figure 3.11: Severity of Attack on Fast-Ethernet Link

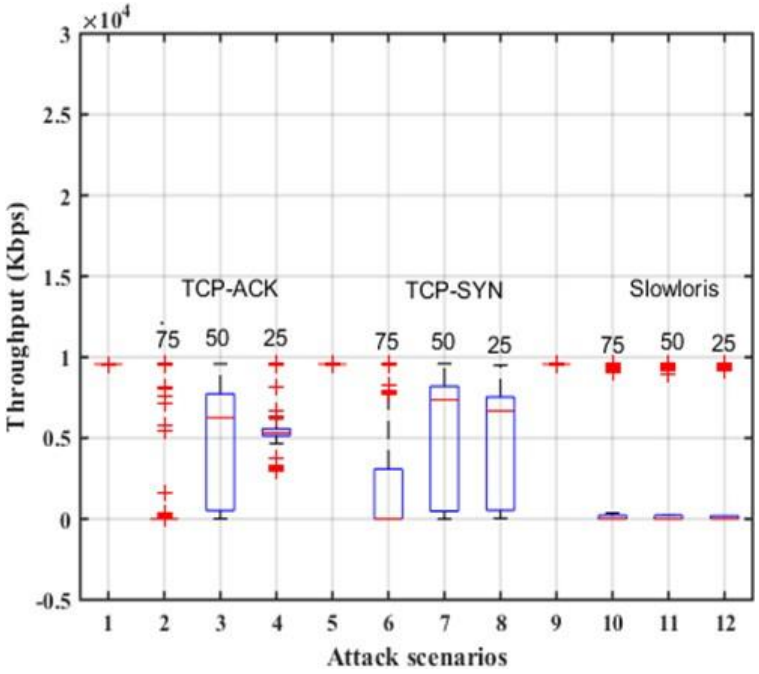

Figure 3.12: Severity of Attack on Ethernet Link

3.9.3 DDoS Detection Accuracy

The increase in detection window affects the prediction accuracy. It can be seen from Table 3.4 that increment of 1 minute reduces the prediction accuracy by 46%, 28% and 40% respectively for ACK, SYN and slowloris attack. Also, an increase of 2 min reduces the accuracy by 60%, 52%, 59% respectively for the Ethernet link. The result is significant as it shows that the server's performance can be brought down in under 2 minutes as seen in Figure (3.9-3.10) for the modelled attack scenarios and should not be ignored. Therefore, a suitable detection window of one minute can help detect DDoS attack with 99% accuracy.

The justification for one minute detection window is not far-fetched. It can be seen from Figure. 3.11 and Figure. 3.12 that server capacity degradation occurred in less than 2 minutes. Hence a detection window of one minute with high accuracy is suitable for detection of DDoS flooding attack in SDN. It can be argued that a detection window of 30 seconds or less will be more suitable in detecting DDoS attack. However, this will require fast processing capacity and introduce overhead and resource utilisation on

the controller or data plane devices. In [93], it has been stated that there is an increase in CPU usage with respect to window size. The goal is to reduce overhead as much as possible while getting optimum performance from the controller.

Table 3.4: Detection accuracy for varying window size

Scenarios	DDoS Attack	1 min	2 min	3 min
Ethernet	TCP-ACK	99.88%	53.84%	35.89%
	TCP-SYN	99.96%	71.94%	47.96%
	Slowloris	99.94%	60.06%	40.66%
Fast-Ethernet	TCP-ACK	99.51%	60.89%	41.14%
	TCP-SYN	92.93%	53.34%	35.57%
	Slowloris	98.83%	60.72%	40.40%

3.9.4 Effect of DDoS Attack on Mean Throughput using different Window Size

Table 3.5 and 3.6 presents a detailed description of analysed data in detecting attack with 95% confidence interval. X75, X50 and X25 represents mean throughputs at 75%, 50% and 25% respectively. The results in both cases (FastEthernet and Ethernet) highlight the fact that the mean throughput drops drastically after every minute in the presence of an attack. The result is significant as it shows that an increase in volume of attack reduce the value of throughput which in turn have effect on the mean throughput for sampled window size

Table 3.5: Ethernet simulation table

95% confidence band for ACK	X_{75}	X_{50}	X_{25}
9565 ± 2.1 (after every 1 minute)	9556	9564	9566
	745.6	4189.7	4866
	0.2	2618.3	5241
	0.4	4823.8	5460
	0.6	3600.5	5251
	1.7	3534.4	5246
9565 ± 1.9 (after every 2 minutes)	5151	6877	7216
	0.3	3721.1	5358
	1.5	3567.6	5249
9565 ± 1.9 (after every 3 minutes)	3434	5457	6630
	0.9	3986.3	5246
95% confidence band for SYN	X_{75}	X_{50}	X_{25}
9565 ± 2.2 (after every 1 minute)	9564	9565	9527
	4200	7142	920.1
	0	2194.9	6034
	0.2	728.9	3124.6
	4.6	4567.1	669.3
	0	5707	7258
9565 ± 1.5 (after every 2 minutes)	6882	8353	3723.5
	0.1	1462	4580
	2.3	5137.1	6963.5
9565 ± 1.8 (after every 3 minutes)	4588	6300.5	4493.8
	1.6	3668	5683.8
95% confidence band for Slow-loris	X_{75}	X_{50}	X_{25}
9565 ± 4.5 (after every 1 minute)	9564	9564	9565
	1927.3	2808	4080
	178.5	27.1	15.3
	0	23	9.2
	0	11.5	0
	937.1	0	0
9565 ± 3.3 (after every 2 minutes)	5746	6186	6822.4
	89.3	25	12.3
	468.6	5.8	0
9565 ± 2.4 (after every 3 minutes)	3890	4132.9	4553
	312.4	11.5	3.1

Table 3.6: Fast-Ethernet simulation table

95% confidence band for ACK	X_{75}	X_{50}	X_{25}
91360 ± 321 (after every 1 minute)	91228 19642.8 0 0 0 0	87603 1432 0 0 0 0	89633 2368.2 0 0 0 0
90516 ± 528.1 (after every 2 minutes)	55435.6 0 0	44518 0 0	46001 0 0
89423 ± 401.3 (after every 3 minutes)	36957.1 0	29678 0	30667 0
95% confidence band for SYN	X_{75}	X_{50}	X_{25}
92867 ± 123 (after every 1 minute)	86413 12573 27.2 2224.2 0 0	87705 5775 0 0 0.2 0	86353 11765.8 63 0 0 0
92561 ± 232.5 (after every 2 minutes)	49493 11.3 0	46740 0 0.1	49059 31.5 0
92619 ± 160.9 (after every 3 minutes)	33004.5 741.4	46740 0.1	32727.4 0
95% confidence band for Slow-loris	X_{75}	X_{50}	X_{25}
91182 ± 324.9 (after every 1 minute)	90440 20412.3 62.5 61.6 36.9 7364	88240 27612 27.7 27.5 7.9 12231	91118 37886.2 15.8 18.9 15.9 15.7
90977 ± 305.1 (after every 2 minutes)	55426 62.1 3701	57926 27.6 6119	64502.3 17.3 15.8
91272 ± 244.8 (after every 3 minutes)	36971.6 2488	38626.7 4089	43007 16.8

3.10 Machine Learning Approach to DDoS Attack Detection in SDN

Machine learning is a wide inter-disciplinary area of research which involves learning patterns from datasets for computer to perform a specific task without explicit instructions. Building a predictive machine learning models has found application in detecting DDoS attack in SDN. Five machine learning methods were applied to existing DARPA dataset with SVM giving the highest detection accuracy and less false positive rate [74]. However, the analysis presented is not compared with dataset received from emulated software defined network environment.

An unsupervised artificial neural network, Self-Organising Maps (SOM) has also been applied in detecting flooding attack in SDN [16]. Using 6-tuple features to train the traffic flow, the proposed method is able to achieve high detection rate and low rate of false alarm in less computation (CPU) times as compared to other methods involving KDD-99 dataset.

Application of XGBoost, an extreme gradient boosting algorithm classifier to detect DDoS can be found in [27]. The result shows better detection accuracy and lower false positive rate as compared to three other machine learning algorithms when applied in SDN-based cloud.

Another interesting approach to detecting DDoS attack in SDN is Deep Learning [82], [128]. Although some novel machine learning techniques have been applied to DDoS detection in SDN, most of them employ the use of KDD and NSL-KDD dataset with very few extracting real network dataset from SDN configuration.

All DDoS detection techniques leverage on identifying key features such as traffic flow metrics relevant to identify malicious traffic from benign traffic. Our emulated network scenario is simpler and mimics network architecture obtainable in a mid-sized enterprise network with focus on three key real-time traffic features obtained from emulated SDN architecture in classifying ICMP, UDP and HTTP flooding attacks.

3.11 Proposed solution

In this section, a custom tree topology is built to extract real SDN data. The extracted data is fed into machine learning algorithms to classify DDoS attacks using six machine learning algorithm.

3.11.1 System Architecture and Setup

A custom network topology has been designed using Mininet emulator [90] to address the problems highlighted earlier. Tree topology is considered because it can be easily adopted for wide area network and it offers easy expansion of node.

The custom topology shown in Figure 3.13 is implemented on 32G RAM Intel Xeon E3-1220 processor with Kali Linux as the base operating system. The floodlight controller [49] is deployed in the VirtualBox VM running Ubuntu 18.10 LTS while Mininet software is deployed on VirtualBox VM running Ubuntu 16.10 LTS.

The modelled network comprises of 10 OpenFlow switches and 16 hosts which are connected using 100Mbps link. The essential software tool includes iperf which is used to create a client-server relationship and Low Orbit Ion Canon (LOIC)[22]to generate DDoS flooding attack. Using iperf and ping commands to generate legitimate traffic between the host and server, system properties such as response time, throughput and jitter values generated were recorded per seconds for a duration of 15 minutes.

In the attack scenario, assumption was made that attack is from internal source. Hence, compromised hosts within the network were used to launch HTTP, TCP and UDP flood attack on the server for 15 minutes respectively and results recorded from the server.

3.11.2 Data Collection

Machine learning approach for anomaly detection relies on the profile of normal datasets and flag deviations from such profile as attack [124]. The effectiveness of any machine learning algorithm is based on its performance to identify anomalies. This requires a comprehensive dataset that contains normal and abnormal network traffic. As a result of this, the need for data integrity and accuracy to build a robust detection system is imminent. However, available benchmark DDoS attack dataset poses a challenge in

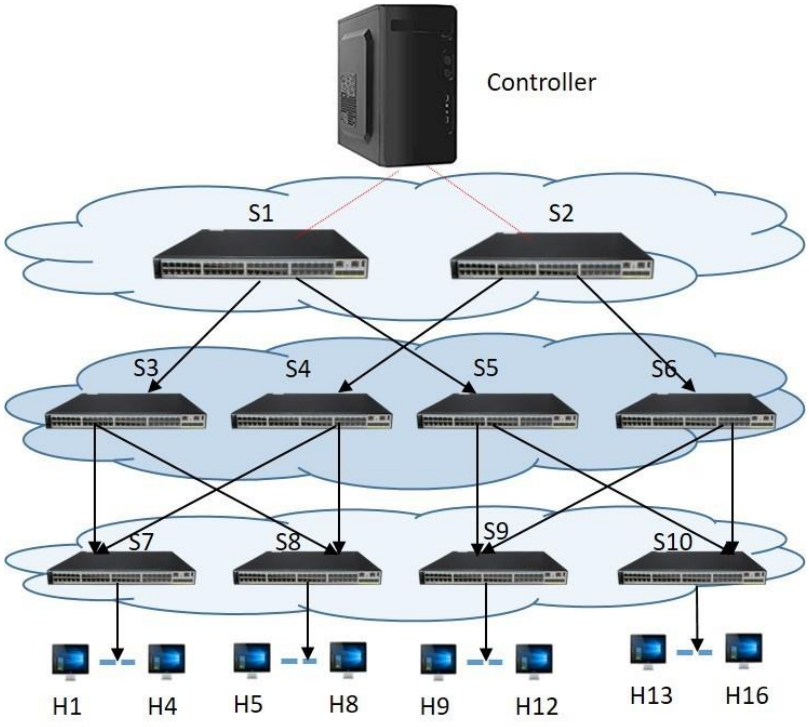

Figure 3.13: Modelled SDN tree architecture

accurately building detection and mitigation algorithm. The KDD99 dataset contains a number of redundant records in the dataset, this in turn often lead to bias in detection results towards the frequent records in training set [86]. Similarly, there exist multiple records which is a factor in changing the nature of the data [87].

A new dataset containing modern DDoS attack in SDN was collected and analysed to overcome the limitation of existing benchmark datasets such as: (1) lack of modern footprint attack fashion in SDN (2) variation in normal traffic recorded with less connected devices few years ago and (3) variation in the attack distribution of old benchmark testing set and new training set [94].

Our collected dataset includes three types of DDoS attack and involves realistic activities of normal traffic that were captured from the server every 15 minutes. The overall data before processing amounts to 1.6 GBytes. Figure 3.14. Shows the data cleaning process employed to arrive at the dataset for classification with a brief overview.

Pre-processing: Network traffic is collected and converted to .txt file

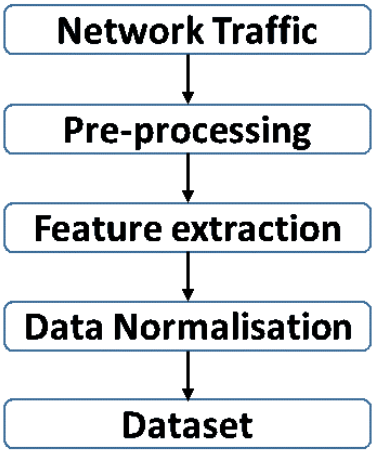

Figure 3.14: Data collection and cleaning steps

Feature extraction: We use Knime to extract basic features (response time, jitter, and throughput) and remove duplicate records

Dataset: Data arranged in row and column format with no missing data.

The identified data integrity challenges form the motivation to create dataset for normal and DDoS attack traffic in SDN environment to evaluate its effect and how accurate attacks can be predicted and classified using ML algorithm.

3.11.3 Classification methodology

Machine learning algorithms iteratively learn from data and automates models to find hidden insight and predict or identify anomaly without being explicitly programmed. The ability to identify deviation in traffic flow makes it suitable for detecting DDoS flooding attack.

There are lots of approaches to machine learning usage in terms of classification and prediction. However, the most important is selecting the model that best fits the training data. Accuracy of the predictive model is important because it determines the quality of resultant predictions and forms the scientific evidence for decision making and policy. Six machine learning algorithm (linear and non-linear) is used in this experiment and are discussed below:

3.11.3.1 Logistic Regression

Logistic regression (LR) is a form of predictive modelling technique which can be used to describe data and explain the statistically significant relationship between dependent variable and independent variable. LR is very efficient and does not require too much computational resources.

Rather than using ordinary least square method to fit model and derive coefficients, LR coefficients are usually estimated using the maximum likelihood method to iteratively fit the model. Hence, LR has low variance and is less prone to overfitting. LR assumes a Gaussian distribution for numeric input variables and is suitable for modelling binary classification problems.

3.11.3.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a statistical technique for binary and multiclass classification with a focus on exploiting the separability among known categories by maximising the distance between means and minimising the variation within each category. LDA makes predictions by modelling the distribution of predictors (set of inputs) separately in each of the response classes and then use Bayes' theorem to estimate the probability. The class with the highest probability is the output class predicted.

One of the advantages of LDA is that it has closed form solutions that are easily computed and no hyper parameters to tune [142].

3.11.3.3 K Nearest Neighbour

K nearest neighbour (KNN) is easy to implement feature similarity supervised learning algorithm that can be used for classification and regression analysis. It makes a general assumption that similar features exist in close proximity. KNN use distance metric to find K similar instances in the training data for new instances and takes the mean outcome of the neighbours as prediction [30].

The accuracy of KNN prediction relies on selecting the right k value. As K increases, predictions become stable due to majority voting up to a certain point without introducing error. Conversely, as the value of K decreases to 1, predictions become unstable.

KNN is robust to noisy training data and learns complex model easily [9]. The main drawback of KNN is the complexity in searching the nearest neighbours for each sample (high computation cost).

3.11.3.4 Naive Bayes

Na¨ıve Bayes calculates the prior probability of each class and the conditional probability of each class given each input value. In Bayesian analysis, these probabilities are estimated for new data and the final classification is produced by multiplying the prior probability with conditional probability [105].

Na¨ıve Bayes can handle an arbitrary number of independent variables which makes it suitable in classifying data with high inputs dimensionality. In analysing real-valued continuous data, Gaussian distribution may be assumed to estimate the probability of input variables using the Gaussian Probability Density Function (PDF).

Although Na¨ıve Bayes is easy to implement, it suffers from the general assumption that predictors are independent which is almost impossible to get in real life cases [26].

3.11.3.5 Classification and Regression Tree

Classification and regression tree (CART) is a decision tree algorithm proposed by Breiman et al. for predicting continuous dependent variables and categorical predictor variables [17]. The models are obtained by recursively partitioning two child nodes, beginning with the root node that contains the whole learning sample and fitting a simple prediction model within each partition [81].

Split points are chosen by evaluating each attribute and value in the training data in order to minimise the cost metric. In CART, prediction error is typically measured by the squared difference between the observed and predicted values.

CART gives comprehensive information and all possible solutions based on the dataset. However, variations in input data can at times cause large changes in the tree or possibly require redrawing the tree.

3.11.3.6 Support Vector Machine

Support vector machin (SVM) is a supervised machine learning technique for both regression and classification tasks. The objective of SVM is to choose hyperplane

in an N-dimensional space (where N – represents number of features) that optimally differentiate classes with the maximum margin. Hyperplanes are decision boundaries that help classify data points. Data points closest to the hyperplanes are called support vectors and they influence orientation and positioning of the hyperplane [131].

SVM is capable of solving problems with nonlinear decision boundaries and has been extended to support multiple classes using different kernel functions such as radial basis, Gaussian and polynomial function. SVM can be subject to overfitting if the kernel parameters are not carefully tweaked.

All the discussed machine learning techniques are implemented using Scikit-learn platform [106].

3.12 Result and Discussions

3.12.1 Evaluation metrics

To evaluate the performance of the six ML algorithm applied on the obtained dataset, we use primary performance indicators such as accuracy, precision and recall values. The performance indicators are calculated using four different measures, True Positives (TP), True Negative (TN), False Positive (FP) and False Negative (FN):

- TP: outcome where model correctly predicts positive class
- TN: outcome where model correctly predicts negative class
- FP: outcome where model predicts positive class wrongly
- FN: outcome where model predicts negative class wrongly

Each ML technique has its unique characteristics to learn, predict and evaluate data points to classify and detect attacks based on the applied tuning parameters.

Accuracy: Accuracy represents the total number of correct predictions divided by the total number of cases. It can be represented mathematically as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.3}$$

Precision: Precision is the total number of True Positives (TP) divided by total number of predicted positives.

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

Recall or sensitivity: Recall is the proportion of correct positive classification (TP) from the total number of actual positives.

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

Recall and precision is a more stringent form of how good a classification algorithm is.

3.12.2 Experimental results

3.12.2.1 Attributes correlation matrix

Correlation between each pair of attributes gives an indication of how related the changes between them are. Figure 3.15, shows how changes between throughput, jitter and response time are related. It is worth noting that each variable is positively correlated with each other as seen in the diagonal line from top left to bottom right.

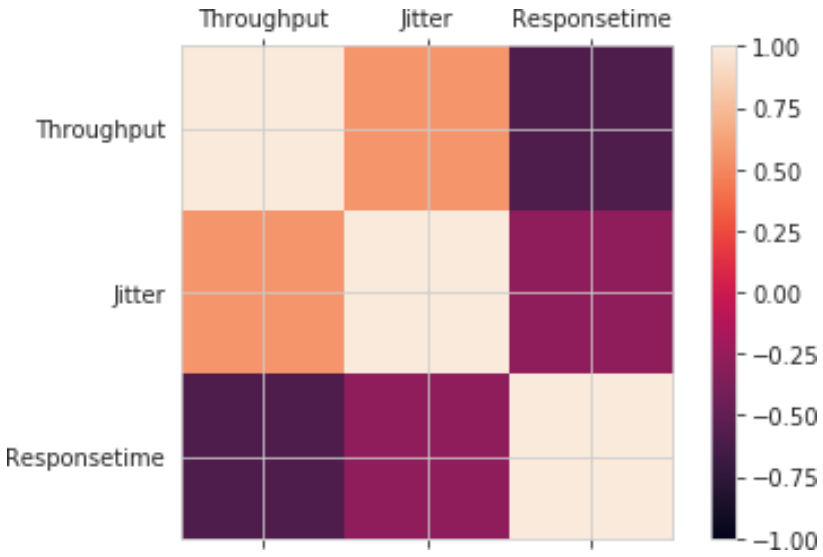

Figure 3.15: Correlation matrix plot of throughput, jitter and response time

3.12.2.2 Prediction accuracy

To evaluate the complexity of the dataset in terms of accuracy, six ML algorithms (LDA, LR, KNN, SVM, NB and CART) are employed. The algorithms are implemented in Scikit-learn. Using k-fold cross validation, we set values of k = 3 , 5, and 10 to evaluate the performance of the selected ML algorithms. Figure 3.16 represents the comparison between the 3, 5 and 10 fold cross validation. For all the algorithm, 10-fold cross validation gave satisfactory accuracy. Hence, we use 10-fold cross validation for training and testing our dataset.

Figure 3.17 shows the box plot comparison of the algorithms. The CART algorithm achieved the highest accuracy (i.e. 98.47%) followed by KNN with 97.01% and the lowest was LDA with an accuracy of 96.07%.

Overall, the results of the accuracy of these algorithms show that CART can detect UDP, TCP and HTTP flood attack from normal (attack free) SDN traffic flow having a small number of features with promising accuracy.

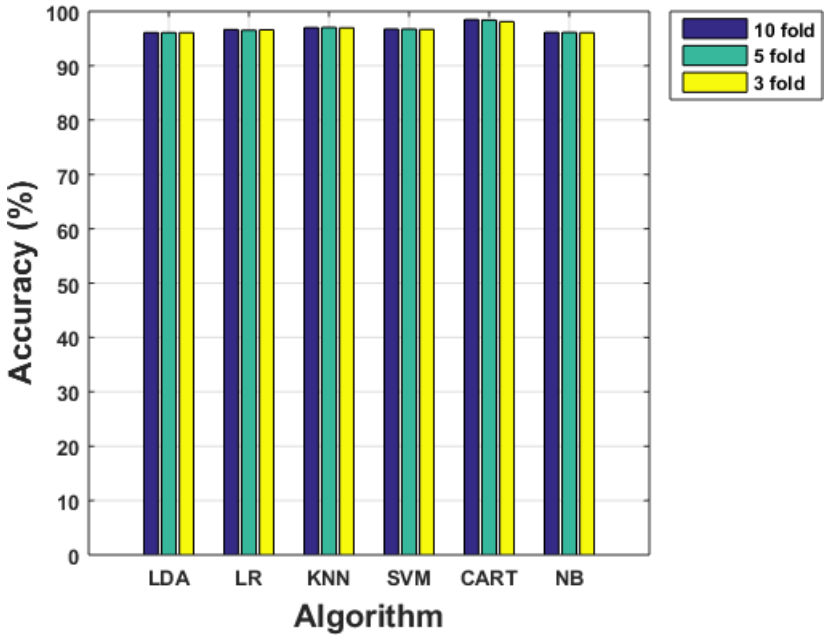

Figure 3.16: Algorithm prediction accuracy for k = 3, 5 and 10

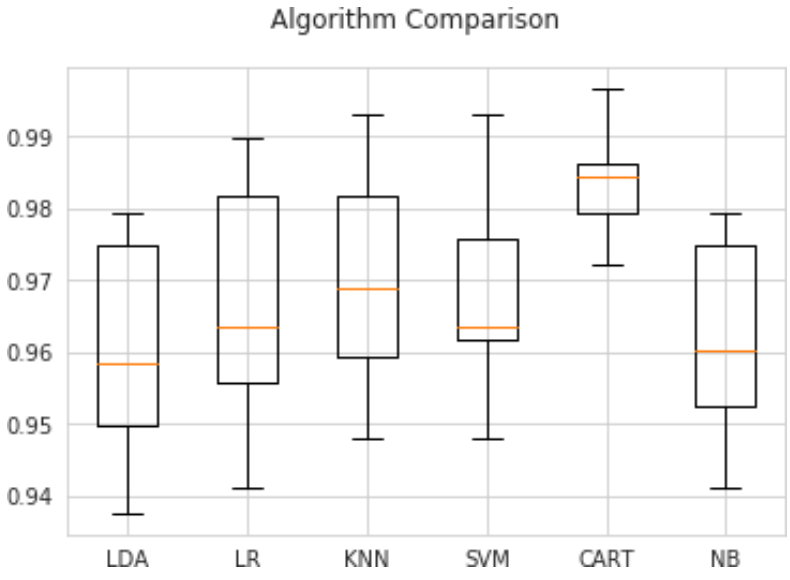

Figure 3.17: Box plot comparing algorithm performance

3.12.2.3 Precision and Recall

Relying on only accuracy may not be sufficient in selecting the optimum classifier especially for real time DDoS attack data. Hence, the need for precision and recall. Figure 3.18 and 3.19 shows the precision and recall comparison of the six ML algorithms for each class of attack. With the exception of LDA and LR, all other algorithms achieved 100% precision in identifying normal traffic. Similarly, all algorithm had perfect precision for UDP flood except NB which is slightly lower. CART also achieve high precision and recall rate in classifying other three classes of attack.

It is worthy of note that all classifiers performed well with high precision and recall values. In general, CART produced high accuracy, good precision and recall rate more than other classifiers. Hence, a better classifier for detecting flooding DDoS attack in SDN with promising performance results.

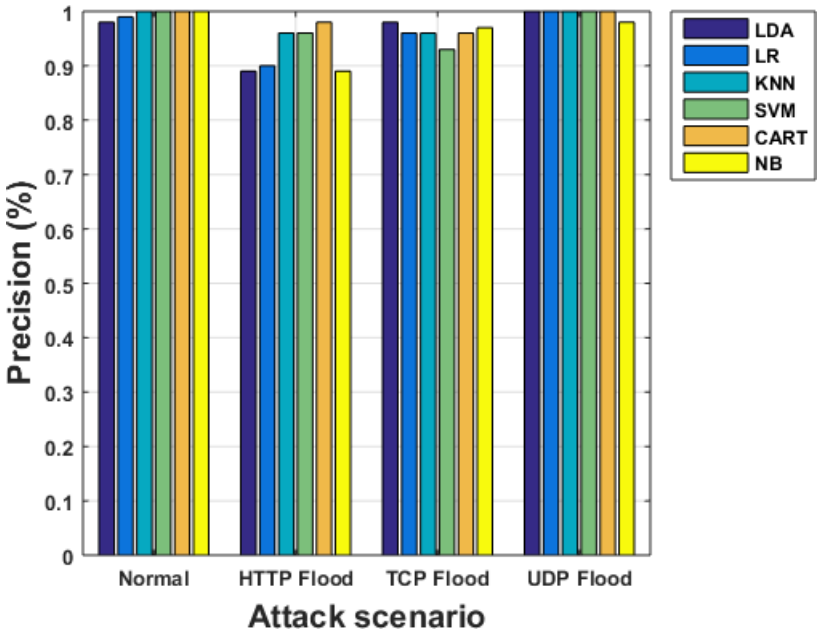

Figure 3.18: Performance of classifier in terms of precision

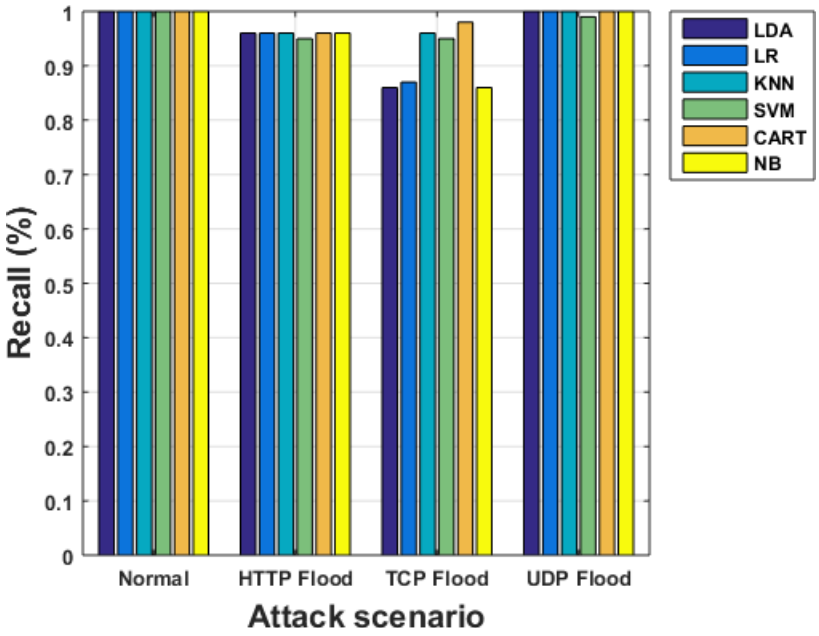

Figure 3.19: Performance of classifier in terms of recall

3.13 Summary

In this section, effect of DDoS attack on SDN is examined. Related work on attack detection in both traditional network and SDN is reviewed. Firstly, custom topology is designed using mininet and three DDoS(SYN, ACK and slowloris) attack launched on the server. A statistical approach is designed to detect attacks and attack window size of 1 minute gave more accurate detection accuracy. Similarly, tree topology is designed and machine learning based approach is applied to the extracted traffic. The models were validated using six machine learning algorithm and CART algorithm achieved high detection accuracy of 98.47% with only three features namely: throughput, jitter and response time when compared to other machine learning algorithm deployed. Hence, CART exhibits strong potential DDoS flooding attack detection in SDN environments.

Chapter 4

Reconnaissance, Attack Launch and Mitigation

4.1 Introduction

In today's rapidly evolving network, attackers are often one step ahead. A good means of classifying security attacks is in terms of passive and active attacks. Passive attacks attempt to learn or make use of the system without affecting the host resources while active attacks alter resources, system operations or shut it down completely in severe case. Hence, protection of networks and their services from unauthorised access, destruction, modification and disclosure becomes imminent. Proactive detection and prevention mechanisms can help keep activities of malicious users in check.

4.2 Understanding Network Environment

The battle for a secure network against DDoS attack remains a daunting task to which network security specialists must always be at alert. To stay one step ahead of malicious users, it is necessary to have a good understanding of the network environment. Unfortunately, due to financial incentives and readily accessible DDoS attack code with stable releases over the years, malicious users are launching variants of DDoS attack and exploiting vulnerabilities in networks. As a result, the race seems unending and open avenues for innovation in securing networks. Having a good understanding

of the network environment simplifies the attack process for malicious users. Getting to know the environment and launching an attack can be broken down into four steps [45]:

- Information gathering: this stage involves sourcing information publicly available using social engineering to gain information about target. The drawback of this stage surprisingly is having too little data or too much information to make sense out of the exercise. Data gathered in this stage can be IP address, software version, network equipment manufacturer and types of protocol running on network devices. This stage is often referred to as footprinting as well.

- Scanning: scanning involves utilising data obtained from footprinting to locate active hosts. The first activity involves port scanning for open ports and potential service running on it. Port scanning is then followed by vulnerability scan to identify specific weaknesses and services and software of targets.

- Exploitation: vulnerabilities detected during scanning is exploited to gain access to the network where attack is launched. Depending on how vulnerable the network is, guest account privilege can be escalated here to full administrative right to take charge of the network resources.

- Covering tracks: Malicious users try to eradicate evidence of gaining access to targets. This is done by deleting system log files created during the hacking phase to prevent being noticed or detected by intrusion system in the network. In some cases, malicious users often plant backdoor in order to gain access without going through the initial stages.

The enumerated steps can be performed using Linux distributions such as Kali Linux or Backtrack. In this experiment, Kali Linux is the tool of choice.

4.2.1 Available Attack tools for gathering information

Several penetration testing procedures can be observed to assess the overall security of an organisation's system and to locate network/system vulnerabilities. Table 4.1 and 4.2 shows open source web and DDoS attack tools that can be used to gather information from network and launch attack [13].

Table 4.1: Open source tools to gather information

Tools	Definition
NMAP	Free security scanner showing which hosts are available on the network and creates a network map of hosts
BRUTUS	Flexible remote password cracker utilities that guess passwords by using dictionary and permutations thereof
NESSUS	A vulnerability scanner that provides ability to audit network, produce a list of vulnerabilities and compliant security tools
Google hacking	Provides advance operators to query and obtain data using ever-widening reach of Google search engine

Table 4.2: Open source DDoS Attack tools

Tools	Definition
LOIC	Open source stress tool to flood server with TCP packets or UDP packets to disrupt service
HULK	Open source Web server DDoS attack tool
Mirai	Malware tool to infect IoT devices to launch DDoS attack
Net-Weave	Booter written in .NET with USB spreading capabilities to launch TCP and UDP flood

The lists in Table 4.1 and 4.2 are by no means exhaustive. New tools are being developed to scan and secure network on a rolling basis.

4.3 Model Formalisation

DDoS presents a form of security attack where multiple attacks from different locations in the network target a victim. Since DDoS attack exhausts the resources of the victim e.g. server, the server refuses new connection from legitimate hosts. It is worth stating that the server resource exhaustion could be bandwidth or buffer size of the victim server. In our architecture, a single controller manages 10 switches and 16 hosts respectively using OpenFlow protocol [127].

Let the transfer of information between the switches in the network with respect to

link path be represented mathematically as follows:

$$T_{switch} = C_i.[S_1L_1 + S_2L_2 + S_3L_3 + ... + S_{10}L_{10}] \quad (4.1)$$

where C_i = no of controllers in the network
assuming equal link i.e. $L_1 = L_2 ... = L_{10}$ *eqn* (1) *becomes*

$$T_{switch} = C_iL \sum_{i=1}^{10} S_i \quad (4.2)$$

Similarly,

$$T_{host} = C_i.[H_1J_1 + H_2J_2 + H_3J_3 + ... + H_{16}J_{16}] \quad (4.3)$$

where link $J_1 = J_2 = J_{16}$

$$T_{host} = C_iJ \sum_{i=1}^{16} H_i \quad (4.4)$$

Combining eqn. (4) and (2) gives total flow of information in the network as:

$$T_{network} = T_{switch} + T_{host} \quad (4.5)$$

$$T_{network} = C_iL \sum_{i=1}^{10} S_i + C_iJ \sum_{i=1}^{16} H_i \quad (4.6)$$

Number of controller in the network = 1, therefore $C_i = 1$

$$T_{network} = L \sum_{i=1}^{10} S_i + J \sum_{i=1}^{16} H_i \quad (4.7)$$

Table 4.3: model parameters description

Notation	Meaning
C_i	No of controllers in the network
H_i	No of hosts present in the network
S_i	No of switches present in the network
L_i	Link between switches
J_i	link between host
T_{switch}	Transmission of information between switches
T_{host}	Transmission of information between hosts
$T_{network}$	Transmission of information within the network

4.3.1 System Architecture

The choice of modelling complexity of a network depend on the amount of information to be conveyed, the medium of transmission and the depth and width of the interface (level of interaction) of the system. Each of these is interrelated in common network topologies available today. This also applies to SDN. For this experiment, a Fat-Tree topology is designed in mininet [90]. Mininet is an open source network emulator devoted entirely to OpenFlow architecture and SDN implementation. It can create a realistic virtual network, running a real Linux kernel, open Vswitch and application code. Star, torus, linear and other custom topologies can be designed using mininet. However, Fat-tree topology is chosen for this experiment as it conforms to practical hierarchical core, distribution and access layer recommended network design. Fat-tree is one of the most widely-used network topologies. It is a combination of two or more star networks connected together with a bus and it supports future extension of network. Fat-tree topology is ideal for workstations located in groups.

In Figure. 4.1, the controller is responsible for providing all of the switches with the information needed to populate their Ternary Content Addressable Memory (TCAM) table. Hence, packet forwarding problems associated with loops would be avoided. The SDN controller updates each of the OpenFlow switches in the network using OpenFlow protocol with the content of the flow table created by the controller. As a result, the controller has a global view of the network and set of matching instructions also referred to as flow can be pushed to OpenFlow switches simultaneously to

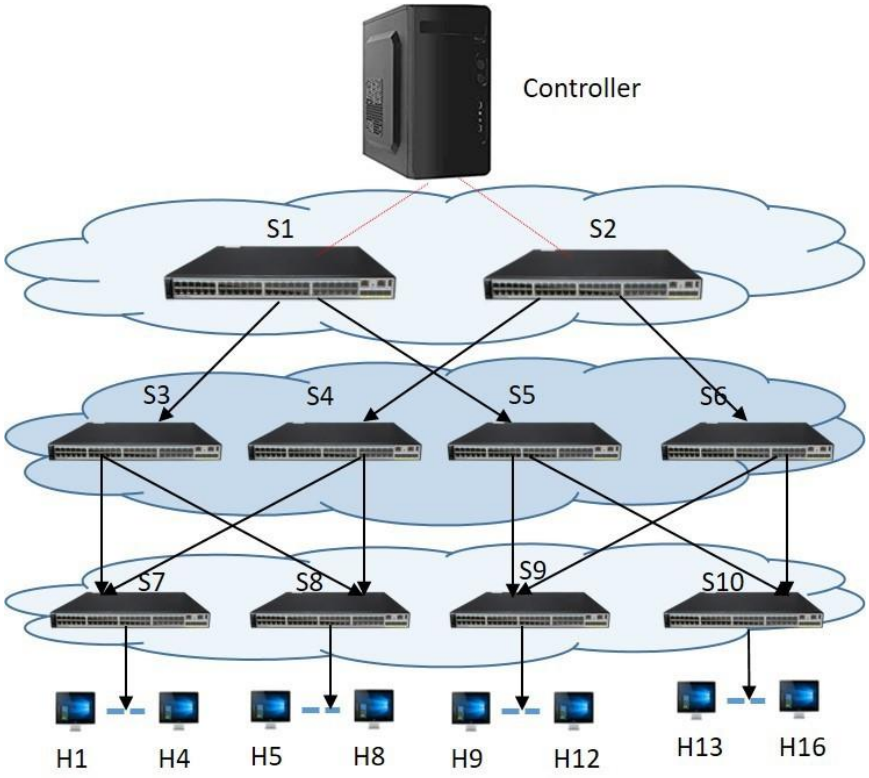

Figure 4.1: Fat-tree network topology

mitigate attack or enhance system performance.

Due to the separation of the control and data plane, there exists negligible latency in the signalling protocol. By the time the controller collects statistics from OpenFlow switch, the values within the message are most likely out of date based on polling interval and it may not reflect the real-time state of the switch anymore. Fortunately, for many applications, this slight inaccuracy is tolerable or negligible. These little but negligible delay in statistics collection for use in real-time is as a result of the separation of the control plane from the data plane. Although this can be more pronounced for a controller that is geographically distant from data plane devices with large link latency. This latency will have to be put into consideration for any reactive algorithm to be placed in the controller that relies on statistics collection. Figure. 4.2 Shows the global view of the network from floodlight controller perspective.

The custom fat-tree topology consists of 10 OpenFlow switches connected hierarchically. At the distribution layer, 4 hosts are connected to each OpenFlow switch

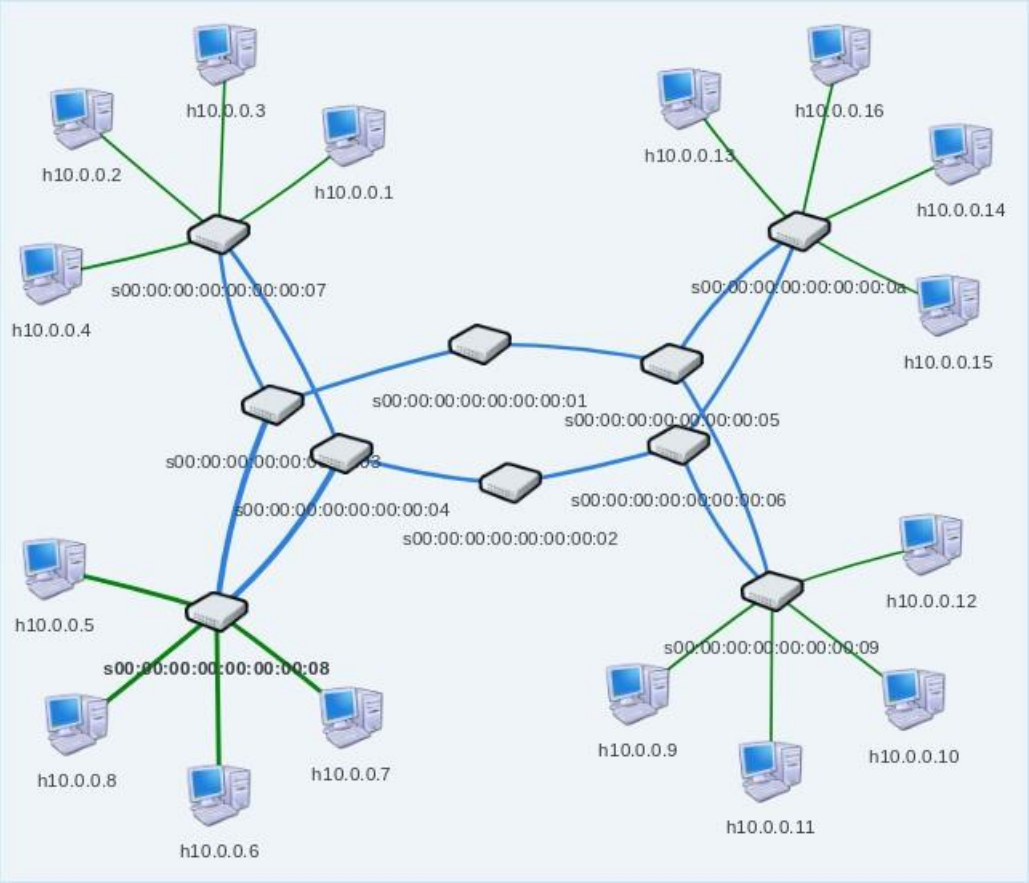

Figure 4.2: Global view of network from controller perspective

making a total of 16 hosts in the network. Each host is tested to ensure connectivity using ‘pingall’ command and the connections were successful.

4.3.2 Experimental Approach

The simulation approach is presented in four stages as shown below:

- **Stage 1**: Create network topology The network topology shown in Fig. 4.1 is designed using mininet emulator and connectivity is established using the following commands:

```
Sudo mn --controller=remote,ip=192.168.1.4

--topo=tree,depth=2,fanout=3--link tc,bw=10

   >net

   >pingall
```

- **Stage 2**: Scan the network using NMAP In this stage, information about the number of active network devices is gathered. Information such as target IP address, open port, device e.t.c. were obtained using intense scan command.

```
nmap -p 1 65535 -T4 -A -v 10.0.0.1/24
```

- **Stage 3**: Launch DDoS attack using LOIC Using one of the hosts to generate constant traffic to and from the server, throughput, jitter, and response time values were measured per second. LOIC was used to launch TCP and UDP flooding attack and CPU utilisation from the controller (CPUT) and server (SCPUT) recorded before the attack (normal scenario) and during an attack.

- **Stage 4**: Mitigate attack In this stage, the controller pushes out reactive block flow based on a predefined countermeasure algorithm to mitigate the flooding attack. Server (SCPUT mitigation) and controller (CPUT mitigation) utilisation values before an attack (without attack) and during an attack (with attack) were recorded. Similarly, the jitter values before the attack (UDPB), during an attack (UDPD) and mitigation(UDPM) were recorded and discussed in section 4.5

4.3.2.1 Hardware and Software Settings

The specific software and hardware configuration information are provided in Table 4.4. All experiments were conducted on high performance PC with adequate computational capabilities. The computer has 32 GB memory, 4 cores Intel Xeon E1234. Kali Linux is installed as the base OS. Oracle VirtualBox is also installed which runs Ubuntu 18.04 version for the floodlight controller [49]. Other software utilised in this experiment are open source as shown in Table 4.4.

Table 4.4: Simulation parameters descriptions

Software and hardware	Specification
CPU	Intel Xeon CPU E3-1220 V3 @3.10Ghz
Memory	32G RAM 500Gb HDD
Kali Linux	Kali-rolling
Oracle virtual box	Virtual environment for simulation
LOIC(Low Orbit Ion Canon)	V 1.0.8
Ubuntu	Bionic V18.04.1 LTS
Mininet Emulator	V 2.2.2

4.4 Active Reconnaissance, Attack Strategy and Countermeasure

In this section, we present the three stage approach followed in gathering information from the network, launching an attack on the network and the associated countermeasure taken to mitigate DDoS attack in the network.

Figure. 4.3 shows the methodology flow diagram used in this model. The attack module represents the active reconnaissance stage where specific information about the network is gathered. The information gathered at this stage is modified and utilised to attack the system. In the controller module, system performance is monitored for abnormal behaviour and a routine is called in the controller to dynamically mitigate attack and restore the system to working order.

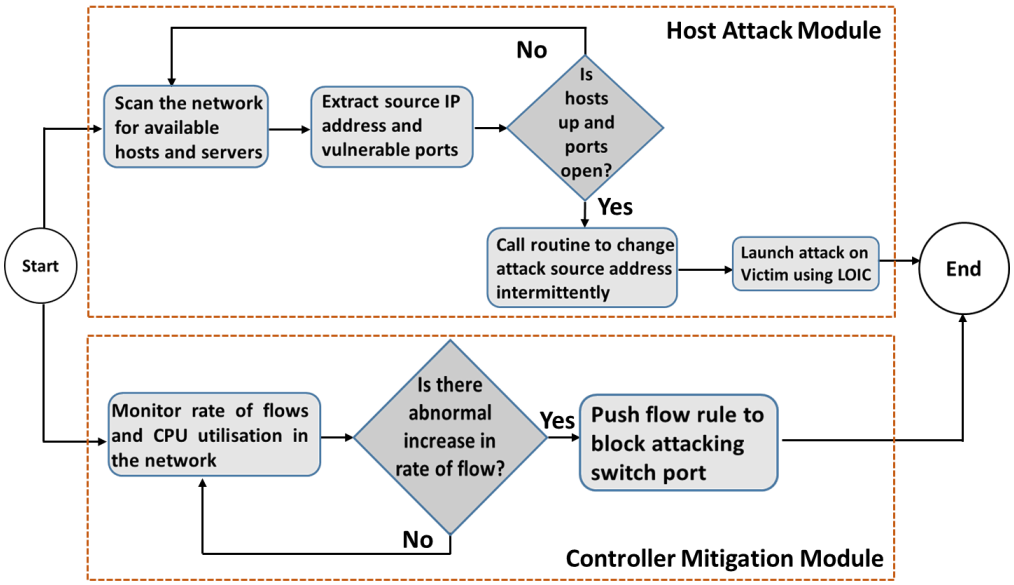

Figure 4.3: Methodology flow chart of the reconnaissance and countermeasure

4.4.1 Active Reconnaissance

Information gathering is an essential step needed to gain access to a network. It involves knowing which information is useful for launching an attack and how to extract it through reconnaissance. One of such information is details of where the targets IP address starts and stop. For this experiment, Zenmap [101] is the network mapper run to see the number of active hosts on the network and the vulnerability scanner to determine the ports open in the network. Zenmap is an open source multi-platform Nmap security scanner with a graphical user interface capable of scanning large networks fast from a single host. As shown in Figure. (4.4-4.5), it can be observed that there are 16 active hosts on the network with IP ranging from 10.0.0.1 to 10.0.0.16 and a clue of the network topology is also provided. The intense scan performed also revealed open port 5566 on the server with 10.0.0.10 IP address. The gathered piece of information paved the way for the attack strategy deployed in stage 2.

4.4.2 Attack Strategy

The known fact about new packets coming to the controller is that the destination hosts/server is within or logically connected to the network of the controller. In this

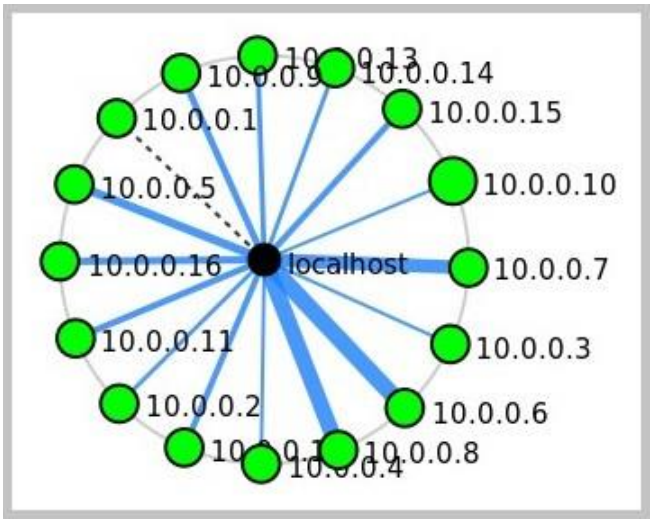

Figure 4.4: Zenmap view of network topology

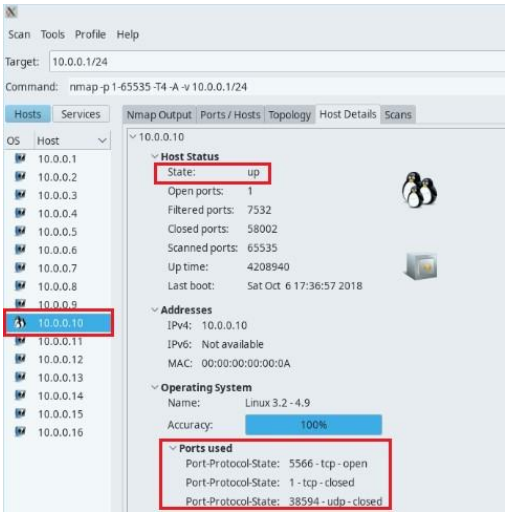

Figure 4.5: Nmap output with details of server open port

experiment, four key assumptions are made:

- 1. Not every host in the network is the attack target
- 2. The attack is orchestrated through an internal network
- 3. During an attack, the volume of attack traffic is much higher than legitimate traffic

- 4. There is a continuous flow of traffic to the server

The assumptions are made to represent what is obtainable in the real day-to-day network activities and attack scenarios. Depending on the nature of the attack, high rate DDoS attacks often have more traffic than legitimate traffic in order to exhaust network resources and render it unavailable. As shown in Figure. 4.1, the attacker in the SDN environment could be a host or a compromised switch. In this experiment, hosts in the network have been used to launch attack on victim's server connected to switch 9. These attacks have been launched using Low Orbit Ion Canon (LOIC) [22]. LOIC is an open source DDoS attack application with a GUI responsible for sending garbage TCP/UDP or HTTP flood requests directed towards victim's server on selected port. This attack is further intensified by running an application that changes attack source IP address within the range of IP address that has been spoofed during the active reconnaissance phase.

4.4.3 Countermeasure

The best way to mitigate any DDoS attack is to prevent the attack from being launched in the first place. There exists two main approach to mitigating DDoS attack; preventive and reactive approach. Due to the centralisation of the SDN controller, it is easy to effectively monitor network health and proactively reacts to anomalies based on the detection system in place. The floodlight controller operates reactive rule insertion by default. The controller monitors packet using packet-in messages and a static flow pusher is used to create a flow proactively prior to malicious packets reaching the OpenFlow switch. This flow pusher is accessible through the REST API and the defined JSON string entry is then added to the controller using an HTTP POST command.

Algorithm 1 Countermeasure Algorithm

Procedure Start
monitor packet in on controller port 6653
for *each Switch in Network* ***do***
collect the network statistics
Calculate average packet in value @ time $t_1 = X_1$
Calculate average packet in value @ time $t_2 = X_2$
if $X_2 > X_1$
push flow to block attacking switch port
Print log of rule insertion
else *continue*
end if
end for
end procedure

4.5 Result and Analysis

In this section, we present and analyse the results as follows:

4.5.1 Effect of DDoS Attack on System Response time

Before launching an attack, based on the assumptions stated in section 4.4.2 that there exists a continuous flow of traffic to the server. Hence, the need to evaluate the average response time of the system before and during an attack. The ICMP error reporting protocol consists of echo requests and echo reply to monitor network problems preventing delivery of IP packets. Round Trip Time (RTT) is the time it takes for a data packet to be sent to a destination plus the time it takes for an acknowledgement of that packet to be received back at the source. The min response time is the fastest time it takes to get a response at the source while the maximum response time represents the time that took the most. The average response time is the sum of all the RTT values found divided by the total number of RTT samples. Using Ping command, 20 probes were sent to the server and the average response time recorded for five scenarios. All packets sent were received. Similarly, an attack was launched and the average response times during the attack period for 5 scenarios are evaluated as shown in Figure. 4.6 For each scenario, the minimum response time before an attack is completely negligible

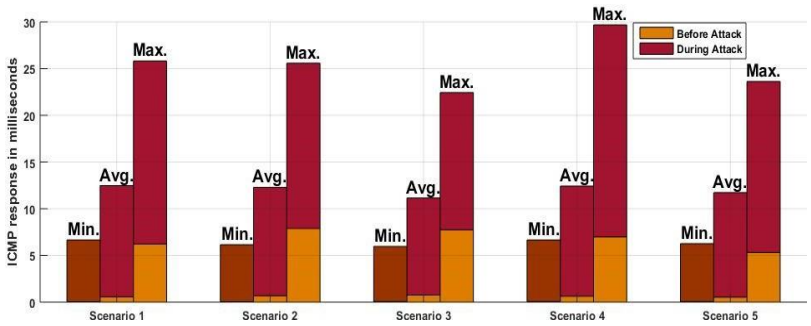

Figure 4.6: Average response time from server before and after attack

compared to minimum response time during an attack. The maximum response time before an attack did not exceed 7.9 milliseconds. This trend is completely different for attack scenario as it reaches a maximum response time of 28 milliseconds. In all the scenarios considered, the response time increased by over 100 percent during attack. Thus, it is evident that DDoS attack can severely impact response time of server within a short period and can result in packet loss or render the server completely unavailable to legitimate users.

4.5.2 Computational Resource Consumption

Figure 4.7 4.8 4.9 4.10 shows the resource consumption rate before an attack, during attack and mitigation. There is no big difference between the controller CPU utilisation and server utilisation before the attack. Utilisation fluctuations are around 8% and 60% on average respectively. Because of the high number of flow during attack, the CPU utilisation quickly reaches a peak of 62% in less than 80 seconds of attack launch and reaches 80% for the victim server respectively. During Mitigation, the utilisation rose by 20% and 10% above the attack utilisation threshold for controller and server respectively. The overall controller utilisation hovers around 8% and 60% for the server and controller when completely mitigated. There is a strong correlation between the normal controller/server utilisation before attack and during mitigation. Hence, low performance overhead introduced as a result of pushing reactive flow to mitigate the flooding attack.

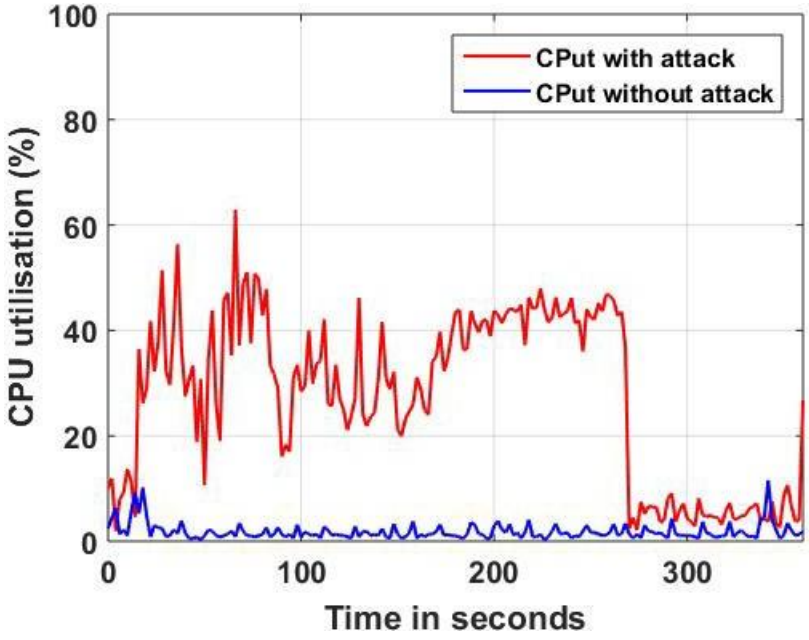

Figure 4.7: Controller CPU utilisation under TCP-based attack

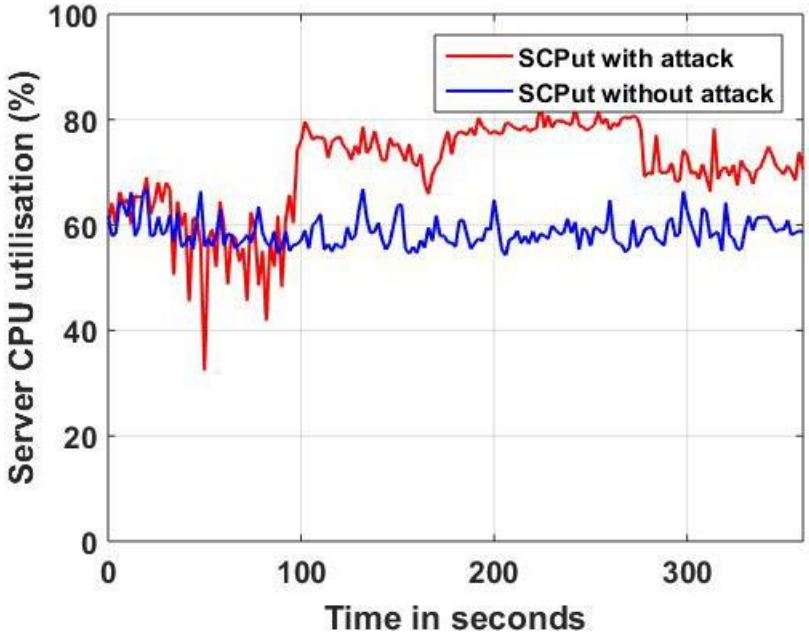

Figure 4.8: Server CPU utilisation under TCP-based attack

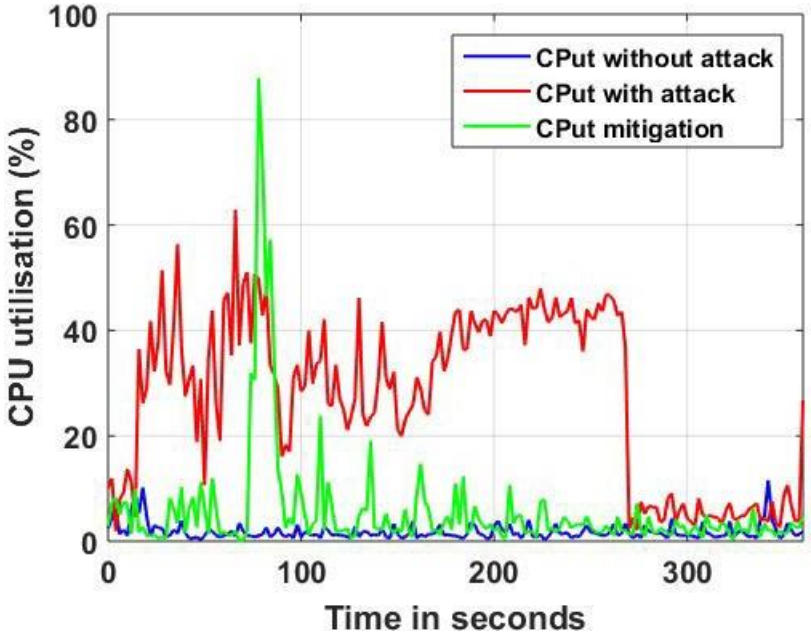

Figure 4.9: Controller CPU utilisation under TCP-based attack

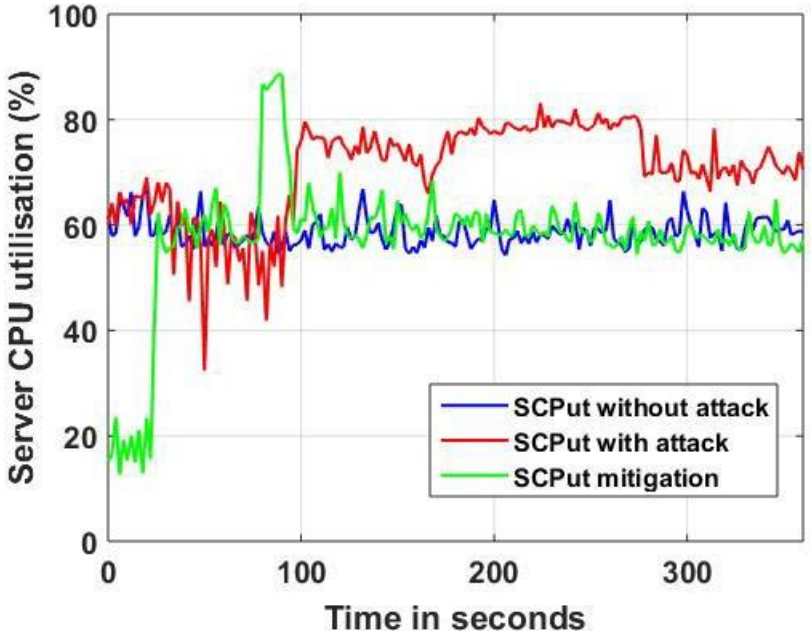

Figure 4.10: Server CPU utilisation under TCP-based attack

4.5.3 Effect of DDoS Attack on Packet Count

From the above assumptions made that there is a continuous flow of traffic to the server, it is evident that rate of packet-in and out at any point in time varies and it is not equal to zero. The rate of flow is seen to be increasing exponentially in Figure 4.11 during the flooding and spoofing attack. The rate of change of packet-in message is monitored for abnormality by the controller and information about the server switch port is kept. A large variation in the value of packet-in message triggers the flow-pusher module to block the attacking source based on predefined rules.

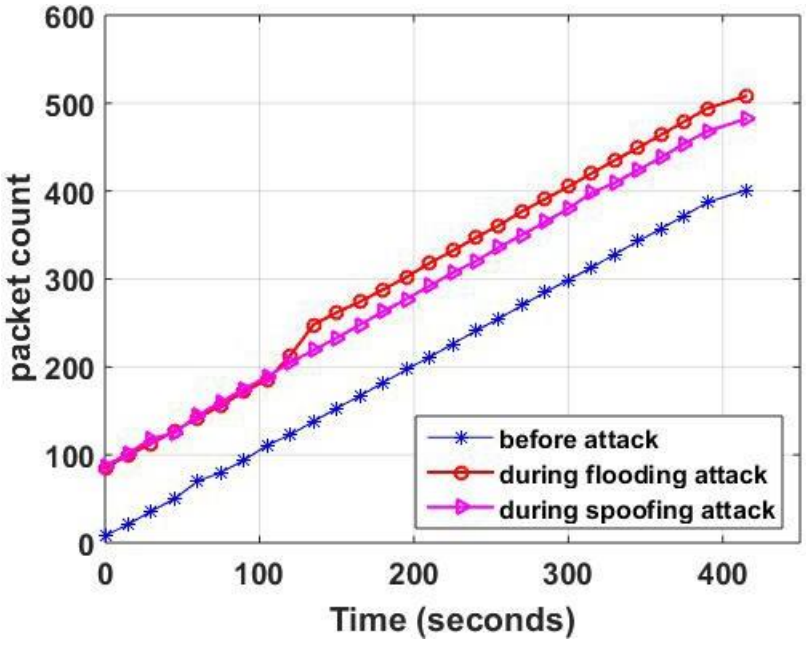

Figure 4.11: Packet count before and during DDoS attack

4.5.4 Effect of DDoS Attack on Jitter

Jitter represents the variation in the delay of received packets. In Figure. 4.12, using a buffer size of 208kbytes, the jitter varies between 0.003ms and 0.015ms before attack. Spiky delay waveform is seen during attack and mitigation phase. The spikes indicate congestion in the network. The congestion window during the flooding attack is large and this will lead to packet drops if the congestion time is more than the packet transmission time. This effect can be severe for a voice application running in SDN during attack.

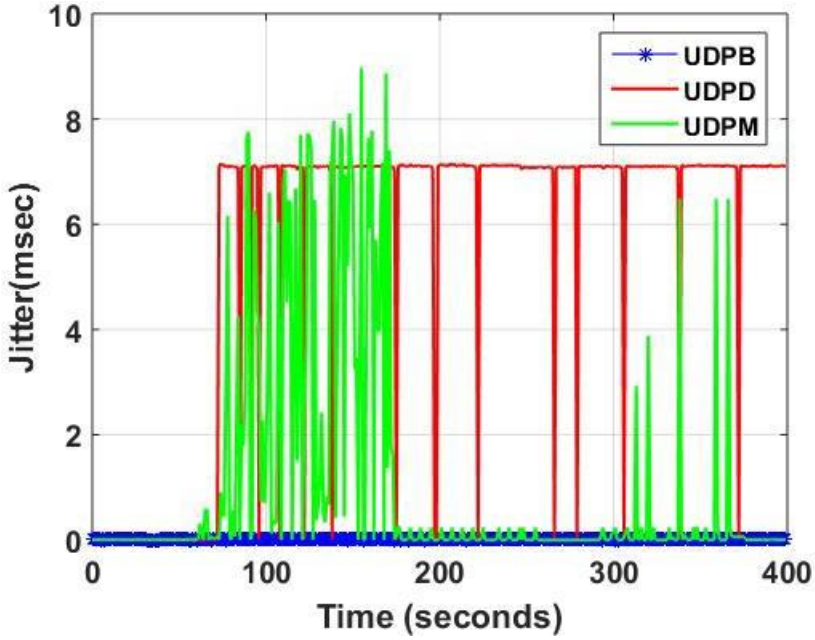

Figure 4.12: DDoS attack effect on Jitter

4.5.5 Effect of DDoS Attack on Throughput

Figure. 4.13 shows a significant drop in throughput value due to the impact of flooding attack on the server. The average throughput for requests made from hosts within the network to the server before the attack is 42 Gbps. TCPB, TCPD and TCPM represent throughput before, during and after attack mitigation respectively. The impact of the attack is felt barely 55 seconds after the flooding attack launch and the server was rendered unavailable for the rest of the transmission. However, when the controller pushes block flow to the switch port connected to the attacking host in the data plane, the server regained its capacity and the network is restored within a short while.

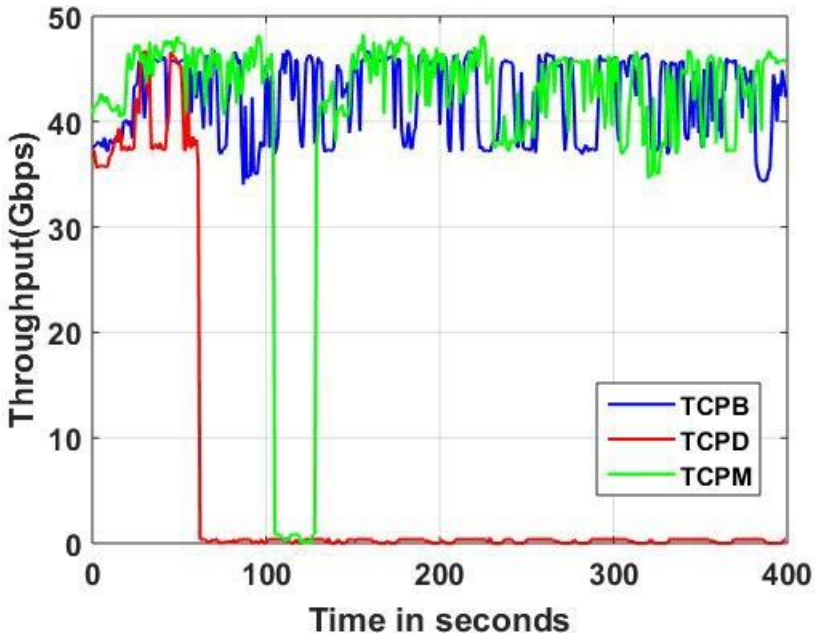

Figure 4.13: Effect of DDoS attack on server throughput

4.6 Summary

In this chapter, active and passive information gathering to gain understanding into network topology is examined. The network topology is constructed from scanning the network and information gathered from the open port is used to launch UDP and TCP flooding attack. As a result, the effect of UDP and TCP flood DDoS attack on SDN has been demonstrated. This study reveals that existing controller and data plane devices are prone to flooding and spoofing attack which degrades network performance within a short while. It also indicates that these attacks can be mitigated by pushing blocking flows from the controller to the attacking switch. Our evaluation shows that additional flow rule insertion to mitigate DDoS flooding attack imposes minimal overhead in terms of CPU utilisation on the controller and server.

Chapter 5

Sensitivity Analysis of Detection Parameters

5.1 Introduction

In the last decade, existing works in the literature and industrial collaboration in the subject of SDN implementation indicates that the wide adoption of the technology is not far from reach. Realistic models and methodologies for understanding network traffic behaviour play an important role in facilitating efficient DDoS attack detection and mitigation. Using simulation data to describe dynamic network traffic characteristic and detect DDoS attacks plays a larger role than traditional mathematical techniques have played in the past. Application of machine learning to network traffic characterisation has made it more realistic to mimic network traffic pattern and develop a robust model against security vulnerabilities for multiple reasons. A primary reason is the reduction of costly investment in data monitoring tools for day-to-day traffic analysis and performance overhead introduced.

Sensitivity analysis is an ad hoc analysis that relies on historical data. Information gathered by network administrators and designers helps in planning and responding to threat to input network parameters deemed sensitive. The full global analysis of all the historical network parameters gathered to monitor and detect DDoS attack can be computationally expensive and it may introduce delay in end-to-end communication if implemented on enterprise network. Hence the need to select a subset of parameters

that seemed likely to have strong effects in the detection of anomaly in network traffic. One-at-a-time local sensitivity analysis (LSA) technique analyses the impact of a single parameter on the cost function at a time, keeping the other parameters fixed and is fast to compute.

5.2 Sensitivity Analysis

Anomaly detection techniques in networks rely on the assumption that variations in network parameters may have some effect on the state of network performance when under attack. Several network features such as time-to-live (TTL), throughput, end-to-end delay, packet drops amongst others are considered input variables to assess the robustness of detection and to ensure appropriate mitigation technique is deployed. This approach is memory-intensive and can increase the computation time coupled with the purchase of a high-performance computing device.

Sensitivity analysis involves the estimation of uncertainty in the output of a model as a result of different sources of uncertainty in the input[116]. Figure5.1 illustrates the basic representation of the relationship between input parameters and output response.

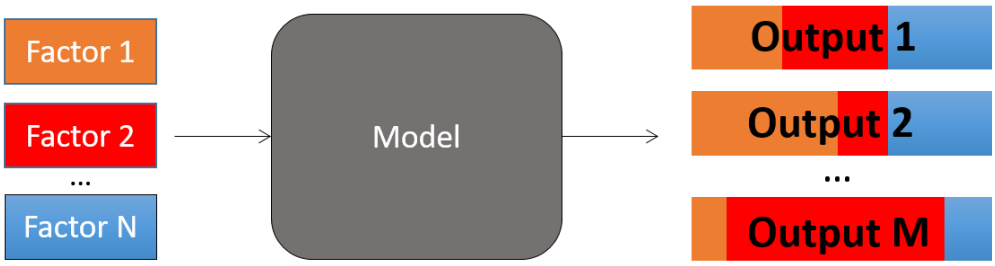

Figure 5.1: Sensitivity analysis of relationship between input and output response

Sensitivity analysis offers an efficient approach to assess extent to which detection results are affected by changes in input network variables. In this context, sensitivity analysis is aimed at priority setting, to identify the key variables that are major influence in predicting whether the network is under attack or secure. As a result, sensitivity analysis provides an understanding of cause and effect reaction between changes in input variables and the corresponding output.

One of the key advantages of sensitivity analysis is that it identifies critical variables that may be given less consideration when designing a robust detection model.

5.2.1 Types of Sensitivity Analysis

When one or multiple inputs have relatively insignificant sensitivity as compared to others, the overall dimension of the neural network for training can be reduced by removing them and a smaller size neural network can be successfully retrained to develop a more efficient model.

5.2.1.1 Local Sensitivity Analysis

Local Sensitivity Analysis (LSA) is the assessment of the local impact of input factors' variation on a model response by concentrating on the sensitivity in the vicinity of a set of factor values [147] [149]. In local sensitivity analysis, the values of other input parameters are kept constant when studying how sensitive an input factor is.

LSA evaluates sensitivity for a single deterministic set of input parameters which is often based on the partial derivatives of the response with respect to the input parameters [62] [73]. Given the model F defined as the following system:

$$\mathbf{y} = \mathbf{F}(\mathbf{x}, \gamma) \tag{5.1}$$

The LSA indicates how independent variable x and parameters $\gamma = [\gamma_1, ..., \gamma_r]$ of F influence dependent variable $\mathbf{y}$.

The main concept of LSA is based on computation, after a training process of influence of pattern attributes x_i, i = 1 ,..., N or model's parameter γ on the output value y_j, j= 1,..., N,where N and J denote the number of features and outputs respectively [150] [76]. This influence is characterised by real coefficients S_{ji}

$$S_{j,i}^{(p)} = \frac{\partial y_j(x_1^{(p)}, x_2^{(p)}, ..., x_N^{(p)})}{\partial x_i} \tag{5.2}$$

Equation 5.2 describes the sensitivity value of the jth neural network output signal on the ith attribute of the input vector $\mathbf{x}$, calculated based on the pth training pattern, p = 1,...,P.

LSA approaches can be informative if there is little uncertainty in model input or if the inputs act linearly or additively [114]

5.2.1.2 Global Sensitivity Analysis

Global Sensitivity Analysis (GSA)is the study of how uncertainty in the output of a model either numerical or otherwise can be apportioned to different sources of uncertainty in the model input [113].

GSA considers the impact of varying parameters simultaneously and uniformly over their full range of possible values [115] [66]. GSA can show the relationship between multiple input parameters and cope well with linear and non-linear response [84]. Unlike local sensitivity analysis, global sensitivity analysis requires more computational work and the approach is often probabilistic.

5.3 Artificial Neural Network Application to Sensitivity Analysis

The concept of Artificial Neural Network (ANN) stems from an understanding of how neurons in the brain function to model a simple neural network using electrical circuits [138].

In intrusion detection and prevention system (IDPS), ANN can predict benign traffic from malicious traffic. Any form of prediction can be improved by learning from the data. These improvements and techniques depend on four major factors [112]:

- What data is to be improved
- What prior information is available
- What representation is used for the data
- What feedback is available to learn from

5.3.1 Neural Network Training Algorithms

There are many different batch training algorithms that can be used to train a network. All have different characteristics and performance in terms of speed, precision and

memory requirement. Table 5.1 presents the lists of algorithm and the associated advantages and disadvantages.

Table 5.1: Comparison of Neural Network training algorithm

Algorithm	Advantages	Disadvantages
Gradient descent	Employs first order algorithm to find minimum of a function	Require many iterations for functions which have long narrow valley structures. slow convergence. Prone to get stuck in local minima.
Newton's method	Require fewer steps than gradient descent to find minimum value of loss function.	Requires more information for evaluation, storage and inversion of Hessian Matrix.
Conjugate gradient	Faster convergence than gradient descent.	Line minimisation can be computationally expensive.
Quasi Newton	It is faster than gradient descent and conjugate gradient. Hessian matrix does not need to be computed and inverted.	It needs to store and update a matrix of size M x M.
Levenberg Marquardt	Works without computing exact Hessian matrix Performs well with loss functions which take the sum of squared errors. Error function is minimised, while the step size is kept small.	Requires a lot of memory when computing Jacobian matrix for big datasets.

Figure 5.2 illustrates the memory-speed comparison of neural network training algorithm. The gradient descent is the slowest training algorithm requiring less memory, while Levenberg-Marquardt is the fastest (but it require a lot of computational memory). For our experiment, we utilise the Levenberg-Marquardt training algorithm. The Levenberg-Marquardt training algorithm is regarded to be one of the most efficient training algorithms for ANNs [58]. It works by combining two algorithms (i.e., gradient descent method and the Gauss–Newton method) and as a result, remedies their individual shortcomings [58][143]. Its major drawbacks are the increased computational cost due to the need to carry out Hessian matrix inversion calculation each time

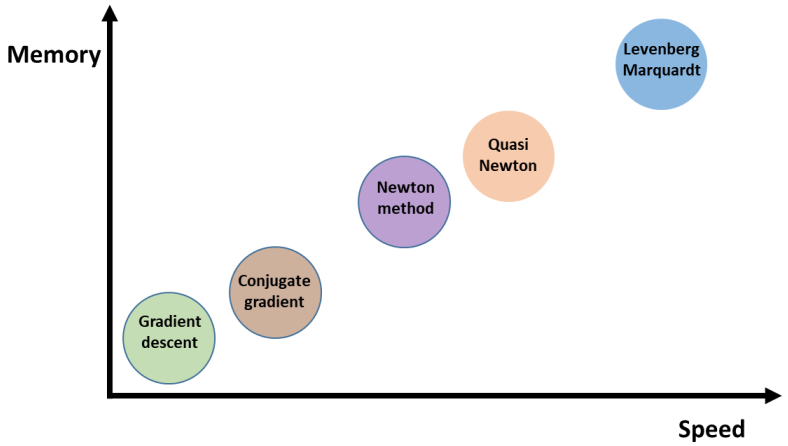

Figure 5.2: Memory speed comparison of neural network algorithm

for weight updating and the storage of the Jacobian matrix whose size is decided by the number of patterns, number of outputs, and the number of weights [143].

For large-sized networks and training patterns, even though the Levenberg–Marquardt algorithm is very efficient, the computational cost may be too expensive or prohibitive to handle the Jacobian matrix storage and the Hessian matrix inversion calculation [143]. For our experiment, there are a few thousands of instances (i.e., 3600 in total), three input parameters and one output. This can be easily classified as small or medium sized problem. Hence, the Levenberg-Marquardt training algorithm is adopted in our experiment due to its speed, stable convergence and less memory consumption (in this case due to a few parameters).

5.4 Description of Dataset

Over time, emphasis has been on the development of algorithm to solve problems. With the growing generation of big data due to migration to 5G and beyond, internet of things (IoT) and cyber-physical processes, it becomes pertinent to develop a means for the accurate representation of data before developing an algorithm that fits the data [126][124].

The dataset for this experiment is generated via the modelled tree topology described in Chapter three (see Figure 3.13) . Four scenarios namely:1. without at-

tack(data collected when there was no attack), 2. With TCP flooding attack(data collected when TCP attack launched), 3. With UDP flooding attack (data collected when UDP attack launched) and 4. With HTTP flooding attack (data collected when HTTP attack launched) scenario were considered. Each experiment was performed for 15 minutes and corresponding network traffic was recorded per second. Hence, there are 900 samples for each scenario to have a total of 3600 data samples. Throughput, jitter and response time features were extracted using KNIME to create our dataset. Tables 5.2 5.3 5.4 5.5 provides the descriptive statistics for the generated data.

Table 5.2: Descriptive statistics of actual simulation data (over 900 data samples) for normal scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	95.1000	95.9000	95.6332	95.6000	0.1402
R_t	0.0320	2.1200	0.2114	0.1980	0.1286
J_t	0.0040	0.4930	0.2271	0.1940	0.0943

Table 5.3: Descriptive statistics of T_p, R_t and J_t(over 900 data samples) for TCP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0	95.9000	0.5441	0.0238	7.0999
R_t	0.2650	678	302.2676	299	110.6598
J_t	0.0040	0.4930	0.2271	0.1940	0.0943

Table 5.4: Descriptive statistics of T_p, R_t and J_t (over 900 data samples) for UDP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	95.1000	95.9000	95.6332	95.6000	0.1402
R_t	0.1980	82.1000	26.3097	24.8000	7.3245
J_t	9.1610	18.4280	10.5100	10.1725	1.0496

It can be seen from Tables 5.2 5.3 5.4 5.5 that the average throughput during attack drops significantly as compared to without attack scenario for TCP and HTTP flooding

Table 5.5: Descriptive statistics of T_p, R_t and J_t(over 900 data samples) for HTTP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0	95.9000	0.7429	0	8.3955
R_t	0.0200	1673	49.1262	23.7000	90.9398
J_t	0.0040	0.4930	0.2271	0.1940	0.0943

attack. Similarly, the jitter is affected adversely for the UDP attack and without attack scenario. In all cases of attack, there is an increase in response time. Thus, it can be deduced that each of these features are sensitive. However, the goal is to establish which is the most sensitive .

5.5 Experimental Approach

In this section, local sensitivity analysis is carried out to determine the sensitivity of throughput, jitter, and response time to DDoS flooding attack. The goal is to determine if truly these metrics are sensitive to attack and which one is the most sensitive. Figure 5.3 shows the methodology flow chart of the 5 stage approach employed in performing local sensitivity analysis.

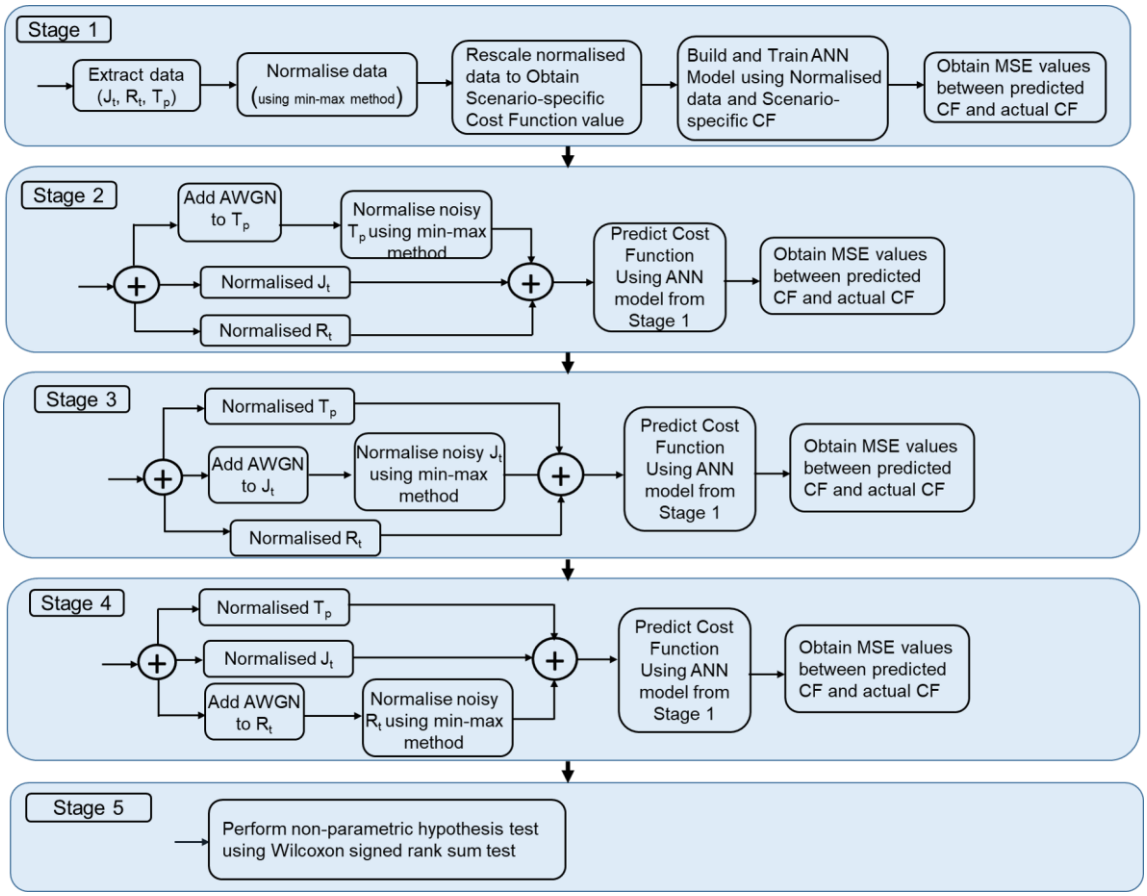

Figure 5.3: LSA methodology flow diagram

- **Stage 1**: At stage 1, throughput, jitter and response time features are extracted and the data is normalised using min-max method. the normalised data is used to obtain cost function values which are then fed as input training data into the ANN where MSE values are obtained.

- **Stage 2 - 4**: In these stages, additive white Gaussian noise(AWGN) is added based on one-at-a-time basis. AWGN is added to throughput(T_p)to have noisy T_p while other factors (Jitter (J_t) and response time (R_t)) were kept constant in stage 2. This process is repeated for noisy J_t (Stage 3) and noisy R_t (Stage 4) while other parameters are kept constant and new cost function values are predicted, respectively in each stage(i.e., Stages 2-4).

- **Stage 5**: Hypothesis test is carried out at this stage to statistically validate any of the inferences made from the deviations.

5.5.1 Data Normalisation

The data were normalised such that each system parameter contributes similar relative numerical weight in order to minimise data redundancy and ensure all target input values have an agreeable metric scale. The data normalisation process employs min-max normalisation method. Min-Max normalisation is a strategy which linearly transforms variable 'X' so that the entire range of values of X from minimum to maximum varies between 0 and 1. It can be expressed mathematically as:

$$X_{normalised} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{5.3}$$

Where X_{min} and X_{max} are the minimum and maximum values in X respectively. Tables 5.6 5.7 5.8 5.9 shows the descriptive statistics of the normalised values for tables presented in section5.4

Table 5.6: Descriptive statistics of normalised T_p, R_t and J_t(over 900 data samples) for without attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0.9917	1	0.9972	0.9969	0.0015
R_t	7.1728e-06	0.0013	1.1440e-04	1.0640e-04	7.6849e-05
J_t	0	0.0265	0.0121	0.0103	0.0051

Table 5.7: Descriptive statistics of normalised T_p, R_t and J_t (over 900 data samples) for TCP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0	1	0.0057	2.4818e-04	0.0740
R_t	1.4645e-04	0.4053	0.1807	0.1787	0.0661
J_t	0	0.0265	0.0121	0.0103	0.0051

Table 5.8: Descriptive statistics of normalised T_p, R_t and J_t (over 900 data samples) for UDP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0.9917	1	0.9972	0.9969	0.0015
R_t	1.0640e-04	0.0491	0.0157	0.0148	0.0044
J_t	0.4970	1	0.5702	0.5519	0.0570

Table 5.9: Descriptive statistics of normalised T_p, R_t and J_t (over 900 data samples) for HTTP attack scenario

Metric	Min	Max	Average	Median	Standard Deviation
T_p	0	1	0.0077	0	0.0875
R_t	0	1	0.0294	0.0142	0.0544
J_t	0	0.0265	0.0121	0.0103	0.0051

5.5.2 Cost function value evaluation

We use the normalised data in section 5.5.1 to build Input-Output correspondence and the normalised values are scaled to the following four scenarios:

- **Scenario 1**: a scale of 1 is assigned to represent without attack
- **Scenario 2**: a scale of 2 is assigned to represent with TCP attack
- **Scenario 3**: a scale of 3 is assigned to represent with UDP attack
- **Scenario 4**: a scale of 4 is assigned to represent with HTTP attack

in order to reflect scenario-specific targets from their corresponding values, a mathematical cost function in terms of throughput, jitter and response time is introduced. The proposed cost function tends toward unity for the worst case scenario (SDN under

severe attack) and approaches zero for the best case scenario (SDN without attack). The cost function can be represented mathematically by:

$$C_F = (abs(T_p - J_t) * R_t) * L_w \tag{5.4}$$

where C_F represents cost function, T_p = Throughput, J_t = Jitter , R_t = Response time and L_w = Weight or Scale.

For the best case scenario (i.e.,normal SDN state), the throughput is maximum, response time is minimum and jitter is minimum. Therefore, T_p approaches 1, J_t and R_t approach zero after normalisation. So, we have the following condition for the best case scenario.

$$Bestcase = \begin{cases} T_p \rightarrow 1 \\ J_t \rightarrow 0 \\ R_t \rightarrow 0 \end{cases} \tag{5.5}$$

Similarly, for the worst case scenario (i.e., SDN under severe attack), the throughput is minimum, response time is maximum and jitter is maximum. Therefore, T_p approaches 0, J_t and R_t approach 1 after normalisation. Hence, we have the following condition for the worst case scenario.

$$Worstcase = \begin{cases} T_p \rightarrow 0 \\ J_t \rightarrow 1 \\ R_t \rightarrow 1 \end{cases} \tag{5.6}$$

Substituting the values in equations 5.5 and 5.6 into equation 5.4 , C_F approaches null (zero) for normal network state and approaches unity (one) when the SDN is under attack. A descriptive statistics of the cost function value over 3600 samples for normal, UDP, TCP, and HTTP flooding attack scenarios is presented in Table5.10.

As shown in Figure 5.4, the cost function value(C_F) associated with normal (without attack) network traffic hovers around zero. This value satisfies the condition for our best case scenario. The other attack scenario has cost function value well above zero with peak values of 0.78 and 0.63 recorded for HTTP and UDP flood traffic respectively.

Table 5.10: Descriptive Statistics of the Cost function value (over 3600 data samples) for normal, TCP, UDP and HTTP attack scenarios

Metric	Min	Max	Average	Median	Standard Deviation
Normal	7.0837e-05	0.0124	0.0011	0.0010	7.5756e-04
TCP	0	0.1528	0.0435	0.0382	0.0251
UDP	0.0014	0.6656	0.2018	0.1968	0.0654
HTTP	0	0.7728	0.0147	0.0065	0.0336

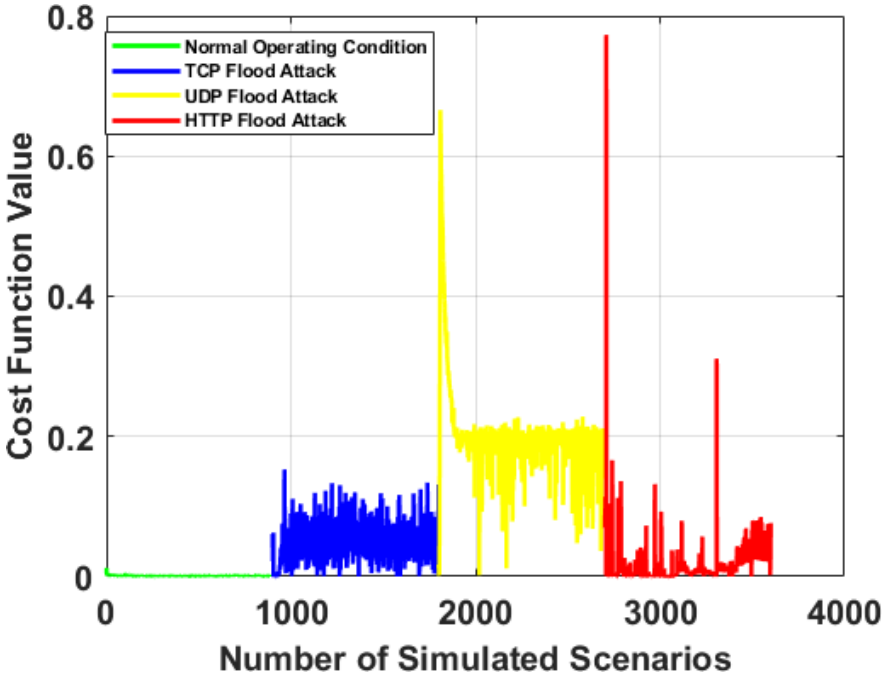

Figure 5.4: Variation in cost function value versus attack

The cost function value(C_F) simply indicates that the system parameters experience changes due to attacks. To ascertain the most sensitive of these parameters due to attacks, local sensitivity analysis is carried out.

5.5.3 AWGN and MSE

Additive White Gaussian Noise (AWGN) is a statistical noise with a Probability Density Function equal to that of standard normal distribution. AWGN is characterised with bell shaped curve as shown in Figure 5.5 with mean value of zero, standard deviation value of 1 and total area under the curve is 1. For the local sensitivity analysis,

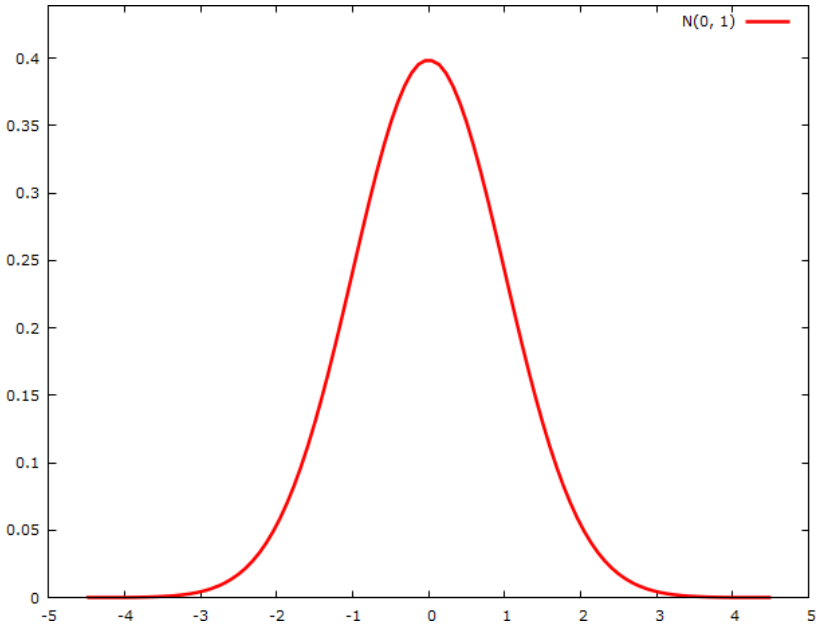

Figure 5.5: AWGN distribution

AWGN is added to throughput, jitter and response time as shown in stages 2 - 4 of the LSA methodology flow chart(see figure 5.3). Mean Squared Error(MSE) values over 50 run is obtained afterwards. MSE measures the average squared difference between the predicted values and the actual value. MSE is expressed mathematically as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \bar{y_i})^2 \tag{5.7}$$

5.5.4 ANN training

A prediction model is built using ANN with the normalised value discussed in Section 5.5.1 as input and the cost function values described in Table5.10 as the target values.

The ANN model is trained with the three inputs metrics (i.e., T_p, R_t and J_t), 10 hidden layers and a single output (i.e.,C_F)under 9 iterations as shown in Figure 5.7. The best validation performance is at epoch 3 (see Figure 5.6). The number of processing elements per layer, as well as the number of layers greatly influence the training process. Too few processing elements can slow down the learning process and too many can lead to overfitting of the training dataset [7][6]. For our experiment, 2520 data samples (70% of the total data samples) have been used as the training data set, 540 data samples (15% of the total data samples) have been used as the validation data set and 540 data samples (15% of the total data samples) have been used as the test data set according to the data portioning approach recommended in several works [19][28]. Figure 5.6 shows that the ANN model is correct and acceptably accurate.

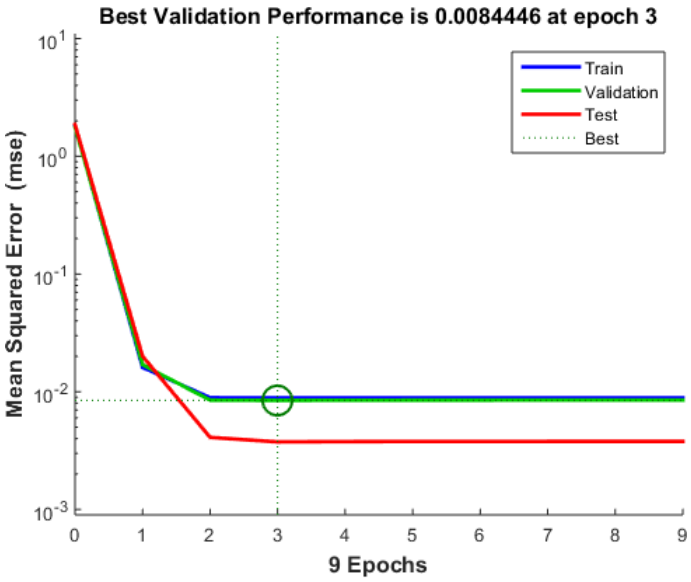

Figure 5.6: A typical plot of MSE vs number of Epochs

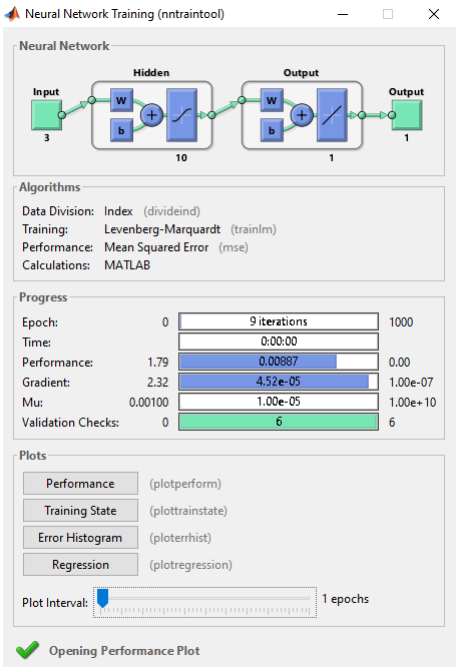

Figure 5.7: ANN training model

5.6 Result and Discussions

The sensitivity of throughput, jitter and response time is evaluated using deviation of newly predicted target value from actual target values obtained and the mean squared error value of the prediction model. Tables 5.11 5.13 5.12 show the impact of adding AWGN to our impact metrics. For the 50 independent statistical runs to validate the statistical significance of this experiment, it can be seen that jitters standard deviation value is considerably more than what is obtainable in the T_p noisy and R_t noisy respectively (See the appendices for the complete tables).

Table 5.11: Local sensitivity analysis for noisy T_p, normalised R_t, normalised J_t (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.2907	0.6820	-0.0230	-0.0632	0.0905
2	-0.1138	0.6911	0.0090	-0.0281	0.0908
3	-0.0876	0.7104	0.0102	-0.0265	0.0915
4	-0.5090	0.6848	-0.0166	-0.0583	0.0900
5	-0.1144	0.6929	-0.0195	-0.0619	0.0895
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
45	-0.1828	0.7074	0.0107	-0.0279	0.0902
46	-0.1252	0.7173	-0.0016	-0.0435	0.0897
47	-0.1248	0.7038	-0.0100	-0.0521	0.0897
48	-0.1589	0.7035	0.0113	-0.0242	0.0917
49	-0.2439	0.6930	-0.0182	-0.0603	0.0894
50	-0.0753	0.7132	0.0036	-0.0390	0.0893

Table 5.12: Local sensitivity analysis for noisy R_t, normalised J_t, normalised T_p (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.1461	0.6995	-0.0017	-0.0445	0.0898
2	-0.0896	0.6993	-0.0107	-0.0528	0.0895
3	-0.0943	0.7007	-0.0090	-0.0513	0.0893
4	-0.5218	0.6938	-0.0113	-0.0531	0.0899
5	-0.1139	0.6994	-0.0083	-0.0499	0.0893
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
45	-0.1440	0.6934	-0.0107	-0.0521	0.0893
46	-0.1094	0.6955	-0.0108	-0.0519	0.0893
47	-0.0889	0.6941	-0.0102	-0.0514	0.0892
48	-0.2157	0.6997	-0.0091	-0.0500	0.0895
49	-0.2500	0.7009	-0.0100	-0.0521	0.0895
50	-0.0836	0.6922	-0.0093	-0.0511	0.0894

Table 5.13: Local sensitivity analysis for noisy J_t, normalised R_t, normalised T_p (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.2783	0.6505	-0.0323	-0.0518	0.1307
2	-0.4036	0.7082	-0.0770	-0.0655	0.1249
3	-0.3362	0.6227	-0.0239	-0.0536	0.0962
4	-0.5330	0.7538	0.0612	0.0708	0.1207
5	-0.1335	0.7024	-0.0008	-0.0401	0.0900
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
45	-0.2275	0.6672	-0.0260	-0.0657	0.0912
46	-0.2108	0.7184	-0.0326	-0.0568	0.0963
47	-0.2364	0.6950	-0.0108	-0.0379	0.0957
48	-0.4165	0.7079	0.0191	-0.0179	0.0912
49	-0.1906	0.7496	0.0007	-0.0350	0.0922
50	-0.0896	0.7296	0.0247	-0.0012	0.0998

5.6.1 Hypothesis test

Wilcoxon test [139] is carried out over 50 runs when J_t, T_p, and R_t are noisy and when they are not noisy. Since the sample size is sufficiently large (that is, 50 in this case), a z-statistic can be used to approximate the probability value (p-value) of the test [52]. This is why Wilcoxon test [139] is an appropriate test for statistical significance in this case. The Wilcoxon test is a non-parametric test that obeys the central limit theorem. It tests the null hypothesis that the normalised data and its noisy version are from continuous distributions with equal medians. A common significance level of 0.05 (i.e., 5%) is selected. If the resultant p-value is equal to or less than 0.05, then, there is strong evidence against the null hypothesis. The p-value obtained from the Wilcoxon test is shown in table 5.14. From table 5.14, it can be seen that the null hypothesis is rejected for all cases. This indicates that noisy J_t, T_p, and R_t are all statistically sensitive.

- S1: normalised T_p, normalised R_t, normalised J_t
- S2: noisy T_p, normalised R_t, normalised J_t

Table 5.14: Descriptive Statistics of the MSE Values Over 50 Statistical Runs

Metric	Min	Max	Average	Median	Standard Deviation	p-value
S1	0.00797	0.00826	0.0080	0.00803	0.000047	N/A
S2	0.00797	0.01145	0.0084	0.00824	0.000634	2.4225e-09
S3	0.00797	0.00843	0.0081	0.00807	0.000066	1.7330e-17
S4	0.00799	0.02951	0.0115	0.02951	0.004858	1.7330e-17

- S3: noisy R_t, normalised J_t, normalised T_p
- S4: noisy J_t, normalised R_t, normalised T_p

A plot of MSE values against the number of runs is shown in Figure 5.8. Using ranksum, the result shows that jitter is the most sensitive to flooding attack followed by throughput and then response time. The work presented in [85] also shows that delay jitter may severely degrade systems performance. It is worthy of note that all parameters evaluated are sensitive to DDoS attack. Hence, adequate prevention and mitigation schemes can be deployed in SDN controller if these features are embedded in the attack detection scheme.

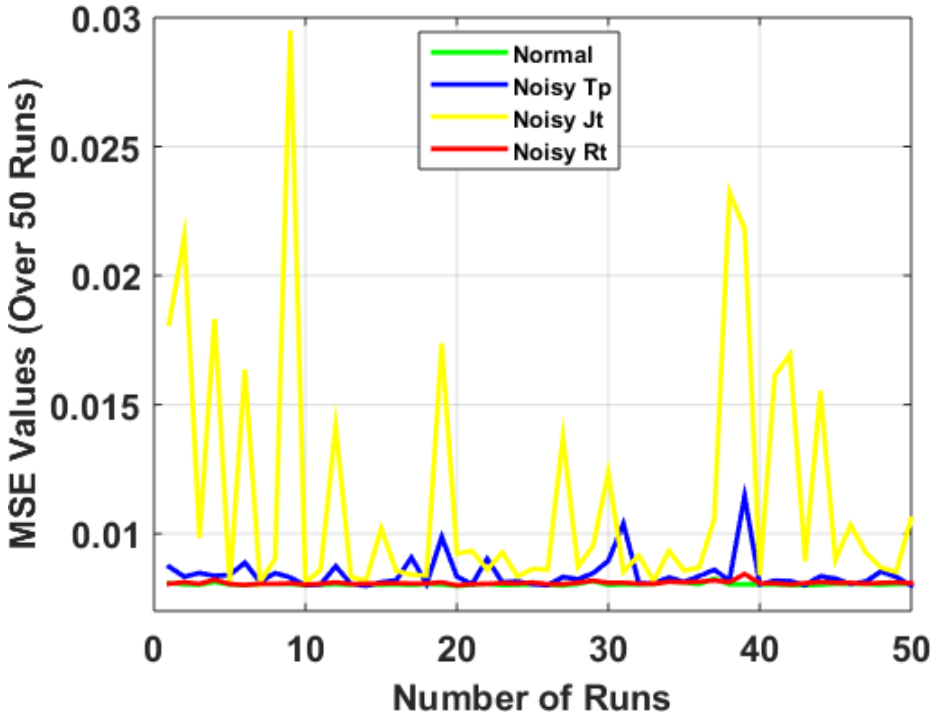

Figure 5.8: Sensitivity analysis of throughput (T_p), jitter (J_t) and response time(R_t)

5.7 Summary

In this chapter, LSA is implemented on real SDN traffic to identify the key metrics that mainly influence the prediction of whether an SDN is under attack or secure. The SDN traffic dataset considered are throughput, response time and jitter, and they are generated from a modelled tree topology in Mininet. The SDN is subjected to a DDoS flooding attack launched using LOIC. An ANN prediction model is built using a min-max feature scaling to derive actual target values from the normalised input parameters. The sensitivity of throughput, jitter and response time is then evaluated using the deviations of newly predicted target values from actual target values when an AWGN is added to the respective SDN traffic dataset. Results obtained show that throughput, jitter and response time are all statistically sensitive to a DDoS flooding attack on the SDN, and jitter is the most sensitive of all the impact metrics considered.

Chapter 6

Conclusions and Future Work

This chapter presents the conclusion of this thesis and recommends future research direction in detecting and mitigating DDoS attacks in SDN.

6.1 Conclusion

Networks have become absolutely essential element in the way we do things. It provides vital communication links that organisations require to run their application and be competitive. Due to an increase in demand for high data rate and real time applications, the networking industry is faced with the challenge of constantly redesigning already complex, vendor-specific equipment with little or no flexibility and interoperability.

SDN is an open technology which promises more innovation, flexible and effective solutions. Although SDN on the surface provides a simple framework for network programmability and monitoring, SDN security is such an important aspect of network and the impact of a security breach cannot be overemphasised. While research into network security has proceeded at a pace in the scientific and mathematical world, its applications and practicality simply haven't kept pace.

In this thesis, design and analysis of anomaly detection and mitigation schemes in SDN is conducted. The main goal is to establish, if any, the impact of DDoS attack on SDN, and experimentally assess attack detection techniques using statistical and machine learning approach. Furthermore, we examine DDoS attack mitigation technique

using reactive flow rule pushed out through the controller and finally, perform sensitivity analysis on the impact metrics (throughput, jitter and response time) to determine which parameter is more sensitive to attack.

The simulation results show that DDoS flooding attack on SDN network can degrade network performance. At first, network throughput is polled within an interval to determine the normal distribution of network data without attack and Confidence Interval (CI) for the normal distribution is obtained. An attack is indicated by a significant deviation in mean throughput value obtained at subsequent interval compared to the without attack mean throughput. The calculation of confidence interval and mean throughput has low overhead and can be easily implemented in the SDN controller to detect an anomaly. Our evaluation shows that, by leveraging on the throughput information from the network server, DDoS attack can be easily detected in real time with an accuracy of approximately 99% when polled at an interval of 60 seconds. In addition, we implement machine learning algorithms to detect DDoS attack. The models were validated using six machine learning algorithm (LR, LDA, KNN, SVM, NB and CART) on emulated network and real SDN dataset containing HTTP flood, TCP flood and UDP flood. CART algorithm achieved a high detection accuracy of 98.47% with only three features namely: throughput, jitter, and response times when compared to other machine learning algorithm deployed. Hence, CART exhibits strong potential DDoS flooding attack detection in SDN environments.

Similarly, we demonstrate the effect of UDP and TCP flood DDoS attack on SDN. This study reveals that existing controller and data plane devices are prone to flooding and spoofing attack which degrades network performance within a short while. It also indicates that these attacks can be mitigated by pushing blocking flows from the controller to the attacking switch. Our evaluation shows that additional flow rule insertion to mitigate DDoS flooding attack imposes minimal overhead in terms of CPU utilisation on the controller and server.

Finally, we perform sensitivity analysis to determine which of the metrics (jitter, throughput and response time) is more sensitive to distortion by introducing white Gaussian noise and evaluating the local sensitivity using feedforward artificial neural network. All metrics are sensitive in detecting DDoS attack. However, jitter appears to be the most sensitive to attack.

For a DDoS attack with more active agents, the attack can be more severe. Hence,

the need for a robust resilient SDN security architecture. While the evaluation of the impact of DDoS attack on SDNs remains a very rigorous endeavour, the work carried out in this thesis offers a primer to the objective evaluation of DDoS attack on SDNs.

6.2 Future Work

The following are the possible extensions of this thesis which include recommendations for future research directions in SDN security:

- **Self healing:** Due to global view advantage of SDN controller, the network downtime can be minimised by timely fault log. The beauty of network self-healing is that service downtime and other network related issues can be resolved immediately without network administrators having to get involved. SDN enabled self healing network will involve less network administrator, hence, cost savings for organisations and improved customer satisfaction. Some research on self-healing network with respect to SDN can be seen in [130] [103] [59].

- **Load balancing:** Load balancing address resource allocation and guarantees that available resources are utilised efficiently. During DDoS attack, overloaded link can be bypassed by finding out less utilised routes and balancing the traffic across it [11]. Some research on load balancing can be found in [1] [148].

- **Integration to cloud:** cloud computing technology has made networking resources on demand and it offers services on a pay-as-you-go basis [125]. leveraging on SDN enabled cloud services and security will offer elastic services to customers with high availability and networking resources in the cloud can be handled effectively by the SDN controller [8]

- **Global Sensitivity Analysis:** As stated in the final chapter, local sensitivity analysis was carried out to determine how sensitive each input parameters are to distortion caused by white Gaussian noise. It would be interesting to see how GSA will handle multiple input parameters simultaneously and show the relationship between these parameters.

References

[1] A. A. Abdelltif, E. Ahmed, A. T. Fong, A. Gani, and M. Imran, "Sdn-based load balancing service for cloud servers," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 106–111, 2018.

[2] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[3] A. Akhunzada, E. Ahmed, A. Gani, M. K. Khan, M. Imran, and S. Guizani, "Securing software defined networks: taxonomy, requirements, and open issues," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.

[4] Y. Al-Hammadi, U. Aickelin, and J. Greensmith, "Dca for bot detection," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 1807–1816.

[5] M. B. Al-Somaidai and E. B. Yahya, "Survey of software components to emulate openflow protocol as an sdn implementation," *American Journal of Software Engineering and Applications*, vol. 3, no. 6, pp. 74–82, 2014.

[6] C. Aldrich, *Exploratory analysis of metallurgical process data with neural networks and related methods*. Elsevier, 2002, vol. 12.

[7] M. K. S. Alsmadi, K. B. Omar, S. A. Noah *et al.*, "Back propagation algorithm: the best algorithm among the multi-layer perceptron algorithm," *International*

Journal of Computer Science and Network Security, vol. 9, no. 4, pp. 378–383, 2009.

[8] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, and G. Wang, "Meridian: an sdn platform for cloud network services," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 120–127, 2013.

[9] G. E. Batista, M. C. Monard *et al.*, "A study of k-nearest neighbour as an imputation method." *HIS*, vol. 87, no. 251-260, p. 48, 2002.

[10] N. Z. Bawany, J. A. Shamsi, and K. Salah, "Ddos attack detection and mitigation using sdn: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.

[11] M. Belyaev and S. Gaivoronski, "Towards load balancing in sdn-networks during ddos-attacks," in *2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*. IEEE, 2014, pp. 1–6.

[12] K. Benton, L. J. Camp, and C. Small, "Openflow vulnerability assessment," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 151–152.

[13] H. Berger and A. Jones, "Cyber security & ethical hacking for smes," in *Proceedings of the The 11th International Knowledge Management in Organizations Conference on The changing face of Knowledge Management Impacting Society*. ACM, 2016, p. 12.

[14] D. K. Bhattacharyya and J. K. Kalita, *DDoS attacks: evolution, detection, prevention, reaction, and tolerance*. Chapman and Hall/CRC, 2016.

[15] R. Bifulco and M. Dusi, "Position paper: Reactive logic in software-defined networking: Accounting for the limitations of the switches," *2014 Third European Workshop on Software Defined Networks*, pp. 97–102, 2014.

[16] R. Braga, E. de Souza Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow." in *LCN*, vol. 10, 2010, pp. 408–415.

[17] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[18] S. Brief, "Sdn security considerations in the data center," 2013.

[19] I. Brown and C. Mues, "An experimental comparison of classification algorithms for imbalanced credit scoring data sets," *Expert Systems with Applications*, vol. 39, no. 3, pp. 3446–3453, 2012.

[20] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 15–28.

[21] A. T. Campbell, I. Katzela, K. Miki, and J. Vicente, "Open signaling for atm, internet and mobile networks (opensig'98)," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 1, pp. 97–108, 1999.

[22] L. O. I. Canon, "Praetox Technologies Low Orbit Ion Cannon." [Online]. Available: https://github.com/NewEraCracker/LOIC/

[23] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[24] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks." in *USENIX Security Symposium*, vol. 49, 2006, p. 50.

[25] G.-Y. Chan, C.-S. Lee, and S.-H. Heng, "Discovering fuzzy association rule patterns and increasing sensitivity analysis of xml-related attacks," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 829–842, 2013.

[26] J. Chen, H. Huang, S. Tian, and Y. Qu, "Feature selection for text classification with naïve bayes," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5432–5435, 2009.

[27] Z. Chen, F. Jiang, Y. Cheng, X. Gu, W. Liu, and J. Peng, "Xgboost classifier for ddos attack detection and analysis in sdn-based cloud," in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2018, pp. 251–256.

[28] D. Chicco, "Ten quick tips for machine learning in computational biology," *BioData mining*, vol. 10, no. 1, p. 35, 2017.

[29] O. Coker and S. Azodolmolky, *Software-defined Networking with OpenFlow: Deliver Innovative Business Solutions*. Packt Publishing Ltd, 2017.

[30] P. Cunningham, "k-nearest neighbour classifiers. mult classif syst," 2007.

[31] N.-N. Dao, J. Park, M. Park, and S. Cho, "A feasible method to combat against ddos attack in sdn network," in *2015 International Conference on Information Networking (ICOIN)*. IEEE, 2015, pp. 309–311.

[32] S. Das, G. Parulkar, and N. McKeown, "Rethinking ip core networks," *Journal of Optical Communications and Networking*, vol. 5, no. 12, pp. 1431–1442, 2013.

[33] N. Dayal, P. Maity, S. Srivastava, and R. Khondoker, "Research trends in security and ddos in sdn," *Security and Communication Networks*, vol. 9, no. 18, pp. 6386–6411, 2016.

[34] N. Dayal and S. Srivastava, "Analyzing behavior of ddos attacks to identify ddos detection features in sdn," in *2017 9th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE, 2017, pp. 274–281.

[35] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *NDSS*, vol. 15, 2015, pp. 8–11.

[36] T. Dierks, "The transport layer security (tls) protocol version 1.2," 2008.

[37] D. Dittrich, "The dos project's 'trinoo' distributed denial of service attack tool," 1999.

[38] P. Dong, X. Du, H. Zhang, and T. Xu, "A detection method for a novel ddos attack against sdn controllers by vast new low-traffic flows," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–6.

[39] D. D. Dorfman and E. Alf Jr, "Maximum-likelihood estimation of parameters of signal-detection theory and determination of confidence intervals—rating-method data," *Journal of mathematical psychology*, vol. 6, no. 3, pp. 487–496, 1969.

[40] A. Doria, F. Hellstrand, K. Sundell, and T. Worster, "General switch management protocol (gsmp) v3," 2002.

[41] A. Doria, J. H. Salim, R. Haas, H. M. Khosravi, W. Wang, L. Dong, R. Gopal, and J. M. Halpern, "Forwarding and control element separation (forces) protocol specification." *RFC*, vol. 5810, pp. 1–124, 2010.

[42] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: a classification," in *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No. 03EX795)*. IEEE, 2003, pp. 190–193.

[43] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[44] G. Eason, R. Kloti, V. Kotronis, and P. Smith, "Openflow: A security analysis," in *IEEE ICNP*, 2013.

[45] P. Engebretson, *The basics of hacking and penetration testing: ethical hacking and penetration testing made easy*. Elsevier, 2013.

[46] R. Enns, "Netconf configuration protocol-rfc 4741," *RFC Editor*, 2006.

[47] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[48] M. Fernando, P. Esteves, C. Esteve *et al.*, "Software-defined networking: A comprehensive survey," *PROCEEDINGS OF THE IEEE*, 2015.

[49] Floodlight, "Floodlight controller." [Online]. Available: http://www.projectfloodlight.org/

[50] D. Gavrilis and E. Dermatas, "Real-time detection of distributed denial-of-service attacks using rbf networks and statistical features," *Computer Networks*, vol. 48, no. 2, pp. 235–245, 2005.

[51] A. Ghasemi and S. Zahediasl, "Normality tests for statistical analysis: a guide for non-statisticians," *International journal of endocrinology and metabolism*, vol. 10, no. 2, p. 486, 2012.

[52] J. Gibbons and S. Chakraborti, "Nonparametric statistical inference, 4th edn. m," 2003.

[53] GinjaChris, "GinjaChris." [Online]. Available: https://github.com/GinjaChris/pentmenu

[54] P. Goransson, C. Black, and T. Culver, *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

[55] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, 2005.

[56] B. Gupta, R. C. Joshi, and M. Misra, "Defending against distributed denial of service attacks: issues and challenges," *Information Security Journal: A Global Perspective*, vol. 18, no. 5, pp. 224–247, 2009.

[57] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid isolation: A slice abstraction for software-defined networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 79–84.

[58] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[59] K. Hasan, S. Shetty, A. Hassanzadeh, M. B. Salem *et al.*, "Self-healing cyber resilient framework for software defined networking-enabled energy delivery system," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2018, pp. 1692–1697.

[60] S. Hashem, "Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 1. IEEE, 1992, pp. 419–424.

[61] T. Hinrichs, N. Gude, M. Casado, J. Mitchell, and S. Shenker, "Expressing and enforcing flow-based network security policies," *University of Chicago, Tech. Rep*, vol. 9, 2008.

[62] A. Hunter, L. Kennedy, J. Henry, and I. Ferguson, "Application of neural networks and sensitivity analysis to improved prediction of trauma survival," *Computer methods and programs in biomedicine*, vol. 62, no. 1, pp. 11–19, 2000.

[63] R. L. Iman and W. Conover, "Small sample sensitivity analysis techniques for computer models. with an application to risk assessment," *Communications in statistics-theory and methods*, vol. 9, no. 17, pp. 1749–1842, 1980.

[64] R. L. Iman and J. C. Helton, "An investigation of uncertainty and sensitivity analysis techniques for computer models," *Risk analysis*, vol. 8, no. 1, pp. 71–90, 1988.

[65] T. S. S. International Telecommunication Union, "Security architecture for systems providing end-to-end communications," *ITU-T Rec. X. 805*.

[66] B. Iooss and P. Lemaˆıtre, "A review on global sensitivity analysis methods," in *Uncertainty management in simulation-optimization of complex systems*. Springer, 2015, pp. 101–122.

[67] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 127–132.

[68] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[69] R. H. Jhaveri, N. M. Patel, Y. Zhong, and A. K. Sangaiah, "Sensitivity analysis of an attack-pattern discovery based trusted routing scheme for mobile ad-hoc networks in industrial iot," *IEEE Access*, vol. 6, pp. 20 085–20 103, 2018.

[70] R. d. S. M. Júnior, A. P. Guimaraes, K. M. Camboim, P. R. Maciel, and K. S. Trivedi, "Sensitivity analysis of availability of redundancy in computer networks," *CTRQ 2011*, p. 122, 2011.

[71] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "Jess: Joint entropy-based ddos defense scheme in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2358–2372, 2018.

[72] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 15–27.

[73] J. Kirch, C. Thomaseth, A. Jensch, and N. E. Radde, "The effect of model rescaling and normalization on sensitivity analysis on an example of a mapk pathway model," *EPJ Nonlinear Biomedical Physics*, vol. 4, no. 1, p. 3, 2016.

[74] R. Kokila, S. T. Selvi, and K. Govindarajan, "Ddos detection and analysis in sdn-based environment using support vector machine classifier," in *2014 Sixth International Conference on Advanced Computing (ICoAC)*. IEEE, 2014, pp. 205–210.

[75] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "Ddos in the iot: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[76] P. A. Kowalski and M. Kusy, "Sensitivity analysis for probabilistic neural network structure reduction," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 5, pp. 1919–1932, 2017.

[77] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55–60.

[78] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *arXiv preprint arXiv:1406.0440*, 2014.

[79] M. Kuerban, Y. Tian, Q. Yang, Y. Jia, B. Huebert, and D. Poss, "Flowsec: Dos attack mitigation strategy on sdn controller," in *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*. IEEE, 2016, pp. 1–2.

[80] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, "Ddos attack detection method using cluster analysis," *Expert systems with applications*, vol. 34, no. 3, pp. 1659–1665, 2008.

[81] R. J. Lewis, "An introduction to classification and regression tree (cart) analysis," in *Annual meeting of the society for academic emergency medicine in San Francisco, California*, vol. 14, 2000.

[82] C. Li, Y. Wu, X. Yuan, Z. Sun, W. Wang, X. Li, and L. Gong, "Detection and defense of ddos attack–based on deep learning in openflow-based sdn," *International Journal of Communication Systems*, vol. 31, no. 5, p. e3497, 2018.

[83] S. Lim, S. Yang, Y. Kim, S. Yang, and H. Kim, "Controller scheduling for continued sdn operation under ddos attacks," *Electronics Letters*, vol. 51, no. 16, pp. 1259–1261, 2015.

[84] K. G. Link, M. T. Stobb, J. Di Paola, K. B. Neeves, A. L. Fogelson, S. S. Sindi, and K. Leiderman, "A local and global sensitivity analysis of a mathematical model of coagulation and platelet deposition under flow," *PloS one*, vol. 13, no. 7, p. e0200917, 2018.

[85] M. Long, C.-H. Wu, and J. Y. Hung, "Denial of service attacks on network-based control systems: impact and mitigation," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 2, pp. 85–96, 2005.

[86] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.

[87] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[88] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[89] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *International workshop on recent advances in intrusion detection*. Springer, 2011, pp. 161–180.

[90] Mininet, "Download/Get Started with Mininet." [Online]. Available: http://mininet.org/download/

[91] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[92] J. T. Moore, M. Hicks, and S. Nettles, "Practical programmable packets," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 1. IEEE, 2001, pp. 41–50.

[93] S. M. Mousavi and M. St-Hilaire, "Early detection of ddos attacks against sdn controllers," in *2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 77–81.

[94] N. Moustafa and J. Slay, "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: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[95] S. Namal, I. Ahmad, A. Gurtov, and M. Ylianttila, "Enabling secure mobility with openflow," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. IEEE, 2013, pp. 1–5.

[96] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich, "Delegating network security with more information," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 19–26.

[97] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: dynamic access control for enterprise networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 11–18.

[98] A. Networks, "Network Security Infrastructure Report." [Online]. Available: https://www.arbornetworks.com/report/

[99] E. Ng, Z. Cai, and A. Cox, "Maestro: A system for scalable openflow control," *Rice University, Houston, TX, USA, TSEN Maestro-Techn. Rep, TR10-08*, 2010.

[100] W. W. Ng, R. K. Chang, and D. S. Yeung, "Dimensionality reduction for denial of service detection problems using rbfnn output sensitivity," in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 03EX693)*, vol. 2. IEEE, 2003, pp. 1293–1298.

[101] Nmap, "Zenmap." [Online]. Available: https://nmap.org/zenmap/

[102] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[103] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "Self-healing and sdn: Bridging the gap," *Digital Communications and Networks*, 2019.

[104] Open Networking Foundation, "OpenFlow Switch Specification (Version 1.5.1)," vol. 1, pp. 1–36, 2015. [Online]. Available: http://www.opennetworking.orghttps://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf

[105] M. Panda and M. R. Patra, "Network intrusion detection using naive bayes," *International journal of computer science and network security*, vol. 7, no. 12, pp. 258–263, 2007.

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[107] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[108] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 121–126.

[109] T. L. F. PROJECTS, "THE LINUX FOUNDATION PROJECTS." [Online]. Available: https://www.opendaylight.org/

[110] B. Rashidi, C. Fung, and E. Bertino, "A collaborative ddos defence framework using network function virtualization," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2483–2497, 2017.

[111] E. Rogers, "Diffusion of innovations, edition of the free press," *The fourth*, 1995.

[112] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[113] A. Saltelli, "Sensitivity analysis for importance assessment," *Risk analysis*, vol. 22, no. 3, pp. 579–590, 2002.

[114] A. Saltelli and P. Annoni, "How to avoid a perfunctory sensitivity analysis," *Environmental Modelling & Software*, vol. 25, no. 12, pp. 1508–1517, 2010.

[115] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, and S. Tarantola, *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[116] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, "Sensitivity analysis in practice: a guide to assessing scientific models," *Chichester, England*, 2004.

[117] A. Sangodoyin, B. Modu, I. Awan, and J. P. Disso, "An approach to detecting distributed denial of service attacks in software defined networks," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2018, pp. 436–443.

[118] A. Sangodoyin, T. Sigwele, and P. Pillai, "Dos attack impact assessment on software defined networks," *Wireless and Satellite Systems*, p. 11.

[119] M. Santos, B. de Oliveira, C. Margi, B. N. Astuto, T. Turletti, and K. Obraczka, "Software-defined networking based capacity sharing in hybrid networks," 2013.

[120] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[121] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*. Ndss, 2013.

[122] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 413–424.

[123] K. Slavov, D. Migault, and M. Pourzandi, "Identifying and addressing the vulnerabilities and security issues of sdn," *Ericsson Technology Review*, vol. 92, no. 7, 2015.

[124] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 305–316.

[125] J. Son and R. Buyya, "A taxonomy of software-defined networking (sdn)-enabled cloud computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, p. 59, 2018.

[126] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 4, pp. 70–73, 2014.

[127] O. switch specification, "Download/Get Started with Mininet." [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.0.pdf

[128] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in sdn-based networks," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 202–206.

[129] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE communications Magazine*, vol. 35, no. 1, pp. 80–86, 1997.

[130] P. Thorat, S. M. Raza, D. T. Nguyen, G. Im, H. Choo, and D. S. Kim, "Optimized self-healing framework for software defined networks," in *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*. ACM, 2015, p. 7.

[131] S. Vishwanathan and M. N. Murty, "Ssvm: a simple svm algorithm," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, vol. 3. IEEE, 2002, pp. 2393–2398.

[132] VMware, "Network Virtualization and Security Software." [Online]. Available: https://www.vmware.com/products/nsx.html

[133] A. Voellmy and J. Wang, "Scalable software defined network controllers," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 289–290, 2012.

[134] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "Ddos attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[135] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed ddos detection mechanism in software-defined networking," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 310–317.

[136] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 171–172.

[137] D. Wetherall, "Active network vision and reality: lessons from a capsule-based system," in *Proceedings DARPA Active Networks Conference and Exposition*. IEEE, 2002, pp. 25–40.

[138] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[139] F. Wilcoxon, "Individual comparisons by ranking methods. biom bull 1: 80–83," 1945.

[140] P. Xiao, W. Qu, H. Qi, and Z. Li, "Detecting ddos attacks against data center with correlation analysis," *Computer Communications*, vol. 67, pp. 66–74, 2015.

[141] G. Yao, J. Bi, and P. Xiao, "Source address validation solution with openflow/nox architecture," in *2011 19Th IEEE international conference on network protocols*. IEEE, 2011, pp. 7–12.

[142] J. Ye, R. Janardan, and Q. Li, "Two-dimensional linear discriminant analysis," in *Advances in neural information processing systems*, 2005, pp. 1569–1576.

[143] H. Yu and B. M. Wilamowski, "Levenberg-marquardt training," *Industrial electronics handbook*, vol. 5, no. 12, pp. 1–16, 2011.

[144] S. Yu, Y. Tian, S. Guo, and D. O. Wu, "Can we beat ddos attacks in clouds?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2245–2254, 2013.

[145] C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, and C. YanRen, "A novel design for future on-demand service and security," in *2010 IEEE 12th International Conference on Communication Technology*. IEEE, 2010, pp. 385–388.

[146] S. Zerkane, D. Espes, P. Le Parc, and F. Cuppens, "Vulnerability analysis of software defined networking," in *International Symposium on Foundations and Practice of Security*. Springer, 2016, pp. 97–116.

[147] X. Zhou and H. Lin, *Local Sensitivity Analysis*. Cham: Springer International Publishing, 2017, pp. 1130–1131. [Online]. Available: https://doi.org/10.1007/978-3-319-17885-1_703

[148] Y. Zhou, K. Zheng, W. Ni, and R. P. Liu, "Elastic switch migration for control plane load balancing in sdn," *IEEE Access*, vol. 6, pp. 3909–3919, 2018.

[149] J. M. Zurada, A. Malinowski, and I. Cloete, "Sensitivity analysis for minimization of input data dimension for feedforward neural network," in *Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94*, vol. 6. IEEE, 1994, pp. 447–450.

[150] J. M. Zurada, A. Malinowski, and S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, no. 2, pp. 177–193, 1997.

Appendix A

Results of Local sensitivity analysis of T_p noisy, J_t noisy and R_t noisy for 50 runs

Table A.1: Local sensitivity analysis for T_p noisy, normalised R_t, normalised J_t (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.2907	0.6820	-0.0230	-0.0632	0.0905
2	-0.1138	0.6911	0.0090	-0.0281	0.0908
3	-0.0876	0.7104	0.0102	-0.0265	0.0915
4	-0.5090	0.6848	-0.0166	-0.0583	0.0900
5	-0.1144	0.6929	-0.0195	-0.0619	0.0895
6	-0.1281	0.7314	-0.0024	-0.0315	0.0942
7	-0.1177	0.6969	-0.0114	-0.0529	0.0895
8	-0.1248	0.6730	-0.0141	-0.0516	0.0910
9	-0.1677	0.7188	-0.0065	-0.0457	0.0909
10	-0.2079	0.6964	-0.0054	-0.0469	0.0893
11	-0.1413	0.7068	-0.0036	-0.0449	0.0895
12	-0.1582	0.6808	-0.0221	-0.0578	0.0908
13	-0.0767	0.7080	0.0012	-0.0400	0.0897
14	-0.0764	0.7064	-0.0014	-0.0427	0.0893
15	-0.1062	0.7064	0.0001	-0.0420	0.0901
16	-0.2337	0.6871	-0.0126	-0.0543	0.0896
17	-0.1832	0.6967	-0.0237	-0.0620	0.0923
18	-0.1964	0.6935	-0.0072	-0.0483	0.0894
19	-0.2736	0.7796	0.0292	-0.0042	0.0950
20	-0.1658	0.6703	0.0019	-0.0355	0.0913
21	-0.1491	0.7115	-0.0012	-0.0438	0.0896
22	-0.1402	0.6364	-0.0170	-0.0478	0.0935
23	-0.1023	0.7022	0.0012	-0.0391	0.0901
24	-0.1567	0.7126	-0.0123	-0.0535	0.0894
25	-0.1739	0.6962	-0.0088	-0.0496	0.0893
26	-0.1628	0.6986	-0.0062	-0.0478	0.0892
27	-0.0838	0.7068	0.0095	-0.0278	0.0907
28	-0.0949	0.7559	0.0092	-0.0290	0.0903
29	-0.1309	0.7172	-0.0091	-0.0451	0.0916
30	-0.1619	0.7354	-0.0079	-0.0392	0.0942
31	-0.3041	0.7020	0.0162	-0.0231	0.1006
32	-0.0808	0.6946	-0.0086	-0.0499	0.0895
33	-0.0943	0.7100	0.0005	-0.0411	0.0898
34	-0.5080	0.6945	-0.0121	-0.0549	0.0903
35	-0.2063	0.6948	-0.0107	-0.0527	0.0895
36	-0.1662	0.7003	-0.0161	-0.0581	0.0900
37	-0.9084	0.7033	-0.0169	-0.0585	0.0912
38	-0.1037	0.7497	0.0053	-0.0349	0.0903
39	-0.2242	0.7425	-0.0207	-0.0351	0.1050
40	-0.1298	0.7061	-0.0028	-0.0450	0.0896
41	-0.1020	0.7055	-0.0071	-0.0482	0.0902
42	-0.1680	0.6896	0.0004	-0.0371	0.0903
43	-0.1595	0.6956	-0.0042	-0.0474	0.0894
44	-0.1175	0.6898	-0.0100	-0.0505	0.0908
45	-0.1828	0.7074	0.0107	-0.0279	0.0902
46	-0.1252	0.7173	-0.0016	-0.0435	0.0897
47	-0.1248	0.7038	-0.0100	-0.0521	0.0897
48	-0.1589	0.7035	0.0113	-0.0242	0.0917
49	-0.2439	0.6930	-0.0182	-0.0603	0.0894
50	-0.0753	0.7132	0.0036	-0.0390	0.0893

Table A.2: Local sensitivity analysis for R_t noisy, normalised J_t, normalised T_p (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.1461	0.6995	-0.0017	-0.0445	0.0898
2	-0.0896	0.6993	-0.0107	-0.0528	0.0895
3	-0.0943	0.7007	-0.0090	-0.0513	0.0893
4	-0.5218	0.6938	-0.0113	-0.0531	0.0899
5	-0.1139	0.6994	-0.0083	-0.0499	0.0893
6	-0.0945	0.6979	-0.0071	-0.0487	0.0892
7	-0.0981	0.6940	-0.0096	-0.0507	0.0892
8	-0.1146	0.6971	-0.0098	-0.0506	0.0892
9	-0.1631	0.6996	-0.0095	-0.0512	0.0893
10	-0.1984	0.7058	-0.0057	-0.0466	0.0894
11	-0.1032	0.6975	-0.0080	-0.0494	0.0894
12	-0.1517	0.7066	-0.0099	-0.0519	0.0895
13	-0.0881	0.6998	-0.0084	-0.0498	0.0893
14	-0.0793	0.6946	-0.0093	-0.0503	0.0893
15	-0.1021	0.6980	-0.0090	-0.0505	0.0894
16	-0.2043	0.7023	-0.0098	-0.0506	0.0893
17	-0.1486	0.7044	-0.0007	-0.0415	0.0898
18	-0.1946	0.6948	-0.0096	-0.0503	0.0893
19	-0.1317	0.6960	-0.0104	-0.0519	0.0895
20	-0.1294	0.6967	-0.0050	-0.0461	0.0893
21	-0.1607	0.7009	-0.0087	-0.0499	0.0893
22	-0.0946	0.6924	-0.0101	-0.0515	0.0892
23	-0.1305	0.7038	-0.0056	-0.0483	0.0897
24	-0.1373	0.7020	-0.0080	-0.0496	0.0894
25	-0.1825	0.6964	-0.0092	-0.0506	0.0896
26	-0.1729	0.6990	-0.0078	-0.0491	0.0893
27	-0.0916	0.7021	-0.0081	-0.0496	0.0893
28	-0.1033	0.7043	-0.0096	-0.0514	0.0895
29	-0.1202	0.7047	-0.0118	-0.0539	0.0897
30	-0.1071	0.7102	-0.0088	-0.0505	0.0895
31	-0.2968	0.6928	-0.0096	-0.0504	0.0895
32	-0.0793	0.6947	-0.0088	-0.0496	0.0893
33	-0.0814	0.6985	-0.0099	-0.0512	0.0892
34	-0.5114	0.6970	-0.0102	-0.0509	0.0896
35	-0.1939	0.6960	-0.0112	-0.0525	0.0894
36	-0.1635	0.7019	-0.0101	-0.0509	0.0896
37	-0.7064	0.6965	-0.0091	-0.0505	0.0900
38	-0.1086	0.7015	-0.0082	-0.0487	0.0897
39	-0.1278	0.7052	-0.0159	-0.0573	0.0905
40	-0.1604	0.6974	-0.0096	-0.0506	0.0893
41	-0.1781	0.7041	-0.0084	-0.0500	0.0895
42	-0.0909	0.6905	-0.0082	-0.0499	0.0892
43	-0.1863	0.6982	-0.0038	-0.0456	0.0898
44	-0.0860	0.7011	-0.0084	-0.0501	0.0897
45	-0.1440	0.6934	-0.0107	-0.0521	0.0893
46	-0.1094	0.6955	-0.0108	-0.0519	0.0893
47	-0.0889	0.6941	-0.0102	-0.0514	0.0892
48	-0.2157	0.6997	-0.0091	-0.0500	0.0895
49	-0.2500	0.7009	-0.0100	-0.0521	0.0895
50	-0.0836	0.6922	-0.0093	-0.0511	0.0894

Table A.3: Local sensitivity analysis for J_t noisy, normalised R_t, normalised T_p (over 3600 data samples) for 50 statistical runs

No of runs	Min	Max	Average	Median	Standard Deviation
1	-0.2783	0.6505	-0.0323	-0.0518	0.1307
2	-0.4036	0.7082	-0.0770	-0.0655	0.1249
3	-0.3362	0.6227	-0.0239	-0.0536	0.0962
4	-0.5330	0.7538	0.0612	0.0708	0.1207
5	-0.1335	0.7024	-0.0008	-0.0401	0.0900
6	-0.3098	0.6035	-0.0655	-0.0718	0.1099
7	-0.1327	0.6900	-0.0092	-0.0515	0.0896
8	-0.1724	0.6709	-0.0271	-0.0663	0.0911
9	-0.5250	0.6866	-0.1055	-0.0780	0.1356
10	-0.2824	0.6954	-0.0008	-0.0413	0.0900
11	-0.1689	0.6948	-0.0204	-0.0610	0.0905
12	-0.3031	0.7189	-0.0511	-0.0571	0.1076
13	-0.1432	0.6935	-0.0101	-0.0510	0.0907
14	-0.1075	0.6804	-0.0133	-0.0568	0.0895
15	-0.1131	0.7846	0.0043	-0.0342	0.1009
16	-0.2020	0.6969	0.0063	-0.0280	0.0923
17	-0.2129	0.6884	-0.0096	-0.0487	0.0912
18	-0.1135	0.7454	0.0056	-0.0320	0.0914
19	-0.3325	0.6425	-0.0747	-0.0896	0.1086
20	-0.1571	0.7642	0.0068	-0.0265	0.0958
21	-0.2269	0.6976	-0.0288	-0.0698	0.0922
22	-0.1246	0.7236	-0.0044	-0.0384	0.0926
23	-0.2143	0.6970	0.0207	-0.0112	0.0941
24	-0.1207	0.7043	-0.0066	-0.0431	0.0911
25	-0.1758	0.6967	-0.0123	-0.0432	0.0921
26	-0.1305	0.6945	0.0141	-0.0185	0.0917
27	-0.0850	0.7126	0.0452	0.0298	0.1082
28	-0.1426	0.6511	-0.0113	-0.0409	0.0927
29	-0.1435	0.6817	-0.0368	-0.0753	0.0904
30	-0.2022	0.7572	0.0150	-0.0066	0.1103
31	-0.1580	0.6980	-0.0162	-0.0568	0.0909
32	-0.2035	0.6875	-0.0215	-0.0599	0.0932
33	-0.0918	0.6951	-0.0092	-0.0505	0.0900
34	-0.6251	0.6365	-0.0271	-0.0646	0.0927
35	-0.1935	0.6802	-0.0224	-0.0655	0.0898
36	-0.3200	0.6875	-0.0080	-0.0365	0.0929
37	-0.8707	0.6955	-0.0220	-0.0444	0.1005
38	-0.1098	0.8213	0.0793	0.0825	0.1302
39	-0.2530	0.7218	0.0422	0.0240	0.1416
40	-0.4004	0.6906	-0.0036	-0.0431	0.0915
41	-0.1805	0.7446	0.0564	0.0499	0.1138
42	-0.2959	0.7020	-0.0802	-0.0986	0.1025
43	-0.1863	0.6472	-0.0226	-0.0638	0.0918
44	-0.0888	0.6865	0.0600	0.0537	0.1093
45	-0.2275	0.6672	-0.0260	-0.0657	0.0912
46	-0.2108	0.7184	-0.0326	-0.0568	0.0963
47	-0.2364	0.6950	-0.0108	-0.0379	0.0957
48	-0.4165	0.7079	0.0191	-0.0179	0.0912
49	-0.1906	0.7496	0.0007	-0.0350	0.0922
50	-0.0896	0.7296	0.0247	-0.0012	0.0998

Appendix B

LSA code in Mfile format

```
clear
clc
for test_run=1:50
    close all force
     simulation_data=xlsread('simulation_data.xlsx');
      % Throughput mathematical/statistical evaluation and normaliz
    Tp_data=simulation_data(:,1);
    Tp_data_norm=norm_func(Tp_data);
     % Jitter mathematical/statistical evaluation and normalization
    Jt_data=simulation_data(:,2);
    Jt_data_norm=norm_func(Jt_data);
       % Response mathematical/statistical evaluation and normaliza
    Rt_data=simulation_data(:,3);
    Rt_data_norm=norm_func(Rt_data);
    input=[Tp_data_norm,Jt_data_norm,Rt_data_norm];
    input=input';
    %Building the NN
    trainFcn = 'trainlm';
    hiddenLayerSize = 10;
    net = feedforwardnet(hiddenLayerSize,trainFcn); % ANN Model
       % net.input.processFcns = {'removeconstantrows','mapminmax'}
```

```
% net.output.processFcns = {'removeconstantrows','mapminmax'};
index=randperm(3600); % Over fitting
TrainData=[];
for i=1:2520
    TrainData(:,i)=input(:,index(i));
end
   ValidData=[];%15% for validation
for i=1:540
    ValidData(:,i)=input(:,index(i+2520));
end
   TestData=[];
for i=1:540 %15% for test
    TestData(:,i)=input(:,index(3060+i));
end
   new_data_input=[TrainData,ValidData,TestData];
    % Calculate Targets (Based on proposed Mathematical Cost Fu
  Lw_data=simulation_data(:,4); % Rescaling to distinguish simu
Lw_data=Lw_data*10;
    norm_simulation_data=[Tp_data_norm,Jt_data_norm,Rt_data_nor
    CF=zeros;
CF_new=zeros;
for i=1:size(norm_simulation_data,1)
    sample_data=norm_simulation_data(i,:);
    Tp=sample_data(1);
    Jt=sample_data(2);
    Rt=sample_data(3);
    Lw=sample_data(4);
    CF(i,:)=(abs(Tp-Jt)*Rt)*Lw;
end
targets=CF';
x=new_data_input;
t=targets;
net.divideFcn = 'divideind';
```

```
net.divideParam.trainInd = 1:2520;
net.divideParam.valInd = 2521:3060;
net.divideParam.testInd = 3061:3600;
net.performFcn = 'mse';
[net,tr] = train(net,x,t); % training ANN
y = net(x);
Normal_e = gsubtract(t,y); % deviations from the actual targets
Normal_performance = mse(net,t,y) % this is the MSE error value
%     view(net)
avg_Normal_y=mean(y);
med_Normal_y=median(y);
min_Normal_y=min(y);
max_Normal_y=max(y);
std_Normal_y=std(y);
avg_Normal_e=mean(Normal_e);
med_Normal_e=median(Normal_e);
min_Normal_e=min(Normal_e);
max_Normal_e=max(Normal_e);
std_Normal_e=std(Normal_e);
store_Normal_performance(test_run,:)=Normal_performance;
Normal_y_avg_result(test_run,:)=avg_Normal_y;
Normal_y_med_result(test_run,:)=med_Normal_y;
Normal_y_min_result(test_run,:)=min_Normal_y;
Normal_y_max_result(test_run,:)=max_Normal_y;
Normal_y_std_result(test_run,:)=std_Normal_y;
Normal_e_avg_result(test_run,:)=avg_Normal_e;
Normal_e_med_result(test_run,:)=med_Normal_e;
Normal_e_min_result(test_run,:)=min_Normal_e;
Normal_e_max_result(test_run,:)=max_Normal_e;
Normal_e_std_result(test_run,:)=std_Normal_e;
% Visualize cost function trend
plot(1:900,CF(1:900),'g-','LineWidth',2)
hold on
```

```
plot(901:1800,CF(901:1800),'b-','LineWidth',2)
hold on
plot(1801:2700,CF(1801:2700),'y-','LineWidth',2)
hold on
plot(2701:3600,CF(2701:3600),'r-','LineWidth',2)
hold off
  grid on
  set(gca,'FontSize',15,'FontWeight','bold');
xlabel ('Number of Simulated Scenarios','FontSize', 15, 'FontWe
ylabel ('Cost Function Value','FontSize', 15,'FontWeight','bold
legend1 = legend('Normal Operating Condition','TCP Flood Attack
set(legend1,'FontSize',10);
  %% Local Sensitivity Analysis
%% Noisy Tp
    % Throughput mathematical/statistical evaluation and normal
Tp_data=simulation_data(:,1);
% A_wnoise = A + sqrt(variance)*randn(size(A)) + meanValue;
% Variance = std ^ 2; std=1 for standard normal distribution
% mean = 0; standard normal distribution
rng('default') % reset random seed
rng('shuffle') % shuffle random seed using CPU time (Always bes
Tp_data_noise = Tp_data + sqrt(1)*randn(size(Tp_data)) + 0; %
Tp_data_norm=norm_func(Tp_data);
Tp_data_norm_noise=norm_func(Tp_data_noise);
input=[Tp_data_norm_noise,Jt_data_norm,Rt_data_norm];
input=input';
index=randperm(3600); % all simulation indexes
% [Training Data:Validation Data:Test Data]=[70%:15%:15%] to av
% 70% Training Data
TrainData=[];
for i=1:2520
    TrainData(:,i)=input(:,index(i));
end
```

```
% 15% Validation Data
ValidData=[];
for i=1:540
    ValidData(:,i)=input(:,index(i+2520));
end
% 15% Test Data
TestData=[];
for i=1:540
    TestData(:,i)=input(:,index(3060+i));
end
new_data_input=[TrainData,ValidData,TestData];
x=new_data_input;
Tp_y = net(x); % the prediction model is the same, only the inp
Tp_e = gsubtract(t,Tp_y); % deviations from the actual targets
Tp_performance = mse(net,t,Tp_y) % this is the MSE error value,
avg_Tp_y=mean(Tp_y);
med_Tp_y=median(Tp_y);
min_Tp_y=min(Tp_y);
max_Tp_y=max(Tp_y);
std_Tp_y=std(Tp_y);
avg_Tp_e=mean(Tp_e);
med_Tp_e=median(Tp_e);
min_Tp_e=min(Tp_e);
max_Tp_e=max(Tp_e);
std_Tp_e=std(Tp_e);
store_Tp_performance(test_run,:)=Tp_performance; % mse
Tp_y_avg_result(test_run,:)=avg_Tp_y;
Tp_y_med_result(test_run,:)=med_Tp_y;
Tp_y_min_result(test_run,:)=min_Tp_y;
Tp_y_max_result(test_run,:)=max_Tp_y;
Tp_y_std_result(test_run,:)=std_Tp_y;
Tp_e_avg_result(test_run,:)=avg_Tp_e;
Tp_e_med_result(test_run,:)=med_Tp_e;
```

```
Tp_e_min_result(test_run,:)=min_Tp_e;
Tp_e_max_result(test_run,:)=max_Tp_e;
Tp_e_std_result(test_run,:)=std_Tp_e;
%% Noisy Jt
% Jitter mathematical/statistical evaluation and normalization
Jt_data=simulation_data(:,2);
% A_wnoise = A + sqrt(variance)*randn(size(A)) + meanValue;
% Variance = std ^ 2; std=1 for standard normal distribution
% mean = 0; standard normal distribution
rng('default') % reset random seed
rng('shuffle') % shuffle random seed using CPU time
Jt_data_noise = Jt_data + sqrt(1)*randn(size(Jt_data)) + 0; %
Jt_data_norm=norm_func(Jt_data);
Jt_data_norm_noise=norm_func(Jt_data_noise);
input=[Tp_data_norm,Jt_data_norm_noise,Rt_data_norm];
input=input';
index=randperm(3600); % all simulation indexes
% [Training Data:Validation Data:Test Data]=[70%:15%:15%] to av
% 70% Training Data
TrainData=[];
for i=1:2520
    TrainData(:,i)=input(:,index(i));
end
% 15% Validation Data
ValidData=[];
for i=1:540
    ValidData(:,i)=input(:,index(i+2520));
end
% 15% Test Data
TestData=[];
for i=1:540
    TestData(:,i)=input(:,index(3060+i));
end
```

```
new_data_input=[TrainData,ValidData,TestData];
x=new_data_input;
Jt_y = net(x); % the prediction model is the same, only the inp
Jt_e = gsubtract(t,Jt_y); % deviations from the actual targets
Jt_performance = mse(net,t,Jt_y) % this is the MSE error value,
avg_Jt_y=mean(Jt_y);
med_Jt_y=median(Jt_y);
min_Jt_y=min(Jt_y);
max_Jt_y=max(Jt_y);
std_Jt_y=std(Jt_y);
avg_Jt_e=mean(Jt_e);
med_Jt_e=median(Jt_e);
min_Jt_e=min(Jt_e);
max_Jt_e=max(Jt_e);
std_Jt_e=std(Jt_e);
store_Jt_performance(test_run,:)=Jt_performance;
Jt_y_avg_result(test_run,:)=avg_Jt_y;
Jt_y_med_result(test_run,:)=med_Jt_y;
Jt_y_min_result(test_run,:)=min_Jt_y;
Jt_y_max_result(test_run,:)=max_Jt_y;
Jt_y_std_result(test_run,:)=std_Jt_y;
Jt_e_avg_result(test_run,:)=avg_Jt_e;
Jt_e_med_result(test_run,:)=med_Jt_e;
Jt_e_min_result(test_run,:)=min_Jt_e;
Jt_e_max_result(test_run,:)=max_Jt_e;
Jt_e_std_result(test_run,:)=std_Jt_e;
%% Noisy Rt
% Response mathematical/statistical evaluation and normalizatio
Rt_data=simulation_data(:,3);
% A_wnoise = A + sqrt(variance)*randn(size(A)) + meanValue;
% Variance = std ^ 2; std=1 for standard normal distribution
% mean = 0; standard normal distribution
rng('default') % reset random seed
```

```
rng('shuffle') % shuffle random seed using CPU time
Rt_data_noise = Rt_data + sqrt(1)*randn(size(Rt_data)) + 0; %
Rt_data_norm=norm_func(Rt_data);
Rt_data_norm_noise=norm_func(Rt_data_noise);
input=[Tp_data_norm,Jt_data_norm,Rt_data_norm_noise];
input=input';
index=randperm(3600); % all simulation indexes
% [Training Data:Validation Data:Test Data]=[70%:15%:15%] to av
% 70% Training Data
TrainData=[];
for i=1:2520
    TrainData(:,i)=input(:,index(i));
end
% 15% Validation Data
ValidData=[];
for i=1:540
    ValidData(:,i)=input(:,index(i+2520));
end
% 15% Test Data
TestData=[];
for i=1:540
    TestData(:,i)=input(:,index(3060+i));
end
new_data_input=[TrainData,ValidData,TestData];
x=new_data_input;
Rt_y = net(x); % the prediction model is the same, only the inp
Rt_e = gsubtract(t,Rt_y); % deviations from the actual targets
Rt_performance = mse(net,t,Rt_y) % this is the MSE error value,
avg_Rt_y=mean(Rt_y);
med_Rt_y=median(Rt_y);
min_Rt_y=min(Rt_y);
max_Rt_y=max(Rt_y);
std_Rt_y=std(Rt_y);
```

```
    avg_Rt_e=mean(Rt_e);
    med_Rt_e=median(Rt_e);
    min_Rt_e=min(Rt_e);
    max_Rt_e=max(Rt_e);
    std_Rt_e=std(Rt_e);
    store_Rt_performance(test_run,:)=Rt_performance;
    Rt_y_avg_result(test_run,:)=avg_Rt_y;
    Rt_y_med_result(test_run,:)=med_Rt_y;
    Rt_y_min_result(test_run,:)=min_Rt_y;
    Rt_y_max_result(test_run,:)=max_Rt_y;
    Rt_y_std_result(test_run,:)=std_Rt_y;
    Rt_e_avg_result(test_run,:)=avg_Rt_e;
    Rt_e_med_result(test_run,:)=med_Rt_e;
    Rt_e_min_result(test_run,:)=min_Rt_e;
    Rt_e_max_result(test_run,:)=max_Rt_e;
    Rt_e_std_result(test_run,:)=std_Rt_e;
end
% Perf Values % mse values (over 50 Runs)
avg_Normal_perf=mean(store_Normal_performance);
med_Normal_perf=median(store_Normal_performance);
min_Normal_perf=min(store_Normal_performance);
max_Normal_perf=max(store_Normal_performance);
std_Normal_perf=std(store_Normal_performance);
avg_Tp_perf=mean(store_Tp_performance)
med_Tp_perf=median(store_Tp_performance)
min_Tp_perf=min(store_Tp_performance)
max_Tp_perf=max(store_Tp_performance)
std_Tp_perf=std(store_Tp_performance)
avg_Jt_perf=mean(store_Jt_performance);
med_Jt_perf=median(store_Jt_performance);
min_Jt_perf=min(store_Jt_performance);
max_Jt_perf=max(store_Jt_performance);
std_Jt_perf=std(store_Jt_performance);
```

```
avg_Rt_perf=mean(store_Rt_performance);
med_Rt_perf=median(store_Rt_performance);   %when Rt is noisy, eval
min_Rt_perf=min(store_Rt_performance);
max_Rt_perf=max(store_Rt_performance);
std_Rt_perf=std(store_Rt_performance);
plot(1:50,store_Normal_performance,'-g','LineWidth',2)
hold on
plot(1:50,store_Tp_performance,'-b','LineWidth',2)
plot(1:50,store_Jt_performance,'-y','LineWidth',2)
plot(1:50,store_Rt_performance,'-r','LineWidth',2)
ylim([0.007 0.03])
grid on
set(gca,'FontSize',15,'FontWeight','bold');
xlabel ('Number of Runs','FontSize', 15, 'FontWeight','bold')
ylabel ('MSE Values (Over 50 Runs)','FontSize', 15,'FontWeight','bo
% legend1 = legend('Normal Operating Condition','Noisy Tp','Noisy
legend1 = legend('Normal','Noisy Tp','Noisy Jt','Noisy Rt');
set(legend1,'FontSize',10);
%legend('Location','northeastoutside')
% Hypothesis Test
% p-value
[p_Tp,h_Tp] = ranksum(store_Normal_performance,store_Tp_performance
[p_Jt,h_Jt] = ranksum(store_Normal_performance,store_Jt_performance
[p_Rt,h_Rt] = ranksum(store_Normal_performance,store_Rt_performance
```

Appendix C

FaTree topology code

```
"""Custom topology
 Directly connected switches plus hosts for fat tree topology:
   host --- switch --- switch --- host
Adding the 'topos' dict with a key/value pair to generate
our newly definedtopology enables one to pass
in '--topo=mytopo' from the command line.
"""
from mininet.topo import Topo
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch
class MyTopo( Topo ):
    "FaT tree topology."
    def__init__( self ):
        "Create custom topo."
        # Initialize topology
        Topo._init__( self )
        # Add hosts
        #leftHost = self.addHost( 'h1' )
        #rightHost = self.addHost( 'h2' )
h1=self.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
h2=self.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
```

```
h3=self.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
h4=self.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
h5=self.addHost('h5', cls=Host, ip='10.0.0.5', defaultRoute=None)
h6=self.addHost('h6', cls=Host, ip='10.0.0.6', defaultRoute=None)
h7=self.addHost('h7', cls=Host, ip='10.0.0.7', defaultRoute=None)
h8=self.addHost('h8', cls=Host, ip='10.0.0.8', defaultRoute=None)
h9=self.addHost('h9', cls=Host, ip='10.0.0.9', defaultRoute=None)
h10=self.addHost('h10',cls=Host,ip='10.0.0.10',defaultRoute=None)
h11=self.addHost('h11',cls=Host,ip='10.0.0.11',defaultRoute=None)
h12=self.addHost('h12',cls=Host,ip='10.0.0.12',defaultRoute=None)
h13=self.addHost('h13',cls=Host,ip='10.0.0.13',defaultRoute=None)
h14=self.addHost('h14',cls=Host,ip='10.0.0.14',defaultRoute=None)
h15=self.addHost('h15',cls=Host,ip='10.0.0.15',defaultRoute=None)
h16=self.addHost('h16',cls=Host,ip='10.0.0.16',defaultRoute=None)
# Add switches
        #leftSwitch = self.addSwitch( 's3' )
        #rightSwitch = self.addSwitch( 's4' )
s1 = self.addSwitch('s1', cls=OVSKernelSwitch)
s2 = self.addSwitch('s2', cls=OVSKernelSwitch)
  s3 = self.addSwitch('s3', cls=OVSKernelSwitch)
        s4 = self.addSwitch('s4', cls=OVSKernelSwitch)
        s5 = self.addSwitch('s5', cls=OVSKernelSwitch)
        s6 = self.addSwitch('s6', cls=OVSKernelSwitch)
        s7 = self.addSwitch('s7', cls=OVSKernelSwitch)
        s8 = self.addSwitch('s8', cls=OVSKernelSwitch)
s9 = self.addSwitch('s9', cls=OVSKernelSwitch)
        s10 = self.addSwitch('s10', cls=OVSKernelSwitch)
        # Add links between switches
        #self.addLink( leftHost, leftSwitch )
        #self.addLink( leftSwitch, rightSwitch )
        #self.addLink( rightSwitch, rightHost )
        self.addLink(s1, s3)
        self.addLink(s1, s5)
```

```
        self.addLink(s2, s4)
        self.addLink(s2, s6)
        self.addLink(s3, s7)
        self.addLink(s3, s8)
        self.addLink(s4, s7)
        self.addLink(s4, s8)
        self.addLink(s5, s9)
        self.addLink(s5, s10)
        self.addLink(s6, s9)
        self.addLink(s6, s10)
# Add links between hosts
        self.addLink(h1, s7)
        self.addLink(h2, s7)
        self.addLink(h3, s7)
        self.addLink(h4, s7)
        self.addLink(h5, s8)
        self.addLink(h6, s8)
        self.addLink(h7, s8)
        self.addLink(h8, s8)
        self.addLink(h9, s9)
        self.addLink(h10, s9)
        self.addLink(h11, s9)
        self.addLink(h12, s9)
        self.addLink(h13, s10)
        self.addLink(h14, s10)
        self.addLink(h15, s10)
        self.addLink(h16, s10)
topos = { 'mytopo': ( lambda: MyTopo() ) }
```