Efficient Malicious Packet Detection in Software Defined Networks

Perekebode Amangele

A thesis submitted for the degree of

Doctor of Philosophy

Department of Computer Science and Electronic Engineering

University of Essex

December 2022

Summary

The emergence of software defined networking is proving to be a strong platform for future networks due to its advantages with regards to management of enterprise networks. One of the key areas where SDN is altering the management of networks is in the area of security. Additionally, the use of machine learning methods for network security is becoming increasingly investigated by both academia and industry. However, existing machine learning based intrusion detection systems can be highly computationally intensive. This forms the motivation for this research that combines the SDN architecture with machine learning to provide greater efficiency for intrusion detection. In this work, a novel solution using a hierarchical machine learning approach based on a SDN topology is presented. In the proposed solution, the SDN architecture uses machine learning at the controller that allows an efficient malicious packet detection mechanism at the edge of the network when compared to a flat traditional architecture. The methodology deployed includes a unique sub-flow classification based on supervised learning. The solution is demonstrated with the widely used CICIDS 2017 and the CICIDS 2018 datasets. The thesis shows significant savings in the traffic processed at the edge for the purposes of intrusion detection. Results show as much as a 74-98% savings in traffic processed for intrusion detection resulting in as much as 1000 times decrease in processing time (or increase in prediction rate) for malicious packet detection. The results also demonstrate that the savings have no significant impact on the overall accuracy of the system. This represents significant savings in network resources allowing for scarce network/computing resources to be deployed to other services. This work is a significant contribution to machine learning based network security research and forms a strong basis for future research in the security of next generation networks.

Acknowledgements

My profound gratitude to my wife, Margaret and my daughter Preye who have both being incredibly helpful and my major source of motivation.

My sincere appreciation to Professor Martin Reed for being a tremendous and helpful guide and supervisor.

I also acknowledge my sponsors, the Petroleum Technology Development Fund (PTDF), Nigeria.

May thanks to all those who helped to make this journey a success whose names I may not be able to mention here.

Thank you all and God bless.

List of Acronyms

BOTNET - Robot Network

CART - Classification and Regression Tree

CIC-IDS - Canadian Institute of Cybersecurity - Intrusion Detection Dataset

DDoS - Distributed Denial of Service

DoS - Denial of Service

eCDF - Empirical Cumulative Distribution Function

FTP - File Transfer Protocol

HIDS - Host Intrusion Detection System

HOIC - High Orbit Ion Cannon

IDS - Intrusion Detection Systems

IP - Internet Protocol

KNN - K-Nearest Neighbour

LDA - Linear Discriminant Analysis

LOIC - Low Orbit Ion Cannon

LR - Logistic Regressor

ML - Machine Learning

NB - Naive Bayes

NIDS - Network Intrusion Detection System

PCAP - Packet Capture

PSCAN - Port Scan

SDN - Software Defined Networks

SQL - Structured Query Language

SSH - Secure Shell

SVC - Support Vector Classifier

TCAM - Ternary Content-Addressable Memory

TLS - Transport Layer Security

WAN - Wide Area Networks

XSS - Cross-Site Scripting

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The emergence of Software-defined Networking (SDN) [2] promises to address several of the difficulties associated with managing traditional networks. This is because, IP networks, despite their widespread adoption on the internet architecture, are complex and difficult to manage [3]. This simplified network management, stems largely from the introduction of centralised control and network programmability. The SDN landscape also promises to alter network security administration as network administrators look to take advantage of SDN to improve security services on the network. SDNs offer increased ability to collect network statistics data which allows OpenFlow [4] [5] enabled forwarding devices to collect traffic information in a timely manner which is very useful for security mechanisms responsible for intrusion detection. This, in addition to the centralised processing capabilities and network-wide view available to SDN further allows the SDN to serve as a platform to improve the security mechanism of networks.

Existing network security solutions include the use of firewalls, antivirus and intrusion detection systems [6]. Network intrusion detection systems (NIDS) [7] are a critical security mechanism in enterprise networks. Several researchers have explored the implementation of intrusion detection systems in the fast growing SDN landscape [8] [9] [6] [7] [10]. A major motivation for this research is the conviction that the SDN architecture provides some unique features which potentially improves the secure administration of networks. This work will explore this improvement in terms of early attack detection and efficiency. One area of network intrusion detection that has received a lot of interest recently is machine learning based intrusion detection systems. Existing machine learning based network security research mostly focus on applying machine learning algorithms on historic data with little recourse to the real-time implications of such studies on the IP network. Most

research work in this domain, also do not take into consideration, the possible integration of their evaluations in real network architectures. Work done in this thesis is aimed at bridging this gap in the literature. Also, it is critical to note that the time it takes to identify an attack usually results in a small amount of attack leakage [11]. One of the main contributions of this work is to propose mechanisms to reduce attack leakage resulting from delays in attack detection.

This work focuses on improving the security of SDN based enterprise networks using a novel hierarchical intrusion detection system. The specific areas of improvement include real-time intrusion detection using IP flow information. This involves sub-flow classification (flow-based) of IP traffic which targets IP traffic classification at levels as low as 1% of traffic flow duration. Another area of improvement achieved by this work is in reduced edge processing for intrusion detection. Enterprise networks still feature fully intelligent edge devices even when instances of SDN solutions are deployed. This work pushes that boundary a little further with a view to reducing the security workload at the edge. This work also demonstrates potentially improved intrusion detection rate by deploying a novel hierarchical intrusion detection solution based on the software defined network architecture.

The hierarchical intrusion detection solution deploys machine learning [12] for its operation. The need for the use of Machine Learning as a network security tool is to improve automation. This further breaks the human loop in the network security pipeline. This eliminates the need to have domain experts looking at the data and trying to infer the presence of an malicious traffic. Machine learning also adds additional intelligence powered by the data-driven learning process. This work will form a strong platform for advancements in Artificial Intelligence (AI) based Cybersecurity.

1.1 Problem Statement

With the increasing volume and complexity of network traffic and applications, network intrusion detection requirements have also become more challenging. The future of network intrusion detection favours a flow based approach as against a knowledge based or payload based approach for reasons such as data encryption amongst others. One of the ways to leverage the flow based approach is by the use of machine learning methods. However, machine learning based intrusion detection systems can be highly computationally intensive. Figure 1.1 gives a fundamental illustration of this problem. It is not unusual for an edge intrusion detection system to be required to process packets at about 100Gb/s. This requires a lot of computational power to be able to

keep up with real-time applications. This potentially impacts significantly on the total cost of ownership (TCO) [13] of the network. This leads to the research question: given the computationally intensive nature of machine learning based IDS, can we leverage on the SDN architecture to provide an efficient malicious packet detection solution at the edge of the network?

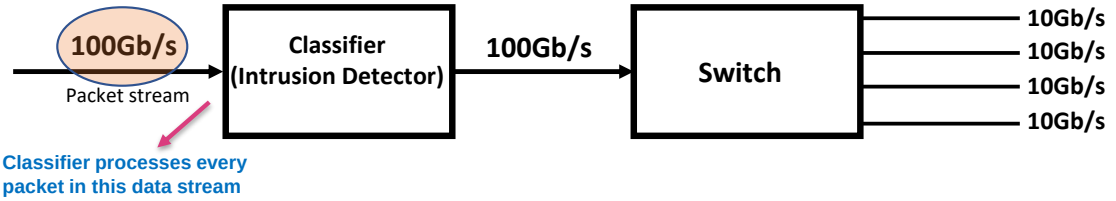


Figure 1.1: Inefficient Intrusion Detection without Hierarchical Solution

1.2 Proposed 2-Stage Hierarchical Machine Learning Based SDN Security Solution

This thesis proposes a novel, two-stage hierarchical machine learning process, integrated into SDN architecture for efficient network traffic intrusion detection and mitigation. One major advantage of using SDN, is its ability to effect network-wide security rules as opposed to the local policy implementation of traditional networks [14]. In addition, the centralised nature of the SDN controller holds significant bearing towards the application of machine learning for network management, monitoring and security. The work done in this thesis explores a unique approach in this regard. Hence, the SDN controller is leveraged to offer a lightweight machine learning instance which combines with a second stage edge classifier to provide efficient intrusion detection. Also, an advantage of using machine learning methods are their ability to use flow-based information which eliminates the rigours of packet payload inspection techniques used to create rule-based intrusion detection systems. This solution is based on a unique combination of SDN and machine learning. Figure 1.2 illustrates a potential deployment of the proposed hierarchical solution in a campus implementation. The proposed solution would also be applicable to a data center and other enterprise network segments.

The idea behind this solution is to deploy a lightweight traffic classifier at the controller with the aim of detecting anomalies early on in the lifespan of the flow. The results of this early detection allows for only a select portion of packets to be sent to the edge classifier. This offers a considerable reduction in processing requirement for intrusion detection at the edge. A high level

illustration of the proposed solution is shown in Figure 1.3. Please note that the values shown in this figure are solely for illustration. Actual experimental values are presented in Chapters 5 and 6. The work of this thesis in Chapters 4-6 shows a considerable reduction in processing effort at the edge with an almost insignificant impact on the overall accuracy of detection.

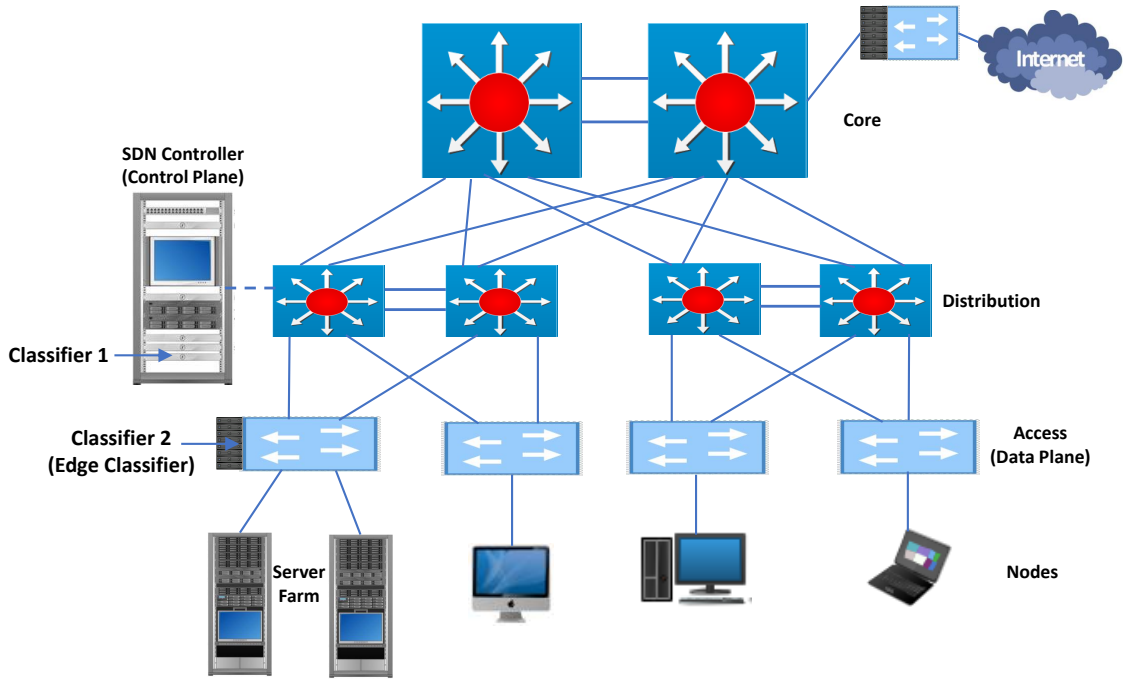


Figure 1.2: Enterprise Campus Implementation

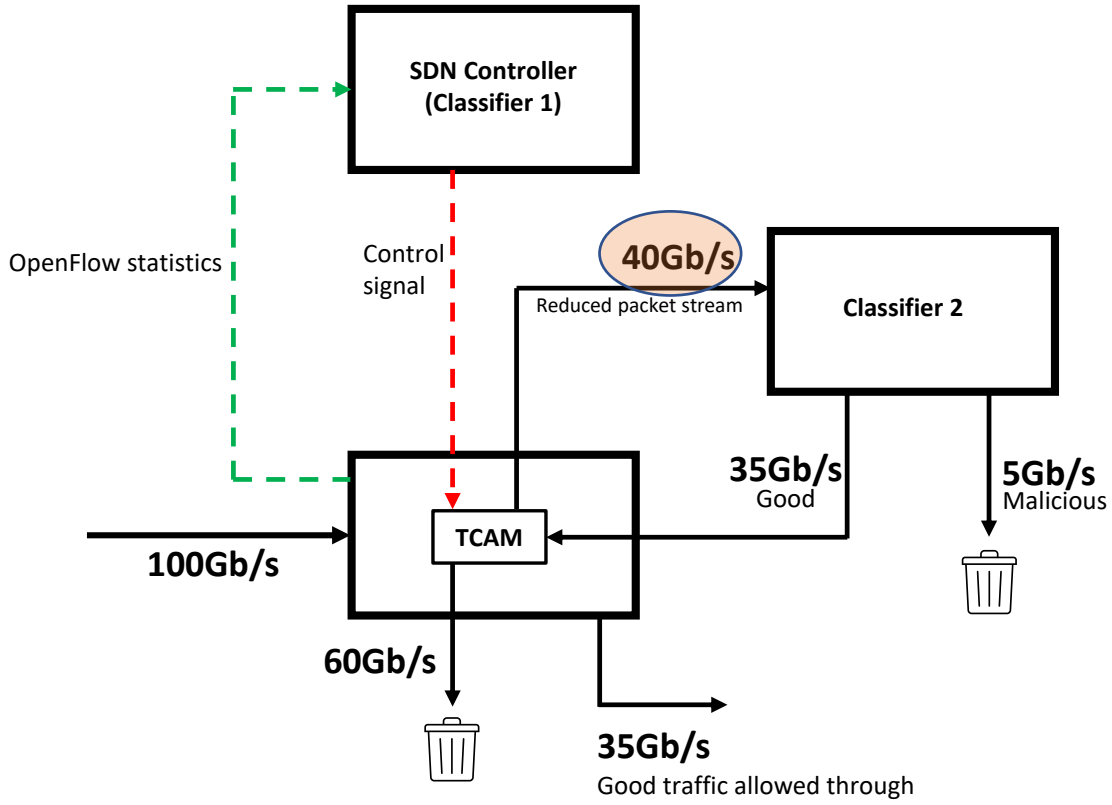


Figure 1.3: Efficient Intrusion Detection using Hierarchical Solution

1.3 Research Methodology

This work will take a modelling approach using realistic data flows. The data flows will be obtained using IP packets from existing intrusion detection datasets that are described in Chapter 2. Due to difficulties in obtaining real-time and real-life intrusion detection datasets, we experiment with existing historic data, by re-engineering the data to create the flexibility to simulate a real-time system. The model is implemented using python based frameworks to process packets into flows. The complete process for creating this experimental model is described in Chapter 5. Afterwards, existing machine learning algorithms from SciKit Learn [15] are used to simulate the malicious packet detection process. It should however be noted that this thesis is about the application of machine learning and not about the machine learning itself. In practice, for final deployment, a great deal of optimisation of the machine learning may be required as is commented in Chapter 7. The entire solution is modelled in python to allow for sufficient flexibility to demonstrate the concepts required to demonstrated that the research question is solved.

1.4 Research Scope, Outline and Contributions

This works provides an efficient intrustion detection system utilising SDN. It should be noted that this work does not focus on securing SDN but focuses on leveraging SDN to secure the enterprise network. In addition, this work does not aim to exhaustively investigate the optimisation of the machine learning process but rather focuses on using machine learning as a tool for achieving specialised network security.

The main contributions of this thesis are:

- Proposes a novel hierarchical machine learning based SDN security solution that offers improved packet processing efficiency when compared to non-hierarchical traditional machine learning based intrusion detection systems.
- Using a sub-flow based classification that goes beyond the approach of most works that carry out machine learning based intrusion detection using *historic*[1] static data.
- The proposed solution forms the basis for achieving near real-time attack detection in SDN.
- Offers improved understanding on the response of different attack types towards sub-flow based traffic classification.
- Creates a modification to the IPFIX architecture to allow for sub-flow traffic generation. This enables sub-flow based real-time malicious packet detection.
- This work forms a strong platform for future research in the security of next generation networks.

This work has been published:

- P. Amangele, M. J. Reed, M. Al-Naday, N. Thomos and M. Nowak, "Hierarchical Machine Learning for IoT Anomaly Detection in SDN," 2019 International Conference on Information Technologies (InfoTech), 2019, pp. 1-4, `https://doi.org/10.1109/InfoTech.2019.8860878` (Work from Chapter 3)
- P. Amangele and M. J. Reed, "Federated learning for intrusion detection in software defined networks," to be submitted for publication to IEEE International Conference on Communications 2023. (Work from Chapter 4 and 5)

[1] *i.e.,* all the packets/flow data after the attack is complete

1.5 Thesis outline

After this introduction, Chapter two gives a review of relevant literature while identifying gaps in literature with regards to work carried out in this thesis. The chapter also reviews key background concepts and technologies required for this work. Chapter three provides the platform for the design of the intrusion detection model that will be deployed in the proposed solution. Chapter four gives a detailed description of the experimental design. This includes the data re-engineering and the traffic classification processes. Chapter four also presents results obtained from the simulation of the proposed hierarchical SDN security solution. Chapter five presents the processes and results obtained from modifying the hierarchical solution for improved detection efficiency. Finally, Chapter 7 comments on the results achieved, describes limitations of the work and suggests future work.

Chapter 2

Background

2.1 Introduction

This chapter explains the background technologies and concepts which have been investigated or utilised in this work. Such concepts include: Software Define Networking (SDN), network security, machine learning, intrusion detection systems, intrusion detection datasets amongst others. This is necessary to provide the foundation knowledge required to understand the work done in later chapters. This chapter also discusses similar research works which have been carried out in the area of machine learning based intrusion detection systems and other network security applications. This chapter also highlights the gap in literature and technology, which this work addresses.

2.2 Network Security Concepts

Network security is a critical subset of the cybersecurity [16] landscape. The American National Research Council describes a cyberattack as the intentional modification, disruption, disassembly and destruction of computer systems and networks or information and programs transmitted or contained in computer systems and networks [17]. According to [18], Cybersecurity can be defined as securing hardware, software, data and information that exists in an online system (internet) from any form of breach. While [16] refers to Cybersecurity as the act of ensuring that the cyberspace is safe from damage or threat, [18] defines Cybersecurity as securing hardware, software, data and information that exists in an online system (internet) from any form of breach. According to the UK's National Cyber Security Centre, 39% of businesses report identifying a cyber attack with the average cost of each cyber attack estimated to be about £4,200 [19]. Recent security breaches

include the May 2022 attack on Cisco, where the attackers conducted a series of sophisticated voice phishing attacks on Cisco's network [20] [21]. In a similar report, Finland's parliamentary website was the target of a DDoS attack in August 2022, following the country's move to join NATO [22].

It can be seen that the network is a constant parameter in every definition and description of Cybersecurity. Cybersecurity generally seeks to protect data and applications domiciled on the network and also data in transit across networks. Therefore network security techniques contribute significantly towards achieving a healthy cyberspace. Network security breaches have the potential to disrupt e-commerce, compromise business data, breach people's privacy and distort the integrity of information [23]. In extreme cases, such breaches may lead to a threat to national security. Anomaly detection [24] [25] is an important subset of network security. This research focuses on network traffic intrusion detection in SDN.

2.2.1 The Network Security Model

The broad requirement for network security is expected to follow the CIA (confidentiality, integrity, availability) model [26]. This is also known as the CIA triad. This is shown in Figure 2.1. The actual origin of this model seems unclear. However, the term 'CIA' as relates to information security has become very popular since the turn of the 20th century with the massive expansion of the information ecosystem. Today, the CIA triad is a widely accepted model in both information security and Cybersecurity scenarios [27]. The three components of the CIA triad represents the three important network security factors [28]. As Figure 2.1. shows, the intersection of these three components is a fundamental requirement for for a secure network infrastructure. Work done in this research impacts on all three components of the CIA triad. The novel hierarchical machine learning security solution proposed in this research improves real-time intrusion detection in SDN. It also improves the efficiency of attack detection. The specific attacks and the datasets will be introduced later, however, the attack types addressed in this thesis are listed below under the appropriate CIA categories.

Confidentiality

Data confidentiality is the ability to keep the data in an information system secret [29]. This is usually achieved using techniques such as Data Encryption and Access Control [26]. This component of the CIA triad ensures that only users who are authorised to access specific data and other

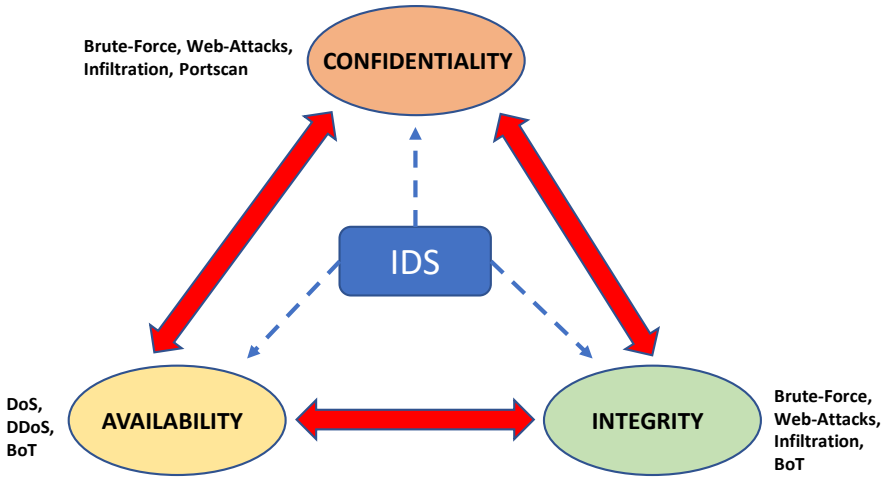


Figure 2.1: Network Security Model

resources on the network are able to do so. Similarly, and equally importantly, the confidentiality requirement also dictates that users who are are not explicitly authorised are actively prevented from gaining access to specific network resources. An effective intrusion detection system contributes to data confidentiality in an information system. Timely detection of malicious traffic can prevent unauthorised users from accessing exclusive network resources thus maintaining the confidentiality requirement of the network. The attack types utilised in this thesis that fall under this category include brute force attacks [30] [31], web attacks [32], infiltration and portscanning [33] [34]. Later we will see brute force attacks within the data set in this thesis include credential stuffing by using common username/passwords in SSH, FTP and web applications; this would allow an attacker to gain access to systems and at least break confidentiality. The web attacks (in addition to brute force) include SQL injection, cross-site scripting (XSS) which can be used to obtain data maliciously. Infiltration is a generic term for gaining access to a system, in this thesis the utilised data set sent an email with a payload used to scan and exploit internal systems [1]. Finally portscanning is where an attacker scans available systems to determine their open ports and the services running on those ports. This is a reconnaissance attack that is usually the first step in breaking confidentiality. Portscan attacks can also serve as an enabler for DDoS and other network based threats [33].

Integrity

Data integrity is the prevention of data in an information system from alteration and is usually implemented using the hash function (MD, MD5, SHA-1, SHA-126) [29] [35] [36]. Also, data

integrity is the assurance that data in an information system has not been modified, either in storage or in transit, either by accident or intentionally [37]. This is one of the most critical requirements in any information system as this forms the basis for cyber attacks that may lead to breaches resulting in financial fraud with devastating effects. Different strategies are required to prevent the modification of data in transit and data in storage. Different network security strategies prevent data in motion from being intercepted and modified by man-in-the-middle [38] attacks. Also, different network intrusion detection systems (IDS) [39] prevent unauthorised entities that may have the intention of unlawfully modifying data, from accessing such stored data. The attack types used in the thesis that fall under this category include again brute force attacks, web attacks and infiltration, as explained above under confidentiality, they can all lead to integrity issues as well.

Availability

The data availability component of the CIA triad ensures the timely and reliable access to data and other network resources [37]. This means that authorised users must have uninterrupted access to relevant network resources. In network security, availability is one of the strategies that ensures information security. This is because network security strategies protect network devices and ensures that data and other critical network resources are available. An effective intrusion detection system can prevent Denial of Service (DoS) [40] attacks which may be intended to render critical network resources to be unavailable to legitimate users. In this thesis, the primary attack types from the dataset which address availability are DoS [41], DDoS [42] [33] and Bot attacks [43] [44]. Note that Bot attacks are used to launch a large number of attacks, the volume often appears like a form of DoS hence it is useful to include them under Availability, however, they may be used to also launch high-volume attacks that target confidentiality or integrity. In fact, it can be argued that the other earlier attack types can also be used for availability attacks as well, for example an SQL injection that deletes a data base. Additionally, availability attacks can be used as first steps to mask confidentiality/integrity attacks.

A DDoS attack is a particular category of availability attacks that attempts to disrupt and deny network services to legitimate users by overwhelming the target resource with extremely high volume of malicious requests [45]. This usually results to huge losses for service providers and businesses. With the increasing sophistication of networks, DDoS attacks have also become increasingly sophisticated. Attackers now utilise malicious software applications to gain control of

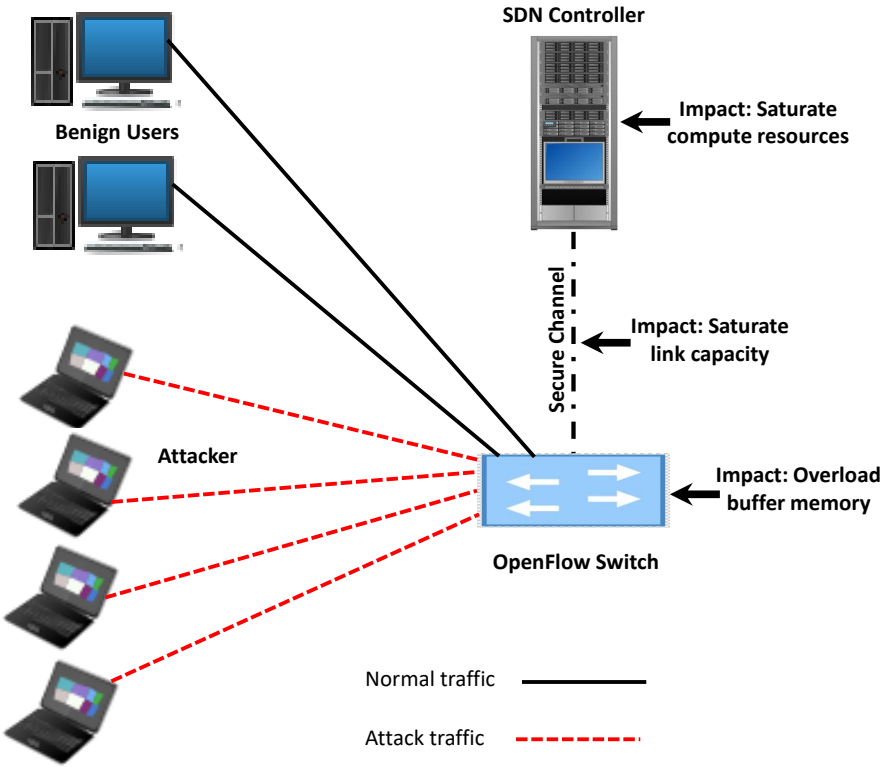


Figure 2.2: DDoS Attack in SDN

a collection of machines within a network [46]. These compromised machines are then remotely manipulated to launch simultaneous attacks to the victim. In addition to loss of revenue resulting from down time caused by DDoS attacks, businesses also incur additional costs due to recovery operations [47]. The common types of DDoS attacks include; http flood attacks, Slow and Low-rate DDoS attacks, Syn Flooding attacks, UDP Flooding attacks, ICMP Flooding attacks and DHCP Flooding attacks. Figure 2.2 illustrates the general architecture of DDoS attacks with reference to SDN. DDoS attacks are included in the work of this thesis as the chosen dataset includes DDoS attack generated using both the so called *High Orbit Ion Cannon* and *Low Orbit Ion Cannon* to create HTTP application layer flooding attacks from multiple source addresses.

2.3 Intrusion Detection Systems

In today's world, corporations and even countries leverage on the internet and the enterprise network, to expand business frontiers. The internet and other advancements in computing and networking technologies have also had a profound impact on governance and national/international security. Governance and business needs have driven enterprises and governments to develop

complicated computing and networking systems [48]. This is even made more demanding with the emergence of internet of things (IoT) [29]. These networks incorporate a complicated array of technologies including very high end servers, massive storage systems, wireless technologies, unified voice, data and video services, etc in addition to other sophisticated applications. Modern demands have required these networks to be more open to diverse array of users such as employees, business partners, customers, citizens etc. The different network access points incorporated to achieve this makes modern networks more more vulnerable to attacks and breaches [48]. This has led to the development of various intrusion detection systems(IDS) and intrusion prevention systems(IPS) [49] [50]. However, this thesis will focus more on intrusion detection systems.

An intrusion detection system is a device or software application responsible for detecting actions that may compromise the confidentiality, integrity and availability of a computing system or network. Its objective is to observe the activities in a network and identify possible breaches or attacks [51]. An intrusion detection system can also be described as any mechanism that observes and controls suspicious actions and policy violations in a network [52]. There are two main types of intrusion detection systems: Host based intrusion detection systems (HIDS) [53] and network based intrusion detection systems (NIDS) [54]. For HIDS, the IDS is deployed separately on a single host and is responsible for monitoring the activities and characteristics of the particular host [55]. A network based IDS is usually located at network points such as routers and firewalls and monitors traffic for a specific network segment in order to detect network intrusions. Both types of IDSs sends alerts to network administrators and users if suspicious activities are detected.

There are three major mechanisms used for intrusion detection systems: signature based IDS, anomaly-based IDS. and hybrid detection [6]. Each category is defined by how it identifies network attacks [51] [56] [57]. Figure 2.3 shows the different detection mechanisms of network intrusion detection systems. The signature and anomaly based systems are described briefly in the next subsections. The hybrid detection system integrates both signature based and anomaly based techniques in attack detection.

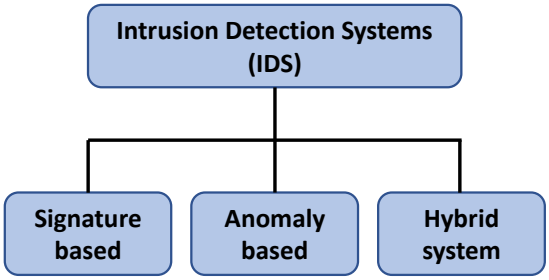


Figure 2.3: Classification of Intrusion Detection Systems

2.3.1 Signature-based IDS

Signature-based IDS also called misuse detection maintain a database of the knowledge base or rules used for detecting known attack types. Signature based detection techniques can further be classified into knowledge based techniques and machine learning based techniques. In knowledge based technique, network traffic or host audit data (such as log outputs) are compared against predefined attack patterns or rules contained in the existing database. However, this method of intrusion detection has its disadvantages. The major disadvantage is that knowledge based IDS can only detect attacks whose signatures are available in the IDS database. The dynamic nature of network attacks means that new attacks can go undetected with a resultant increase in the false negative rate. The continuous emergence of new attacks makes the signature update complicated. Another disadvantage is the amount of time it takes to process the signature lookup as every instance requires looking through the entire database. Additionally, knowledge based IDS is a payload-based traffic identification technique which examines the entire payload of the packets. The evolution of transport layer security (TLS) and other secure communication techniques, ensures that network traffic is fully encrypted and hence payload based traffic identification mechanisms are becoming less applicable. Also legal issues relating to payload inspection have become more crucial in recent years. These amongst other reasons have given rise to the need for flow based traffic identification mechanisms. Knowledge based techniques can be further classified into signature matching, state transition analysis and rule based expert systems. Figure 2.4 gives a hierarchical view of the different techniques in signature based IDS.

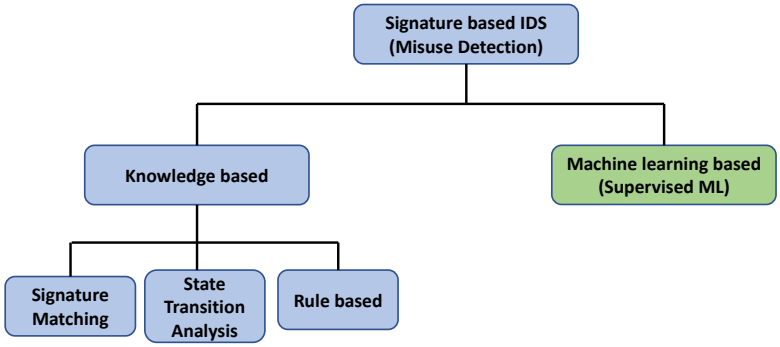


Figure 2.4: Classification of Intrusion Detection Systems

Machine learning based IDS has been attracting increased attention recently. Machine learning based detection (under Signature based IDS also called misuse detection) utilises a learning based system to identify attack types based on learned normal and attack behaviour. Misuse detection utilises supervised machine learning methods. The goal of this method is to create a general representation of known attack types. This method provides very high detection accuracy for known attacks. As expected, this technique would struggle to detect unknown attacks. Just like knowledge based IDS, the machine learning based IDS also requires regular update and maintenance of the signature database which impacts the operational overhead Generally, some of the advantages of deploying a machine learning based IDS over traditional knowledge based IDS may be summarized as follows:

- Traditional knowledge based IDS can be easily bypassed by implementing slight variations in known attack patterns. However, machine learning IDS based on either continuously trained supervised learning or unsupervised learning can easily identify the different attack variants as it learns the characteristics and behaviour of the traffic flow.
- The complex properties of the attack behaviour as embedded in the traffic can be captured by machine learning based IDS. This improves the accuracy and speed of detection when compared with traditional knowledge based systems.
- Very high detection accuracy of known attacks.
- Machine learning based IDS are not affected by encrypted communication protocols which may have been deployed on the network

However, one of the disadvantages of machine learning IDSs are that machine learning tends to be computationally highly intensive. Indeed this gives one of the strongest motivations for this

work: how to apply machine learning for intrusion detection in a manner that increases efficiency. It should be noted that advantages resulting from unsupervised machine learning techniques are not listed here. Work done in this thesis is limited to machine learning IDS based on supervised learning as many researchers comment that supervised learning is most appropriate for specific attack detection, as opposed to general anomaly detection we consider next, see the survey of the area by Mishra *et al.* [6]. Further discussion on machine learning based IDS techniques is provided in Section 2.9.

2.3.2 Anomaly-based IDS

Anomaly based intrusion detection systems are based on the hypothesis that the behaviour of attack traffic is different from that of normal traffic [58]. This type of IDS models the normal behaviour of the system and looks to identify patterns whose behaviour is not normal or expected [59]. This method utilizes collected historical data related to the behaviour of legitimate traffic or users to create a model or baseline traffic pattern. Incoming traffic is compared with the model and any significant deviation from the model is classified as an anomaly. A major advantage of this method of intrusion detection is that it can detect new types of attack and thus has a high capacity for low false negative rate. This advantage is the result of the system's ability to model the normal operation of a network and detect deviations from the model. Some disadvantages of this technique include the intrinsic complexity if the system and its high false positive rate. Anomaly detection techniques are broadly classified into three types: statistical techniques, machine learning based techniques and finite state machine based techniques. It should be noted that the machine learning techniques used in an anomaly-based IDS are mostly semi-supervised and unsupervised learning methods. However, this thesis does not aim to cover anomaly detection. Although the general high-level approach could be potentially re-engineered to provide anomaly detection.

2.4 What is an IP Flow?

The main purpose of a data network is the transmission of data from one point represented by a device to another point. This transfer of data is possible through the routing of IP packets from from one point to another. IP flow is an important concept in the context of this research. The machine learning based intrusion detection solution proposed in this work is a flow-based system, hence a thorough understanding of an IP flow is essential. According to the Internet Engineering

Task Force as stated in RFC 7011, "a flow is defined as a set of packets or frames passing an Observation Point in the network during a time interval" [60]. An IP flow is essentially a stream of data packets defined by a specific set of properties. Generally, a flow is defined by the 5-tuple properties of source IP address, destination IP address, source port number, destination port number, and transport layer protocol. These properties bunch all the packets in one direction of communication in a single socket. The source and the destination IP addresses specify beginning and the end of the Observation Point as stipulated in the RFC definition. This is illustrated in Figure 2.5.

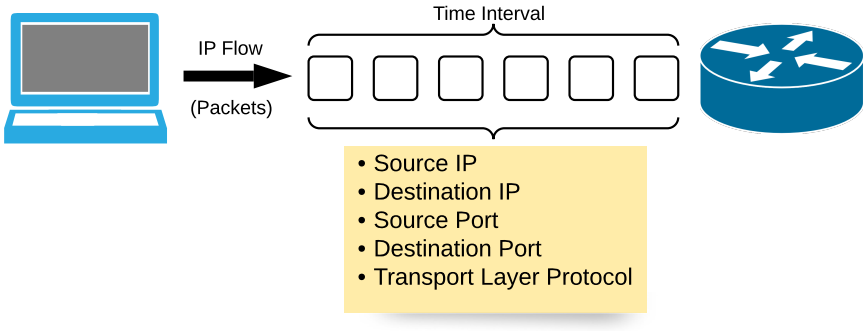


Figure 2.5: IP Flow

2.4.1 IP Flow Record

A flow record is a collection of information about a specific flow instance. The collected information about a flow is generally grouped into two categories: measured properties of the flow and characteristic properties of the flow. The measured properties of the flow are the statistical measurements derived from the flow of the packets such as total number of packets, total number of bytes, packet inter arrival times etc. Characteristic properties are the intrinsic properties of the flow packets such as source and destination IP addresses. A sample flow record from the CICIDS2017 Dataset is shown in Figure 2.6

	FlowID	SourceIP	DestinationIP	SrcPort	DstPort	Timestamp	FwdPkts	BwdPkts	FlowDur	FwdPktLer	BwdPktLer	FwdPktLer	BwdPktLer	FwdPktLer	BwdPktLer
0	192.168.	192.168.1	192.168.10.3	63567	53	05/07/2017 10:17	4	2	0.06061	71	155	0	0	71	155
1	192.168.	192.168.1	173.241.242.22	46124	443	05/07/2017 02:10	11	6	6.15606	77.18182	646.1667	71.2573	759.7174	40	52

Figure 2.6: Sample Flow Record Showing Some of the Representative Fields

2.4.2 Flow Duration

The IETF in RFC5102 [61] describes flow duration as the time difference between the first observed packet of a flow and the last observed packet of the same flow. There are various conditions that may define flow duration. According to RFC5102, the following properties of a flow determine the flow duration:

- **Idle timeout:** A flow may be terminated if no packets belonging to the flow have been observed within a specific time interval. There is no specified standard value for this time interval and may depend on the flow metering system or the intended application.
- **Active timeout:** A flow may be terminated if it has reached a specific time interval even if there is still a continuous flow of packets. Once again, there is no specified standard value for this time interval and may depend on the flow metering system or the intended application.
- **End of flow detected:** A flow may be terminated if the metering process detects signals which indicate the end of the flow. An example of such signal is the TCP FIN flag.
- **Forced end:** A flow may be terminated by an external event such as a shutdown of the metering process which may be initiated by an external application.
- **Lack of resources:** A flow may be terminated for lack of resources available to either or both of the metering process and the exporting process. Such resource may be storage or processing capacity.

In this work, the active timeout parameter will be used to define flow duration. This will be elaborated further in subsequent sections.

2.5 Flow Monitoring and Export

Network monitoring techniques are generally classified into two categories: active and passive. In the active approach, tools like ping [62] and traceroute [63] are used to introduce traffic into the network to perform different network management procedures [64]. In the passive approach, existing network traffic is observed as it passes a specific measurement point. The passive approach of network monitoring is further divided into packet capture and flow monitoring (or flow export). Packet capture involves the complete capture of individual IP packets or frames as they traverse a measurement point on the network. This method offers the most detail regarding network traffic

but is generally expensive to perform given the extremely high data rates of modern networks resulting in substantial processing and storage requirements. Flow monitoring on the other hand is more scalable and adaptable to high speed networks. In this approach IP packets are aggregated into flows as defined in Section 2.4. These flows are then exported and stored and subsequently used for various network analysis. This method is less expensive and offers significant reduction in data to the tune of 1/2000 of the original traffic(packet) volume [64]. Additionally, flow export ensures more privacy than packet capture because only packet header information is utilised. The different traffic monitoring techniques are illustrated in Figure 2.7. The analysis in this work will be based on the flow export approach.

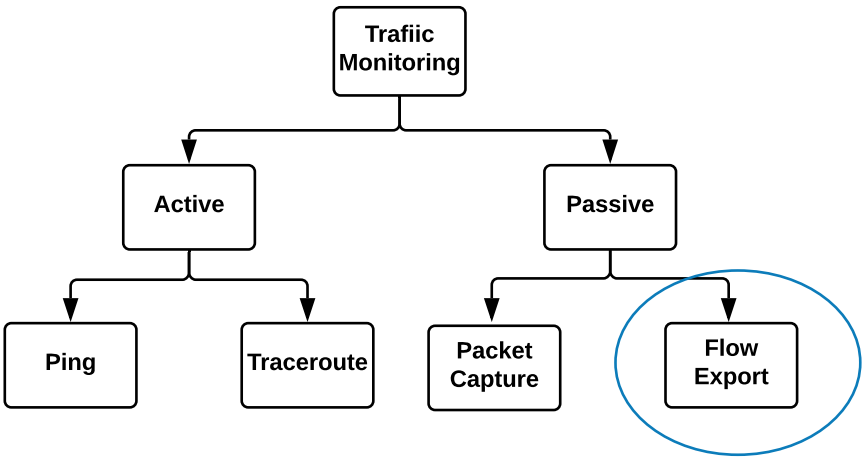


Figure 2.7: Traffic Monitoring Techniques

There are different network protocols and technologies that have been developed to implement the export of IP flow traffic for various purposes such as network monitoring, security and other network administration tasks. The two most common protocols for flow monitoring are IP Flow Information eXport(IPFIX) and Netflow.

2.6 IP Flow Monitoring for Network Security

IP flow exports are useful for the detection of most network attacks and are implemented in systesm such as Netflow of IPFIX [65]. A common characteristic of these attacks is that they may affect certain measured properties of an IP flow record. These metrics include packet and byte counts and number of active flows in a given time interval. These attacks may also affect other characteristic properties of an IP flow record such as suspicious port numbers as well as suspicious

source and destination IP addresses of the traffic. Flow based intrusion detection systems analyse IP traffic flow properties and detect attacks. Also, compared to conventional NIDS, flow based NIDS process a significantly lower volume of data which makes flow based IDS the logical option for high speed networks. However, it may be argued that IP traffic flows do not carry sufficient information for adequate network intrusion detection as compared to packet inspection [66]. This can be dependent on the intended application. Flows are generally believed to posses limited information as regards network interactions. However, with available information, it is possible to model communication patterns between network devices. For many network attacks, the available information is sufficient. In addition, the modelling of attack traffic patterns from available flow information has improved with the use of machine learning techniques.

In this thesis, the flow monitoring implemented for for the proposed hierachical SDN security solution is based on SDN counters available to the OpenFlow protocol and also to a flow export and monitoring mechanism based on the IPFIX standard.

2.6.1 Internet Protocol Flow Information eXport(IPFIX)

Internet Protocol Flow Information Export (IPFIX) is an industry standard flow export protocol. It specifies a common, universal standard for IP flow information export from network nodes. The IPFIX protocol defines standard templates on how IP flow information should be formatted and transferred from an exporter to a collector [67]. This allows network operators and vendors to implement a common standard for IP traffic flow information export [68]. This is necessitated by the fact that network flow data is becoming increasingly central to network management and security. A flow record is a representation of the connection between two sockets and represents a finer-grained view of network traffic than that provided by interface-level counters queried by the Simple Network Management Protocol (SNMP) [69].

The Internet Engineering Task Force specifies the requirements and architecture for IPFIX in RFC 3917 [67] and RFC5470 [70] respectively. An overview of the IPFIX architecture is shown in Figure 2.8. Since the work done in this research does not involve a testbed, the flow monitoring architecture in this work does not completely align with Figure 2.8. In this work, the entire IPFIX architecture and procedures are simulated in python. Further details of this will be provided in Chapter 4 where IPFIX is assumed to obtain the flow features used for the edge classifier, later called the second stage classifer from Chapter 4 onwards.

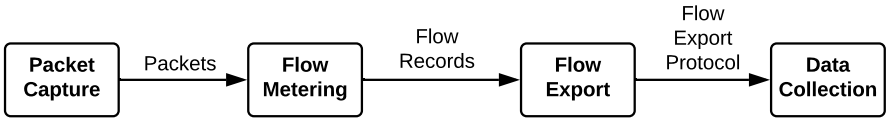


Figure 2.8: IPFIX Architecture

2.6.2 Netflow

Netflow is a Cisco proprietary protocol that allows the collection of network traffic features across an interface or an observation point. Cisco's Netflow v9 is the foundation upon which the IPFIX protocol is built. Although there were third-party compatible implementations of Netflow, the IPFIX protocol was built out of Netflow v9 to allow for standard industry-wide compatibility across flow export protocols [64]. Other candidate protocols evaluated by the IPFIX Working Group were CRANE, Diameter, LFAP, and Streaming IPDR [71]. The basic output for Netflow is a flow record as defined in Section 2.4.1. Netfow defines several different formats for IP flow records. The most recent format provided by Netflow version 9 is the template-based flow record format. A major benefit of the template-based format is that new traffic features can be added more easily an this allows for backward compatibility with previous versions and third-party application compatibility [72]. However, the IPFIX protocol shall be used as the reference for this research work.

2.7 An Overview of SDN

Over the past few decades, the traditional network architecture has remained unchanged. On the other hand, the increasingly dynamic nature of network services, applications and requirements, has put a reasonable burden on conventional network. Most traditional network devices have control and forwarding functionalities operating on the same device. Software Defined Networks (SDN) is a next generation networking concept which offers greater flexibility and control compared to traditional networks [73]. SDN is based on the concept of centralizing control plane intelligence while keeping the data plane separate [74]. This allows the network nodes to keep their switching fabric (the data plane) while transferring their intelligence (switching and routing capabilities) to the SDN controller. SDN provides logically centralised control over the network and separates the control and data planes, by abstracting the lower-level functionality allowing network management and control to be directly programmable [73] [75].

This is achieved by extracting the networks control logic (control plane) from the underlying switches and routers that perform the actual task of traffic forwarding (data plane). With this separation, network nodes become simple forwarding devices and the control function is implemented in a centralised controller [75].

2.7.1 SDN Architecture

The SDN architecture consists of three layers [73] namely: the application layer, the control layer and the infrastructure layer [76]. This is shown in fig 2.9

Infrastructure Layer: also called the data plane, consists of the forwarding devices which can be accessed through an open interface by which packet forwarding is done [77]. The decision making capabilities have been extracted from the devices and they rely on the control plane for forwarding decisions. Much of the work done in this thesis involves operations at the data plane.

Control Layer: the main element of the control plane is the logically centralised SDN Controller which coordinates the communication between the data plane and the application layer. The control plain is considered the 'brain' of SDN which allows the programming of network control and management ensuring a flexible network administration. Similarly, a significant portion of work done in this thesis involves operation at the control plane.

Application Layer: the application layer sits above the control layer and holds the applications that deliver network services. The application layer maintains a global view of the network through the control plane and provides the interface for user applications to interact with the network [78] [76]. However, work done in this thesis does not focus on the application plane.

This work focuses mainly on the control and forwarding planes and the communication between them.

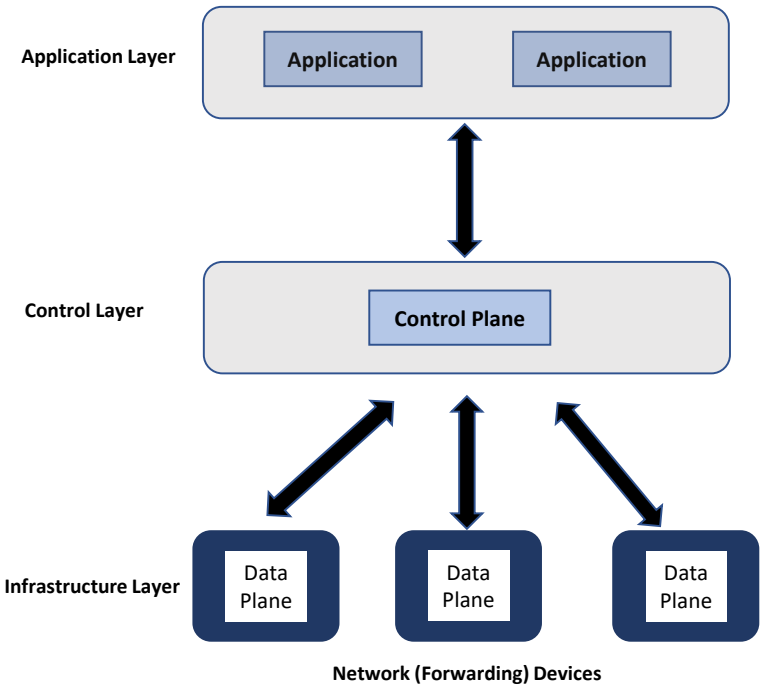


Figure 2.9: SDN Architecture

2.7.2 OpenFlow

The OpenFlow protocol was developed as a result of the need to have programmable networks [79]. The open and vendor-neutral nature of SDN requires a secure communication channel between the Controller and both planes above and below the controller. OpenFlow is the first standard communication protocol that was developed to interface between the control and data layers of the SDN architecture [80]. OpenFlow provides an open protocol which allows the SDN controller or the network administrator to program different routers and switches; OpenFlow achieves this by enabling software-based access to the flow tables that provide forwarding instructions to switches and routers [74]. Using this feature, network administrators can quickly modify network topology and implement packet filtering amongst other management and control tasks. The interaction between the controller and the OpenFlow switch or forwarding device(s) is shown in Figure 2.10.

The OpenFlow protocol supports the following message types [81]: (1) **Controller-to-switch messages** which are initiated by the SDN controller and are used to directly manage or inspect the state of the forwarding devices on the network. An important example of this message type in the context of this work is the Modify-State message (2) **Asynchronous messages** which are initiated by the switch and are used to update the controller about network events such as the

arrival of a packet or of other switch state changes. An important example is the Packet-in message (3) **Symmetric messages** which are unsolicited messages initiated by either the controller or the forwarding device. These are mostly hello type network messages and other routine network control messages.

2.7.3 OpenFlow Switch

The concept of the OpenFlow switch exploits the fact that most Ethernet switches and routers contain flow-tables. The OpenFlow Switch is the general term for referring to dedicated OpenFlow or OpenFlow-enabled forwarding devices. Dedicated OpenFlow switches do not support the usual Layer 2 and Layer 3 functionalities while the OpenFlow-enabled switches are the regular Ethernet switches and routers that have the OpenFlow capability as an added feature. The OpenFlow switch consists of one or more flow tables and a group table. These tables perform packet lookup and forwarding. The OpenFlow switch also contains one or more OpenFlow Channels which manages communication with the controller. The controller manages the switch and the switch communicates with the controller via the OpenFlow Protocol [81]. Figure 2.10 shows the main components of an OpenFlow switch. The work done in this research will be a compromise between the two classes of OpenFlow switches as will be explained in later sections.

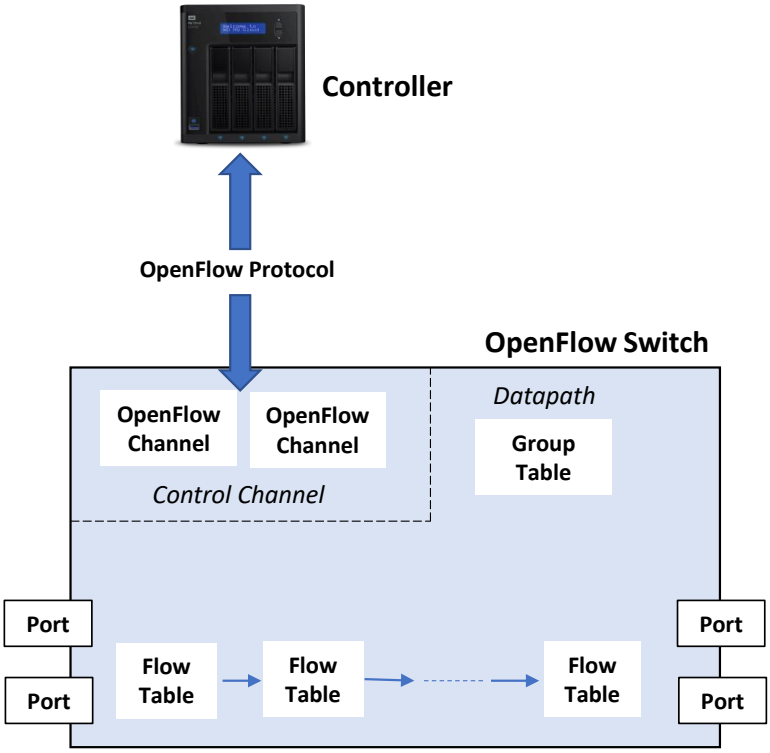


Figure 2.10: OpenFlow Switch

2.7.4 Flow Entry

One of the key components of the OpenFlow pipeline is the flow table. Each flow table consists of several flow entries. Each flow entry has the following fields [81]: (1) A **Match Field** that contains characteristic features that define the flow. This may consist of ingress ports and packet header information. This is used to match against packets (2) **Instructions** which defines how the packets in the flow should be processed and (3) **Counters** which holds statistics that keep track of the number packets and bytes in the flow [79]. The counters are also updated when the packets are matched (4) **Priority** which defines the order of matching precedence of the flow entry (5) **Timeouts** which sets the maximum amount of time before which a flow is removed by the switch (6) **Cookie** which is a control field used by the controller to filter flow entries based on flow statistics and other control requests (7) **Flags** dictate the way flow entries are managed. Figure 2.11 shows the main components of a flow entry table.

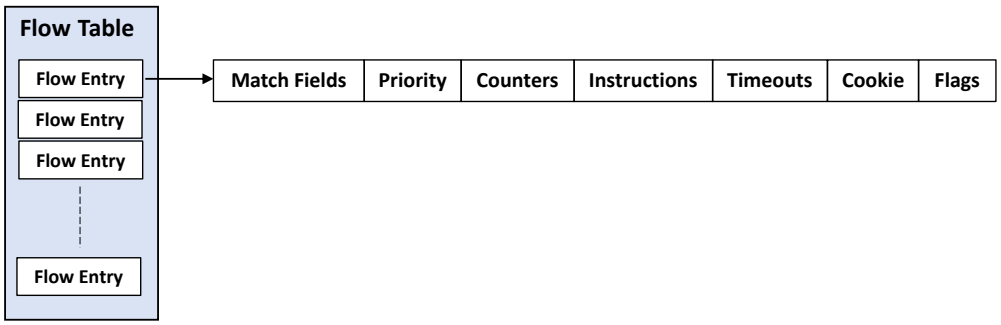


Figure 2.11: Main Components of a Flow Entry

2.7.5 SDN Security

There are generally two sides of SDN security; first, that the peculiar characteristics of SDN can be utilised to improve network security, secondly, that the SDN architecture in itself introduces security vulnerabilities [82]. The concept of software defined networking detaches the control and data planes within a network device. This separation provides a means for dynamic management of computer networks. The network control logic is then implemented in a software-based controller which provides dynamic control of network configurations and operations [10]. Key features of SDN include the programmability of the data plane and the controller's global view of the network. Additionally, the OpenFlow Switch Specification [81] which defines the southbound protocol responsible for communication between the control and data planes is able to provide network flow statistics at the data plane. These network flow statistics are able to support network monitoring for security and other network management functions. As a result of this SDN feature, network attacks can be detected by collecting traffic flow information and examining the flow statistics. There are various techniques to analyse flow statistics with respect to attack detection, however, this work will focus on the use of machine learning for flow analysis. Once attack is detected, SDN-specific characteristics allow the implementation of intrusion protection systems by modifying flow rules to filter or block malicious traffic.

One of the major advantages of SDN is centralised control. The ability of the SDN Controller to deploy network-wide policies and directly program network control ensures a more flexible and agile network solution. The centralised controller can instruct forwarding devices to allow or block traffic (as well as deciding the route). This feature of SDN is key to network security and can be leveraged to implement a more robust and intelligent network security solution. One solution which will be considered in this research is Machine Learning based SDN security solution.

Despite its ability to enhance network security, the SDN architecture introduces new security vulnerabilities into the network. The centralized controller becomes a central point of failure and becomes a prime target for attack [83]. Software bugs in the controller could lead to various problems such as malicious remote access to the controller, redirecting packets (loss of confidentiality) and DDoS attacks. Generally, any attack on the controller is capable of taking down the entire network. In the data plane (infrastructure layer), too many packets representing new flows could be sent rapidly to the forwarding device resulting in TCAM (flow table) [3] exhaustion attacks. This makes the TCAM to fill up and either stops new flows or causes the new flows to be processed very slowly. Also, communication channels between isolated planes can become another point of attack [83].

2.8 Network Edge

The edge of the network is an important concept in this work. There are different definitions of a network edge. Researchers in [84] describe the edge network as the local network as against the core network. A network edge can further be described as the point where an enterprise network connects to a third party network infrastructure. The third party network could be the internet, the WAN network or the cloud infrastructure. However, the network edge referred to in this work would follow a slightly different definition. In the context of this work, the network edge refers to the entry point into the enterprise core network. Edge devices could include routers, switches and a variety of other wired and wireless access devices. Based on Figure 1.2, the network edge can be illustrated as shown in Figure 2.12.

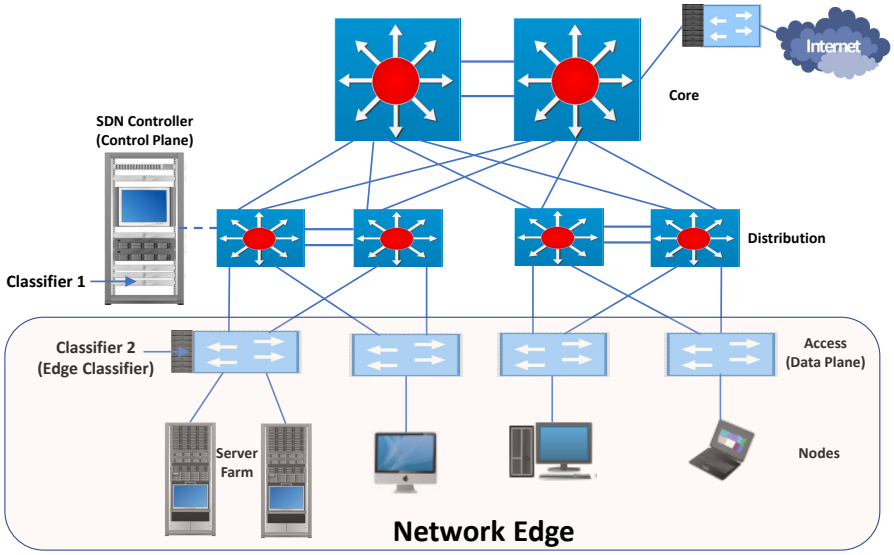


Figure 2.12: The Network Edge

2.9 Machine Learning Techniques for Network Security

Supervised machine learning methods have shown more success in network security applications. Although, this requires access to large volumes of labelled data which are very difficult to generate [85].

Existing research in traffic flow identification utilise human experience, thresholds and rule-based mechanisms to achieve traffic classification [86]. Unfortunately, these methods require extensive domain knowledge, and the complexities of modern networks and the volume and nature of today's network traffic make these methods increasingly difficult to implement. Exploring machine learning techniques can present new methods for more effective traffic classification. Machine learning has been proven to be very useful in proffering solutions in fields that have large representative data [87]. The field of networking is not exempted in this trend as there has been an upsurge of network related datasets in recent years.

Work done in this research have been carried out using supervised machine learning. Supervised machine learning is mainly split into classification and regression. The malicious packet detection problem is considered a classification problem hence classification technique is utilised in this work. IP traffic classification has been used for a variety of network management operations including network security. Other areas where traffic classification is applied include QoS and performance monitoring. Traffic classification involves the ability to accurately segregate IP traffic based on pre-

defined categorizations [87]. The classical method of traffic classification involves the use of Internet Assigned Numbers Authority (IANA) [88] registered port numbers for traffic classification. With the increasing complexity of network communication technology, this technique has been shown to be ineffective. Some of the reasons for this include port negotiation, traffic encapsulation, manipulation of well-known port numbers which could all be deployed to bypass firewalls [89]. There are three main traffic classification techniques that leverage on machine learning; payload-based traffic classification, host-behaviour-based traffic classification, and flow feature-based traffic classification. Work done in this research utilises flow-feature based traffic classification which is further sub divided as shown in Figure 2.13. The supervised complete flow-feature based TC and the early and sub-flow-based TC are both used in this work and are shown in red.

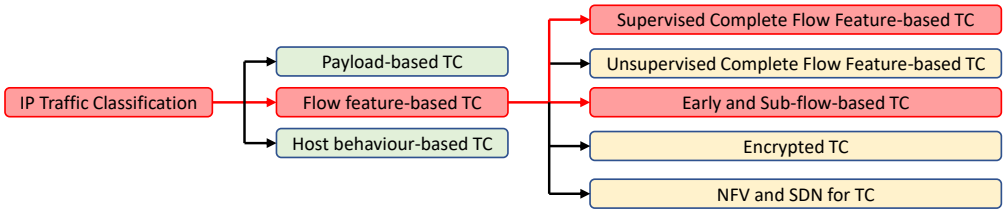


Figure 2.13: Machine Learning- Based Traffic Classification Techniques

Work done in this research evaluates the effectiveness of various machine learning algorithms using the CICIDS2017 and the CICIDS2018 IDS datasets and simulated in a distributed-learning SDN architecture.

A number of different machine learning algorithms have been used for network traffic classification and prediction. The dominant algorithms utilised in this research are decision trees and random forest classifiers. Hence discussions in this section will be limited to these two algorithms if the reader would like a general overview of machine learning algorithms excellent texts are [90] [91].

2.9.1 Classification and Regression Tree (CART)/Decision Tree Classifier

A decision tree is a popular machine learning method. As the name implies, decision trees make decisions based on a tree structure. Typically, decision trees comprise of a single root node, multiple internal (decision) nodes and multiple leaf (terminal) nodes. The leaf denotes the decision outcome while every other node corresponds to a test on an attribute [90]. The typical structure of a decision tree is shown in Figure 2.14.

The root nodes represent the entire samples in the dataset. This continuously gets divided into two or more child nodes. The samples in each decision node are split into child nodes based on statistical calculations carried out on the features. The route from a root node to a leaf node is a decision sequence. The goal is to generate a tree model capable of predicting unseen samples. The decision tree model is created recursively. The recursion terminates when any of the three conditions are achieved: 1) when all samples in a given node belong to the same class, in which case further splitting is not required; 2) when all the samples in a given node possess identical feature values making further splitting impossible; 3) when there are no samples in a given node, also making further splitting impossible.

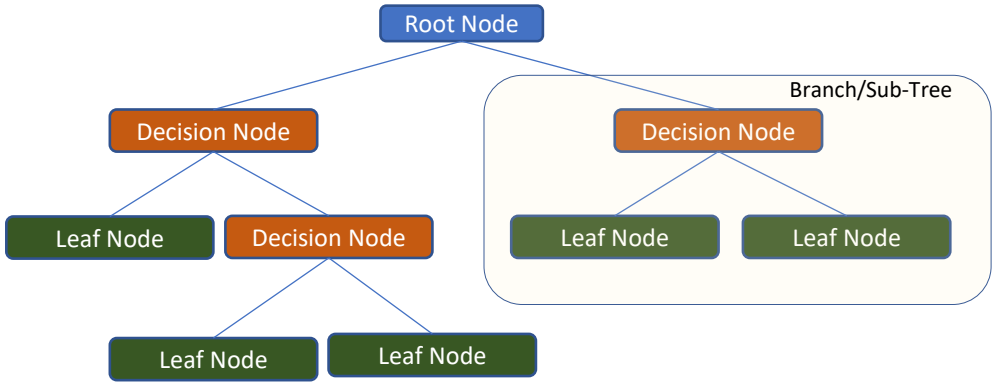


Figure 2.14: Decision Tree Structure

The major challenge in the implementation of the decision tree algorithm is the selection of the optimal splitting feature. This is the feature or attribute which is considered at the root node and is considered as the most important feature for the classification of the dataset. The same attribute selection process also applies at every node.

There are two common attribute selection measures used by decision tree algorithms: 1) Information Gain and 2) Gini Index. The Classsification and Regresion Tree (CART) algorithm which is utilised in this work uses the gini index for its attribute selection. Also, the gini index attributes are considered to be continuous which represents the nature of the dataset utilised in this work.

2.9.2 Gini Index

The Gini index is a popular method for selecting the splitting feature of a decision tree. The Gini value Gini(D) of a dataset D, represents the likelihood of two samples randomly selected from the

data belonging to separate classes and is given by [90]

$$\text{Gini}(D) = 1 - \sum_{i=1}^{c} (P_i)^2 \tag{2.1}$$

where P_i is the probability of the ith class and c is the number of classes. The lower the Gini value Gini(D), the higher the purity of the data D and the less likely for the node to be split. If a feature is used to split the data, producing the child nodes labelled $1 \dots U$, then the data represented at one of these child nodes u is denoted D^u.

To select the split feature, the Gini index of splitting datset D with feature b is given as

$$\text{Gini}\_index(D, b) = \sum_{u=1}^{U} \frac{|D^u|}{|D|} \text{Gini}(D^u) \tag{2.2}$$

where discrete feature b has U possible values $\{b^1, b^2, ..., b^U\}$ and

D^u = all samples in D with the value b^u for feature b *i.e.,* all the data split by node u, as defined above.

D = total number of samples in data and

Gini(D^u) = Gini value for D^u at node u.

This is calculated for all the features and the feature with the lowest Gini index is selected as the splitting feature. Next, the split point or value is calculated. Split points for the selected feature are chosen and the algorithm calculates the Gini index for each split point. The split point with the lowest Gini index represents the optimal point and is selected. The Gini index for each split point is thus derived, much the same way as with discrete value features. An example showing the calculation of the Gini value from experiments carried out in this research is given in Section 4.3.1. The utilisation of decision tree classifiers for network security is shown in [92], [93], [94].

2.9.3 Random Forest classifier

Random Forests is a Decision Tree based ensemble learning algorithm. Ensemble methods in machine learning involves the use of multiple similar models which have been trained independently. The individual predictions of each model are then combined to give the final prediction output. All the concepts of the Decision Tree algorithm apply here. Random Forest algorithm is a combination of multiple Decision Trees forming a forest of trees. It builds decision trees with the available data and extracts the prediction from each tree [95]. Some level of randomness is generated when creating the models in order to provide more accurate results. The required

randomness is achieved by training each model on a different subset of the available data. The algorithm will then present a prediction output based on the aggregation of the predictions from the forest of trees [96]. Methods utilised by the Random Forest algorithm to combine the outputs of the multiple decision trees include: averaging, voting, and combining by learning [90]. A simple graphical illustration of the Random Forest algorithm is shown in Figure 2.15. A random forest classifier will be specifically used in Chapter 6.

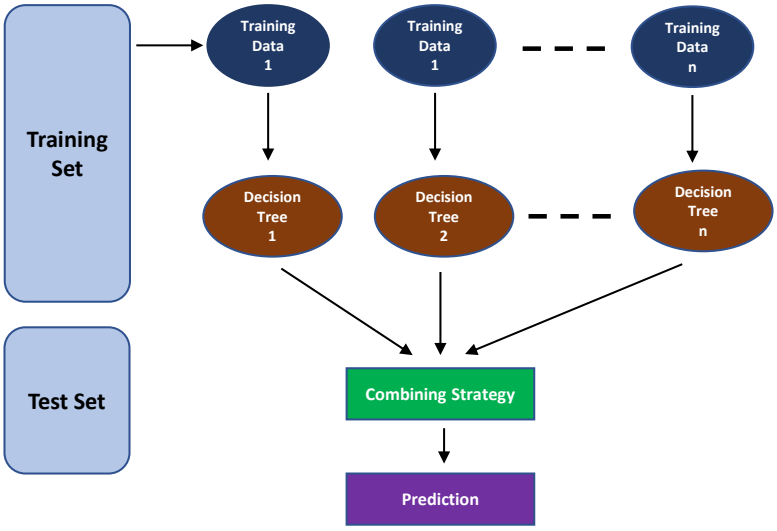


Figure 2.15: Random Forest Algorithm Structure

2.10 Machine Learning Based Network Security

In recent years, the share volume of network attacks have overwhelmed security analysts. As a result, network administrators and network security researchers have been exploring the use of machine learning to automate intrusions and anomaly detection, but progress made in the past have been slow. However, with the emergence of big data, the use of machine learning for network security is showing significant promise [85].

The detection mechanisms used for network intrusion detection systems (IDS) are of three types: misuse detection, anomaly detection and hybrid detection. Machine learning algorithms can be used to implement both the misuse detection and the anomaly detection [6]. Supervised learning methods are used for misuse detection while semi and unsupervised methods are used for anomaly detection. Supervised machine learning techniques utilises a learning based approach to discover attack profiles based on learned attack and benign profiles. The goal of supervised learning approach is to create a model representation of established attack profiles. Generally, misuse

detection techniques are unable to detect new and unknown attacks. However, it compensates with its high detection accuracy of well known attacks [92]. Just like signature based IDSs, the ML based IDS also requires regular update and maintenance of its detection models which adds to the overhead of its implementation. This research focuses on the machine learning based technique for misuse detection systems.

In traditional networks, Machine Learning has been used to detect malicious traffic and classify network attacks accordingly [14]. As networks become increasingly larger and more complex, and applications also growing significantly, network management across distributed sites is becoming increasingly difficult [86]. Machine Learning methods have demonstrated significant potential in the classification of network traffic and are widely used for classification and prediction problems [97]. The machine learning approach eliminates the need to manually analyse and encode traffic anomaly patterns as well as the rigours and uncertainties in statistically profiling normal traffic [98]. For SDN, the centralised nature of the controller holds significant bearing towards the application of machine learning for network management, monitoring and security. The work done in this thesis explores a unique approach that utilises this fact. The major advantage for using machine learning algorithms for SDN security is its ability to effect network-wide security rules as against local policy implementation of traditional networks [14]. Another advantage is its ability to use flow-based information which eliminates the rigours of packet payload inspection techniques to create rule-based intrusion detection systems.

Several researchers have carried out research works on machine learning based IDSs and other network security applications. Researchers in [99] explored the use of the CatBoost and LightGBM algorithms to perform binary classification of network traffic to detect attacks. Both algorithms also perform well when utilised for the multi-label classification. The CatBoost algorithm returns a slightly higher accuracy but requires up to 20 times more train time for binary classification and approximately 3 times more train time for multi-label classification. The CICIDS dataset was used for this work comprising typical network attacks such as DoS, DDoS, Portscan, BruteForce, WebAttacks, Bot and Inflitration. Kim et al in [92] have proposed a multiple classifier method which integrates a hierarchical combination of misuse detection and anomaly detection model as against just combining their results. The misuse model is implemented using the decision tree classifier DTC4.5 while the anomaly detection module is implemented using a collection of one class SVM algorithm. The proposed hybrid IDS was evaluated using the NSL-KDD data set. Results from this work show that the training and testing times for the anomaly detection model are 50%

and 60% respectively compared to conventional models. Additional results also demonstrate that the proposed hybrid solution delivers better detection rate for known and unknown attacks when compared to conventional methods while maintaining a low false positive rate.

S. Nanda et al in [97] proposed the use of Machine Learning algorithms to detect potentially harmful connections and likely attack destinations. This solution utilizes four widely used Machine Learning algorithms: the Bayesian Network, C4.5, Nayes-Bayes and Decision Table to make the required predictions with experimental results showing that the Bayesian Network returned an average prediction accuracy of 91.68%. The Machine Learning output is then used to define adequate security rules on the SDN Controller to block potential attack traffic.

In [100], L. Barki et al utilize the Support Vector Classifier and the Neural Networks Classifier to detect harmful DDoS traffic to the SDN Controller. The proposed solution is implemented using Mininet and Ryu Controller. Simulation of the results shows effectiveness of the solution on different topologies.

S. Gangadhar et al [14] extends the application of Machine Learning to improve traffic tolerance in SDNs by extending the functionality of the SDN Controller to integrate a resilient framework, ReSDN. The ReSDN deploys Machine Learning to detect DDos traffic in a real-time system achieving high levels of traffic tolerance.

Work done in [101] highlights the vulnerability of the SDN Controller to link flooding attack (LSA) and demonstrates how Deep Learning techniques can be used to mitigate these attacks. A novel solution, Cyberpulse is proposed, which conducts network surveillance by classifying network traffic using machine learning and deep learning methods and implemented as an additional module in the Floodlight controller. For every traffic flow in the network, Cyberpulse examines the flow statistics on the control channel and classifies the flow using a multi-layer learning approach which is able to correctly identify traffic flows that reveal LSA characteristics. Experimental results show Cyberpulse to have high accuracy and high false positive rate when compared to other competing solutions on realistic networks executed using mininet.

Researchers in [98] proposed a network intrusion detection and response system, Eunoia, based on the Random Forest classifier and applied in SDN. The system consists of data preprocessing, predictive data modelling, decision making and response subsystems. The system responds to anomaly detection by utilising reactive routing to optimize limited SDN resources. Experimental results demonstrate the ability of the SDN controller to mitigate known and unknown attacks that can not be eliminated by signature-based intrusion detection systems.

In [55], an IoT-based intrusion detection and mitigation framework, IoT-IDM is introduced. The framework provides network-level for smart home environments by identifying and mitigating potential attacks. This is implemented by deploying SDN architecture in IoT-IDM and leveraging on its network programability and visibility. Machine learning techniques are then utilised to detect compromised smart devices on the home network upon which IoT-IDM generates pushes specific mitigation policies to underlying forwarding devices.

One finding from the review of the literature, was that none of these proposed solutions demonstrate any real-time considerations of the flow based approaches. The real-time implication of flow-based network intrusion detection systems in SDN is yet to be extensively researched. Also, the concept of utilising the unique advantage of the SDN topology to implement efficient intrusion detection systems is yet to be extensively investigated. This work aims to contribute to that gap by investigating techniques that will allow near real-time flow based intrusion detection in SDN while also improving the efficiency of the detection process.

2.11 Summary

This chapter reviewed relevant research works on intrusion detection systems for networks. The chapter also highlighted the advantages of machine learning based intrusion detection systems as against traditional signature based systems. The chapter also reviewed research works which tested their proposed machine learning based solutions on various publicly available datasets. However, there is a significant gap in most of these research works. Most of the proposed machine learning based solutions are tested on static, historical datasets which does not reflect the real-time nature of real life production networks. No research work has investigated sub-flow based machine learning security solutions for SDN with reference to specific attack types. Also very little has been done in the area of exploring the hierarchical combination of the SDN controller and the edge devices towards improving the security performance of modern networks. These gaps and a few more will be addressed in this thesis.

The chapter also presented brief description of the different attack types present in the dataset which will be used in this thesis and also a brief description of the machine learning algorithms that will be used. Relevant SDN and IP traffic flow concepts are also presented.

Chapter 3

Methodology

3.1 Introduction

This chapter presents details of the experimental methodology used to implement the security solution proposed in this work. The chapter also presents the datasets and other tools and applications used in this work. It is important to note that Chapter 4 utilises a slightly different methodology from Chapters 5 and 6. In Chapter 4, the original traffic flow information contained in the dataset has been used for the experimentation and evaluation. The methodology for this is described in Section 3.3. The methodology for the work done in Chapters 5 and 6 involve the re-engineering of the IP packets to recreate the traffic flows. This is described in Section 3.4.

3.2 Intrusion Detection Datasets

Intrusion Detection Systems (IDS) are a focal point of this research. There is a shortage of reliable datasets for carrying out performance analysis and evaluation of proposed IDS and IPS systems [1]. Most real world datasets cannot be shared for reasons relating to privacy. Based on studies carried out, there are about eleven publicly available since 1998 [29]. Many of these datasets are outdated and largely no longer reliable to use. Some of these datasets include; DARPA (Lincoln Laboratory 1998-1999) [102], DEFCON (The Shmoo Group, 2000-2002) [103] and ADFA (University of New South Wales) [104]. Some of the attributes that make these datasets unreliable to use include limited attack variety, insufficient volume of data, anonymous packet and payload content, a lack of feature set and Meta data and such other characteristics that may not correctly depict recent trends of network attacks.

The work done in [1] generates a reliable intrusion detection dataset (CICIDS2017) which addresses most of the shortfalls from previously available IDS datasets. The dataset contains benign traffic and seven standard network attack flows which correlates contemporary real world scenarios. In addition, the dataset has the following attributes; it complies with the framework proposed in [1] for a reliable intrusion detection benchmark dataset, is completely labeled, has about 80 network traffic features and has an adequate Meta Data. This dataset will be used in this thesis to evaluate the proposed hierarchical SDN security solution.

3.2.1 Dataset Analysis

The work done in this thesis will be carried out using the CICIDS2017 and the CICIDS2018 datasets. Information on how the dataset was generated can be found in [105], [1], [106] and [107]. Both datasets are essentially a continuation of the other with the 2018 version version containing additional attack profiles. The purpose of both datasets is for network security and intrusion detection research, hence covers a wide range of attacks. The CICIDS2017 has been used in Chapter 4 while the CICIDS2018 has been used in Chapters 5 and 6.

The CICIDS2017 dataset comprises broadly of seven attack categories with some of the attacks further broken down into different types. The seven attack categories are: BOT, DDOS, DOS (DoS Slowloris, DoS Hulk, DoS Golden Eye, Heartbleed, DoS Slowhttptest), FTP Patator (ssh-patator, ftp-patator), Infiltration, Portscan and Web Attack (Web Attack SQL, Web Attack XSS, Web Attack Brute Force). In addition to the already mentioned attacks, the 2018 version contains the DDoS-LOIC and DDoS-HOIC variations of the DDoS attack. These attack types (and others) are described well by [42] [108] [41] [44] [109] [110] [23] [45]. A breakdown of the composition of the attack traffic flows in the dataset is shown in Tables 3.1 and 3.2. The dataset consists of raw packet capture files (as PCAP format files), system logs of each machine deployed in the testbed, and CSV files representing vector space models [111] [112] of the data. The PCAP files are very important to this work as it holds the raw captured IP packets. PCAP is the default file format for packet capture tools like Wireshark. The PCAP files are initially analysed using Wireshark to extract some packet features such as timestamps which are later used in the data re-engineering procedures. It is important to note that the IP packets contained in the dataset are approximately 1.1 tera bytes packet capture (PCAP) files. The size of the PCAP files is one of the reasons that makes this work computationally intensive as will be pointed out in subsequent sections of this thesis. The original dataset CSV files are used for the work in Chapter 4. For the

Table 3.1: CICIDS2017 Dataset Summary

Attack	**Sub Type**	**Count**	**Total**
DDOS	DDOS	128027	
	BENIGN	97718	**225745**
BOT	BOT	1966	
	BENIGN	189067	**191033**
PORTSCAN	PORTSCAN	158930	
	BENIGN	127537	**286467**
INFILTRATION	INFILTRATION	36	
	BENIGN	288566	**288602**
FTP PATATOR	SSH-PATATOR	5897	
	FTP-PATATOR	7938	
	BENIGN	432074	**445909**
WEBATTACK	WEBATTACK SQL	21	
	WEBATTACK XSS	652	
	WEB ATTACK BRUTE FORCE	1507	
	BENIGN	168186	**672703**
DOS	DOS SLOWLORIS	5796	
	DOS HULK	231073	
	DOS GOLDENEYE	1029	
	HEARTBLEED	11	
	DOS SLOWHTTPTEST	5499	
	BENIGN	440031	**672703**

works in Chapters 5 and 6, the original IP packets from the raw packet capture files have been re-engineered to recreate a more flexible version of the traffic flows. More details of how this has been done can be found in Section 3.4. This modified dataset is used in Chapters 5 and 6.

3.2.2 Attack Profiles

The different attack scenarios and profiles found in the dataset have been generated using various tools and applications. Table 3.3 gives a list of the different attack profiles and the respective tools with which they were generated. The table also shows the duration of each attack and other testbed information.

Table 3.2: CICIDS2018 Dataset Summary

	No. Flows		**No. Flows**
BruteForce Wed-14-02-2018		**DoS Fri-16-02-2018**	
Benign	**5478650**	Benign	**5363486**
FTP-BruteForce	**193360**	DoS-Hulk	**91557**
SSH-Bruteforce	**94237**	DoS-SlowHTTPTest	**105550**
Total	**5766247**	**Total**	**5560593**
DDoS-LOIC Tue-20-02-2018		**DDOS-HOIC Wed-21-02-2018**	
Benign	**6061102**	Benign	**6105843**
DDoS-LOIC-HTTP	**260079**	DDOS attack-HOIC	**967187**
DDoS-LOIC-UDP	**1668**	DDOS attack-LOIC-UDP	**1527**
Total	**6322849**	**Total**	**7074557**
Web Attack Thu-22-02-2018		**Web Attack Fri-23-02-2018**	
Benign	**6338216**	Benign	**6095665**
Brute Force -Web	**137**	Brute Force -Web	**123**
Brute Force -XSS	**43**	Brute Force -XSS	**73**
SQL Injection	**17**	SQL Injection	**33**
Total	**6338413**	**Total**	**6095894**
Infiltration Wed-28-02-2018		**BoT Fri-02-03-2018**	
Benign	**6943937**	Benign	**6281634**
Infilteration	**44**	Bot	**143011**
Total	**6943981**	**Total**	**6424645**

3.3 Data Pre-processing

Proof of the concept proposed in this work requires evaluation with Intrusion Detection Datasets. Obtaining reliable and recent Intrusion Detection Datasets for studies of this nature have always proved to be difficult. Two IDS Datasets made available to the research community by the Canadian Institute of Cybersecurity was chosen for this work. The two datasets are: the CICIDS2017 Dataset and the CICIDS2018 Dataset. The traffic profile of the CICIDS2017 Dataset has already been presented in Section 3.2.1.

The complete dataset comprises of a set of csv files and a set of traffic capture files in PCAP format [113]. This section describes work done on the csv files. The csv files presents the data in a vector space model. The entries of the data in this form are IP traffic flows. Each flow entry carries a label characterising it as either benign or malicious (based on the attack profile). However, the vector space model as presnted at this stage is not Machine Learning ready. The bulk of the work in this section is to make the data Machine Learning ready. This process is accomplished using the Python data processing tools Pandas and Numpy [114], [115], [116]. This

Table 3.3: Dataset Attack Profile

Attack	**Tools**	**Duration**	**Attacker Machine**	**Victim Machine**
Bruteforce attack	FTP – Patator, SSH-Patator	1 Day	Kali linux	Ubuntu 16.4 (Web Server)
DoS attack	Hulk, GoldenEye, Slowloris, Slowhttptest	1 Day	Kali linux	Ubuntu 16.4 (Apache)
DDoS+ PortScan	Low Orbit Ion Canon (LOIC) for UDP, TCP, or HTTP requests	2 Days	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)
Web attack	Damn Vulnerable Web App (DVWA) In-house selenium framework (XSS and Brute-force)	2 Days	Kali linux	Ubuntu 16.4 (Web Server)
Infiltration attack	First level: Dropbox download in a windows machine, Second Level: Nmap and portscan	2 Days	Kali linux	Windows Vista and Macintosh
Botnet attack	Ares (developed by Python): remote shell, file upload/download, capturing, screenshots and key logging	1 Day	Kali linux	Windows Vista, 7, 8.1, 10 (32-bit) and 10 (64-bit)

section of the experimental framework consists of the following procedures:

1. Data Cleanup - Some of the data entries returned values like 'not a number' (NaN), infinity and blank values. These values are not compatible with the Machine Learning algorithms and as such would have to be replaced or deleted to make the data compatible for Machine Learning. Although the volume of these entries were not significant enough to to impact the final Machine Learning results, they were replaced by either the mean feature value or by zero.

2. Label Coding - The labels (or the target values, y) on the dataset come with attack names. These names have to be encoded to make the data ready for Machine Learning. In this procedure, the target labels on the dataset have been encoded with values 0 for benign packet and between 1 and $N - 1$, where N is the number of classes [117]

3. Delete Unwanted Columns - The following feature columns were deleted from the data; 'FlowID', 'SourceIP', 'DestinationIP', 'SrcPort', 'DstPort', and 'Timestamp'. The 'FlowID', 'SourceIP' and 'SrcPort' are not in machine readable format and are characteristic properties of the flow. In fact including the IP addresses, ports and FlowID would severely bias the results as these are highly correlated to the classes (attack or benign) as was the nature of

the testbed used to generate the dataset. They are also not critical to the intrusion detection process. This is because the intrusion detection process is designed to receive only measured properties of the flow to perform traffic classification.

3.3.1 Feature Selection

The CICIDS2017 dataset is presented with 80 network traffic flow features. A description of the 80 features are given in [118]. The 80 features have either been extracted or calculated by the creators of the dataset using the CICFlowmeter [119]. To ensure an efficient classification model, a best short feature set is selected. This short feature set should be able to deliver a comparable level of classification accuracy as with the complete feature set. Feature selection plays a critical role in machine learning and is often performed as a data pre-processing step for creating robust learning models [120] [121]. The creators of the dataset utilized Random Forest Regressor for feature selection. The results do not show any metric to measure the accuracy of this process. To improve the feature selection process, we utilize the Mean Decrease Accuracy method [122] [123]. Although, Mean Decrease Accuracy method is also an implementation of Random Forests, this method is not directly integrated into Scikit-learn.

The Mean Decrease Accuracy method directly measures the impact of each feature on the accuracy of the model [122]. It achieves this by sequentially excluding each feature and calculating the impact on the accuracy of the model. This provides an indication of how important each feature is. The lower the accuracy of the model upon exclusion of a feature, the more important is the feature for classification. Tables 3.4 and 3.5 present the results of both feature selection methods. Table 3.6 shows a comparison of F1 scores between the the two methods. It is seen that with the CICIDS2017 method [1], there is a 20.79% reduction in F1 score as the features are reduced from 12 to 3. However, with the MDA method, there is an insignificant reduction in F1 score with a similar reduction in features. The results obtained from the Mean Decrease Accuracy method are used to evaluate the intrusion detection model for the proposed hierarchical malicious packet detection solution. The Mean Decrease Accuracy method of feature selection has been applied to all the attack categories and the results recorded in Appendix A. In the following sections, we will consider this in more detail across different machine learning models and the different attacks.

To design and evaluate the proposed intrusion detection model using the available dataset, the following process is utilized; the performance and accuracy of the selected features is tested with six common Machine Learning algorithms. The result of this process informs the choice of an optimal

Table 3.4: Feature Selection using Random Forest Regressor for DDoS as suggested by [1]

Feature	**Importance**
Fwd Packet Length Max	0.5746
Subflow Fwd Bytes	0.2516
Total Length of Fwd Packets	0.1675
Init_Win_bytes_forward	0.0022
Destination Port	0.0011
Init_Win_bytes_backward	0.0005
Bwd Packet Length Mean	0.0003
FIN Flag Count	0.0003
Bwd Packet Length Min	0.0002
Flow IAT Min	0.0002
Bwd Packets/s	0.0002
Avg Bwd Segment Size	0.0002
Fwd IAT Std	0.0001

Table 3.5: Feature Selection using Mean Decrease Accuracy for DDoS

Feature	**Importance**
Fwd Packet Length Max	0.815
Destination Port	0.4995
Init_Win_bytes_forward	0.1707
Subflow Fwd Bytes	0.0408
Total Length of Fwd Packets	0.0344
Bwd IAT Total	0.0279
Bwd IAT Mean	0.0223
Fwd IAT Std	0.0114
Bwd Packet Length Mean	0.0095
Packet Length Std	0.0092
Packet Length Variance	0.0043
Idle Min	0.004
Flow IAT Mean	0.0037

combination of feature set and Machine Learning algorithm for use in building the intrusion detection model for the proposed 2-stage Machine Learning SDN architecture. Figure 3.1 shows the performance evaluation score of the different algorithms based on different combination of feature sets. The first combination contains all 80 traffic features, the second and third combinations comprise the first 6 and the first 3 most important features from the feature selection process. Table 3.7 gives a list of the most common features when considering the first 6 most important features of each attack type. Initial results indicate that the Decision Tree Classifier (CART) and the K-Nearest Neighbor (KNN) both return a high evaluation score of approximately 99% across all feature set combinations.

Table 3.6: Comparison of F1 Scores for Different Feature Selection Methods

	MDA F1 Score	**CICIDS2017 F1 Score**
1st 12 Features	0.9997	0.9998
1st 6 Features	0.9994	0.792
1st 3 Features	0.9994	0.792

Table 3.7: Most common features when considering the first 6 most important features of each attack type

Feature	**Attacks**
Flow Duration	BoT, Infiltration
Bwd IAT Std	BoT, Infiltration
Init_Win_bytes_forward	BoT, Patator, DDoS
Init_Win_bytes_backward	BoT, DDoS
Total Length of Fwd Packets	PSCAN, DDoS
Fwd Packet Length Max	PSCAN, DDoS
Subflow Fwd Bytes	PSCAN, DDoS
Fwd IAT Min	Patator, Infiltration

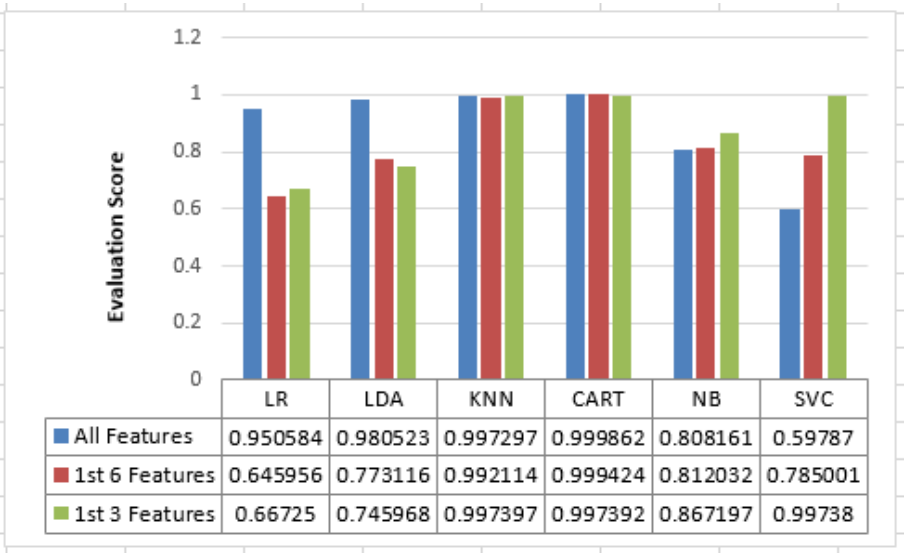


Figure 3.1: Performance Evaluation of the 6 Machine Learning Algorithms for DDoS

3.4 Experimental Framework

This section presents the experimental framework used to demonstrate and evaluate the proposed hierarchical intrusion detection solution in SDN. This solution is completely modelled in Python. The reason for modelling this solution in Python is to allow for sufficient flexibility to demonstrate

the concepts required to prove the hypothesis. Network simulation applications such as Mininet, Opnet Modeller and GNS3 were initially considered to evaluate the proof of concept of this work. However, after preliminary investigations, it was evident that these applications would not deliver the flexibility required.

3.4.1 IP Packet Processing

This is the first step in the experimental framework. The IP packet processing is performed using the Scapy Library in Python. This procedure involves converting the raw data into a vector space model. The raw data from this dataset is in the form of IP packets captured and stored in PCAP files. In order to process the PCAP files, an IP flow meter has been designed in python. This was done using the Scapy library in Python. The IP flow meter reads the pcap files and reconstructs the flows based on the flow conditions as specified by the authors of the dataset. The reason for reconstructing the flows from the IP packet capture is to create the ability to truncate each flow at selected time intervals creating subflows in the process. This is achieved by inserting a subflow generator into the flow meter that was designed. The authors of the dataset have created the threat vectors and have specified the details of each threat.

Using these details it was possible to link the attack vectors (and thus the label of the attack) so specific hosts in the large number of packet capture files; consequently, this was used to label the flows descriptions produced by the flow meter. This IP packets reverse-engineering process is illustrated at a high level in Figure 3.2. From the Figure, csv2 is the output of the re-engineering process. An illustration of csv2 is shown in Figure 3.3.

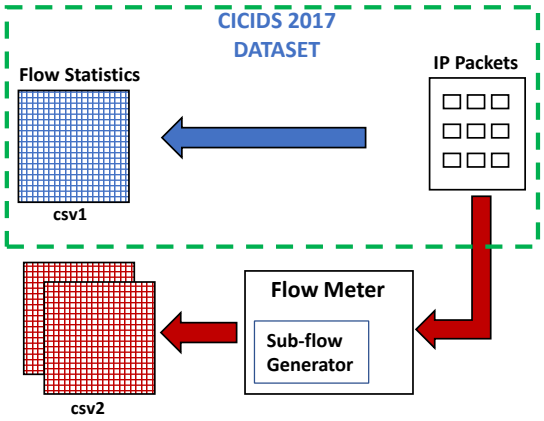


Figure 3.2: CICIDS2017 Dataset Re-engineering

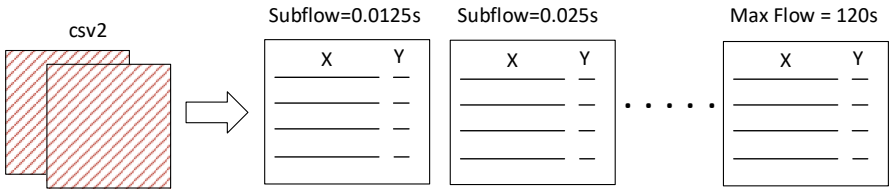
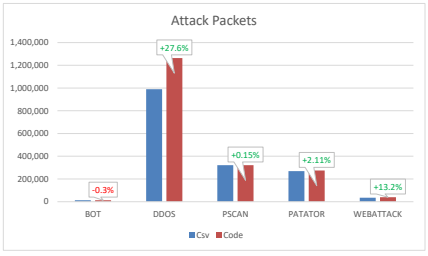


Figure 3.3: CSVs Representing IP Subflows

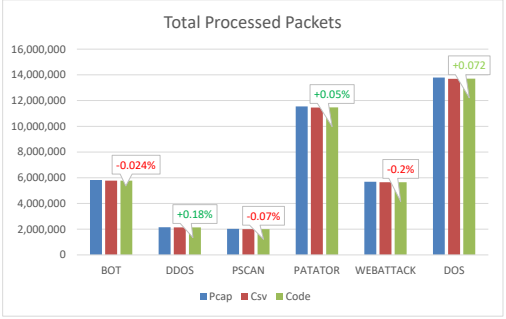
The following procedures are carried out in this section of the experimental framework:

1. Recreate IP flows by reverse engineering the IP packets using PCAP files from the dataset.
2. Relevant statistics for each flow is calculated during the reverse engineering process.
3. Match the flows recreated with the flows already recorded in the csv file. This is necessary in order to match the labels correctly on the recreated flows.

Further details of these procedures are given in Appendix B. As flow extraction from the PCAPs was essential to this work, a significant amount of work was invested in carefully checking that the flow meter was working correctly. For example the flows captured by the thesis flow meter was compared to the CSV files generated by the data set authors which used their CICFlowmeter. Relevant sanity checks are carried out on the re-engineered data to ensure adequate representation of the original data. It was found that the flows discovered were similar, however, it appears that the CICFlowmeter did not capture all of the threat packets, as shown in Figure 3.4. This was confirmed by manually checking to see that some packets were omitted by the CICFlowmeter. This gave further validation that reengineering the flows was required.



(a) Total Attack Packets



(b) Total Processed Packets

Figure 3.4: Comparison of packet counts between CICFlowmeter for Re-engineered Data (method used in this thesis)

3.4.2 Subflow Times

As explained in Section 3.4.1, the traffic is sampled at 14 different subflow time intervals including the maximum flow time of 120s. The different subflow timeintervals are: 0.0125, 0.025, 0.05,0.1,0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 30.0, 60.0 and 120 seconds. These have been chosen in a semi-logarithmic manner in order to emphasise subflows at lower times closer to the start of the flow. This is done this way because the CICIDS2017 dataset literature [1] which was discussed in Section 3.2.1 did not consider subflow durations in its work. Also, subsequent works such as [99] which was discussed in Section 2.10, and are based on the CICIDS dataset also have not considered this limitation. This is required in order to properly investigate the real time detection properties of the proposed hierarchical solution. This means that the two-stage intrusion detection process shown in Figure 3.5 is repeated fourteen times for attack type.

3.4.3 Maximum Flow Duration

One major consideration in the experimental framework is the maximum flow duration. This is built into the flow meter that has been designed in python. The captured packets are processed into flows and the flow duration is determined as explained in Section 2.4.2. As explained in Section 2.4.2, the active timeout parameter has been used to determine the maximum flow duration for this experiment. The IPFIX Standard does not specify any fixed time frame as the standard IP flow duration. The IPFIX Standard allows for the maximum flow duration to be determined by the type of application and the metering process. In this experiment, the maximum flow duration has been chosen as 120s. The major reason for this is because the original authors of the dataset have chosen this value as the maximum flow duration for their capture and it is desirable for the re-engineered data to replicate the original data as much as possible. Also, since this research focuses on near real time intrusion detection, there is more focus on the early stages of the flow and 120s sufficiently covers this.

3.4.4 Feature Extraction

The flow features for the re-engineered IP flows are either extracted or calculated by the flow meter. A total of forty six flow features are extracted. The flow features comprise of characteristic flow properties and measured flow properties as discussed in Section 2.4.1. The characteristic flow features do not form part of the input to the machine learning model. The main reason for this is because these features will create a high level of bias on the machine learning model.

Additionally, most of the characteristic properties of the flow are not directly in a readable format for the machine learning algorithms. A total of six characteristic flow features have been extracted; the flow ID, the source and destination IP addresses, the source and destination port numbers, and the timestamp. The remaining forty flow features are calculated and form the input of the machine learning algorithms. The flow properties of each sub-flow are calculated separately and represents a snapshot of the flow within that time window and are unique for each subflow. The flow properties are calculated with reference to the IPFIX Standard which was introduced in Section 2.6.1. A total of forty six flow features were calculated by the flow meter. The complete list of extracted flow features is given in Appendix C.1.

It should be noted that the first stage intrusion detection will be carried out using limited traffic flow features. This limitation arises from the OpenFlow protocol itself. OpenFlow was designed to be a flow-based configuration protocol for packet forwarding devices rather than a data export or network monitoring tool [64]. According to the Open Networking Foundation's OpenFlow Switch Specification [81], the OpenFlow protocol provides the following five Flow Stat Fields on the OpenFlow Switch:

1. OXS_OF_DURATION: This field represents the elapsed time the flow entry has been installed in the switch.

2. OXS_OF_IDLE_TIME: This field: represents the elapsed time for which the flow entry has not matched any packets.

3. OXS_OF_FLOW_COUNT: This field is only used when the aggregated approach [124] to flow entry is utilised.

4. OXS_OF_PACKET_COUNT: This field represents the total number of packets matched by a flow entry.

5. OXS_OF_BYTE_COUNT: This field represents the total number of bytes matched by a flow entry.

Fields 1-3 above are specific to the flow entry on the OpenFlow Switch. Only 4 and 5 are original flow properties of the actual packet stream. This concept is what is utilised in the OpenFlow based traffic classification which is performed at the first stage of the hierarchical intrusion detection solution. This clearly indicates that the gross flow level information available through OpenFlow counters are the packet count and the byte count. These are part of the forty six flow features that have been extracted by the flow meter.

3.4.5 Experimental Design

This section presents the complete experimental design. It shows the different components of the complete workflow and their inter-connectivity. The experimental framework simulates a hierarchical intrusion detection system in SDN. This is illustrated in Figure 3.5. The first stage intrusion detection module is domiciled in and simulates the SDN controller. The second stage intrusion detection module is co-located at the edge (forwarding) device. These intrusion detection modules are Machine Learning models obtained by training a Machine Learning Algorithm with the data processed in Section 3.4.1. The flow meter receives the IP packets and reconstructs the flows based on flow and threat vector information provided in the original dataset. This has been described in Section 3.4.1. The IP packet stream shown in the figure is contained in the packet capture (PCAP) files described in section 3.2.1. A critical part of the flow meter is the sub-flow generator which generates fourteen sub-flows $F_{(i)}$ as specified in Section 3.4.2. Each sub-flow with OpenFlow features $F_{(i)OF}$ is spilt into train and test data, $F_{(i)OF-train}$ and $F_{(i)OF-test}$, with a 40% test data ratio. The train portion of each sub-flow is used to train the machine learning algorithm to create the first stage model. The output of the first stage model is processed as dictated by the decision box. The idea behind processing the output from the first stage has been introduced in Section 5.5 and will be discussed further in Section 5.6. The predicted attack flows are dropped and the predicted benign flows are first translated to full features and are then passed on to the second stage classification. The reason for the translation of features is because the flow monitoring at the second stage is not based on OpenFlow and hence not limited to just OpenFlow features as was the case in the first stage. The total classified results from the first and the second stages are then combined to give the final hierarchical classification output. This combination is necessary because only the negative classifications were sent to the second stage but the final hierarchical results have to be presented and analysed in terms of the overall positive and negative classifications. Figure 3.6 illustrates the non-hierarchical method. Note that this figure also contains the flow meter and the sub-flow generator. This ensures that sub-flows of identical duration to those from the hierarchical solution are created. The same feature extraction procedures and the same train-test split size as was used in the hierarchical experiment is also used for here. This allows for a balanced comparion between the two methods. It should be noted that the processing of the data to create the models and test results is computationally intensive. Each attack dataset typically represent 6 hours of packet flows that have to be processed by the flow meter for each sub-flow. Then the features have to be collected and used to train the

model, then the model is tested. Separate models are created for each sub-flow and the process repeated for each attack. The entire process is then repeated for the non-hierarchical approach but with the non-hierarchical approach utilising only one classifier. The comprehensive results of the experiment is given in Section 5.6. Please note that this process as described in Figure 3.5 has been modified in Chapter 6. The details of the modification have been provided in Chapter 6.

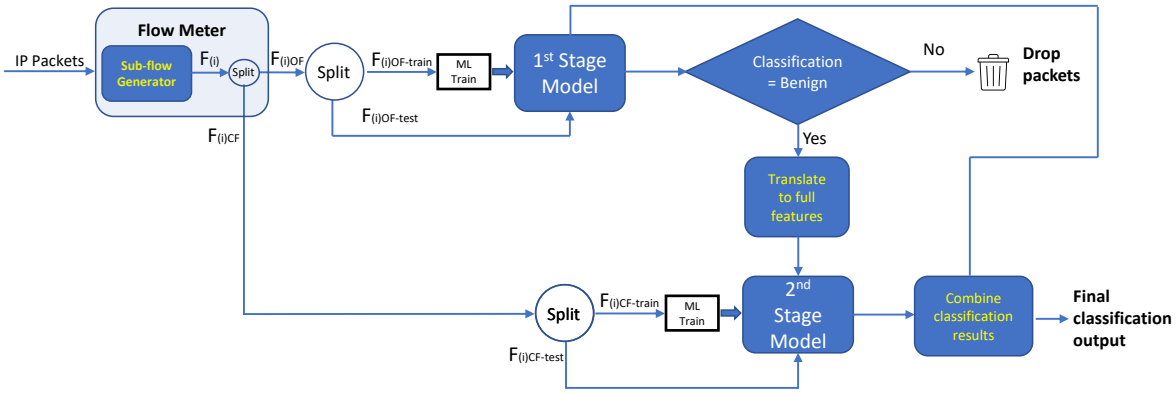


Figure 3.5: Experimental Design for Hierarchical Solution

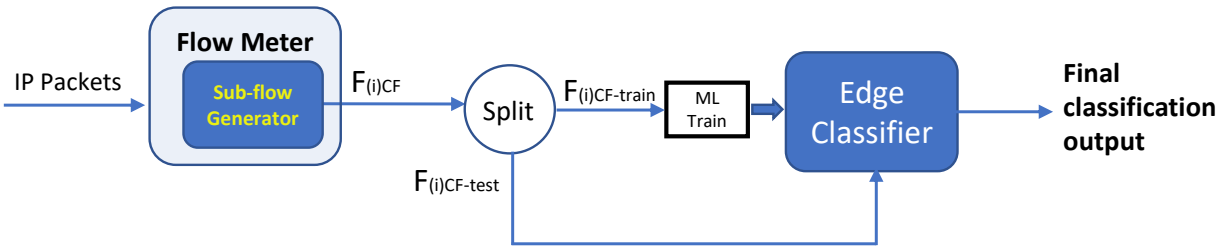


Figure 3.6: Experimental Design for Non-Hierarchical Approach

3.5 Summary

The main contribution of this chapter is the introduction of a novel methodology for evaluating intrusion detection solutions. This involves the creation of a sub-flow generator in the flow meter design based on the IPFIX standard which does not provide for sub-flow generation. This chapter also presented a unique experimental framework which allowed for distributed machine learning instances which would be used to simulate a hypothetical SDN topology to be used for the work in Chapters 5 and 6. Also, the methodology described in this chapter forms the basis for the first stage design framework presented in Chapter 4.

Chapter 4

Intrusion Detection Model Design

4.1 Introduction

This chapter explores the machine learning models that will be applied later in Chapter 5 to the hierarchical solution, briefly introduced in Chapter 1. The main aim of this chapter is to focus on the machine learning model itself before it is later applied to the hierarchical approach. To obtain this model, different machine learning algorithms are evaluated based on the unique requirements of the proposed hierarchical security solution. The unique requirements for the hierarchical solution are:

1. Fast detection times to allow for timely intrusion detection, this is particularly important as the proposed solution entails a two-stage hierarchical machine learning process;
2. Fast model train time to allow for quick deployment and live update of the model
3. Good recall to ensure that a high proportion of the attack traffic is detected.

Five machine learning algorithms have been chosen for this evaluation. The five algorithms have been selected across linear and non-linear algorithms. The five machine learning models evaluated are; Linear Discriminant Analysis, Logistic Regression, Decision Tree Classifier, K-Nearest Neighbours Classifier and Gaussian Naive Bayes. The evaluation is made using the intrusion detection dataset CICIDS2017 [1], which has been introduced in Section 3.2.1. This choice of algorithms was inspired by the literature search documented in Chapter 2, a good summary is given by Mishra et al. [6]. One notable machine learning technique which has not been considered is that of deep neural networks. As pointed out by Mishra et al., while deep neural networks are very attractive

for image classification, they have limitations for very fast packet processing due to the high rate of traffic requiring very intensive processing using graphic processing units (GPUs). In the future, when GPU based deep neural networks become cheaper, it might be a solution that could be investigated.

Work done in this chapter has been carefully synthesised to develop a design framework. Although this design framework is original to work done in this thesis, it can be adapted to any dataset and to any machine learning algorithms. The design framework is shown in Figure 4.1.

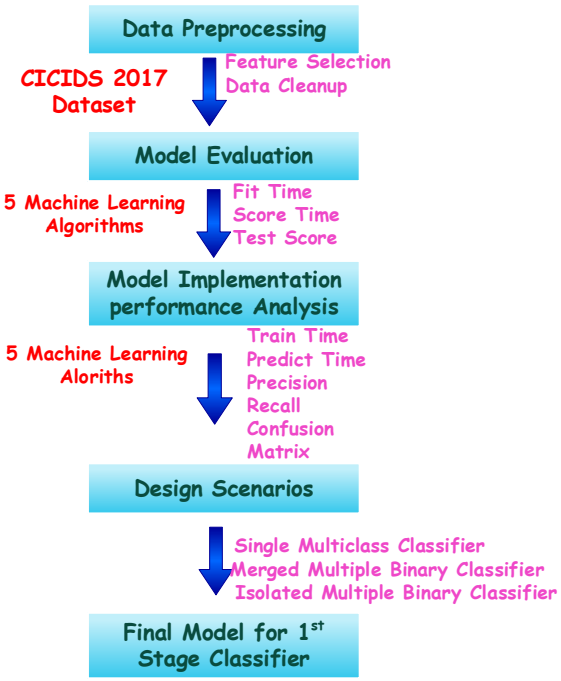


Figure 4.1: 1st Stage Design Framework

The data prepossessing stage involves preparing the CICIDS2017 dataset into a format suitable for Machine Learning. This section also involves reducing the dimensionality of the data through a feature reduction process. In the model evaluation section, five Machine Learning algorithms are chosen and evaluation parameters such as fit time, score time and test score are used to evaluate the performance of the different algorithms. The model implementation performance section simulates actual deployment of the five algorithms to create classification models which are deployed in an SDN architecture. The parameters of interest for this analysis are the train time, predict time, prediction accuracy, recall and F1 score. The confusion matrix is also utilised for this stage of analysis. Further down on the framework, three different design scenarios are then created as shown in Figures 1.29, 1.30, 1.31. A single scenario from this section is adopted

as the first stage classifier based on additional evaluation results as presented in Section 1.7. The adopted design scenario captures output from all previous blocks of the framework.

The methodologies presented in this chapter utilise standard features of the Python library scikit-learn [15] unless otherwise stated.

At the end of this chapter a brief emulation will demonstrate a preliminary estimation of how the machine learning could be applied to a hierarchical model to test the approach.

4.2 Model Evaluation

The process of model evaluation involves training and evaluating the model using the train portion of the dataset. This is carried out using K-fold cross validation with a split size of 10. In this process, the evaluation metrics of fit time, score time and test score are used. The fit time is the mean time the train data takes to fit the model on across the entire cross validation split. Note that with a split size of 10, a tenth of the data is used as a test set. The score time is the mean time expended by the model on each test set across the entire cross validation split. The test score is the mean accuracy score on the test set across the entire cross validation split. This is illustrated in Figure 4.2. Upon satisfactory response of these metrics, the resulting model is used to simulate model implementation. The following sections will provide more detail of the model evaluation process across different attack types. The use of cross validation allows the repeatability of the model across different data. In practice, for the data sets used here the variation was very small, and thus the basic statistics on this variability are only shown in the first two cases, for all others the variability was equally small.

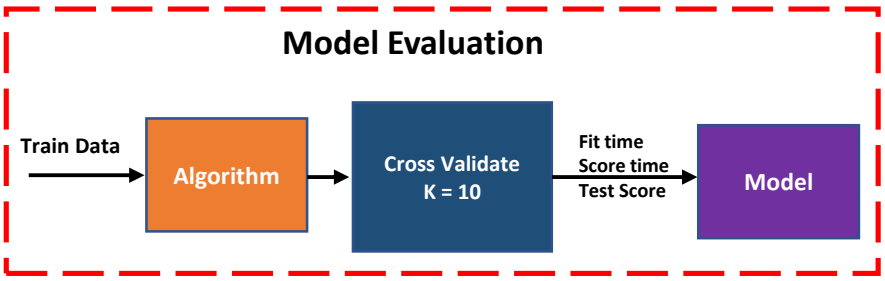


Figure 4.2: Model Evaluation using k-fold Cross Validation

4.2.1 Model Evaluation Results for DDoS

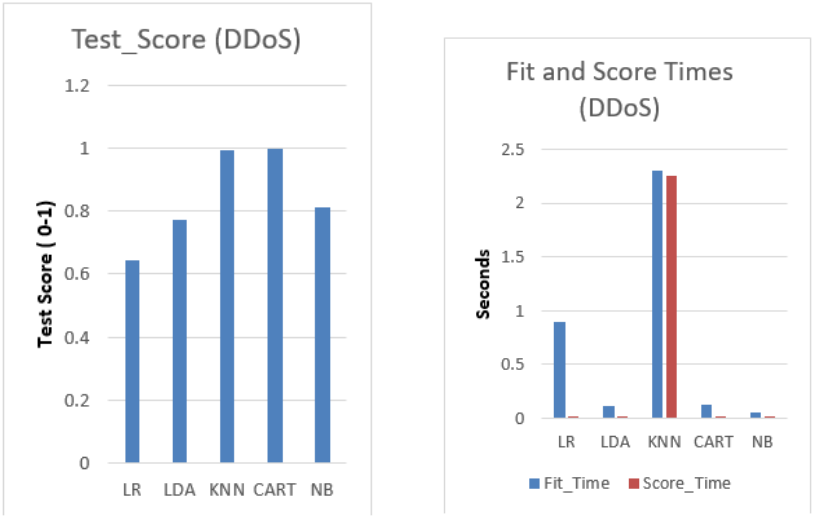
As explained in Section 4.2, the evaluation score shown in Figure 3.1 has been split into the more specific components of the fit time, score time, and test score. This second phase evaluation utilizes the first six features from the feature selection process.

Results for DDoS once again show CART and KNN returning very high test scores (average 0.99) but this second phase evaluation results show a significant difference in fit and score times as shown in Table 4.1. The results show that CART returns a significantly lower fit and score times than KNN. Figures 4.3 further splits the results into 2 graphs. Figure 4.3a shows the test scores and figure 4.3b shows the fit and score times for for the model evaluation for DDoS attacks. The score time is a parameter of significant importance for the design of the intrusion detector model. This is because the classifiers need to make classifications in real-time at a rate comparable to the traffic flow rate.

The evaluation score times for CART and KNN (the closest competitors for DDoS) are shown in Figure 4.4 and shows a factor of nearly 1000 in favour of CART. This suggests CART to be a more likely suitable algorithm for the design of the classifiers of the hierarchical model. Also, the attack types considered in this research is limited to the available dataset.

Table 4.1: Performance Evaluation of 5 Machine Learning Algorithms for DDoS Attacks

MODEL EVALUATION (DDoS)						
Algorithm	**Fit_Time (s)**	**Score_Time (s)**	**Test_Score**			
			Mean	**Std Dev**	**Min**	**Max**
LR	0.890995	0.001988	0.667	0.002	0.663	0.671
LDA	0.105983	0.002194	0.773	0.003	0.768	0.778
KNN	2.301868	2.258479	0.992	0	0.992	0.993
CART	0.12518	0.003014	0.999	0	0.999	1
NB	0.048441	0.00269	0.812	0.002	0.808	0.815



(a) Test Score (b) Fit and Score Times

Figure 4.3: Evaluation Results for DDoS showing Test Score, Fit Score and Fit Time

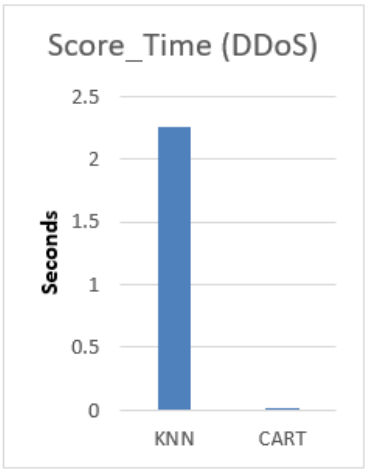


Figure 4.4: Performance Evaluation of the Top 2 Machine Learning Algorithms for DDoS Attacks showing Score Time

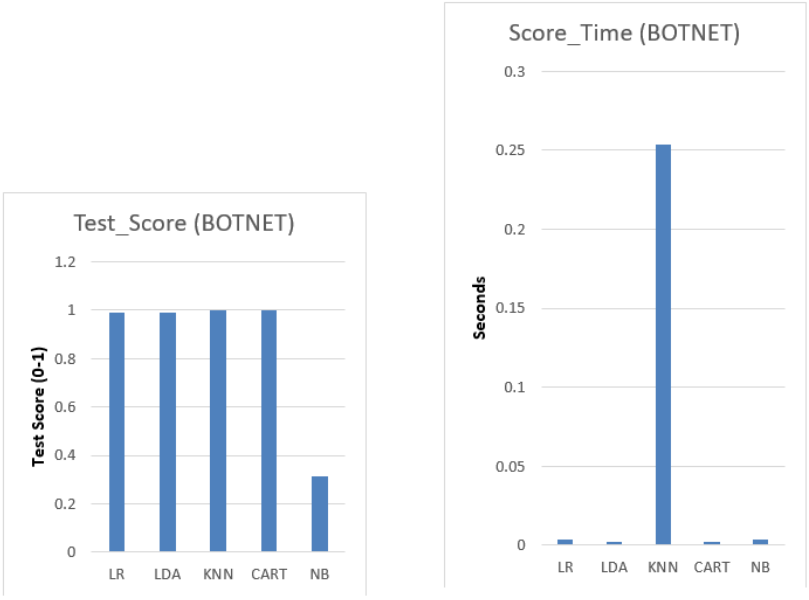
4.2.2 Model Evaluation Results for BOTNET

Model evaluation results for the BOTNET attack is shown in Table 4.2. These results follow a similar trend to DDoS. The results for BOTNET show CART with the highest test score of 99.94% and a score time of 0.00216s. LDA returns a marginally faster time of 0.00209s but has a lower test score of 98.96%. Once again KNN returns a very good test score of 99.68% but returns a high score time of 0.253s which is higher by an approximate factor of 100 to the CART score time. The reason for this large difference in score time is because the KNN algorithm calculates

distance from the reference sample to all training samples. Hence when the training data is very large, this results in a decline in classification speed [125] as evidenced by the the high score times recorded in this results. Figure 4.5 further illustrates the comparison of test score and score time between the different algorithms.

Table 4.2: Performance Evaluation of 5 Machine Learning Algorithms for BOTNET Attacks

MODEL EVALUATION (BOT)						
Algorithm	**Fit_Time (s)**	**Score_Time (s)**	**Test_Score**			
			Mean	**Std Dev**	**Min**	**Max**
LR	1.708155	0.003533	0.99	0.001	0.988	0.991
LDA	0.074206	0.002095	0.99	0.001	0.988	0.991
KNN	1.110726	0.253919	0.997	0	0.997	0.998
CART	0.188147	0.002161	0.999	0	0.999	1
NB	0.036608	0.003665	0.315	0.012	0.307	0.35



(a) Test Score (b) Score Time

Figure 4.5: Evaluation Results for BOTNET

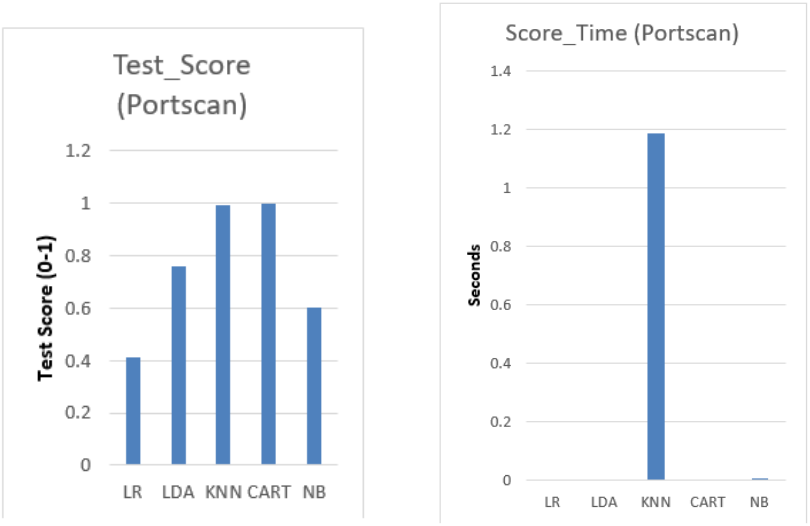
4.2.3 Model Evaluation Results for Portscan

Model evaluation results for the Portscan attack are shown in Table 4.3. These results follow a similar trend to DDoS and BOT with CART returning the most impressive metrics. The results for BOTNET show CART once again with the highest test score of 99.80% and a score time of

0.00399s. LDA yet again returns a marginally faster time of 0.00309s but this time returns a much lower test score of 76.12%. Similar to the two previous attack scenarios, KNN returns a very good test score of 99.32% but returns a much higher score time of 1.18s which makes KNN relatively unsuitable for the classifiers of the proposed Hierarchical Machine Learning SDN security model. Figure 4.6 further illustrates the comparison of test score and score time between the different algorithms.

Table 4.3: Performance Evaluation of 5 Machine Learning Algorithms for PORTSCAN Attacks

MODEL EVALUATION (PSCAN)			
Algorithm	**Fit_Time (s)**	**Score_Time (s)**	**Test_Score**
LR	0.73465	0.004272	0.411039
LDA	0.133543	0.003092	0.761238
KNN	1.846937	1.18761	0.993206
CART	0.278064	0.003995	0.998051
NB	0.063132	0.00619	0.603121



(a) Test Score

(b) Score Time

Figure 4.6: Evaluation Results for Portscan

4.3 Model Implementation Performance

The model implementation is a critical stage of the Classifier design. The model implementation procedure involves applying the hitherto unseen test data on the model. This is illustrated in Figure 4.7. While the model evaluation procedure simulates the process of creating the the model,

the model implementation procedure simulates the actual operation of the model upon deploying the model in a production network. The results of this procedure represents an indication of the actual performance of the algorithms when deployed in real scenario. During model evaluation, the evaluation parameters considered were the fit time, the score time and the test score. In terms of model implementation, the parameters of interest at this stage of design are the train time, predict time, recall, precision and F1 score.

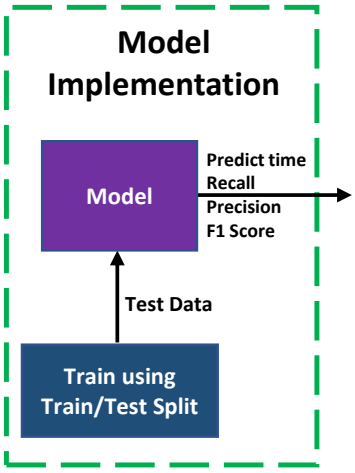


Figure 4.7: Model Implementation

To understand how these metrics are derived, a sample confusion matrix is shown in Figure 4.8. In machine learning, classification results are usually summarized in a confusion matrix [126]. In relation to work done in this research, negative refers to benign flows while positive refers to attack flows. **True negative (TN)** refers to the portion of the classification output that has correctly identified benign flows while **false positive (FP)** is the portion of the output that has incorrectly identified benign flows as attack flows. The **false negative (FN)** is the most important confusion matrix component with respect to work done in this thesis. FN represents the portion of the classification output that has incorrectly identified attack flows as benign flows. This has the greatest security implication in networks and is a critical parameter in the recall metric and other accuracy metrics which would be used in the evaluation of the proposed security solution. Finally, the **true positive (TP)** refers to the portion of the classification output that has correctly identified the attack flows.

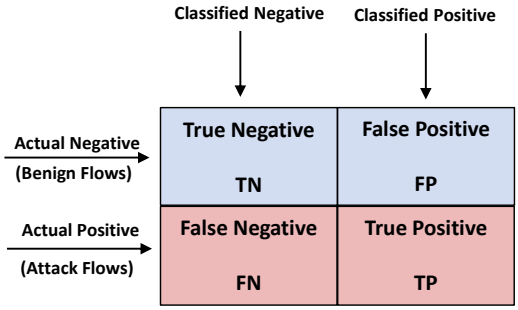


Figure 4.8: Sample Confusion Matrix

Equations 4.1, 4.2 and 4.3 give the calculations for recall, precision and F1 score respectively. As expected, the recall metric is the most critical accuracy metric in the evaluation of the proposed hierarchical intrusion detection solution. The recall metric is a measure of completeness and is an indication of how much of the available attack flows have been correctly classified. From Equation 4.1, it can be seen that the recall metric defines the ratio of accurate attack classifications in relation to the total attack flows present in the dataset. This is indirectly proportional to the FN value, hence a large FN value gives a low recall value. An intrusion detection system with a low recall value offers little protection to the network. Therefore, one of the main goals of the proposed solution is to have a low FN output and a high recall value. A high recall metric is of particular importance in the design of the intrusion detection module because of the need for a very low false negative output. The implication of a low false negative output is that malicious packets or flows are not erroneously allowed through to the network as benign packets or flows. Precision on the other hand gives a measure of how precise the flow classifications have been. Precision gives a measure of how much of the classified flows have been correctly classified. The precision metric defines the ratio of accurate attack classifications in relation to all the attack classifications (both accurate and inaccurate attack classifications) as seen in Equation 4.2. While the precision metric is one of the standard ways of evaluating the accuracy of the intrusion detection model, it does not represent complete protection of the network. Hence, the precision metric is only used for limited analysis in this work. These parameters could be considered to be analogous to the already considered model evaluation parameters. The impact of these metrics on the proposed solution is considered in more depth in the next Section 5.5.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.1}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

The F1-score combines the recall and precision metrics by calculating their harmonic mean. Since the F1-score gives a balanced measure of the precision and recall, it is only used for selected analysis in this work.

$$\text{F1-Score} = \frac{2(Precision * Recall)}{Precision + Recall} \tag{4.3}$$

The more generic accuracy measure given in Equation 4.4 is less critical in this application. This is because in an imbalanced dataset (like the ones used for this study), can return a very high accuracy but the classification results may have completely missed out all the attack traffic. Given that the essence of this study is to detect malicious IP packets represented in the dataset, the generic accuracy value will not be used as the key measure of accuracy. Section 5.5 explains in further detail how the relevant metrics impact on the design and operation of the hierarchical solution.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.4}$$

The Model implementation results further validate the evaluation results. In the Model Implementation results, further reference is made of feature selection. The Model Implementation metrics of train time, predict time and prediction accuracy are presented with different feature selection results. The Model Implementation metrics are presented first using all the features available on the respective datasets, secondly with the first six important features and lastly with the first three important features. Further analysis of the results suggest an optimal use of the first six important features for Model Implementation of all attack categories.

4.3.1 Model Implementation Performance for DDoS

Initial model implementation results for DDoS showing the performance metrics of prediction time and prediction accuracy are presented in Tables 4.4 and 4.5. The results in Table 4.4 show a significant reduction in prediction time across all the algorithms when the features are reduced from all eighty available features to the first six and to the first three. CART returns an 85% reduction in prediction time when all available eighty features are reduced to six features while maintaining its 99.9% prediction accuracy. This conforms to the trend captured in the model evaluation and thus validates the model evaluation results. NB offers a 93% reduction in prediction time but gives

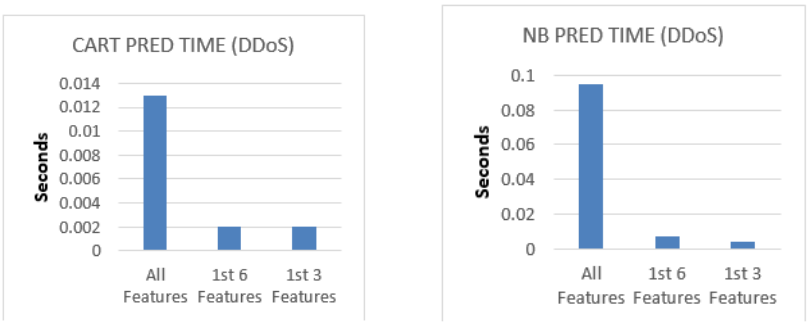
Table 4.4: Prediction Times for DDoS Using Different Feature Selection Results

Predict Time for DDoS (s)			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.008985	0.001996	0.000998
LDA	0.007979	0.000998	0.000998
KNN	5.971206	5.626392	11.625983
CART	0.012965	0.002023	0.002023
NB	0.094778	0.006981	0.003963

Table 4.5: Prediction Accuracy for DDoS Using Different Feature Selection Results

PREDICTION ACCURACY for DDoS			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.9459	0.64111	0.6675
LDA	0.9808	0.77263	0.7445
KNN	0.9978	0.99235	0.9975
CART	0.9997	0.99937	0.9975
NB	0.8061	0.80956	0.8515

a prediction accuracy of 80.6% and 80.9% when reduced from eighty to six features respectively. Although KNN returns a consistently high prediction accuracy of 99.9%, its prediction times are much higher (5.97s and 5.62s) which makes it unsuitable for the proposed Hierarchical Classifier design. LDA and LR return a much reduced prediction accuracy when the feature set is reduced. A graphical comparison between the two nearest candidates for this attack category (CART and NB) are shown in Figure 4.9. Figure 4.10 also clearly shows the prediction time difference between CART and NB when using the preferred choice of six features. Figure 4.11 shows the combined accuracy of all five algorithms across the three feature selection results while Figure 4.12 presents results which include other model implementation metrics such as precision, recall and F1 score. Here it is seen that the critical metric of recall is highest for CART across all other algorithms.



(a) CART Predict Time (b) KNN Predict Time

Figure 4.9: Implementation Results for DDoS

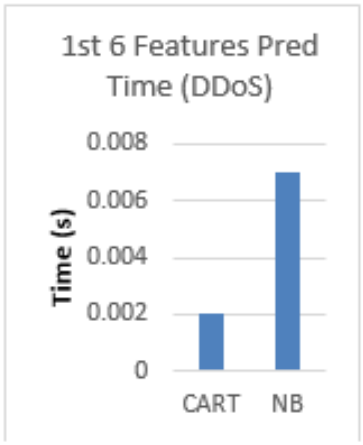


Figure 4.10: Prediction Times for CART and NB Using 1st 6 Feature Set for DDoS

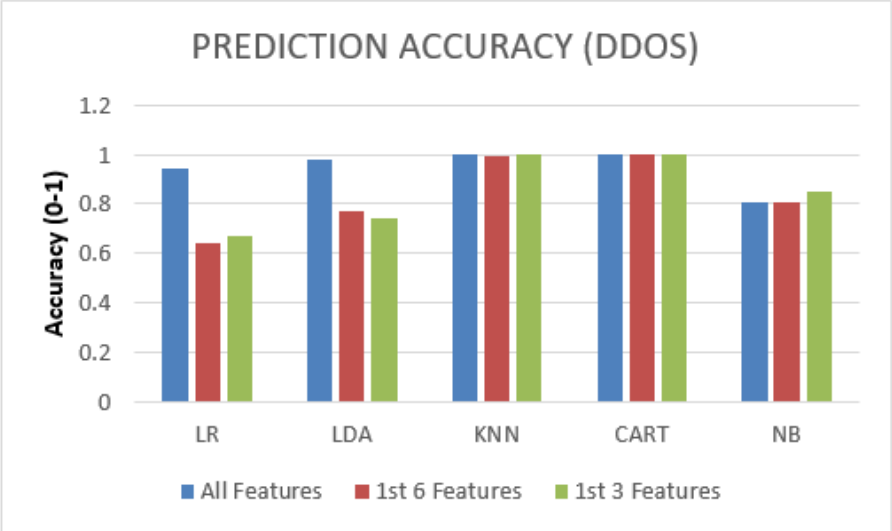


Figure 4.11: Prediction Accuracy for DDoS Across the Different Feature Selection Results DDoS

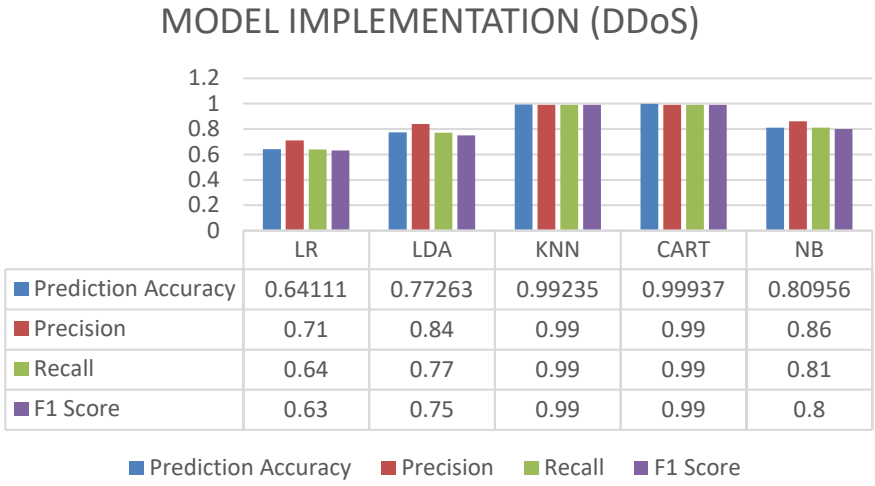


Figure 4.12: Model Implementation Metrics for DDoS

The Model Implementation results presented thus far provides a framework for the choice of an optimal Classifier Model for the design of the 1st stage classifier for the proposed Hierarchical Machine Learning security architecture for SDN. To further expand the framework, additional performance measures need to be evaluated. The confusion matrix puts real numbers to the prediction accuracy results and helps to further explain each model's performance. The confusion matrix for CART implementation for DDoS is presented in Figure 4.13a. It shows a total of 45,143 traffic flows in the test/validation dataset out of which 19,679 entries are benign and 25,464 are DDoS traffic. The CART Classifier failed to classify 18 DDoS traffic flows returning a

false negative of 18 while also returning a false positive of 10, failing to classify 10 benign traffic. The 1st stage Classifier of the proposed Hierarchical Machine Learning Architecture is required to have very low false negative as has been explained in Sections 4.1 and 5.5. From the Model Implementation framework considered thus far, the LDA provides the lowest false negative of 2 as seen in figure 4.13b but provides a very high false positive of 10,262. This explains the reason for the LDA Classifier returning a very high recall of 99.99% on the DDoS traffic (from the output of Scikit Learn but not shown here). While the very low FN (high recall) from LDA is a very useful result, the very high FP (low precision) output will overload the 2nd stage classifier and possibly impact on the overall efficiency of the solution as has been explained in Section 5.5. Further investigation is required to determine the possibility of reducing the FP of the LDA Classifier and hence determining its suitability for use as the 1st stage Classifier. In this regard the CART offers a more balanced combination of FN and FP.

n = 45143	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	19669 TN	10 FP
Actual Yes (DDoS)	18 FN	25446 TP

(a) CART

n = 45143	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	9417 TN	10262 FP
Actual Yes (DDoS)	2 FN	25462 TP

(b) LDA

Figure 4.13: Confusion Matrix for DDoS

For further analysis of the Model Implementation results for CART, a visualisation of the decision tree is presented in Figure 4.14. It is seen that the attribute or feature at the root node is Forward Packet Length Max which was already seen in Table 3.5 as the most important feature from the feature selection process. The train data is 180592 samples (IP flows) as also seen at the root node. The tree has 14 leaf or terminal nodes with all 180592 samples distributed to these nodes after fulfilling required conditions on decision nodes. For further analysis, a section of the decision tree is shown in Figure 4.15. From Figure 4.15, it is seen that the gini value at the root node is 0.491. The gini value at the root node takes into consideration the entire dataset, D. For example, for the DDoS example of Figure 4.15, the CART algorithm calculates this value from Equation 2.1 and is shown thus:

$$Gini(D) = 1 - \sum_{i=1}^{2}(P_i)^2$$

$$Gini(D) = 1 - ((\frac{78047}{180592})^2 + (\frac{102545}{180592})^2) = 0.491$$

While the above calculates the Gini value for the whole dataset across all features, the Gini index for selecting the split attribute, b, for dataset D is calculated using Equation 2.2:

$$Gini\_index(D, b) = \sum_{u=1}^{U} \frac{|D^u|}{|D|} Gini(D^u)$$

which resulted in the choice of Forward Packet Length Max as the most important attribute in the train data. The details of this is not shown on the tree output and is also not shown here due to the large nature of the dataset and the complex nature of the calculations. Further explanation on the operation of the decision tree algorithm is given in Section 2.9.1. The root node is further split to another decision node shown on the left with a sample value of 125641. Since this is a decision node, the process described above is repeated and the node is further split into other child nodes. This process continues recursively until a leaf node is achieved. The root node also splits into a leaf node shown on the right with a sample size of 24951. The gini value for this node is 0 which implies a homogeneous class distribution. In this case all the 54951 samples are classified as benign.

Every machine learning instance of decision tree implemented in this research produces a tree of this nature. In the course of this work, hundreds of decision trees have been generated which all follow the same explanation provided in this section.

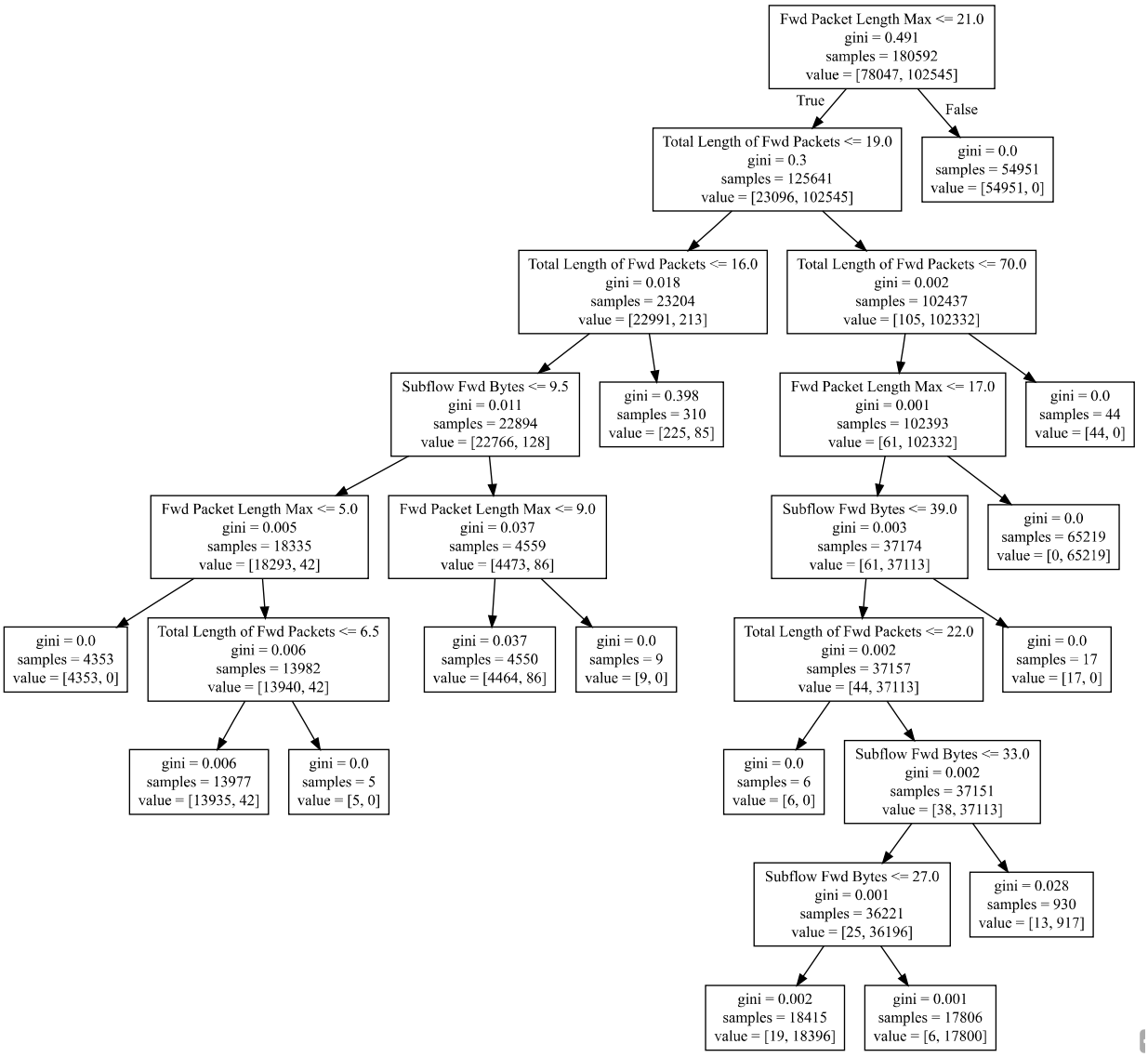


Figure 4.14: CART for DDoS Classification

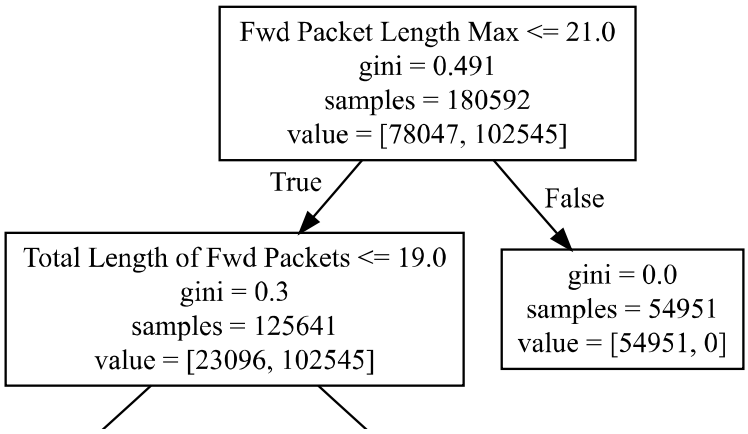


Figure 4.15: CART for DDoS Classification (showing just top three nodes)

4.3.2 Model Implementation Performance for BOTNET

Initial model implementation results for BOTNET attack showing the critical metrics of prediction time and prediction accuracy are presented in Tables 4.6 and 4.7. Similar to the Model Implementation results for DDoS, the results in Table 4.6 show a significant reduction in prediction time across all the algorithms when the features are reduced from all eighty available features to the first six and to the first three. CART returns an 82% reduction in prediction time when all available eighty features are reduced to six features while maintaining its 99.9% prediction accuracy. Although KNN also returns a consistently high prediction accuracy of 99.9%, its prediction times are much higher (5.6s and 0.65s) which makes it relatively unsuitable for the proposed Hierarchical Classifier design. As with the results for DDoS, NB offers a 93% reduction in prediction time but gives very low prediction accuracy across the 3 feature reduction results. LDA and LR return a much reduced prediction accuracy when the feature set is reduced. This once again conforms to the trend captured in the model evaluation and thus validates the model evaluation results. A graphical comparison between the two nearest candidates for this attack category (CART and KNN) are shown in Figure 4.16. Figure 4.17 also clearly shows the prediction time difference between CART and KNN when using the preferred choice of six features. Figure 4.18 shows the combined accuracy of all five algorithms across the three feature selection results while Figure 4.19 presents results which include other model implementation metrics such as precision, recall and F1 score. Here, it is seen that the more critical metric of recall returns a value of over 99% across all algorithms except the Naive Bayes which returns a value of 31.1% recall. However, CART returns a precision of 99.93% (the highest amongst the the other algorithms), which explains the

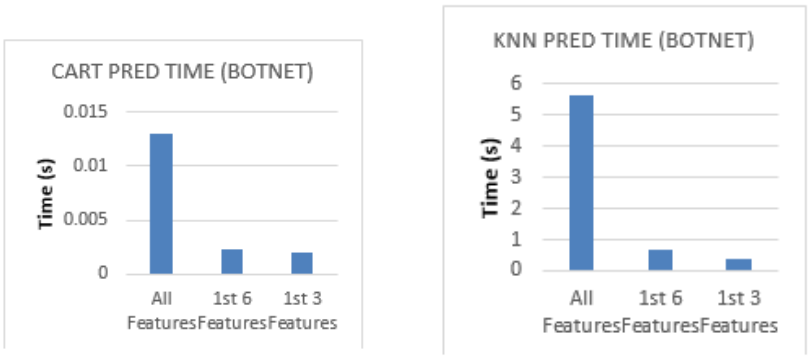
Table 4.6: Prediction Times for BOTNET Using Different Feature Selection Results

PREDICT TIME (BOT)			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.006956	0.000998	0.000997
LDA	0.006983	0.000998	0.000997
KNN	5.654006	0.653103	0.4088
CART	0.01299	0.002338	0.001994
NB	0.084774	0.005985	0.001994

Table 4.7: Prediction Accuracy for BOTNET Using Different Feature Selection Results

PREDICTION ACCURACY (BOT)			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.9881	0.99	0.9899
LDA	0.9827	0.99	0.99
KNN	0.9974	0.9973	0.9954
CART	0.9992	0.9993	0.9952
NB	0.3524	0.3134	0.1769

99.93% F1-Score which is also the highest amongst the other algorithms.



(a) CART Predict Time (b) KNN Predict Time

Figure 4.16: Implementation Results for BOTNET

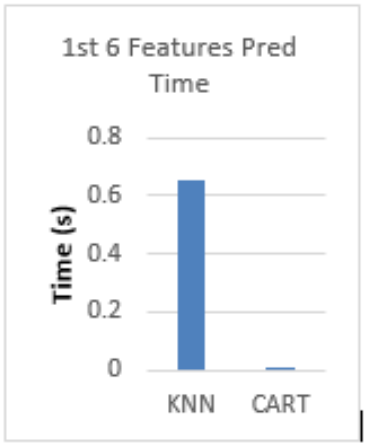


Figure 4.17: Prediction Times for CART and KNN Using 1st 6 Feature Set (BOTNET)

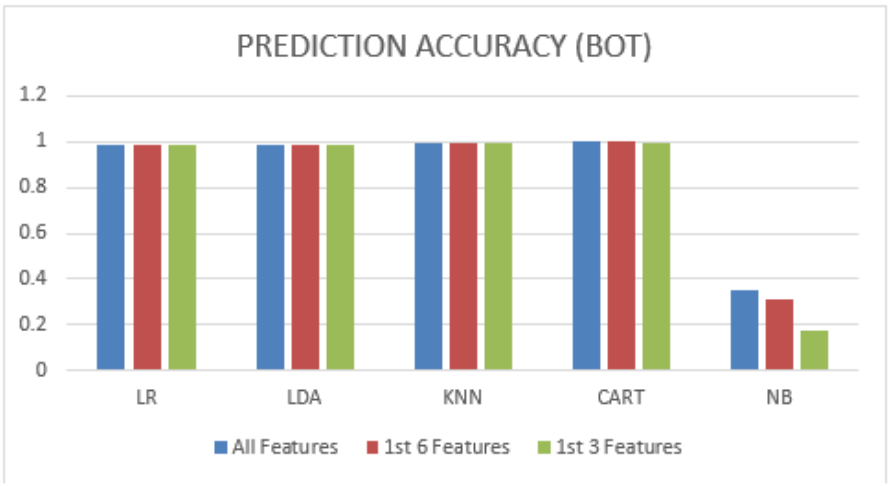
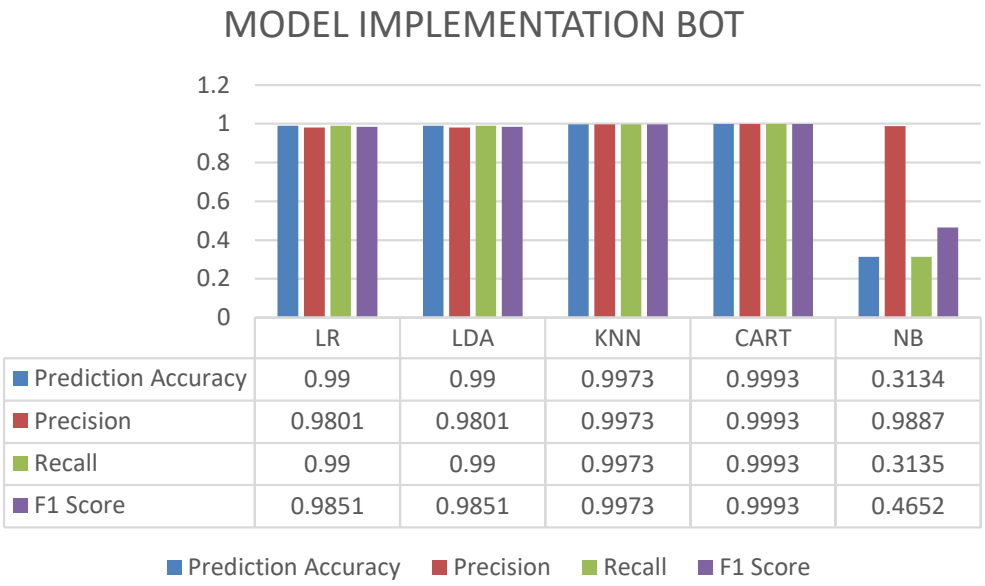


Figure 4.18: Prediction Accuracy for BOTNET across the Different Feature Selection Results



	LR	LDA	KNN	CART	NB
Prediction Accuracy	0.99	0.99	0.9973	0.9993	0.3134
Precision	0.9801	0.9801	0.9973	0.9993	0.9887
Recall	0.99	0.99	0.9973	0.9993	0.3135
F1 Score	0.9851	0.9851	0.9973	0.9993	0.4652

Figure 4.19: Model Implementation Metrics for BOTNET

To further expand the model implementation performance results presented thus far for BOTNET, the Confusion Matrix output from selected classifiers are presented. For BOTNET attacks, the confusion matrix for CART gives the best results. The Confusion Matrix for CART is presented in figure 4.20a and shows that there are 38,183 flows in the test data, of which 37,802 are benign flows while 381 are BOTNET attacks. Of the 37,802 benign flows, the Classifier returned a FP value of 14 while returning a FN value of 11 from the 381 BOTNET attack flows. From figure 4.20b, NB returns a low FN of 17 but returns but a very high FP of 26,196.

n = 45143	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	19669 TN	10 FP
Actual Yes (BOTNET)	18 FN	25446 TP

(a) CART

n = 45143	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	11606 TN	26196 FP
Actual Yes (BOTNET)	17 FN	364 TP

(b) KNN

Figure 4.20: Confusion Matrix for BOTNET

4.3.3 Model Implementation Performance for Portscan

Initial model implementation results for Portscan attack showing the critical metrics of prediction time and prediction accuracy are presented in Tables 4.8 and 4.9. As with previous results, Table 4.8 shows a significant reduction in prediction time across all the algorithms when the features

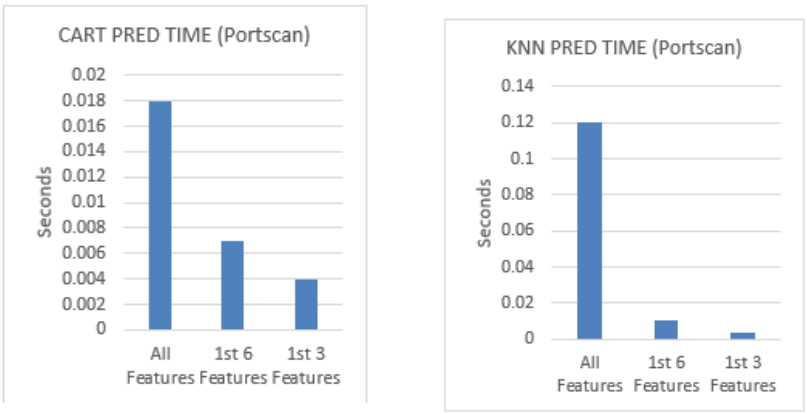
Table 4.8: Prediction Times for Portscan Using Different Feature Selection Results

PREDICT TIME (PSCAN)			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.015584	0.000998	0.000999
LDA	0.015621	0.001001	0.000998
KNN	22.390143	3.288231	1.484005
CART	0.017952	0.007008	0.004017
NB	0.119708	0.010971	0.003989

Table 4.9: Prediction Accuracy for Portscan Using Different Feature Selection Results

PREDICTION ACCURACY (PSCAN)			
Algorithm	**All Features**	**1st 6 Features**	**1st 3 Features**
LR	0.9483	0.4074	0.4055
LDA	0.9886	0.762	0.6421
KNN	0.9992	0.9947	0.9947
CART	0.9998	0.9979	0.9975
NB	0.6907	0.6077	0.6045

are reduced from all eighty available features to the first six and to the first three. CART returns a 61% reduction in prediction time when all available eighty features are reduced to six features while maintaining its 99.9% prediction accuracy. KNN is the only other algorithm that returns a consistently high prediction accuracy of 99.9%, but its prediction times are much higher (22.3s and 3.2s) which makes it relatively unsuitable for the proposed Hierarchical Classifier design. NB, LDA and LR all return a much reduced prediction accuracy when the feature set is reduced. A graphical comparison between the two nearest candidates for this attack category (CART and KNN) are shown in Figure 4.21. Figure 4.22 shows the combined accuracy of all five algorithms across the three feature selection results while Figure 4.23 presents results which include other model implementation metrics such as precision, recall and F1 score. Once again, it it is seen here that CART consistently posts the highest of all the metrics including the critical recall metric, amongst all the other algorithms.



(a) CART Predict Time (b) KNN Predict Time

Figure 4.21: Implementation Results for Portscan

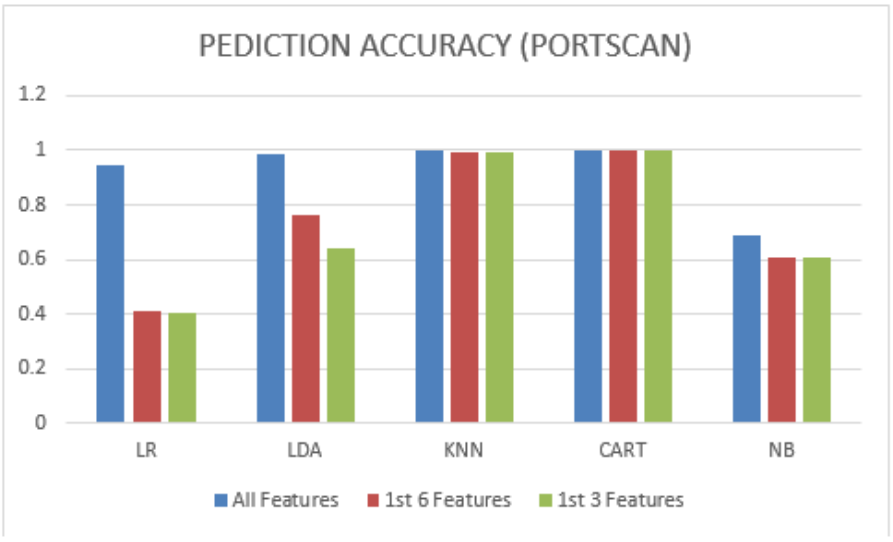
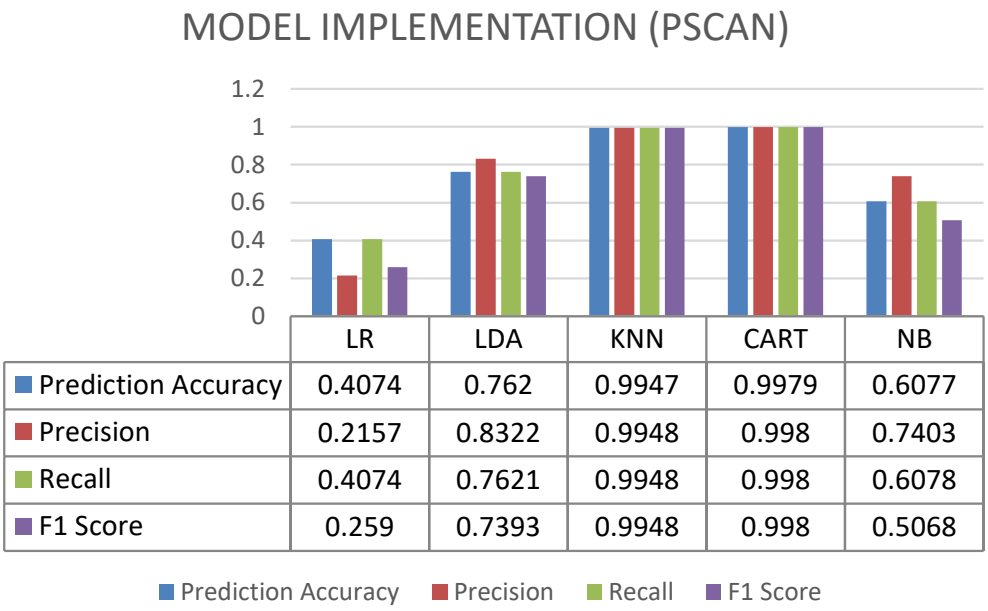


Figure 4.22: Prediction Accuracy for Portscan across the Different Feature Selection Results



	LR	LDA	KNN	CART	NB
Prediction Accuracy	0.4074	0.762	0.9947	0.9979	0.6077
Precision	0.2157	0.8322	0.9948	0.998	0.7403
Recall	0.4074	0.7621	0.9948	0.998	0.6078
F1 Score	0.259	0.7393	0.9948	0.998	0.5068

Figure 4.23: Model Implementation Metrics for Portscan

Considering the Confusion Matrix for Portscan, Figure 4.24b shows that LDA returns the least FN value of 29, resulting in the highest recall value of 99.9% (not shown in the report). However, LDA gives a very high FP value of 13586. While a very low FN and high recall is the desirable requirement for the design of the 1st stage Classifier, further investigations may need to be carried out to determine the suitability of LDA for the 1st stage Classifier design. CART on the other hand provides a more balanced combination of FN and FP and gives a recall value of 99.6%. The confusion Matrix for CART is shown in figure 4.24a.

The work of this section has shown that the CART decision tree algorithm is the most appropriate from those tested as it has high precision and recall with low run-time requirements.

n = 57220	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	25271 TN	0 FP
Actual Yes (Portscan)	117 FN	31832 TP

(a) CART

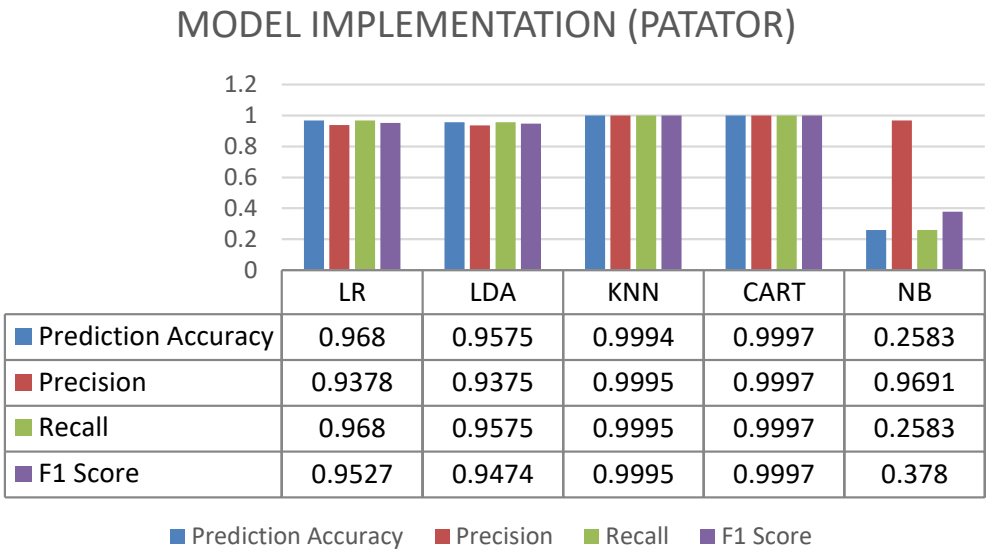
n = 57220	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	11685 TN	13586 FP
Actual Yes (Portscan)	29 FN	31920 TP

(b) LDA

Figure 4.24: Confusion Matrix for Portscan

4.3.4 Model Implementation Performance for Patator

Figure 4.25 shows the model implementation metrics for patator attack. As can be seen, these also favour the choice of the CART algorithm as can be seen from results of the critical metric of recall and other performance metrics. The other analysis shown for the previous attack types have been excluded for this attack as there is no significant deviation in the analysis.



	LR	LDA	KNN	CART	NB
Prediction Accuracy	0.968	0.9575	0.9994	0.9997	0.2583
Precision	0.9378	0.9375	0.9995	0.9997	0.9691
Recall	0.968	0.9575	0.9995	0.9997	0.2583
F1 Score	0.9527	0.9474	0.9995	0.9997	0.378

Figure 4.25: Model Implementation Metrics for Patator

4.4 The Single Model Problem

The single model problem arises from the fact that the machine learning algorithm is trained on traffic with a specific attack profile. Whereas, in a practical scenario, the intrusion detector is exposed to multiple attack profiles. The single model problem refers to using a different attack traffic as input to an intrusion detection model that has been trained on a different attack type. This experiment is necessary to highlight and to better understand the advantage of using a single class classifier; the metrics to focus on are the precision and recall. From the results as shown in Figure 4.26, it is seen clearly that there exists a potential problem whenever the attack type does not match the trained classification model. From the result, it is seen that BOT traffic on a DDoS Model yields impractical 6.7% precision and 38.8% recall values while same BOT traffic on a PSCAN Model yields 1.64% precision and 34.72% recall on the attack traffic. Similar trends occur for the other attack traffic and different models as shown in the figure except for DDoS traffic on PSCAN Model which gives relatively high precision and recall values of 81.22% and

63.15% respectively. A possible reason for this is that port scanning attacks is one of the means by which DDoS is achieved. While typical port scanning attacks can be used to achieve DDoS, not all DDoS attacks are achieved by port scan. Hence the traffic profile of both attacks may contain some similarities which may reflect in the PSCAN model allowing the PSCAN model return a fairly accurate prediction when DDoS traffic is applied to it. Is is also seen that precision and recall for benign traffic across all options are consistently high with very few exceptions. This is because the traffic profile of the benign traffic across all datasets is expected to be similar. An important conclusion for this experiment/simulation is that results could be used to investigate relationships between different attack types. Additionally, this gives justification to seek better design scenarios which is addressed in subsequent sections.

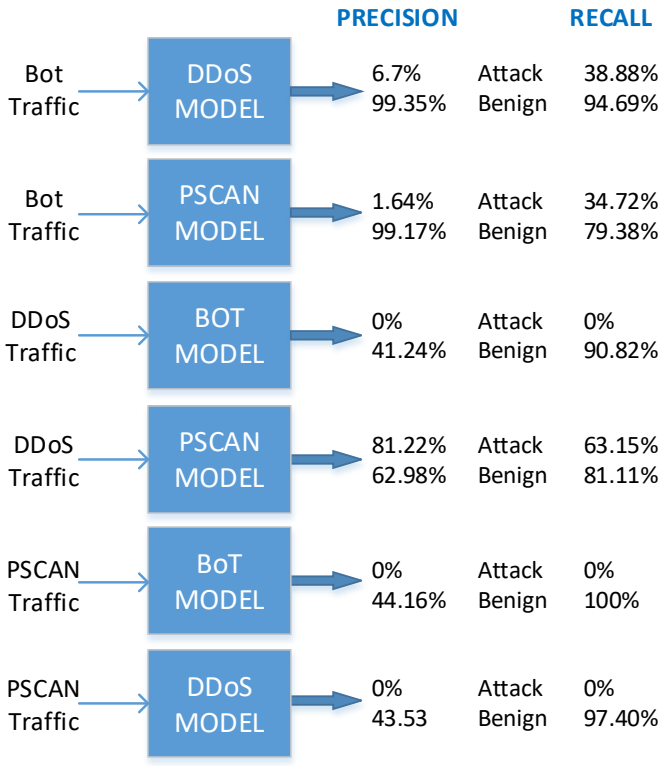


Figure 4.26: The Single Model Problem

4.5 Classifier Design Scenarios

The results in Section 4.3 suggest a decision-tree based approach. The combined results of model evaluation and implementation presented in the previous sections show that the CART model delivers the best combination of results to be the classifier for the intrusion detection module.

Hence, the analysis presented in this section will be entirely based on the Decision Tree model. This section demonstrates the different possible ways the final classifier model could be implemented to develop the intrusion detection module. Once again, the critical metrics are the predict time, precision, recall and F1 score. Here, the predict time has been modified to express rate (flow per second) which can be described as analogous to data rate, in an attempt to standardize the time metric. This is necessary because the different scenarios do not utilise the same size of dataset (in terms of flows), and as such a straight timing measurement could be misleading.

4.5.1 Scenario 1 - Single Multiclass Classifier

In this scenario, the 3 attack datasets of DDoS, BoT and PSCAN are merged into a single composite dataset containing 13 features. The 6 important features (as established in section 3.3.1) of each attack is contained in the composite 13 features. The composite (merged) attack data is used to train a CART model with 4 labels; benign, DDoS, BoT and PSCAN. Figure 4.26 illustrates the single multiclass classifier scenario and also shows a summary of the results. The results show an average 99.87% precision across the 3 attack types resulting in a similar 99.87% recall and f1 scores. Also, the results show that the predict time for Scenario 1 is 0.023967s (3.8m flows/s.)

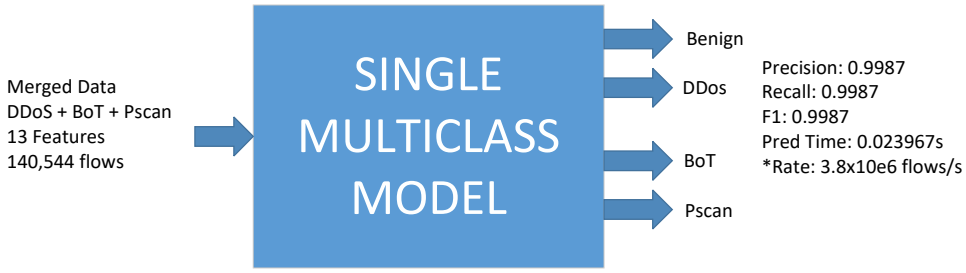


Figure 4.27: Design Scenario 1 - Single Multiclass Classifier

4.5.2 Scenario 2 - Merged Multiple Binary Classifier

In this scenario, 3 separate models are deployed in parallel. Each model is trained on each of the 3 different attacks. Hence, the 3 separate models are for DDoS, BoT and PSCAN. The same composite dataset that was used in scenario 1 is also utilized for this scenario. A filter is placed ingress to each classifier to ensure that each classifier only receives data with the features with which it was trained. The layout for the Merged Multiple Binary Classifier is shown in Figure 4.28 which also shows a summary of the results. From the results, the DDoS portion of this model gives

an attack precision and recall value of 99.91% with a response time of 0.02236s (14.06 flows/s). The BoT portion gives an attack precision of 55.53% and recall of 99.76%. A possible reason for the irregular BoT results is the imbalanced nature of the BoT attack data. For the PSCAN portion of the model, the attack precision and recall are given as 72.25% and 99.95% respectively. The response times for the BoT and PSCAN sections of the model are 0.12169s (10.84m flows/s) and 0.025958s (11.74m flows/s) respectively.

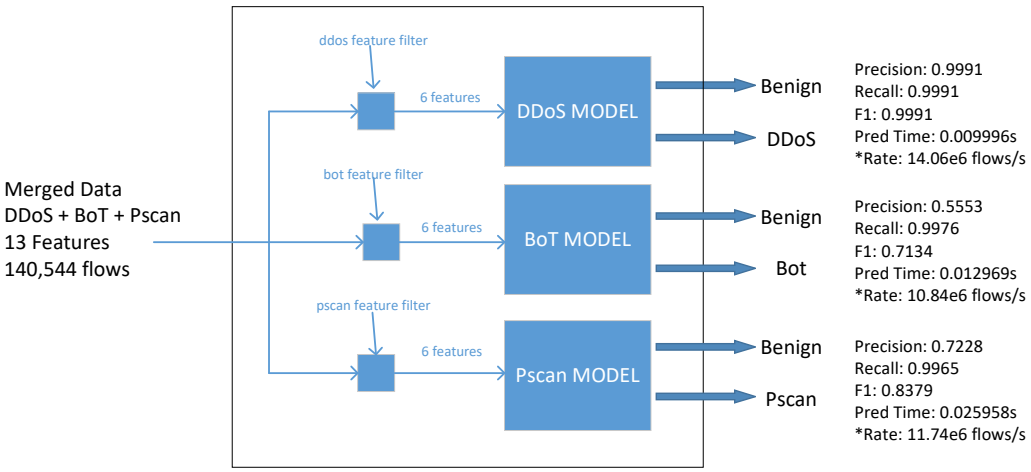


Figure 4.28: Design Scenario 2 - Multiple Parallel Classifier

4.5.3 Scenario 3 - Isolated Multiple Binary Classifier

The implementation of scenario 3 is similar to scenario 2. The difference is in the application of the test attack data. In this case, the 3 attack datasets are kept separate and each is fed to the corresponding sub classifiers (which operate in parallel) and the required metrics of attack precision, recall, F1 score and time/rate are obtained. This scenario gives an average attack precision and recall output of 99.68% each across all 3 sub classifiers with prediction time of 0.002991s (15.09m flows/s), 0.001996s (19.13m flows/s) and 0.002993 (19.12m flows/s) for DDoS, BoT and PSCAN respeectively. The layout of the isolated multiple binary classifier is shown in figure 4.29.

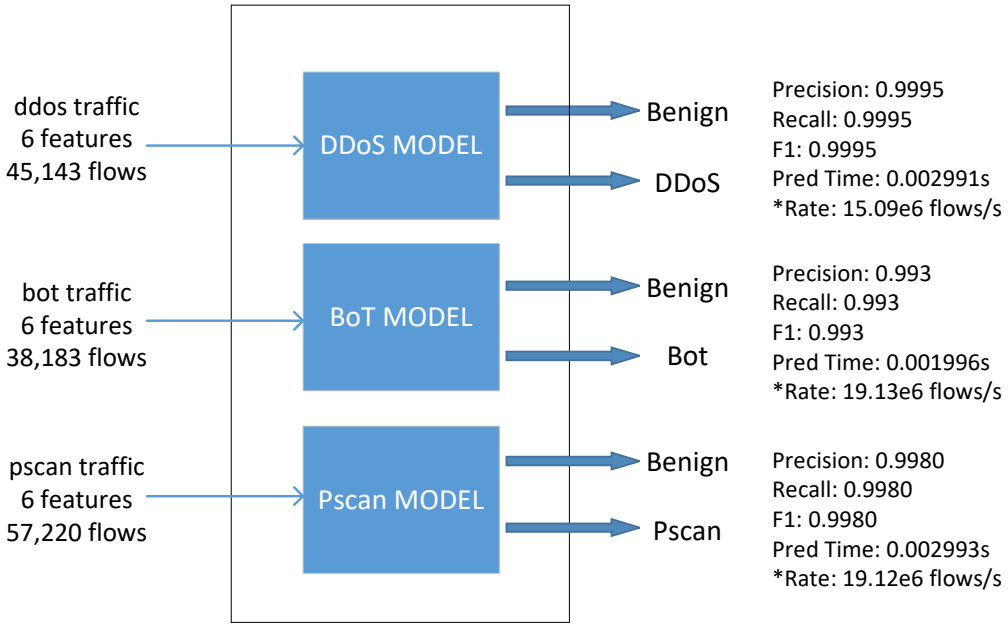


Figure 4.29: Design Scenario 3 - Isolated Parallel Classifier

4.6 Design Scenario Analysis

Scenario 1 offers the indications of being the most practical solution of the three. It gives better precision and recall scores than scenario 2. One of the drawbacks of scenario 2 is the distributed nature of the classification models which has the potential of making it more processor intensive. Scenario 3 has both distributed classification models and distributed datasets, both of which could make implementation quite problematic.

For a composite attack dataset, the straight timing (predict time) metric could be more appropriate. If comparison is being made between 2 models or 2 scenarios with different test datasets (different size or number of flows), then the rate metric might be more appropriate because the straight time will not be absolute, given the different volume of data and hence the different workload of the model. However, the rate metric (flow/s) could also have its own inherent drawback given that each flow is a different size (bytes or packets). Hence flow/s may not be be a standard timing metric across different classifiction models.

4.6.1 Design Scenario Analysis Based on Recall Metric

Recall is a measure of the TP rate and gives a measure of the percentage of attack flows that was correctly predicted/classified. Out of all the attack flows, the recall determines what percentage was correctly classified. Given the nature of the proposed hierarchical solution, this requirement is considered most critical hence the recall metric is given the highest prority value. Equation 4.1 clearly shows recall to be the ratio of TP to all attack flows. In essence Recall answers the question: of all the attack flows what proportion was correctly classified by the Model.

The results for Design Scenario 1 gives the Confusion Matrix shown in Figure 4.31. Calculating the recall for scenario 1 gives:

$$\text{Recall}_{Scenario1} = \frac{25,374 + 352 + 31,643}{25,374 + 352 + 31,643 + 10 + 71 + 3} = \frac{57,369}{57,453} = 0.9985$$

Similar procedure gives recall values for Design Scenario 2 as 0.991, 0.9976 and 0.9965 with an average value of 0.9977.

Also, similar procedure gives recall values for Design Scenario 3 as 0.9995, 0.9930 and 0.9980 with an average of 0.9968.

As already seen in section 4.6, Scenario 1 provides the best results for recall which is critical for the Hierarchical Machine Learning SDN Security solution. However, the differences between the recall values are small, consequently, if run-time performance was critical Scenario 3 might be suitable.

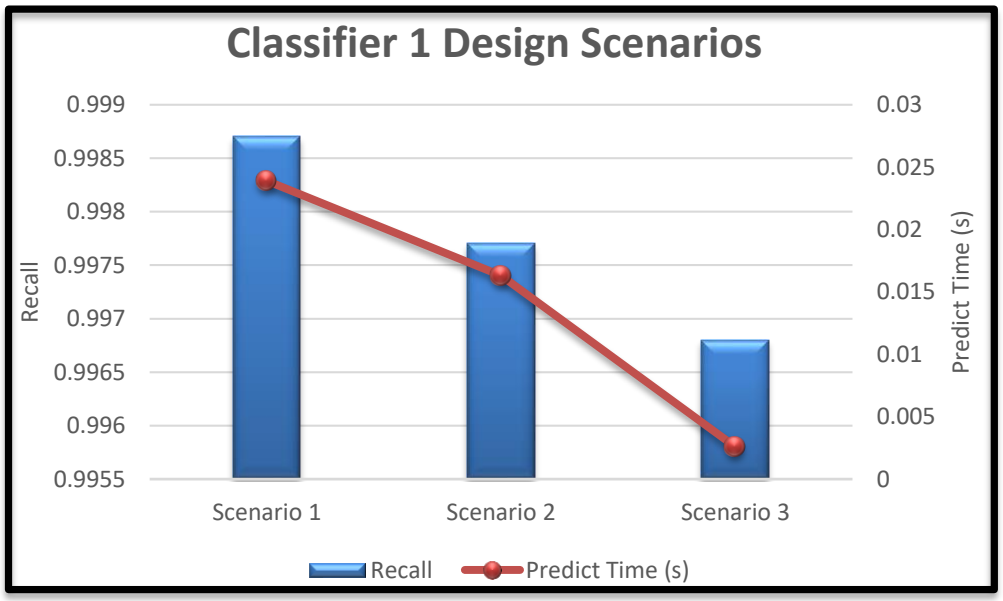


Figure 4.30: Design Scenario 1 Analysis based on Recall Metric

n = 140,544	Predicted Benign	Predicted DDoS	Predicted BOT	Predicted PSCAN
Actual Benign	**82992 TN**	8 FP	85 FP	6 FP
Actual DDoS	10 FN	**25374 TP**	0 FN	0 FN
Actual BOT	71 FN	0 FN	**352 TP**	10 FN
Actual PSCAN	3 FN	0 FN	0 FN	**31643 TP**

Figure 4.31: Design Scenario 1 Confusion Matrix

4.6.2 Design Scenario Analysis Based on F1 Score Metric

The F1 Score incorporates the precision metric into its value in addition to recall. Precision is a measure of how accurate or precise the attack classification has been and is given by Equation 4.2: which gives a ratio of correct attack classification to all attack classification. In essence, precision answers the question: of all the attack predictions given by the Model, what proportion is correct. Hence the F1 Score gives a more balanced measure as against the recall metric which give a more

specific measure. F1 Score is given by Equation 4.3: For Design Scenario 1:

$$\text{F1 Score}_{Scenario1} = \frac{2(0.9987 * 0.9987)}{0.9987 + 0.9987} = 0.9985$$

Similar procedure gives F1 score for Scenario 2 as 0.9991, 0.7134, 0.8379 giving an average of 0.8499. Similar procedure gives F1 score for Scenario 3 as 0.9995, 0.9930, 0.9980 giving an average of 0.9968.

As already seen in Section 4.6, Design Scenario 1 returns a marginal F1 Score advantage over Design Scenario 3 with Design Scenario 2 returning an undesirably low F1 Score. This further emphasises the suitability of Design scenario 1 for the Hierarchical Machine Learning SDN Security solution.

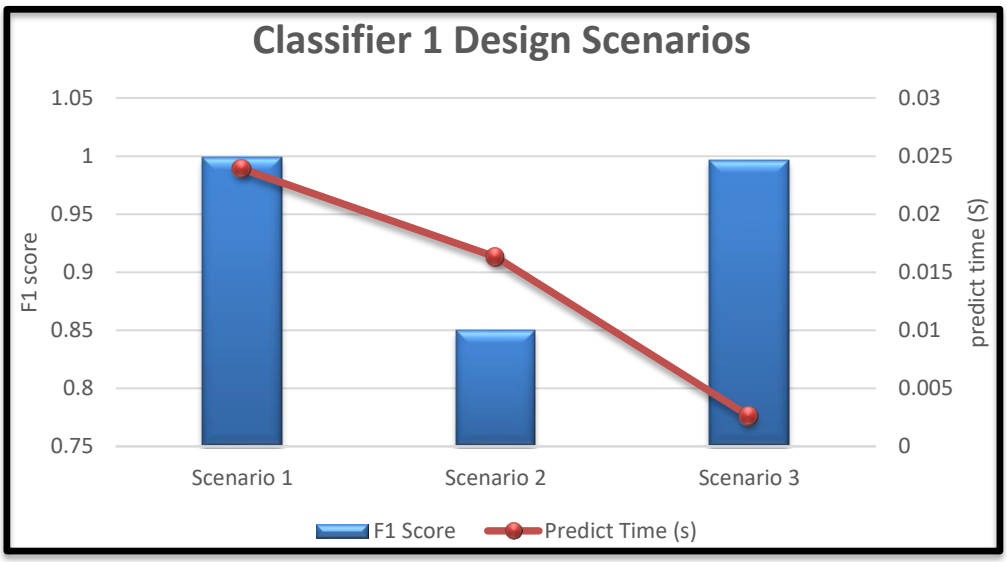


Figure 4.32: Design Scenario 1 Analysis based on F1 Score Metric

4.7 Simplified Emulation of Hierarchical Intrusion Detection Solution

To give motivation for the later chapters, the work in this section looks at initial performance consideration of the hierarchical intrusion detection solution. Figure 4.33 shows the updated version of the hierarchical solution given the results of the intrusion detection module design. The figure shows the Decision Tree Model (CART) being adopted as the classifier for the first and

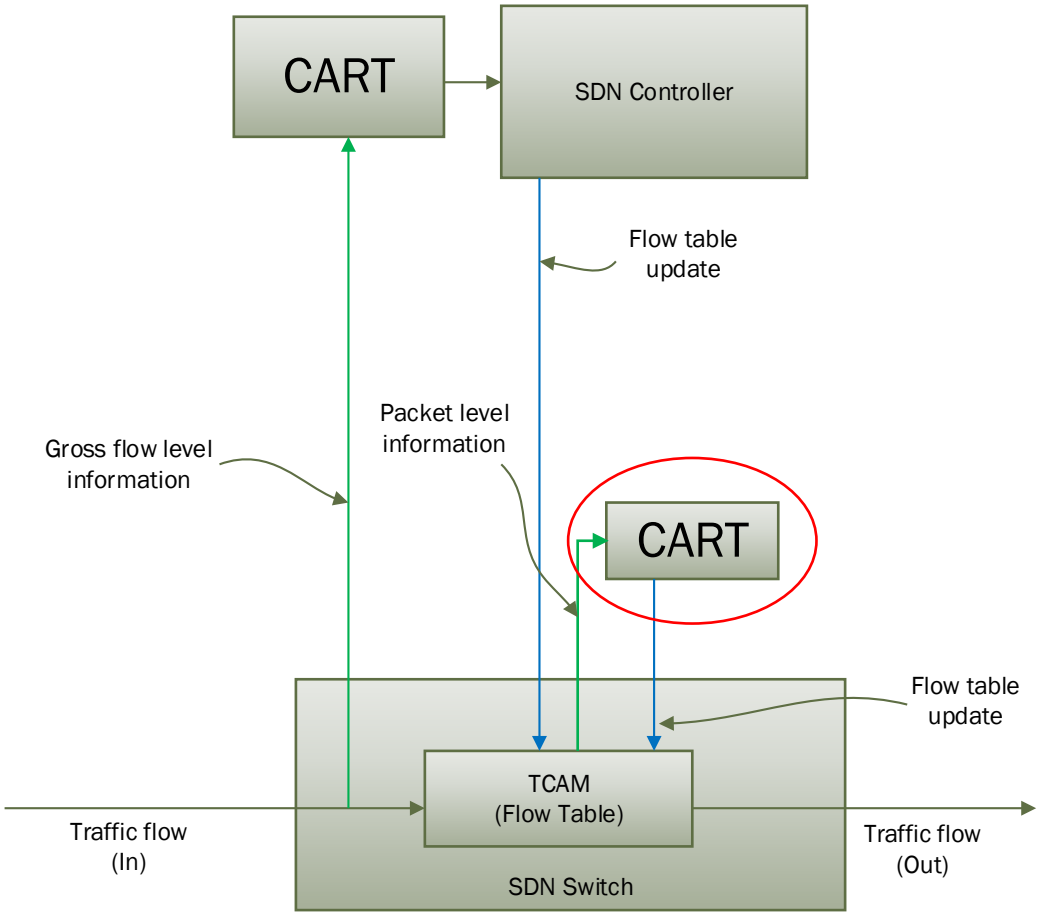


Figure 4.33: Hierarchical Solution Showing Classifier Model

second stage. It is important to state that this does not represent the actual operation of the second stage. This is described in Chapter 5, but is necessary at this stage of the work, to demonstrate the potential benefits of moving from a centralised classifier alone to the proposed two-stage classifier. Figures 4.34 and 4.35 illustrate the difference in operation between the simplified emulation of the second stage and the actual operation of the second stage that is used in the later chapters. In Figure 4.34, gross flow level information from the traffic is fed into the classifier from where traffic is classified as good or bad and the flow table updated to drop bad traffic.

Results for this comparison are shown in Fig. 4.36 for three different attack types: DDoS, BOT and PSCAN. These results are using the selected CART algorithm for Classifier 2 with the 6 most important features (derived from Section 3.3.1 and compare the effort if only implementing the single stage classifier at the edge vs implementing the two stage classifier. The results show a 44% and 43% reduction in traffic volume for PSCAN and DDoS respectively and 85% and 50%

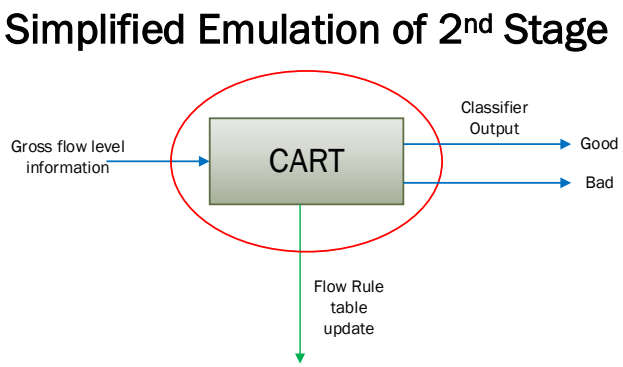


Figure 4.34: Simplified emulation of 2nd stage used to estimate performance benefits of a hierarchical design.

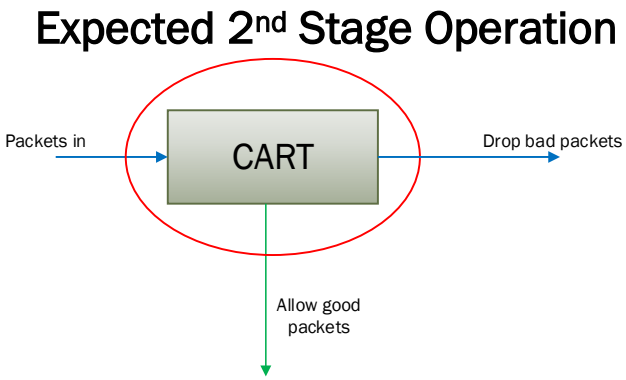


Figure 4.35: Actual 2nd Stage operation as used in later chapters.

reduction in effort for PSCAN and DDoS respectively. The traffic volume is measured by number of flows while effort is measured by predict or classification time. The results show the reduction on effort in the second stage classifier is significant, in particular for the BOT and PSCAN traffic (with BOT nearly zero). Given that effort is directly proportional to traffic volume, the reduced traffic volume resulting from the intrusion detection at the first stage has resulted in reduced effort at the second stage.

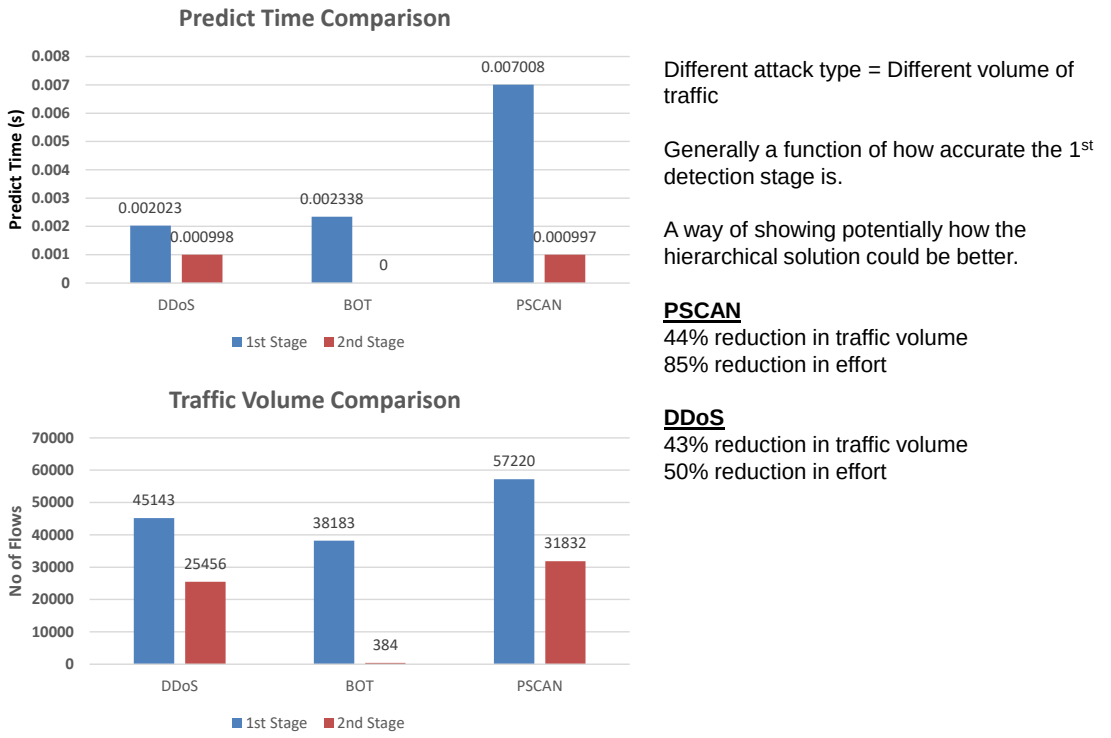


Figure 4.36: 2nd Stage Emulation Results

4.8 Summary

The main contribution of this Chapter is the design of a unique machine learning based intrusion detection model which shows potential for hierarchical operation in SDN. The hierarchical solution requires a machine learning based intrusion detector with quick response time and high recall value. This is designed to allow for a near real-time attack detection at a reasonably high accuracy level. A reasonably low model train time is also required to allow for quick deployment and dynamic update of the model. A machine learning algorithm learns from experience: practical applications need to incorporate some form of feedback mechanism that responds to changes in attack profile. One method to achieve this is by adopting reinforcement learning. This feedback mechanism is

not within the scope of this work. However, to accommodate for this technique within this study is one of the reasons why the model train time has been considered as a design parameter for the intrusion detection model. Among the five machine learning algorithms tested, the non-linear algorithms Decision Tree Classifier provided the best response to these specifications. This solution has been evaluated using the CICIDS2017 dataset.

Also, experiments which support a better understanding of the single model problem was carried out. The experiments show that it is more effective to have different intrusion detector models for different attack profiles. To reduce the effect of this problem, different design scenarios were created. Results from the design scenarios support the use of a single multi-class model which has been trained on a combination of different attack profiles.

Work done in this chapter has been synthesised to develop a design framework that can be adapted to different network traffic datasets and to different machine learning algorithms to create robust and efficient intrusion detector module for network security applications.

The chapter concludes by presenting a simplified operation of the proposed hierarchical security solution. Results from this experiment demonstrates substantial promise in the area efficient intrusion detection in a network security application.

Chapter 5

Hierarchical Intrusion Detection with Categorical Classification

5.1 Introduction

Chapter 4 explored the use of machine learning algorithms for malicious flow detection in network traffic. This chapter will apply the results and concepts from Chapter 4 to an enterprise solution. Figure 5.1 shows the enterprise model of the proposed solution. The use of SDN allows a hierarchical approach to machine learning with the aim of reducing the packet level processing for malicious packet detection at the edge through applying centralised machine learning in the SDN controller. Results from the classifier design analysis from Chapter 4 support a decision-tree based approach and show that it promises a considerable reduction in the per-packet processing at the network edge compared to a single stage classifier as illustrated in Section 4.7. The fast predict time of the decision-tree based classifier also allows for a near real-time detection of attack traffic which is also a function of the attack type.

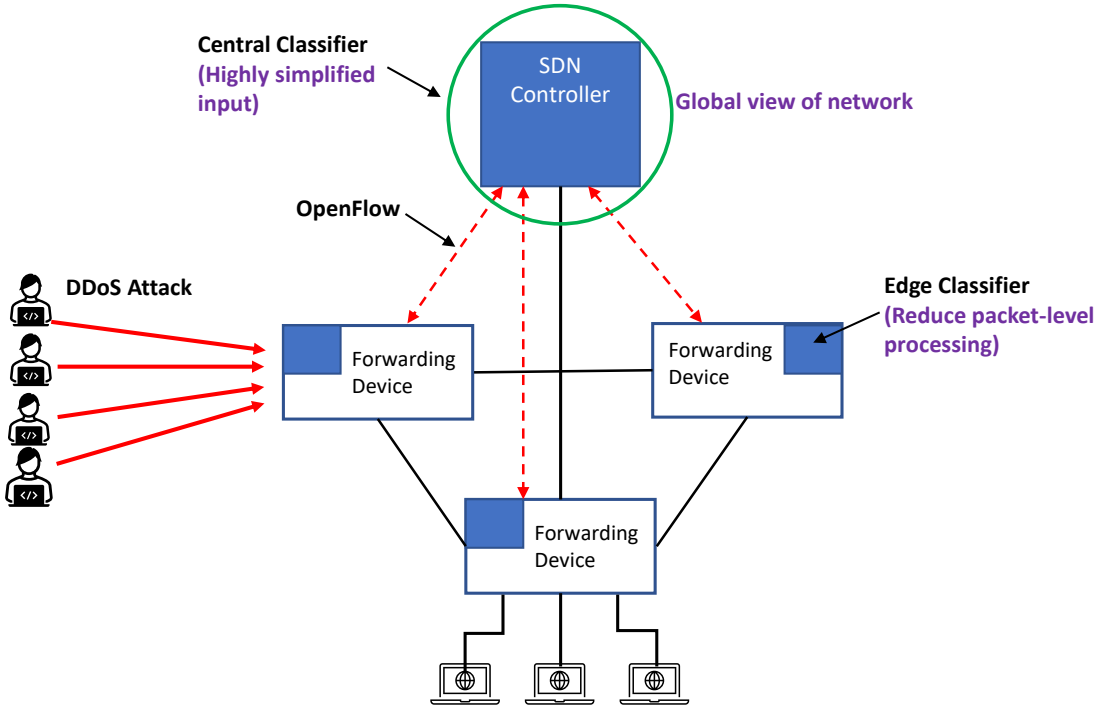


Figure 5.1: Enterprise View of Hierarchical Solution

The proposed SDN intrusion detection solution integrates a 2-stage Machine Learning instance as shown in Figure 5.1. The first instance of machine learning, *i.e.*, Classifier 1, works on summarised network flow traffic features based on gross level flow information available through OpenFlow counters. This is further explained in Section 3.4.4. The central or first stage classifier is used to identify potentially harmful network traffic which is then redirected into a second intrusion detection stage or packets dropped by forwarding device.

Work in this chapter presents results from the intrusion detection process of both the first and second stage classifiers. The general overview of work done in this chapter is as represented in Figure 1.3. At this point, it is unclear if Classifier 2 may also be deployed to provide additional functionalities such as QoS and Routing Optimization or Resource Management [51], which could be a source of further investigation and outside the scope of this work. Previous topologies presented have not focused much on the actual traffic flow and the mechanisms around it. The Machine Learning intrusion detection process is not intended to replace the standard SDN traffic flow decision process but to complement it. Also a standard access control or intrusion detection system will match the properties of this first packet against any pre-configured security rules. This is insufficient for dynamic intrusion detection because an attacker can clone the first packet to bypass the pre-configured security rules. Hence the use of measured aggregate flow statistics

for intrusion detection using machine Learning. However, work done in Chapter 4 is based on completed flows. This represents the limitation of most published research work in this area. This is the major motivation behind re-engineering the data to simulate real-time dynamic traffic flow.

5.2 Limitations of Machine Learning Based IDS Research

Results from work done in Chapter 4 are based on flow statistics derived from completed IP traffic flows without much recourse to flow duration and hence duration of attack traffic. Flow duration as defined in Section 2.4.2 is the time difference between the first observed packet of a flow and the last observed packet of the same flow. According to [45], most detection methods begin investigating network attacks after these attacks have occurred and have already and have already caused considerable damage to the system. Other recent works such as [127], [128], [129], [100], [1], [130], and [131] have also not considered flow duration in their machine-learning based solutions.

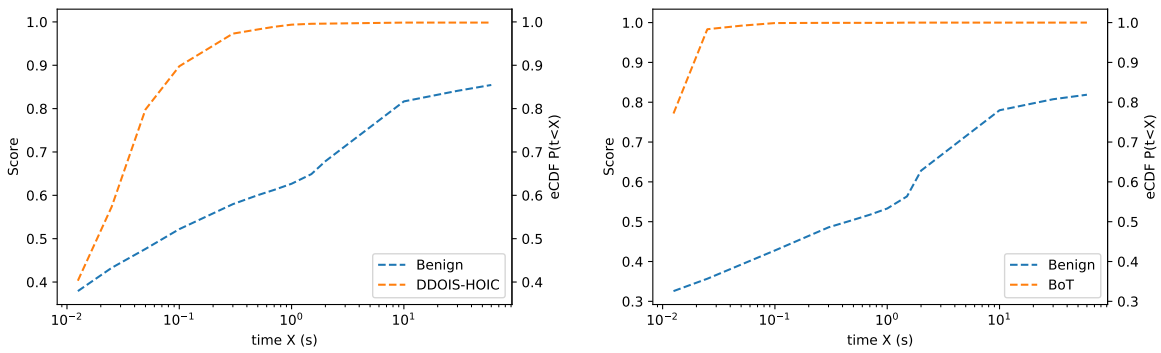
This presents a challenge for real world applications as the attack flow would have already passed through at the time of detection. Reducing the data dimensionality using feature selection is not sufficient to ensure real time intrusion detection. This gives rise to the research question as stated in Section 1.4; what portion of a network traffic flow is required to get reasonable intrusion detection or at what time in the duration of a network traffic flow can a reasonable intrusion detection accuracy be obtained in SDN using Machine Learning techniques? This is one of the questions that the remainder of this research will answer.

5.3 Sub-Flow Enabled Real Time Intrusion Detection

One way to address the limitations discussed in Section 5.2 is to create subsets of IP traffic flow and perform intrusion detection on each subset to determine how well attack traffic will be classified or predicted at intervals within the lifespan of the flow. Many Papers and research works miss this crucial point as stated in Section 5.2 In this section, this concept will be implemented using the CICIDS2018 Datasets. First, the dataset is re-engineered to create the flexibility to alter traffic flow duration. A total of forty six flow features and fourteen sub-flows have been generated for each attack. Detailed explanation of the IP traffic re-engineering process has been given in Section 3.4.1.

5.3.1 Sub-Flow IP Traffic Classification

Due to the flexibility of the re-engineered data, time-based IP traffic subflows are sampled at specific subflow threshold values during the flow's lifespan. The flow statistics for each subflow correctly reflects the threshold value. For the purpose of this analysis, a flow active time of two minutes is assumed for all flows. To properly visualise the temporal distribution of the various IP traffic on the labeled datasets, an empirical cumulative distribution function (eCDF) for flow duration is applied. Sample eCDF for DDOS and BOT traffic are shown in Figure 5.2. From the figure, it is seen that at approximately 10s, nearly 100% of all the DDoS attack would have occurred. This means that any real time malicious traffic detection must occur well before this time. The BoT attack duration is even smaller with nearly 100% of all BoT attack traffic occurring before 0.1s. This concept will be utilised in analysing the results of the hierarchical solution in Section 5.6. A semi-logarithmic scale is used to search sub-flows each of which represent a snapshot of the complete flow at the instance of time. The complete eCDF curves are given in Section 5.6 as part of the results for the hierarchical solution.



(a) eCDF for DDOS Traffic

(b) eCDF for BOT Traffic

Figure 5.2: Empirical Cumulative Distribution Function for Re-engineered Data

The intrusion detection module used for this analysis is the CART Decision Tree Classifier obtained from Chapter 4. As explained in section 5.3.1, the classifier is separately trained on each sub-flow. This is explained by the fact that, in practice, if a classifier is deployed to detect attack traffic at say 2s flow interval, it is expected that the classifier should be trained with traffic data of corresponding characteristics. However, one of the main objectives of this work is to investigate how the classifier responds to a change in flow duration and a change in flow monitoring approach. A summary of the sub-flow classification results for a cross-section of the attack traffic is shown

in Figure 5.3. The classifier works quite well on automated attacks such as DDOS and BOT. As expected, the classifier does not work as well on non-automatically generated attacks e.g. WEB-XSS which is the result of a user accessing a vulnerable page, the attack, from a traffic level, looks identical to "good" traffic. In practice this would need to be detected using "end-point" detection that looks inside packet contents and server log-files. As seen, there is less significant change in F1 score for the automated attacks of DDOS and BOT across the different sub-flows. This is not the case with the Web attacks for which classifier performance significantly degrades at lower sub-flow times. This is one of the concepts that will be utilised in developing the hierarchical solution as seen in Section 5.6. It is seen that the results presented in Figure fig:example3.43 utilise the F1-score. This is used to show the general classifier performance across the different sub-flow times being the harmonic mean of the precision and recall. It should be noted that based on these results, the classifier working well on an attack type does not automatically imply that the hierarchical solution will be suitable for that attack type. Complete results for all sub-flow classifications are provided in subsequent sections. Section 5.6 which deals specifically with the development and analysis of the hierarchical solution will utilise the recall metric as the basis for its accuracy analysis. this is because the recall metric represents a stronger security implication with the hierarchical solution.

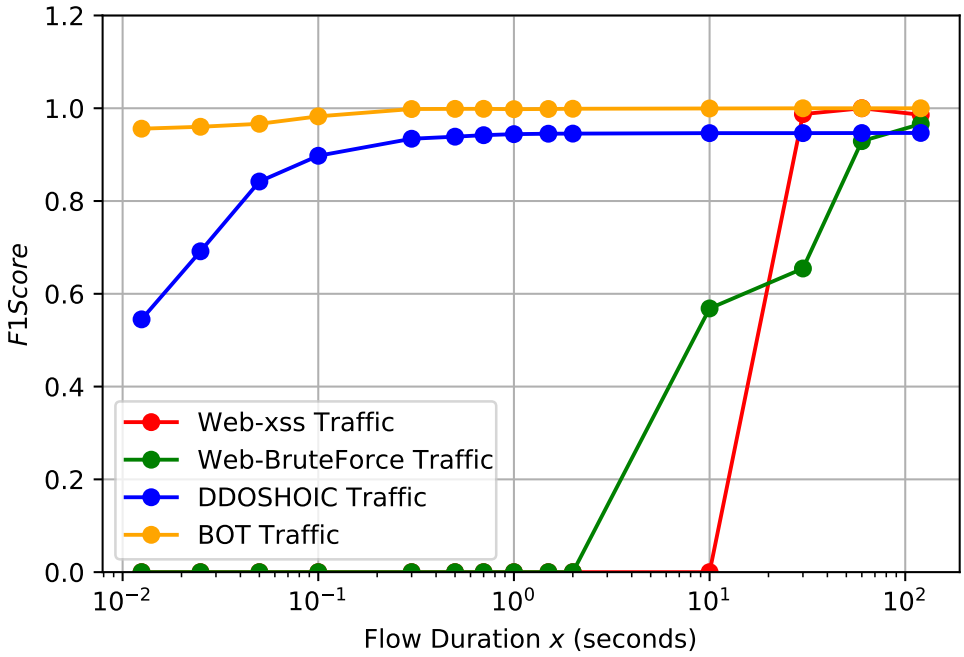


Figure 5.3: Subflow Classification (note DDOS, BOT and PSCAN all have F1 scores close to 1 hence only DDOS is shown)

5.3.2 Traffic Classification Based on Different Flow Monitoring Methods

Once again, the non-automatically generated attacks eg Webattack-Bruteforce is more significantly affected by a change in flow monitoring approach. Figures 5.4 and 5.5 show the effect of classification using OpenFlow monitoring and the Flowmeter for BOT and Webattack traffics. Figure 5.5 shows that there is no significant difference at lower sub-flows for Web attacks given that classifier performance degrades significantly. At higher sub-flows above 1s, classifier performance picks up significantly but shows some variation based on flow monitoring approach. For DDOS attacks shown in Figure 5.4, there is a slight reduction in Classifier performance with OpenFlow monitoring but still considered to give fairly reasonable attack detection rate. This is sufficient to provide reasonable attack detection at lower sub-flow times suggesting a high possibility for real-time attack detection. This shows that there is still ample time within the flow's lifespan to compensate for the drop in classifier accuracy due to the combination of OpenFlow monitoring and sub-flow classification. Once again, this concept will be used in the development of the hierarchical solution as seen in Section 5.6. The precision metric is used for this results as it illustrates

the variation between the two flow monitoring approaches better.

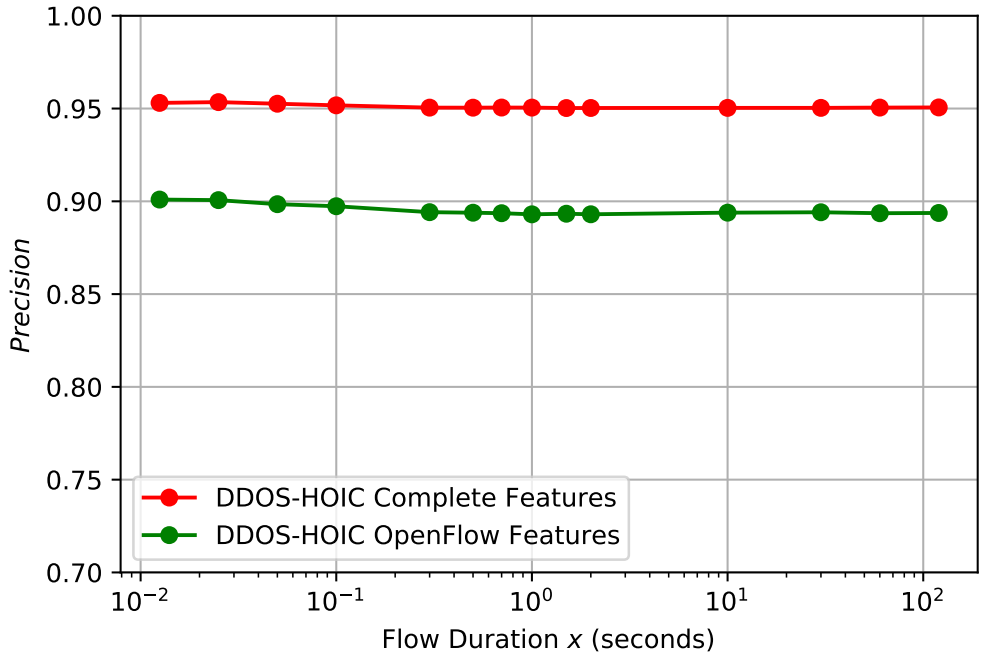


Figure 5.4: Classification results for BOT traffic using OpenFlow and Extended Features (Note: mean attack flow duration was 0.48s

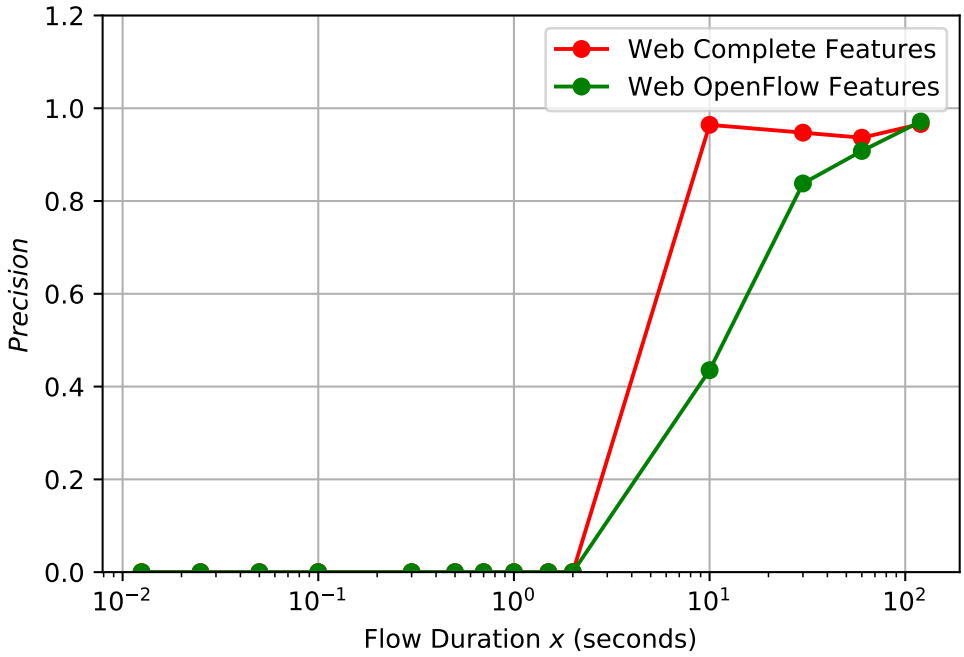


Figure 5.5: Classification results for WEBATTACK traffic using OpenFlow and Extended Features

5.4 Sub-Flow Based Hierarchical Intrusion Detection Solution

The layout of the proposed Subflow based Hierarchical Machine Learning solution is shown in Figure 5.6 and is a disected representation of the experimental procedures in Section 3.4.5.. Classifier 1 is trained and fitted for realistic features that can be captured with OpenFlow. Many papers also ignore this crucial point: that not all flow parameters can be collected in practice. Classification results based on OpenFlow monitoring give varying responses depending on attack type. The second stage detector further reduces the false negative component with the aim of reducing the false negative rate (FNR), thereby increasing the overall attack detection of the system.

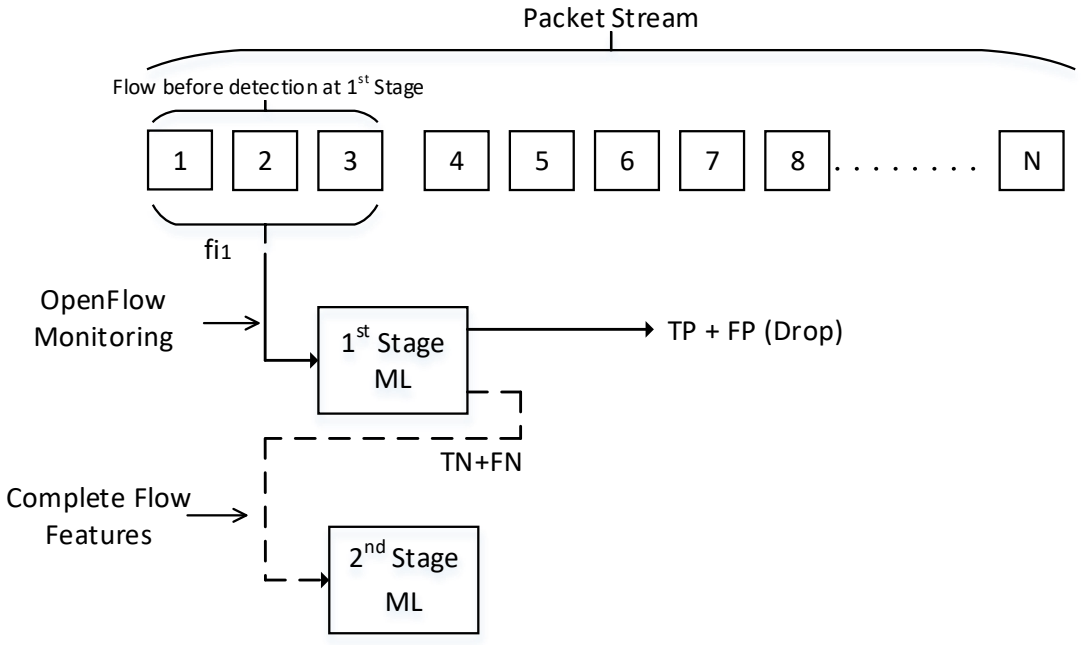


Figure 5.6: Sub-Flow Based Hierarchical Machine Learning Solution 2

5.5 Classification Metrics Based on Hierarchical Solution

Classification metrics based on the Hierarchical solution are presented here based on two operational options:

1) If all classified attack flows (Ps) from 1st stage are sent to classifier 2 and all classified benign flows (Ns) from 1st stage are allowed through

a) In this case, a high False Positive Rate (FPR) in the the first stage means unnecessary overload on classifier 2 which will impact network efficiency. FPR is given by

$$\text{FPR} = \frac{FP}{TN + FP} \tag{5.1}$$

b) Also in this scenario, a high False Negative Rate (FNR) in the first stage will be highly detrimental to network security because substantial attack traffic will be allowed through to network. FNR is given by

$$\text{FNR} = \frac{FN}{FN + TP} \tag{5.2}$$

Figure 5.7 shows an illustration for option 1

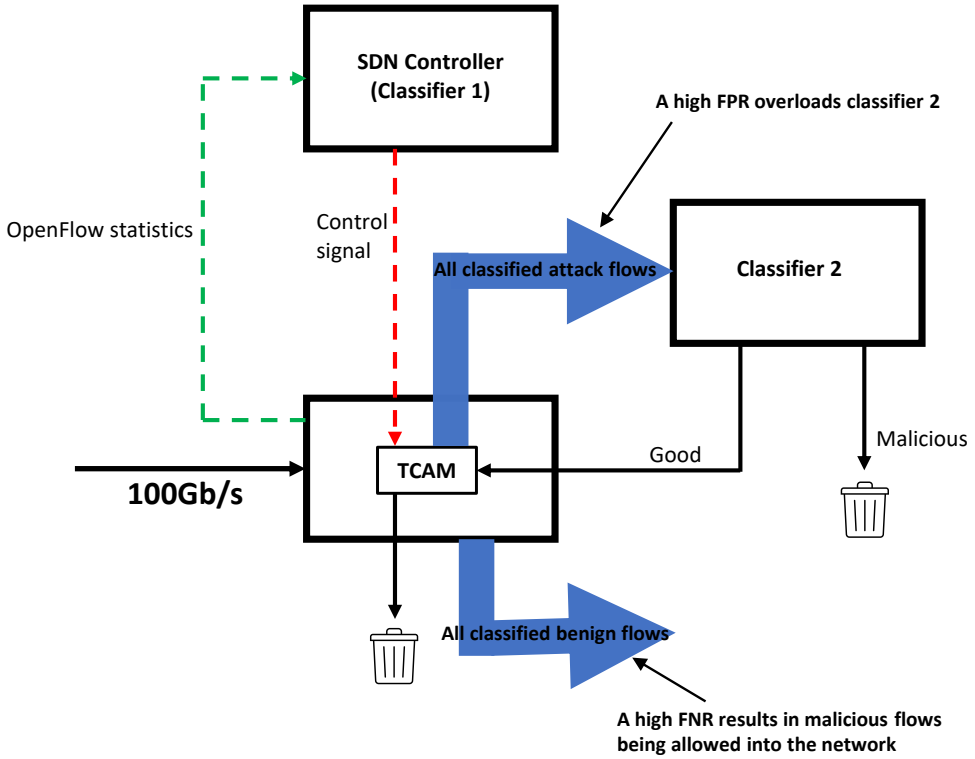


Figure 5.7: Sending all Classified Atttack Flows to 2nd Stage Classifer

2) If all benign flows (Ns) from 1st stage are sent to classifier 2 and all attack flows (Ps) from 1st stage are dropped

a) In this case, a high False Positive Rate (FPR) in the first stage means loss of information as substantial good traffic will be dropped. FPR is defined in Equation 5.1.

b) In this scenario, a high False Negative Rate (FNR) in the first stage which would have been detrimental to network security is addressed by classifier 2. FNR is defined in Equation 5.2

Figure 5.8 shows an illustration for option 2

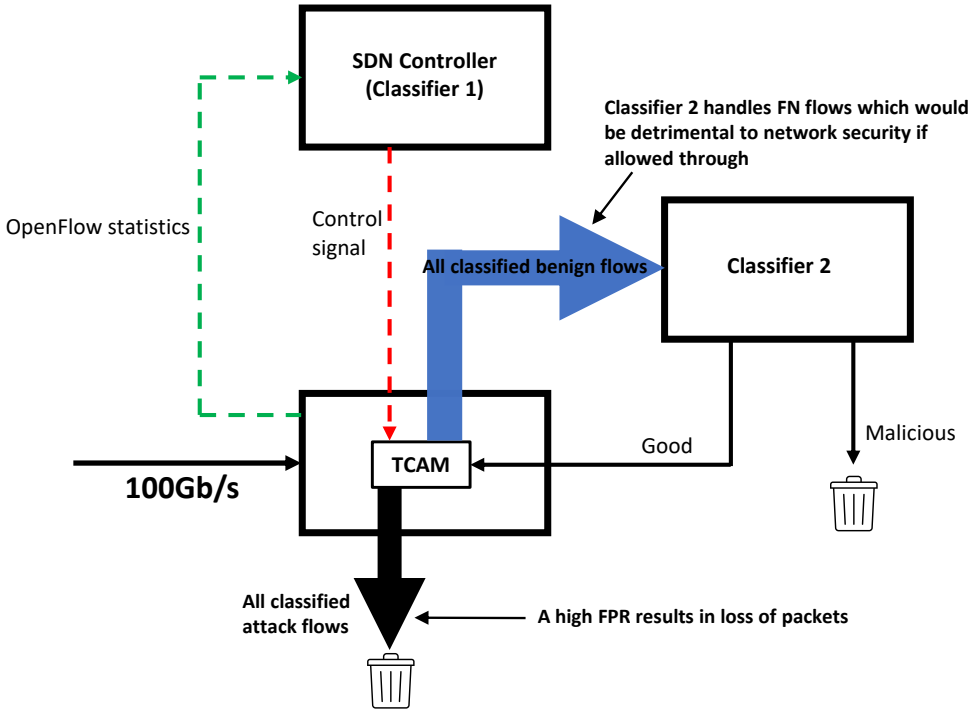


Figure 5.8: Sending all Classified Benign Flows to 2nd Stage Classifer

5.6 Hierarchical Intrusion Detection Results

Results from the previous sections were based on the CICIDS2017 dataset. In this section, detailed (more comprehensive) intrusion detection results based on the 2018 version of the dataset is presented. The reason for this is to further validate the hypothesis with a different and slightly more recent dataset. This section will emphasis on the timely detection of malicious packets using limited flow features based on the flow monitoring limitations of OpenFlow as explained in Section 3.4.4. This depicts/simulates the operation of the first stage of the hierarchical solution. The mean flow duration of the different attack types as profiled in the dataset will be a key parameter for the analysis of the first stage. As expected, the results will be different for the different attack types.

The second stage intrusion detection of the hierarchical solution occurs at the edge of the SDN topology. The intrusion detection model remains the Decision Tree Classifier from the previous stage. The possible need for a different kind of detection model for this stage is recognised and will be considered in future work.

Experimentation and results in this section cover the first and second stage operations. In other

to evaluate the advantage of the hierarchical approach, a non-hierarchical intrusion detection procedure is experimented and the results are presented and compared with results from the hierarchical approach. The results will be presented according to the different attack types which corresponds to specific dates as generated by the authors of the dataset. The results will be presented in terms of the detection parameters (recall, ecdf, predict rate, the confusion matrix. The mode of presentation of the results will be similar for the different attack types. However, the actual trend of results may differ with attack type and the resulting analysis may also differ.

The following derived metrics have been formulated to enhance the analysis of the hierarchical solution:

Detection Cutoff which is the recall value that is considered adequate given the response from the specific experiment. This is calculated as the maximum recall value at less than 10% of the average attack flow duration. It can also be the case that a maximum recall value is reached at a lower sub-flow time in which case it is selected. This ensures that the detection is carried out when at most 10% of the mean attack flow-duration has expired.

Flow Duration Cutoff which is the flow duration at the the detection cutoff.

Flow Duration Cutoff Ratio which is the percentage of the average attack duration which the flow duration cutoff represents.

Attack Flows Completed which is the percentage of the completed attack flows at the detection or flow duration cutoff.

The above represents an *aggressive* intrusion detection cut-off so that detection occurs early in the attack flows, but clearly, better recall/precision results might achieved if a higher (less-aggressive) cut-off was chosen.

5.6.1 SSH BruteForce

The key time-based statistics for the ssh bruteforce attack dataset is given below:

BRUTE_WED14TH data mean flow duration: 14.41s

BRUTE_WED14TH attack traffic mean flow duration: 0.12s

From the above statistics, it is seen that the average attack duration for this profile is 0.12029s while the maximum attack duration is 0.5676s.

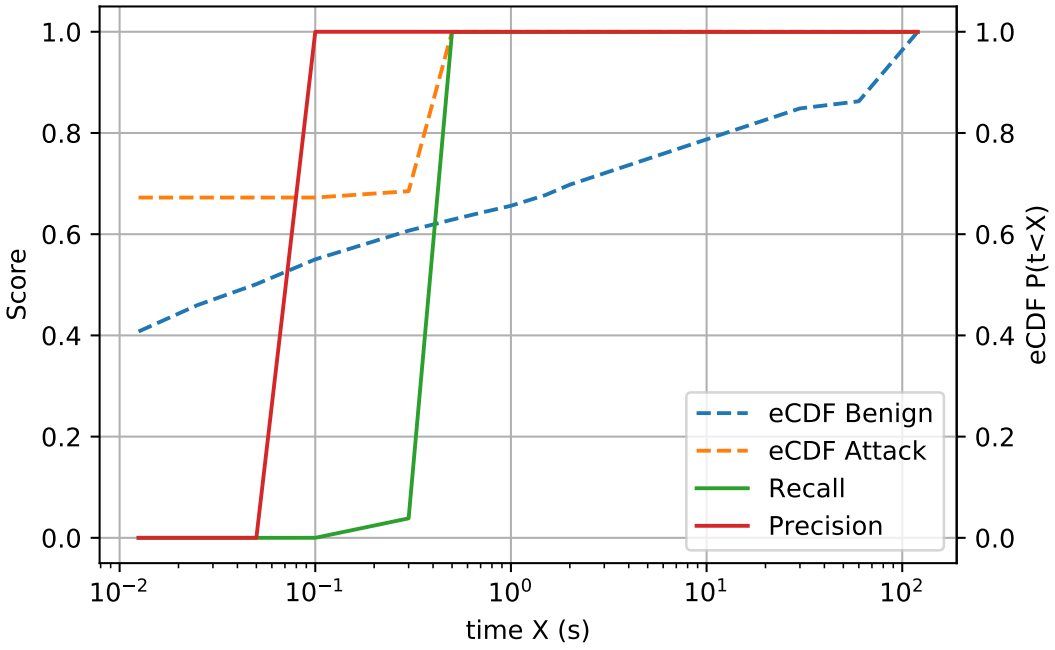


Figure 5.9: 1st Stage Detection for BruteForce Attack

Table 5.1: Derived Detection Metrics for SSH/FTP BruteForce

BRUTE_WED14TH	
Detection Cuttoff (Recall)	0
Flow Duration Cuttoff (s)	0.012
Average Attack Duration (s)	0.12
Attack Flows Completed (%)	68
Flow Duration Cuttoff Percent (%)	10

Table 5.1 shows the key derived metrics from the first stage intrusion detection for the SSH brute-force attack. The metrics show a generally poor detection response on this attack type. The metrics are calculated as follows:

Average Attack Duration (AAD) = 0.12029s

10% of AAD = 0.012s

At 10% of AAD, Recall = 0

Hence Detection Cutoff = 0

Therefore Flow Duration Cutoff (assuming Detection Cutoff at 10% of AAD) = 0.012s.

With approximately 68% Attack Flows Completed at Detection Cutoff of 0 indicates minimal

protection by the hierarchical solution. Analysis from the graph (Figure 5.9) and the calculations above, suggest very short duration attack traffic with the classifier unable to classify flows at these low subflow durations. Hence, the hierarchical solution will provide very little or no protection for the SSH brute-force attack at these low subflow times. A closer look at the graph will reveal that there is an increase in precision beyond 0.05s while recall remains closer to 0. This may give a false impression of high accuracy with the tendency to contradict the results. The fact that recall is closer to 0 but not exactly 0 suggests that there have been a few positive predictions all or most of which would have been correct. This accounts for the high precision at these low recall points. This further validates why precision is not considered as critical as recall as stated in Section 4.3. These results are mostly inadequate for timely attack detection, hence, further work needs to be done to improve the Detection Cutoff for this attack type.

Further consideration of the potential of this solution is seen where all the predicted benign traffic is forwarded to the second stage detection module. As seen from Figure 5.10, results from this experimentation shows that there is reduction in false negative predictions when using a hierarchical intrusion detection approach. Figure 5.10 shows a graphical comparison of the false negative predictions across all the traffic subflows. After the first stage, all benign flows are switched or filtered to the second stage for further classification. It should be noted that the results from the second stage shown, is a combined result from the first and second stage as explained in Section 3.4.5. This explains why the same number of flows reflect on both results from first and second stage (114841 in this case).

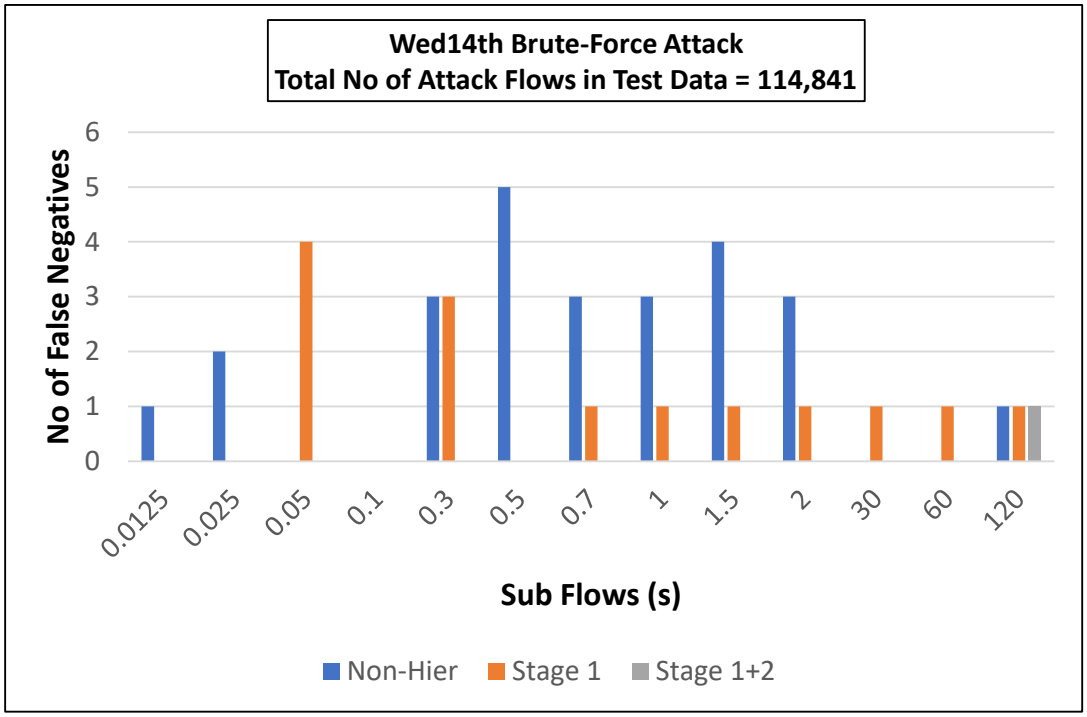


Figure 5.10: Hierarchical Intrusion Detection Results for SSH/FTP BruteForce Attack

As can be seen from Figure 5.10 and Table 5.2, the intrusion detection at the second stage has significantly eliminated the false negative flows seen from the non-hierarchical approach and from the first stage of the hierarchical process. This has the effect of protecting the network better. The confusion matrices for the 0.5s subflow is shown in Figure 5.11 as an example. The advantage is in two fold; the first is the improvement in detection as against using only a one stage detection at the central controller, the second is the reduction in effort as against using a non-hierarchical detection at the edge. The down side of this solution is the increase in false positives as seen in Table 5.2. However, it can be argued that an increase in the false positive rate bears no direct significant security risk to the network, however, it would block some legitimate users. A very interesting outcome in Table 5.2 is that the False Negatives are small and have significantly reduced at the output of the 2nd stage. For example at subflow 0.7 the 1st stage misses 1 flow and passes this to the second stage where this flow is then detected correctly. Generally, the results show that the hierarchical approach significantly improves the result compared to the Non-hierarchical result. This appears a strange result on first-site as both are using the full-features and the same machine learning algorithm. However, there is a key difference between the approaches and that is that in the hierarchical case the results are the process of two models: the first works on the open-flow

data and captures most of the attack flows; then only the few attack flows (and benign) that are not detected by the first stage are then passed to the second stage which has been trained to spot these particular flows i.e., the second stage model has been specifically trained to detect the attack flows that get through the first stage. Consequently, by using two models the system is better trained to deal with variability in the attack flows. Conversely, because all the detected benign traffic is sent to the second stage, any false positives in the first stage are blocked and not sent to the second stage, this means the final outcome from the second stage, in terms of false positives, can never be better than the first stage and, as we see, is slightly worse than the non-hierarchical approach This is because in this case the system has two opportunities to inject false positives i.e. in the first and second stages. This shows a weakness in passing just one category of traffic to the second stage, Chapter 5 will investigate a solution to this.

Table 5.2: Hierarchical Intrusion Detection Results for SSH Brute-Force

WED14TH BRUTEFORCE (No of Attack Flows = 114841) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	1	0	0
0.025	2	0	0
0.05	0	4	0
0.1	0	0	0
0.3	3	3	0
0.5	5	0	0
0.7	3	1	0
1	3	1	0
1.5	4	1	0
2	3	1	0
30	0	1	0
60	0	1	0
120	1	1	1

WED14TH BRUTEFORCE (No of Attack Flows = 114841) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	2	35	37
0.025	3	37	38
0.05	5	36	41
0.1	1	37	38
0.3	6	49	53
0.5	9	35	42
0.7	7	36	41
1	1	25	26
1.5	8	26	32
2	7	26	31
30	3	26	27
60	7	32	37
120	4	33	35

Non-Hier (Subflow = 0.5s BruteForce)		
n = 114841	Predicted No (BENIGN)	Predicted Yes (BruteForce)
Actual No (BENIGN)	2191649 TN	9 FP
Actual Yes (DDoS)	5 FN	114836 TP

1st Stage (Subflow = 0.5s BruteForce)		
n = 114841	Predicted No (BENIGN)	Predicted Yes (BruteForce)
Actual No (BENIGN)	2191623 TN	35 FP
Actual Yes (DDoS)	0 FN	114841 TP

2nd Stage (Subflow = 0.5s BruteForce)		
n = 114841	Predicted No (BENIGN)	Predicted Yes BruteForce
Actual No (BENIGN)	2191616 TN	42 FP
Actual Yes (DDoS)	0 FN	114841 TP

Figure 5.11: Sample Confusion Matrix for BruteForce Detection

5.6.2 Denial of Service (DoS)

The key time-based statistics for the DoS attack dataset is given below:

Dosfri16 data mean flow duration: 15.52s

Dosfri16 attack traffic mean flow duration: 46.51s

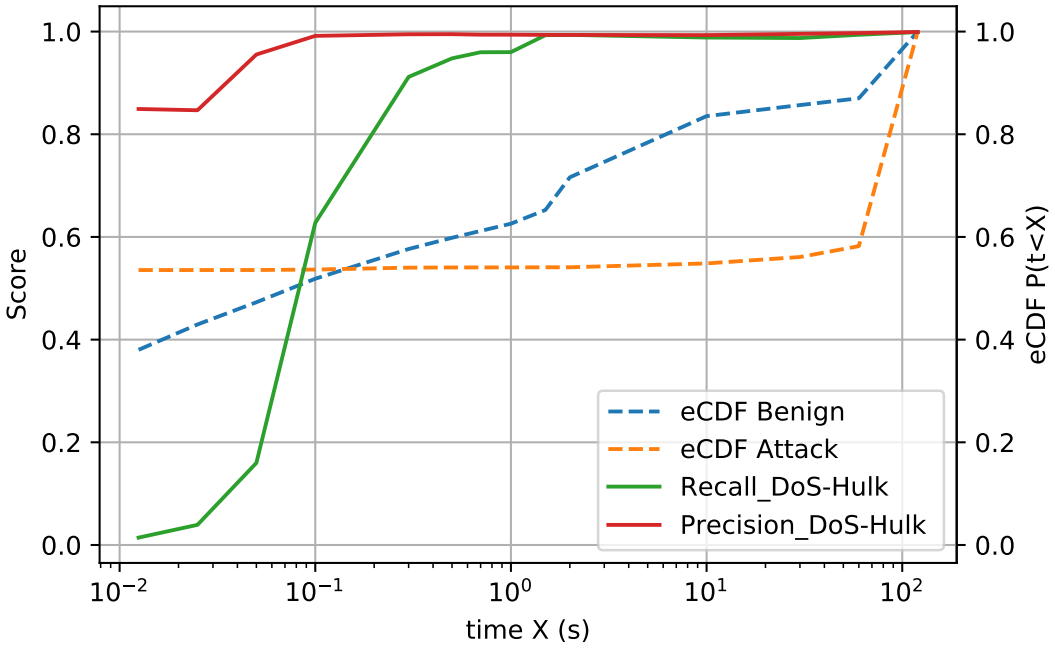


Figure 5.12: 1st Stage Detection for DoS Attack

Table 5.3: Derived Detection Metrics for DoS Attack

DOS_FRI16TH	
Detection Cuttoff (Recall)	0.9998
Flow Duration Cuttoff (s)	1.8
Average Attack Duration (s)	46.5
Attack Flows Completed (%)	55
Flow Duration Cuttoff Percent (%)	3.87

Table 5.3 shows the key derived metrics from the first stage intrusion detection for the DoS attack. The metrics are calculated as follows:

Average Attack Duration (AAD) = 46.5s

10% of AAD = 4.65s

Maximum recall value at less than 10% of AAD occurs at approximately 1.8s and gives a recall

Table 5.4: Hierarchical Intrusion Detection Results for Denial of Service (DoS) Attack

FRI16TH DOS (No of Attack Flows = 78416) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	11	1357	9
0.025	17	1360	11
0.05	12	2365	8
0.1	10	1352	8
0.3	16	1340	7
0.5	16	1332	3
0.7	14	1329	2
1	17	1291	5
1.5	41	111	30
2	49	109	29
10	4	19	3
30	5	12	3
60	6	23	3
120	6	45	2

FRI16TH DOS (No of Attack Flows = 78416) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	1178	2400	3577
0.025	1176	2402	3577
0.05	1178	64	1239
0.1	1181	80	1260
0.3	1179	51	1229
0.5	1185	56	1241
0.7	1181	43	1224
1	1153	44	1197
1.5	55	40	95
2	60	39	99
10	15	49	64
30	7	60	66
60	6	77	83
120	15	88	103

value of 99.98%

Hence Detection Cutoff = 99.98%

Flow Duration Cutoff = 1.8s

Flow Duration Cutoff Percent = 3.87% and

Attack Flows completed = 55%

The metrics show good detection response for this attack type which indicates that the hierarchical solution provides reasonable protection for DOS attack. This represents real-time or at least near real-time attack detection. With such early detection, there is sufficient time within the lifespan of the flow for further investigation of the traffic with the second stage classifier. The results as seen from Figure 5.12 shows appreciable response even at points lower than the Detection Cutoff. For instance, there is still reasonable protection as low as 0.3s which gives a recall of about 91.17%. It is also seen that at much lower sub-flows (less than 0.3s) with much lower recall values, the precision values are reasonably high. The reason for this has already been explained in Section 5.6.1.

Despite the relatively high Detection Cutoff, passing the benign flows to the second stage detection module at the edge results in further detection of attack traffic which hitherto had not been detected. This process reflects as a reduction in the false negative classifications or a reduction in FNR. From Table 5.4 and given the flow duration cutoff of 1.8s (using an approximate 2s subflow),

there is a 40.81% reduction in false negatives as against using a non-hierarchical approach. Also, considering a subflow of 0.3s (which gave reasonable detection as mentioned earlier), there is a 56.2% reduction in false negatives as against using a non-hierarchical approach. In addition, although the focus of the hierarchical solution is to compare with with a non-hierarchical approach, it is seen that there is an approximately 99.4% improvement in false negative from the first stage. The hierarchical intrusion detection solution therefore shows lots of promise with this attack type. Once again Table 5.4 shows an increase in false positives on the second stage. This is for the reason commented on in Section 5.6.1. The positive flows are only seen here because the results from the first and second stages were combined as described in Section 3.4.5 and are presented mainly for the purpose of analysis.

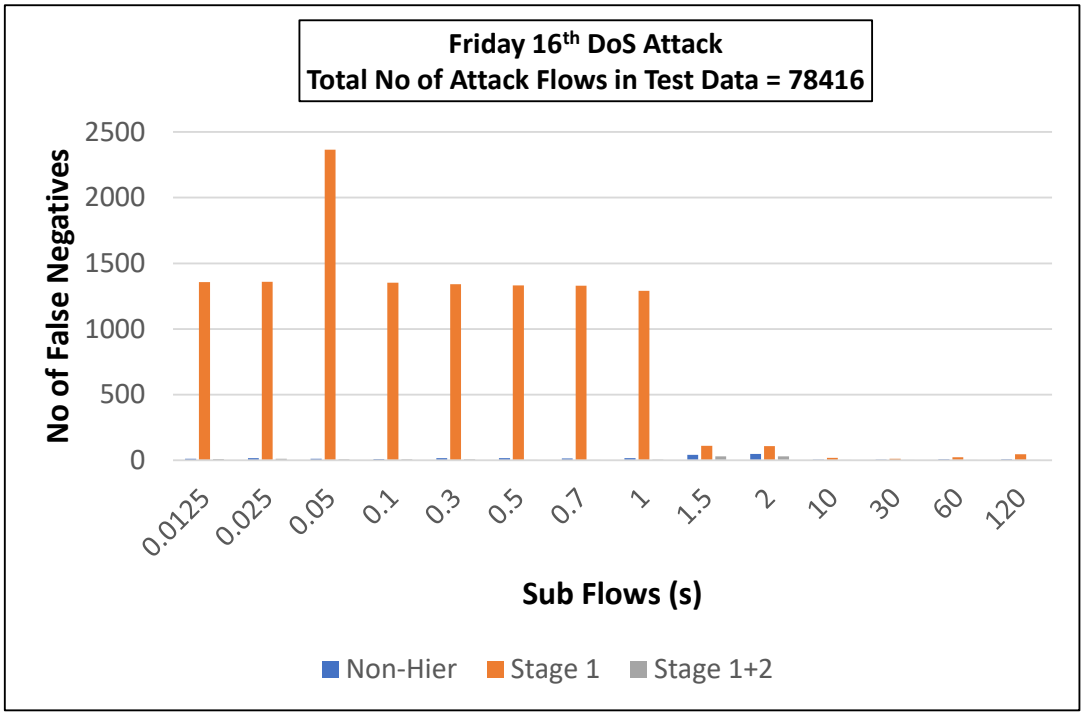


Figure 5.13: Hierarchical Intrusion Detection Results for DoS Attack

Non-Hier (Subflow = 0.5s DoS)		
n = 78416	Predicted No (BENIGN)	Predicted Yes (DoS)
Actual No (BENIGN)	2144637 TN	1185 FP
Actual Yes (DDoS)	16 FN	78400 TP

1st Stage (Subflow = 0.5s DoS)		
n = 78416	Predicted No (BENIGN)	Predicted Yes (DoS)
Actual No (BENIGN)	2145766 TN	56 FP
Actual Yes (DDoS)	1332 FN	77084 TP

2nd Stage (Subflow = 0.5s DoS)		
n = 78416	Predicted No (BENIGN)	Predicted Yes DoS
Actual No (BENIGN)	2144581 TN	1241 FP
Actual Yes (DDoS)	3 FN	78413 TP

Figure 5.14: Sample Confusion Matrix for DoS Attack Detection

5.6.3 Distributed Denial of Service (DDoS)

The key time-based statistics for the DDoS attack dataset is given below:

Ddostue20 data mean flow duration: 15.02s

Ddostue20 attack traffic mean flow duration: 23.41s

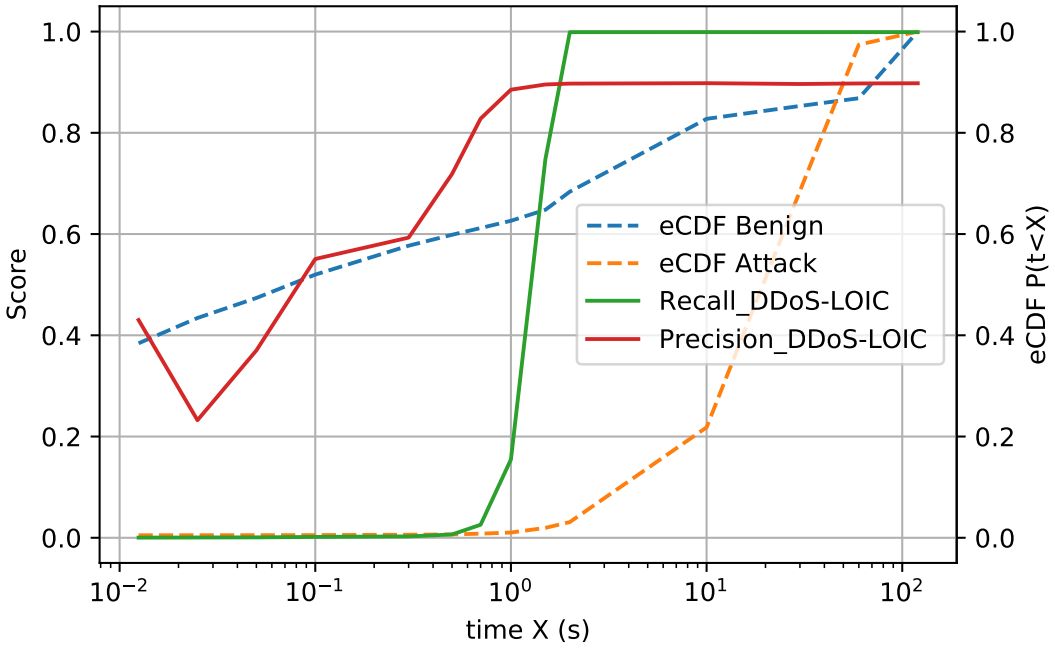


Figure 5.15: 1st Stage Detection for DDoS-LOIC Attack

Table 5.5: Derived Detection Metrics for DDoS Attack

DDOS_TUE20TH	
Detection Cuttoff (Recall)	0.998
Flow Duration Cuttoff	2s
Average Attack Duration	23.4s
Attack Flows Completed	3%
Flow Duration Cuttoff Percent	8.54%

Similar to DoS attack, the first stage detection metrics for DDoS also returns good response. Calculation of the first stage metrics are as follows:

Average Attack Duration (AAD) = 23.41s

10% of AAD = 2.34s

Maximum recall value at less than 10% of AAD occurs at approximately 2s and gives a recall

value of 0.998.

Hence,

Detection Cutoff = 0.998

Flow Duration Cutoff = 2s

Flow Duration Cutoff Percent = 8.54% and

Attack Flows Completed = 3%.

As seen in Table 5.5, these results represent a good response and suggests good protection as offered by the hierarchical solution. This is an improvement on the DoS response which returned a 55% attack flows completed at Detection Cutoff. This represents an even greater potential for the hierarchical solution with this attack type. The combination of an 8.54% flow duration cutoff percent and a 3% attack flows completed represents a near real-time attack detection and supports further investigation using the hierarchical model. Results as seen from Figure 5.15 shows reasonable precision across the sub-flows. As seen from Table 5.6, at the detection cutoff of 2s, there is a 95.4% reduction in false negatives compared to using a non-hierarchical approach. There is also a similar reduction across all the sub-flow times. However, in the case of DDoS, there is no significant increase in the false positive rate. This suggests that the cost in terms of increased false positives is lower for DDoS attack when using the hierarchical solution.

One interesting feature that appears for the first time in Figure 5.15, but will appear in later graphs, is that the trend for the machine learning is not monotonically increasing with the increase in sub-flow time. At first sight this may appear strange: it would seem that the more information you have the better the response would be, and in general terms this is true. However, it is important to remember that packet flows are highly non-linear in nature and that there is a complex inter-play between precision and recall with the machine-learning being retrained for each sub-flow. Consequently, each result at each sub-flow is a unique model and subject to the nature of the traffic flow features present in a subflow. If there had been more computation time available it would have been good to run these multiple times on multiple splits (cross-validation) and show mean curves with error bars, however, the results of each of these experiments represented several days of processing time. This is not surprising since the data sets represent about 6 hours of packets which had to be first processed for each of the sub-flow time to create flow data and then had to be put through the experimental process described earlier for each sub-flow. Consequently, the general trend of these results are important rather than the specific small-changes at each sub-flow time.

Table 5.6: Hierarchical Intrusion Detection Results for Distributed Denial of Service (DDoS) Attack

TUE20TH DDOS (No of Attack Flows = 104481) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	10444	658	589
0.025	10399	621	578
0.05	10341	601	561
0.1	10358	590	547
0.3	10533	584	541
0.5	10297	578	543
0.7	10466	590	546
1	10528	572	537
1.5	11928	596	544
2	11913	585	546
10	11748	594	538
30	11462	590	536
60	11398	570	533
120	11437	576	532

TUE20TH DDOS (No of Attack Flows = 104481) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	10731	11909	11926
0.025	10750	11896	11920
0.05	10738	11907	11931
0.1	10739	11906	11940
0.3	10687	11911	11921
0.5	10692	11905	11917
0.7	10671	11910	11916
1	10624	11906	11922
1.5	10327	11896	11916
2	10357	11923	11932
10	10168	11904	11911
30	9825	11912	11920
60	9729	11908	11913
120	9689	11906	11913

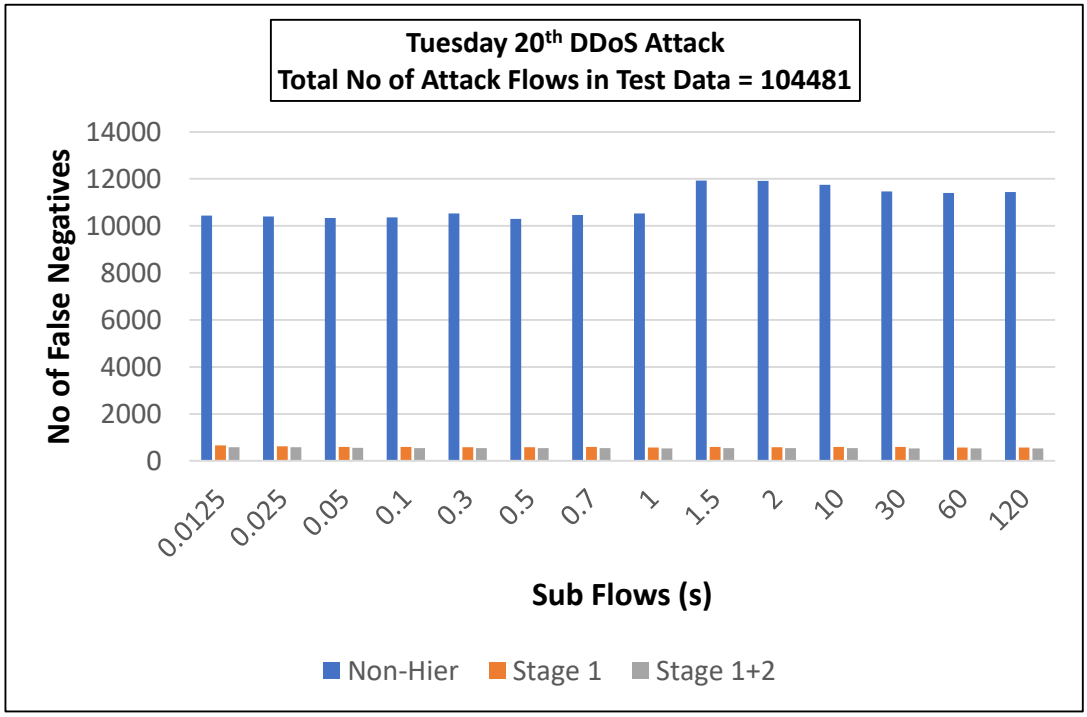


Figure 5.16: Hierarchical Intrusion Detection Results for DDoS Attack

Non-Hier (Subflow = 0.5s DDoS)		
n = 104481	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2413967 TN	10692 FP
Actual Yes (DDoS)	10297 FN	94184 TP

1st Stage (Subflow = 0.5s DDoS)		
n = 104481	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2412754 TN	11905 FP
Actual Yes (DDoS)	578 FN	103903 TP

2nd Stage (Subflow = 0.5s DDoS)		
n = 104481	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2412742 TN	11917 FP
Actual Yes (DDoS)	543 FN	103938 TP

Figure 5.17: Sample Confusion Matrix for DDoS Attack Detection

In continuation for the DDoS attack results, the DDoSHOIC attack is presented next. From the CICIDS2018 dataset, the DDoSHOIC attack profile was created on a different day, Wednesday the 21st, hence the intrusion detection experiments are carried out separately and the results presented here separately.

The key time-based statistics for the DDoSHOIC attack dataset is given below:

Ddoshoicwed21 data mean flow duration: 13.69s

Ddoshoicwed21 attack traffic mean flow duration: 0.239s

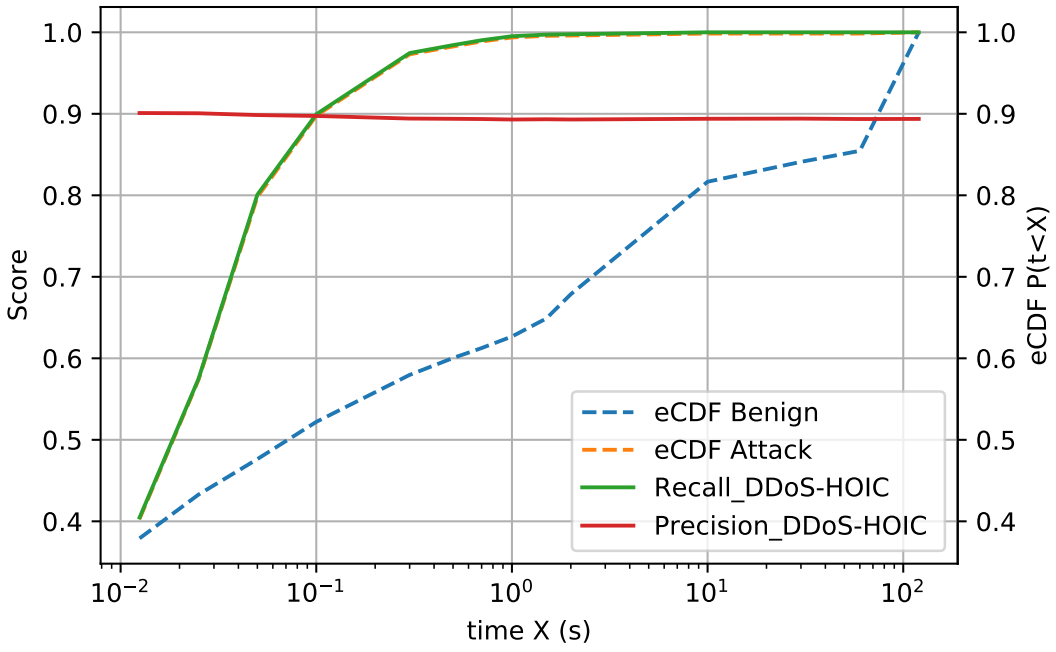


Figure 5.18: 1st Stage Detection for DDoSHOIC Attack

The derived metrics for DDoSHOIC attack are calculated thus:

Average Attack Duration (AAD) = 0.239s

10% of AAD = 0.0239s

Table 5.7: Derived Detection Metrics for DDoSHOIC Attack

DDOS_WED21ST	
Detection Cuttoff (Recall)	0.574
Flow Duration Cuttoff	0.0239s
Average Attack Duration	0.239s
Attack Flows Completed	57.39%
Flow Duration Cuttoff Percent	10.00%

Maximum recall at less than or equal to 10% of AAD occurs at approximately 0.0239 and gives a value of 0.574. Hence the other derived hierarchical metrics are shown in Table 5.7. Interestingly, the mean attack duration for DDoSHOIC is much lower than the DDoSLOIC. This is because the of the HOIC tool which possesses greater firepower and is capable of launching DDoS attacks at a greater rate. This makes the DDoSHOIC more difficult to detect as seen from the results of the derived metrics. 10% of mean attack duration (0.023s) only gives a 0.574 detection cutoff and 57.3% attack flows completed. This provides little room for the hierarchical approach. However, further investigation using the hierarchical model as shown in Table 5.8 shows that a flow duration cutoff of 0.0239s gives an approximately 99.9% reduction in false negatives as against using a non-hierarchical solution. However, there is the relative advantage of an insignificant increase in FPR at the second stage Figure 5.18 has a very interesting characteristic as the eCDF for the attack traffic very closely follows the curve for the recall. At first site this may appear anomalous; however, it is clear in this case that the machine learning result is dependent upon receiving most of the traffic (information) in a flow for this attack type and, when it has most of the flow, it is highly successful in detecting the attack packets (high recall). For example, when the flow time limit of, approximately, 0.04s is used, then only 70% of the flows are complete, and as flows need to be complete to be detected, the recall is also 0.7. This is the first case where the two results track so closely, but it can be seen in some earlier cases the trends are similar, and some later results will also show a similar trend. This is an important result as it shows that for some attacks, detection is only successful when most of the flow has completed. In these cases, while detection may be successful (assuming a long-enough window) it suggests that mitigating purely on the flow information will not be possible as the attack will be complete before any blocking from the detection could take place. Consequently, for these types of attacks other techniques, such as source identification and correlation maybe required so that previous attacks can be added to a block-list to stop future attacks.

Table 5.8: Hierarchical Intrusion Detection Results for Distributed Denial of Service (DDoS) Attack

WED21st DDOSHOIC (No of Attack Flows = 387476) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	25432	985	17
0.025	31989	1007	25
0.05	36449	1179	38
0.1	40879	23394	91
0.3	44171	1217	59
0.5	44028	991	21
0.7	44003	992	20
1	44055	983	11
1.5	44184	975	12
2	43907	949	7
10	44156	74	9
30	44097	59	9
60	44179	75	16
120	43886	66	7

WED21st DDOSHOIC (No of Attack Flows = 387476) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	42819	70494	71117
0.025	41455	71409	71910
0.05	40203	71409	71747
0.1	39491	42396	46785
0.3	38744	46316	46689
0.5	38621	46315	46638
0.7	38591	46288	46613
1	38609	46264	46597
1.5	38653	46275	46598
2	38631	46274	46591
10	38301	46366	46374
30	38250	46373	46381
60	38348	46370	46379
120	38337	46361	46377

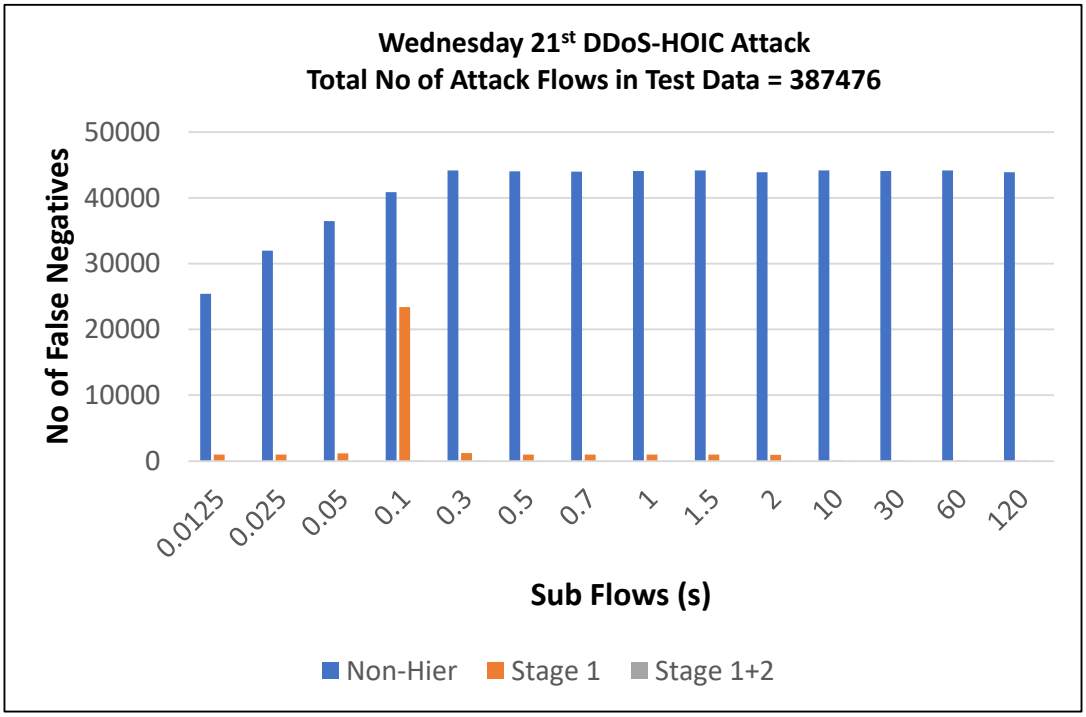


Figure 5.19: Hierarchical Intrusion Detection Results for DDoS Attack

Non-Hier (Subflow = 0.5s DDoS)		
n = 387476	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2403726 TN	38621 FP
Actual Yes (DDoS)	44028 FN	343448 TP

1st Stage (Subflow = 0.5s DDoS)		
n = 387476	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2396032 TN	46315 FP
Actual Yes (DDoS)	991 FN	386485 TP

2nd Stage (Subflow = 0.5s DDoS)		
n = 387476	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2395709 TN	46638 FP
Actual Yes (DDoS)	21 FN	387455 TP

Figure 5.20: Sample Confusion Matrix for DDoS Attack Detection

5.6.4 Web Attack

Just like the DDoS attack in Section 5.6.3, the Web attack is presented in two results based on different forms of Web attack.

The key time-based statistics for the Web attack brute-force is given below:

Webthu22 data mean flow duration: 21.11s

Webthu22 attack traffic mean flow duration: 33.16s

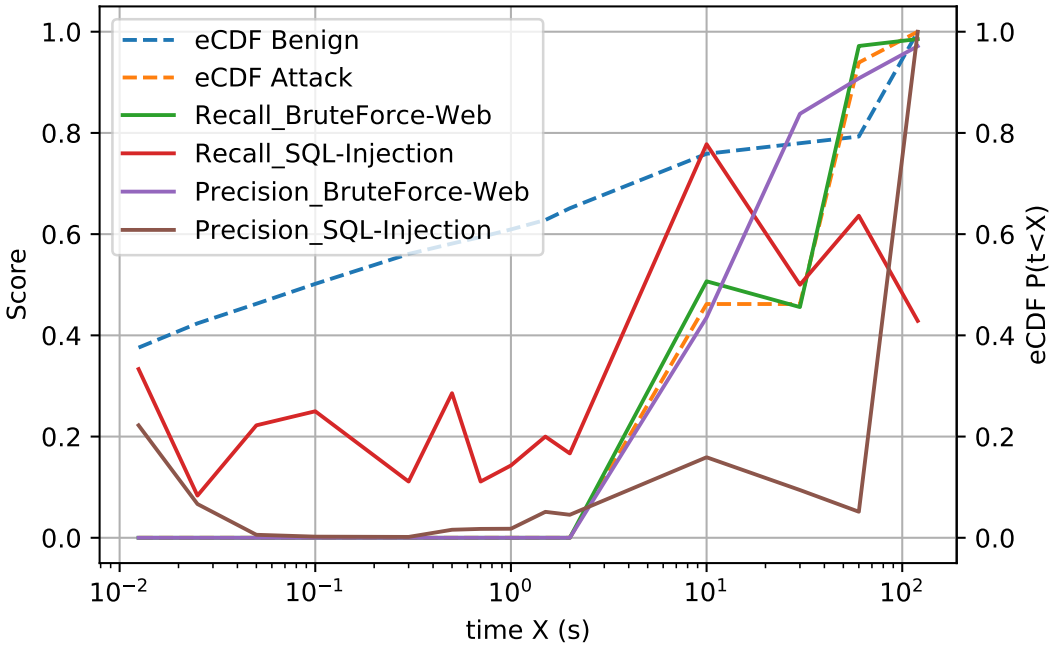


Figure 5.21: 1st Stage Detection for Web Attack (Thus 22nd)

The derived metrics for the Web attack brute-force detection is given in Table 5.9 and calculated thus:

Average Attack Duration (AAD) = 33.16s

10% of AAD = 3.316s

Table 5.9: Derived Detection Metrics for Web Attack (Thus 22)

BRUTEFORCE-WEB_THU22ND	
Detection Cuttoff (Recall)	0.21
Flow Duration Cuttoff (s)	3.31
Average Attack Duration (s)	33.16
Attack Flows Completed (%)	19
Flow Duration Cuttoff Percent (%)	10

SQL-INJECTION-WEB_THU22ND	
Detection Cuttoff (Recall)	0.44
Flow Duration Cuttoff (s)	3.31
Average Attack Duration (s)	33.16
Attack Flows Completed (%)	19
Flow Duration Cuttoff Percent (%)	10

Maximum recall value at less than or equal to 10% of AAD occurs at 3.31s and gives a recall value of 0.21. Hence, the other derived metrics are as shown in Table 5.9.

Similar results are obtained for Web attack Sql-injection but with a slightly better Detection Cutoff of 0.44. Both classes of Web attack however, do not return sufficient Detection Cutoff required to achieve sufficient protection from the hierarchical solution. Although both attacks return a good Attack Flows Detected value (19%) and Flow Duration Cutoff Percent of 10%. However, going forward to investigate the hierarchical performance at a sub-flow time of 2s shows a 33.3% reduction in false negatives and a 57.14% increase in false positives as seen in Table 5.10. This shows that despite the relatively poor detection response from the derived metrics of the first stage, the hierarchical solution still shows some promise for the Web attack types investigated here. It is notable that these category of attacks have poorer performance than the others, in particular the SQL-Injection. There is a fundamental reason for this: the attacks previously shown have traffic patterns that are unique to their type of attack, for example DoS attacks. However, SQL-Injection is fundamentally an application layer attack. With the increasing use of encryption, it is not possible for most traffic (soon probably all) to have content inspected at the network switch layer. Consider specifically an SQL-Injection attack, a specially crafted SQL-injection may have no feature in its packet flow to distinguish it from a benign packet to the same server. However, in the case of automated SQL-Injections it may be possible to determine some information from the general flow if there are general patterns for that particular attack. We can see that this is the case in Fig. 5.21 where there is reasonable recall (low FN) showing that the attacks can be found, however, at the cost of poor precision (high FP), and that almost all the flow has to be observed before good recall is observed.

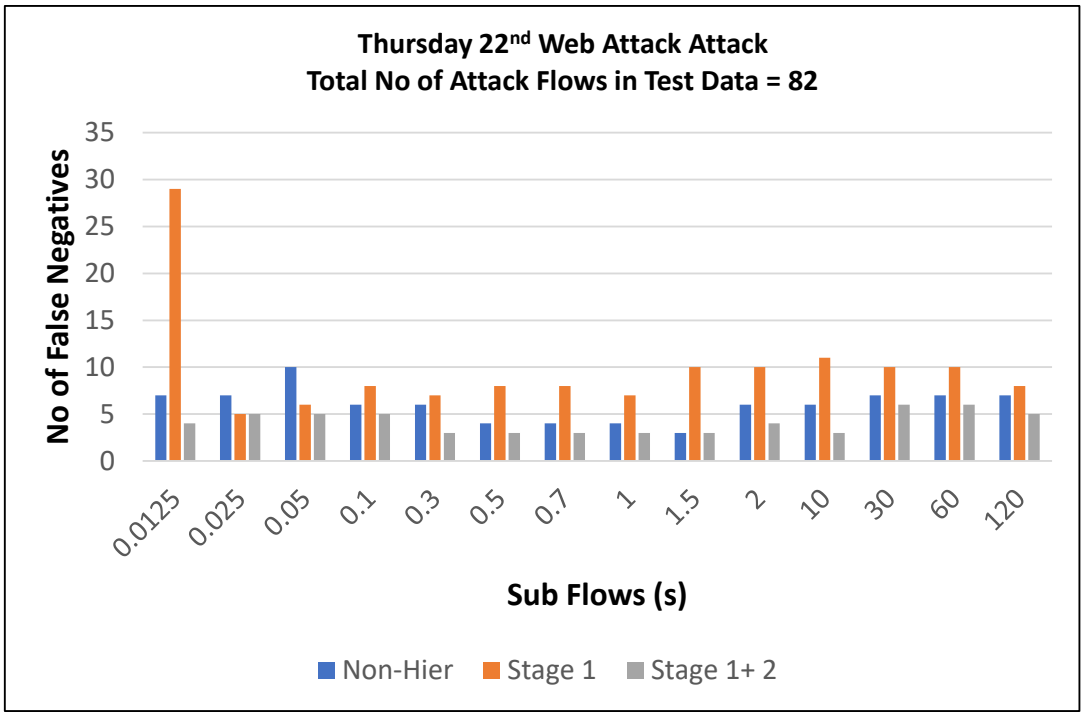


Figure 5.22: Hierarchical Intrusion Detection Results for Web Attack (Thur 22nd)

Table 5.10: Table showing Hierarchical Intrusion Detection Results for Web Attack (Thur 22)

THUR22nd WEB (No of Attack Flows = 82) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	7	29	4
0.025	7	5	5
0.05	10	6	5
0.1	6	8	5
0.3	6	7	3
0.5	4	8	3
0.7	4	8	3
1	4	7	3
1.5	3	10	3
2	6	10	4
10	6	11	3
30	7	10	6
60	7	10	6
120	7	8	5

THUR22nd WEB (No of Attack Flows = 82) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	7	2	9
0.025	9	8	17
0.05	9	9	18
0.1	10	14	20
0.3	7	11	18
0.5	8	12	20
0.7	6	14	20
1	7	13	19
1.5	8	10	18
2	14	8	22
10	5	9	14
30	7	8	14
60	5	3	8
120	6	6	11

Non-Hier (Subflow = 0.5s Web Attack)		
n = 82	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2535276 TN	8 FP
Actual Yes (DDoS)	4 FN	78 TP

1st Stage (Subflow = 0.5s Web Attack)		
n = 82	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2535272 TN	12 FP
Actual Yes (DDoS)	8 FN	74 TP

2nd Stage (Subflow = 0.5s Web Attack)		
n = 82	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2535264 TN	20 FP
Actual Yes (DDoS)	3 FN	79 TP

Figure 5.23: Sample Confusion Matrix for Web Attack Detection

The next results for the Web attacks present Web brute-force and Brute-force XSS. The results for Web brute-force are ignored because they follow much the same pattern as the Web brute-force results presented earlier. The derived metrics for the Brute-force XSS are also quite similar with the other Web attacks exception for the Detection Cutoff which gives a much poorer value of 0.012%. The poor Detection Cutoff offers little protection from the hierarchical solution despite the relatively good Attack flows Completed value of 18%. The derived metrics are shown in Table 5.11. However, investigating the hierarchical solution at 0.0125s cutoff gives an 18.18% reduction in false negatives as against a non-hierarchical approach. The same consideration for false positives gives a 20% increase in false positives. The slight increase in false positive shows a less likely impact on overall network performance.

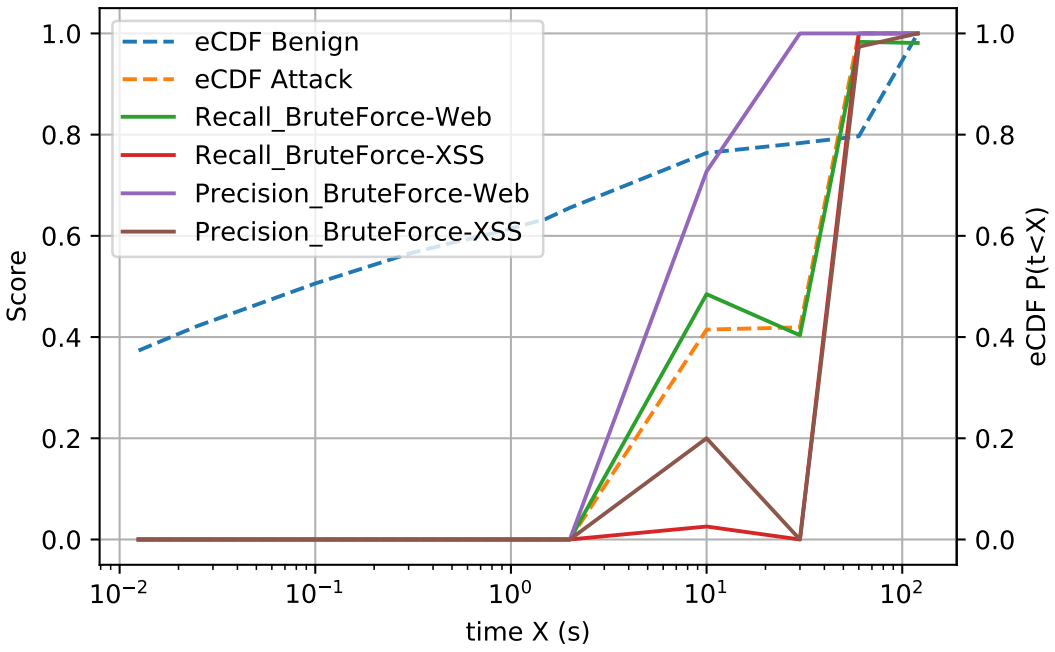


Figure 5.24: 1st Stage Detection for Web Attack

Table 5.11: Derived Detection Metrics for Web Attack (Fri 23)

WEB_FRI23RD	
Detection Cuttoff (Recall)	0.44
Flow Duration Cuttoff	9s
Average Attack Duration	35.08
Attack Flows Completed	41.00%
Flow Duration Cuttoff Percent	25.65%

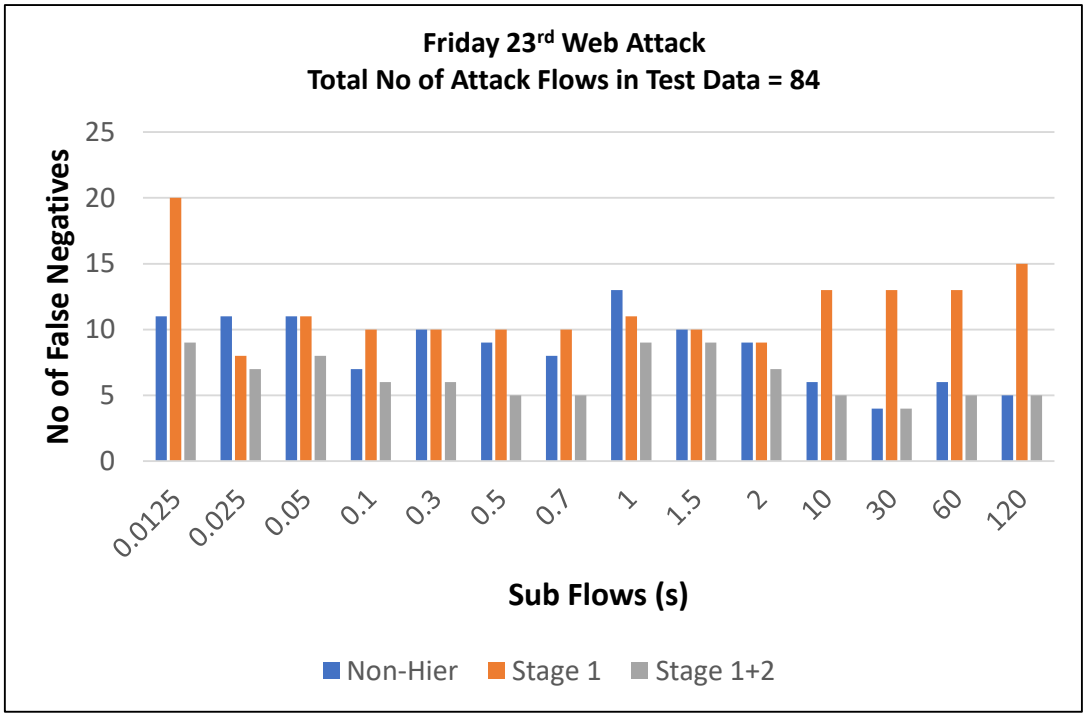


Figure 5.25: Hierarchical Intrusion Detection Results for Web Attack (Fri 23)

Non-Hier (Subflow = 0.5s Web Attack)		
n = 84	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2438269 TN	5 FP
Actual Yes (DDoS)	9 FN	75 TP

1st Stage (Subflow = 0.5s Web Attack)		
n = 84	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2438268 TN	6 FP
Actual Yes (DDoS)	10 FN	74 TP

2nd Stage (Subflow = 0.5s Web Attack)		
n = 84	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2438264 TN	10 FP
Actual Yes (DDoS)	5 FN	79 TP

Figure 5.26: Sample Confusion Matrix for Web Attack Detection

5.6.5 Infiltration Attack

The key flow duration-based statistics for the Infiltration attack dataset is given below:

Infiltrationwed28 data mean flow duration: 4.76s

Table 5.12: Table showing Hierarchical Intrusion Detection Results for Web Attack

FRI23rd WEB (No of Attack Flows = 84) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	11	20	9
0.025	11	8	7
0.05	11	11	8
0.1	7	10	6
0.3	10	10	6
0.5	9	10	5
0.7	8	10	5
1	13	11	9
1.5	10	10	9
2	9	9	7
10	6	13	5
30	4	13	4
60	6	13	5
120	5	15	5

FRI23rd WEB (No of Attack Flows = 84) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	5	2	6
0.025	5	3	7
0.05	5	4	8
0.1	6	8	13
0.3	2	4	5
0.5	5	6	10
0.7	7	7	13
1	3	5	7
1.5	7	10	16
2	9	12	20
10	13	14	26
30	9	6	14
60	9	10	18
120	10	9	18

Infiltrationwed28 attack traffic mean flow duration: 63.21s

The derived metrics are calculated as shown in previous examples and shown in Table 5.13. It is seen that flow duration cutoff is 0.5s which is an impressive 0.79% of the average attack duration. This is well within the threshold for the hierarchical solution. The attack flows completed is also an impressive 9% which satisfies the criteria for the hierarchical model. However, the detection cutoff of 0.35 offers very little protection. Interestingly with this attack type and with further investigation using the hierarchical model, there is a 0% reduction in false negatives at the flow duration cutoff of 0.5s as seen from Table 5.14. At this Cutoff, there is also no change in false positives which suggests negligible overall impact on network performance. As noted earlier, the machine learning performance shown in Fig. 5.27 is highly non-linear and does not increase monotonically with flow duration. For achieving both good recall and precision it would be required to wait until all the flow completes, again, as commented earlier, allowing detection but not mitigation.

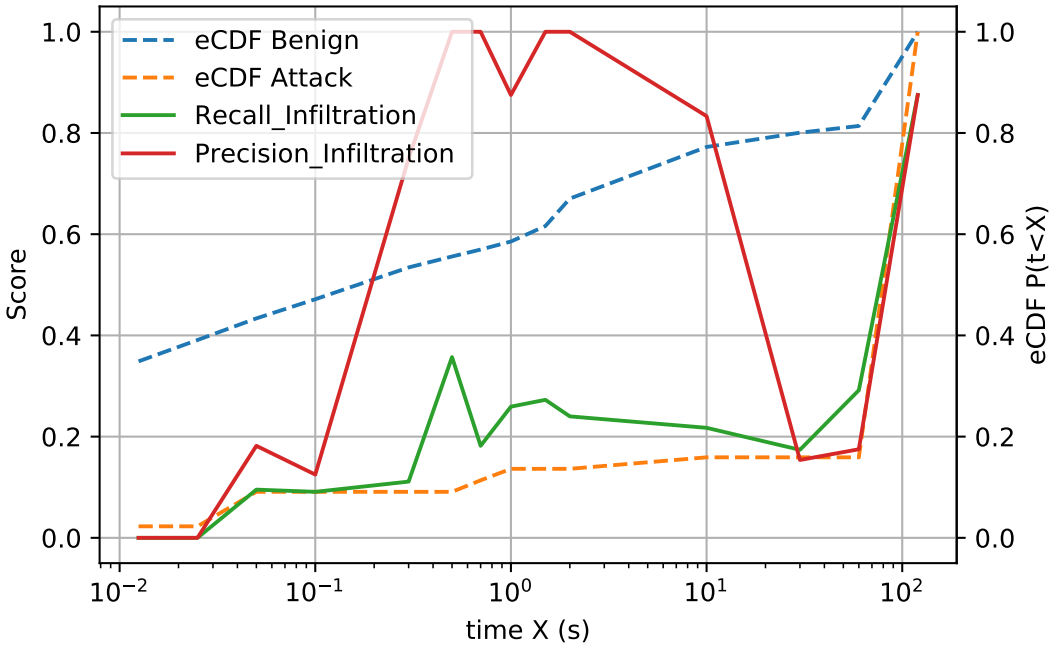


Figure 5.27: 1st Stage Detection for Infiltration Attack

Table 5.13: Derived Detection Metrics for Infiltration Attack

INFILTRATION_WED28TH	
Detection Cuttoff (Recall)	0.35
Flow Duration Cuttoff	0.5s
Average Attack Duration	63.2s
Attack Flows Completed	9.00%
Flow Duration Cuttoff Percent	0.79%

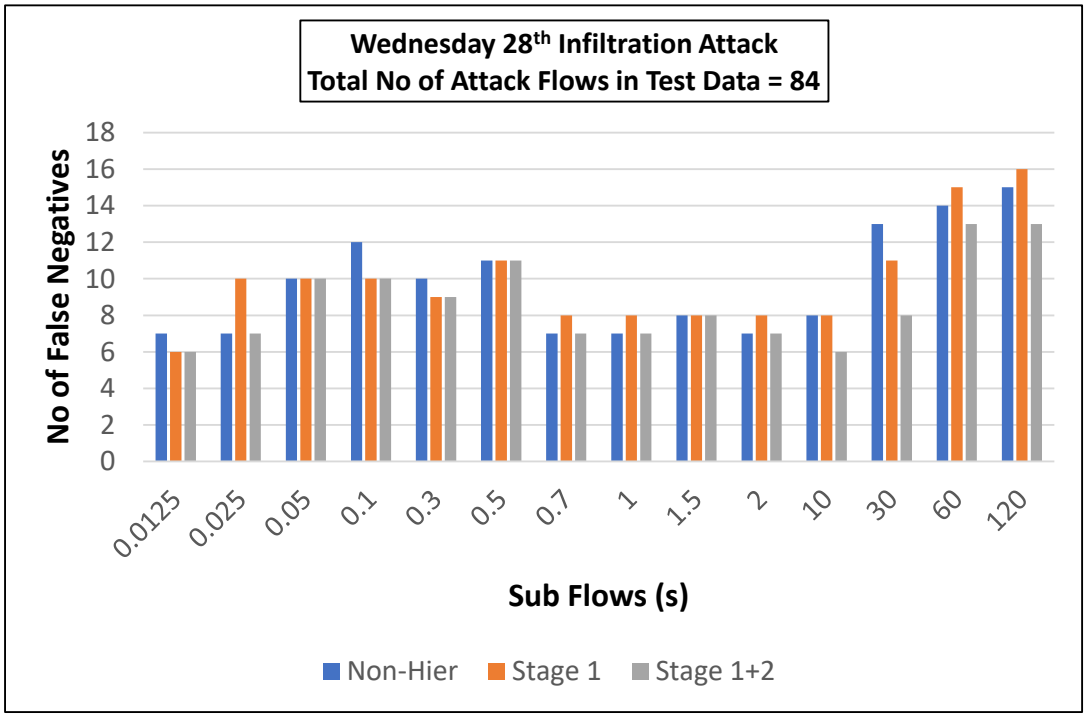


Figure 5.28: Hierarchical Intrusion Detection Results for Infiltration Attack

Table 5.14: Table showing Hierarchical Intrusion Detection Results for Infiltration Attack

WED28TH INFILTRATION (No of Attack Flows = 23) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	7	6	6
0.025	7	10	7
0.05	10	10	10
0.1	12	10	10
0.3	10	9	9
0.5	11	11	11
0.7	7	8	7
1	7	8	7
1.5	8	8	8
2	7	8	7
10	8	8	6
30	13	11	8
60	14	15	13
120	15	16	13

WED28TH INFILTRATION (No of Attack Flows = 23) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	3	0	3
0.025	1	0	1
0.05	1	1	2
0.1	0	0	0
0.3	1	0	1
0.5	0	0	0
0.7	0	0	0
1	1	0	1
1.5	2	0	2
2	1	0	1
10	0	0	0
30	4	3	7
60	1	3	4
120	1	1	2

Non-Hier (Subflow = 0.5s Infiltration)		
n = 23	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2777570 TN	0 FP
Actual Yes (DDoS)	11 FN	12 TP

1st Stage (Subflow = 0.5s Infiltration)		
n = 23	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2777570 TN	0 FP
Actual Yes (DDoS)	11 FN	12 TP

2nd Stage (Subflow = 0.5s Infiltration)		
n = 23	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2777570 TN	0 FP
Actual Yes (DDoS)	511 FN	12 TP

Figure 5.29: Sample Confusion Matrix for Infiltration Attack Detection

5.6.6 BoT Attack

Due to the very low Average Attack Duration (0.0154s) of the BoT attack traffic, the derived hierarchical metrics would be unreliable at this very low flow duration times. Based on the limits of this experiment, it is impossible to analyse the first stage derived metrics for the BoT attack. However, given the high recall at values close to the mean flow duration suggests possible significant good response. Further investigation into the hierarchical solution gives an average reduction of 52.39% in false negatives across all the sub-flow times. The same analysis on the false positives give a 213% average increase across all sub-flow times which suggests a significant impact on network operations.

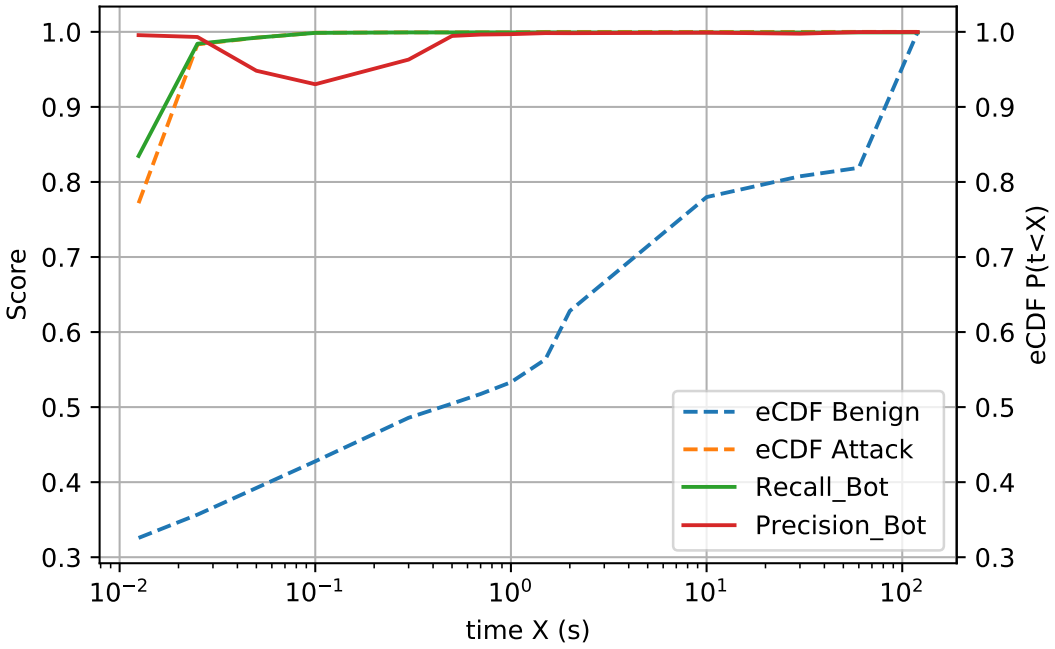


Figure 5.30: 1st Stage Detection for BoT Attack

Table 5.15: Table showing Hierarchical Intrusion Detection Results for BoT Attack

FRI2nd BOT (No of Attack Flows = 57052) FALSE NEGATIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	14	380	11
0.025	11	383	11
0.05	7	175	4
0.1	8	40	7
0.3	4	32	1
0.5	5	32	1
0.7	5	36	5
1	7	36	3
1.5	6	36	2
2	5	38	2
10	7	34	1
30	4	37	4
60	12	45	6
120	8	41	6

FRI2nd BOT (No of Attack Flows = 57052) FALSE POSITIVES			
Subflow	**Non-Hier**	**1st St.**	**2nd St.**
0.0125	19	4	21
0.025	13	26	39
0.05	6	7	12
0.1	4	10	14
0.3	4	4	8
0.5	10	4	14
0.7	4	4	8
1	4	6	10
1.5	3	5	8
2	1	11	12
10	5	6	10
30	10	5	15
60	5	17	22
120	5	14	19

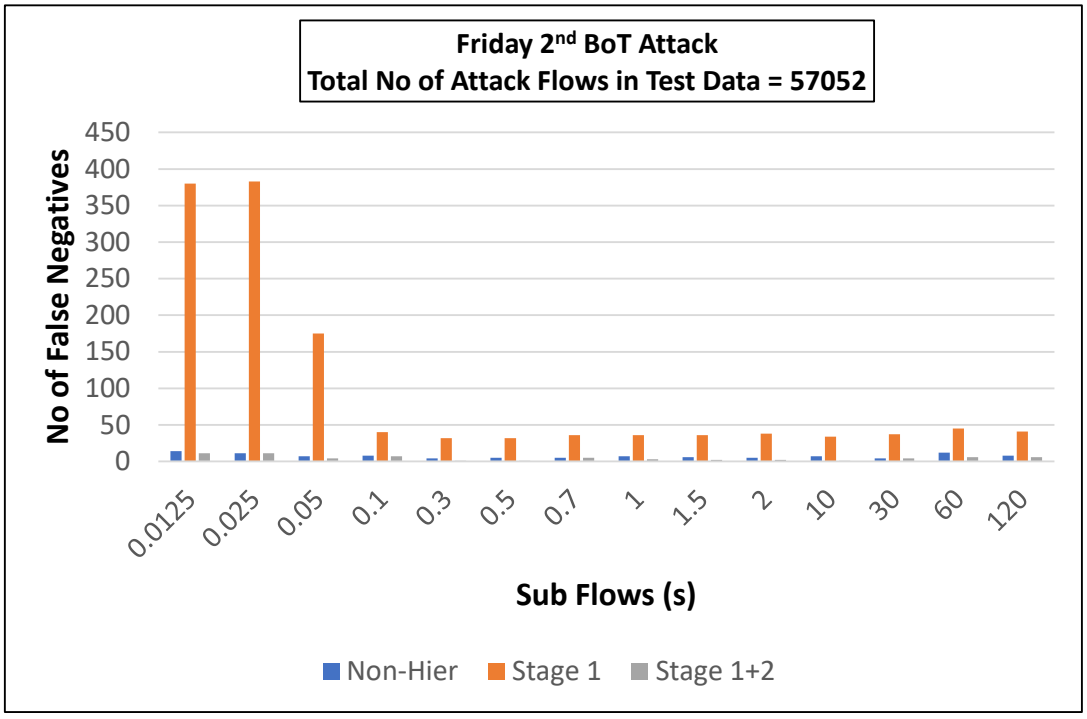


Figure 5.31: Hierarchical Intrusion Detection Results for BoT Attack

Non-Hier (Subflow = 0.5s Infiltration)		
n = 57052	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2512796 TN	10 FP
Actual Yes (DDoS)	5 FN	57047 TP

1st Stage (Subflow = 0.5s Infiltration)		
n = 57052	Predicted No (BENIGN)	Predicted Yes (DDoS)
Actual No (BENIGN)	2512802 TN	4 FP
Actual Yes (DDoS)	32 FN	57020 TP

2nd Stage (Subflow = 0.5s Infiltration)		
n = 57052	Predicted No (BENIGN)	Predicted Yes DDoS
Actual No (BENIGN)	2512792 TN	14 FP
Actual Yes (DDoS)	1 FN	57051 TP

Figure 5.32: Sample Confusion Matrix for BoT Attack Detection

5.7 Summary of Results

The security solution presented in this work comes with a unique mode of operation and unique performance metrics. These performance metrics which are defined in Section 5.6 and results of which are summarised in Table 5.16. The table also represents a summary of the results presented in this chapter. The table can be analysed in three categories based on the unique metrics of evaluation: 1) **real-time implication** which comprises of the detection cutoff (recall), flow duration cutoff percent and attack flows completed percent; 2) **accuracy implication** which comprises of detection cutoff, attack flows completed percent and reduced false negative rate (FNR); 3) **cost implication** which comprises of increased false positive rate (FPR). The cost implication is considered in terms of how much worse this solution would cause other network performance parameters (outside of security) to be. Based on the limitations of this work, the cost is evaluated in terms of increase in FPR which may result in loss of information due to benign packets being dropped.

Based on the different metrics considered, it is seen that the proposed hierarchical SDN solution adapts more suitably to the DoS and DDoS attacks. The other attacks also show good potential but with detection times after a relatively larger proportion of the flows have completed. For the Brute Force attacks a much less aggressive cut-off is required, for example Figure 5.9 shows that nearly all the attack flows would need to be completed to achieve good recall. Other attacks still show potential despite poor suitability, for example in terms of better FN as against using just a single stage and as against using a traditional non-hierarchical approach.

A key point to note in the trend of the overall results is that, across all the attack types, there is a lower lower number of FNs at the end of the second stage when compared to the first stage and when compared to the non-hierarchical implementation. Analysis in Sections 4.3 and 5.5 clearly shows that a low FN output is desirable. This shows that the proposed hierarchical solution has the potential to provide higher recall than the traditional non-hierarchical approach. Crucially

Table 5.16: Summary Table for Hierarchical Detection Results Using Chosen, Agressive, Flow Cut-Off Criteria (Detection at or less than 10% mean flow duration)

Attack Type	Detection Cuttoff (Recall)	Flow Duration Cuttoff (%)	Attack Flows Completed (%)	Reduced FNR	Increased FPR	Overall Suitability
BRUTE_WED14TH	0	10	68	Good	High	Poor
DOS_FRI16TH	0.9998	3.87	55	Moderate	Low	Moderate
DDOS_TUE20TH	0.998	8.54	3	High	Low	High
DDOS_WED21ST	0.574	10	57.39	High	Low	Moderate
WEB_THU22ND	0.21	10	19	Low	Low	Poor
WEB_FRI23RD	0.44	25.65	41	Low	Low	Poor
INFIL_WED28TH	0.35	0.75	9	Low	Low	Poor
BOT_FRI2ND	N/A	N/A	N/A	Moderate	Low	Inconclusive

also, the sequence of results show that using only the first stage classifier may not be enough to offer adequate protection to the network. Another interesting trend of results is the comparison between number of FNs across the non-hierarchical and the first stage. It is seen that some results show a sharp drop in the number of FNs from the non-hier to first stage whereas some others show a sharp rise in the nunber of FNs. Specifically, the DDoS attacks (LOIC and HOIC) show a sharp drop in FNs from non-hier to first stage while the DoS and BoT attacks show a sharp rise in FNs within the same comparison. Also, there is an interesting trend between the FNs across the first and the second stages. In this case, the trend is only downwards from the first to the second stage. This means that there is consistently higher recall when using the hierarchical approach as compared to using only the first stage. This is in addition to a consistently higher accuracy (lower FNs) when using the hierarchical solution as compared to when using the non-hierarchical approach, which is actually one of the main benefits of this solution.

Note that, further work would need to be done to adapt the hierarchical solution to more attack types. Practical deployment would require a choice between attacks detected, sub-flow duration and acceptable recall/precision for a given attack.

5.8 Summary

In this chapter, a novel hierarchical machine learning based security solution in SDN has been developed. The solution showed additional attack detection accuracy compared to a conventional non-hierarchical approach. This is demonstrated by way of a reduction in false negative rate upon deployment of the hierarchical solution as explained in Section 5.6.1. In addition, the hierarchical solution supports real time or near real-time detection in some attacks. This is the first work to use a hierarchical machine-learning solution to achieve real-time attack detection in SDN.

Work done in this chapter also created unique performance parameters to evaluate and analyse the hierarchical solution. These unique evaluation metrics can be adopted in future studies in the area of real time attack detection in next generation networks.

Also, this is the first work that has shown how different attack types respond to sub-flow classification and also how different attack types respond to a change in flow monitoring approach.

This is also the first work to introduce a sub-flow generator in the flow-meter design based on the IPFIX standard. The IPFIX standard does not provide for a sub-flow generator. This is necessary to implement real-time machine learning based security in networks as seen demonstrated for SDN in this work. Hence, this is the first work to use historic data to recreate a real-time scenario to test the effectiveness of using machine learning algorithms to detect specific attack types in SDN security.

This chapter did find instances where the hierarchical approach used in this chapter had poorer performance than the non-hierarchical approach in terms of false positives (lower precision). This was caused by only sending traffic classified as benign to the second stage (blocking all classified as attack). The next chapter will provide a solutions to this issue.

Chapter 6

Improving Intrusion Detection Efficiency Through Probabilistic Classification

6.1 Introduction

Work done in this Chapter highlights the savings in effort at the edge of the network. In a regular network topology, the anomaly detection or intrusion detection system at the edge would be designed to process all packets passing through it in order to make security decisions. This would result in extensive workload for the edge forwarding and security devices. One of the major goals of this work is to improve the efficiency of the intrusion detection process. In the work done in Chapter 5, all classified benign flows or traffic is switched to the second stage classifier. In this chapter, a probabilistic approach is used where the classification probability is used to send only traffic with prediction probability lower than a certain threshold. This further reduces the volume of traffic being processed at the second stage classifier. The final results shown at the second stage is the combination of the first and second stage results. The final results is compared with results from the non-hierarchical approach.

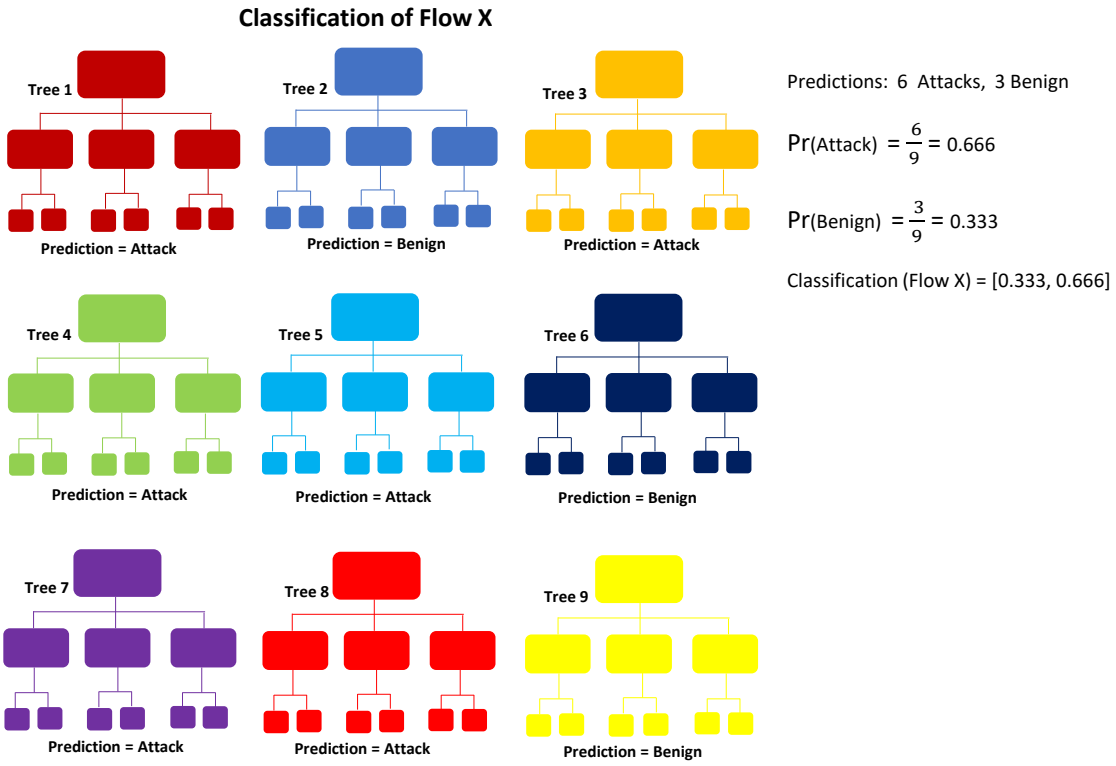


Figure 6.1: Simplified Illustration of Random Forest Probabilities

6.2 Why Random Forest Classifier

The main reason why Random Forest Classifier is used in this chapter is because the Decision Tree Classifier is not intrinsically a probabilistic model. On the Random Forest Classifier, for each traffic flow, each tree predicts a class. The algorithm then calculates different probabilities for each class for that particular flow. The output probabilities for each flow are the fractions of total numbers of trees which predicted a specific class. A simplified illustartion of how Random Forest Classifier calculates its probabilities is shown in Figure 6.1. All calculated class probabilities for each flow sum up to 1. Obviously, the class with the highest probability is the predicted class. Each flow therefore has different probabilities for each class. This concept of different probabilities for each flow is one of the main techniques used in this chapter.

6.3 Probability Threshold Analysis

The concept of classification probability is utilised to control the volume of traffic forwarded to the second stage classifier. The overall idea is to limit the volume of packets to be processed by

the edge classifier while maintaining a similar overall attack detection accuracy.

High probability attack traffic in the context of benign classifications is equivalent to low probability benign classification. Hence the lower the probability of benign classification, the higher the possibility of an attack flow. Therefore, only benign traffic below the threshold is sent to the second stage classifier. Accordingly, all benign traffic higher than the given threshold is allowed through to the network.

With regards to attack traffic, the higher the probability of attack traffic, the higher the possibility of an attack. Consequently, all attack traffic with classification probabilities higher than the threshold is dropped. As with the benign classifications, all traffic with positive classification probabilities lower than the threshold are sent to the second stage classifier. This is illustrated in Figure 6.2. Hence, a threshold is chosen such that all traffic with classification probability below the threshold is sent to the second stage classifier.

Table 6.2 shows a table of analysis to work out what the threshold should be. Table 6.2 shows the percentiles of volumes of flows according to classification probability. The table shows the classification probabilities for DoS and DDoS. For DoS, the result shows that up to 99% of the flows are predicted with certainty while the value for DDoS is 95.4%. Hence, a threshold value slightly lower than certainty (probability = 1) should be chosen, for example, the value of 0.999 has been used as a suggested threshold for the examples shown here. The results shown in Table 6.2 represent values at the flow duration cutoff (calculated in Chapter 5) of both attack types (1.8s and 2s respectively). In practice, the threshold value may need to be optimised as an extensive engineering exercise on real data and would depend on the machine learning algorithm, the traffic or attack type and the system application amongst other factors. This has not been done primarily due to limitation in computational resources and amount of time that this would require. The Table 6.1 summarises the application of classification probability to the hierarchical SDN security solution.

6.4 Malicious Packet Detection Efficiency based on The Probabilistic Approach

The probability approach has resulted in a significant reduction in traffic to the second stage classifier. This reduction is in varying proportions with respect to different attack types and across different sub-flow times. This is mainly responsible for the increase in prediction rate at

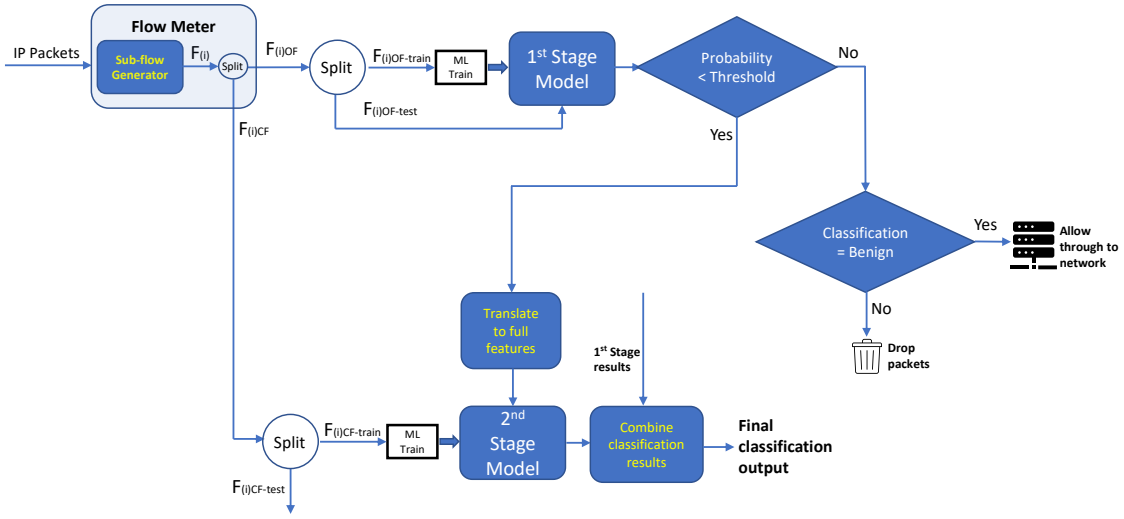


Figure 6.2: Experimental Design based on Probabilistic Method

Table 6.1: Classification Probability Operation for Hierarchical SDN Security Solution

Classification Probability	Description	Action
Benign < Threshold	Less likely benign. Possibly a False Negative	Send to Stage 2
Benign > Threshold	Most likely benign	Allow packets through
Attack < Threshold	Possibly a False Positive	Send to Stage 2
Attack > Threshold	Most likely an attack	Drop packets

the edge when compared to using a non-hierarchical system. The remainder of this section will present the results of the probabilistic approach in more detail. The results are presented in a three graph format. The first graph shows the percentage of total traffic sent to the second stage classifier as a result of the implementing the probability method. The second graph compares the prediction rates between implementing a traditional non-hierarchical approach and the hierarchical solution developed so far. The third graph compares the overall detection accuracy between the traditional non-hierarchical approach and the hierarchical solution. This overall detection accuracy is combination of the classification results from the first and the second stages. This process has been explained in Chapter 5. In terms of the prediction rate, this is calculated as the number of flows processed by a single core Intel Xeon CPU (E5-2680 v4) operating at 2.40 GHz and with 32 GBytes of RAM.

Table 6.2: Traffic Classification Probabilities, Basic Statistics and Probability by Traffic Percentiles

DOS_FRI16TH	
Classification Probability	
mean	0.9999
std	0.00147
min	0.52
0%	0.52
0.1%	0.9877
0.2%	0.9878
0.3%	0.9878
0.4%	0.9878
0.5%	0.9991
1.0%	1
10%	1
50%	1
max	1

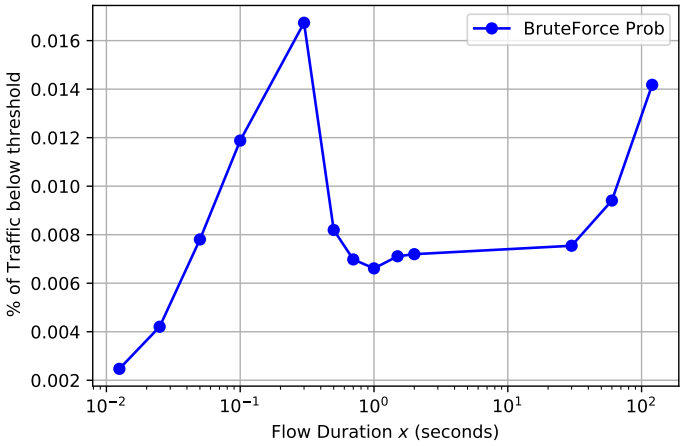
DDOS_TUE20TH	
Classification Probability	
mean	0.9997
std	0.001844
min	0.513
0%	0.513
0.1%	0.9952
1.0%	0.9952
2.0%	0.9952
3.0%	0.9954
4.0%	0.9954
4.5%	0.9954
4.6%	1
50%	1
max	1

6.4.1 Brute-Force Attack

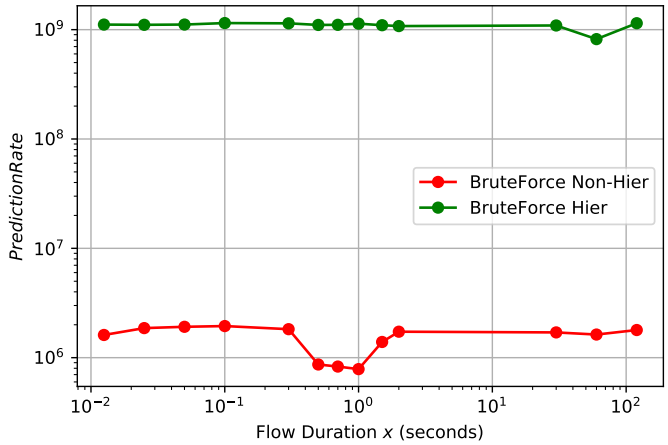
For the Brute-Force attack as shown in Figure 6.3, less than 1% of total traffic is sent to the second stage across all the sub-flow times. This results in an increase in prediction rate by approximately a factor of 1000. This represents a significant savings in CPU resources. As a result of the reduced traffic to the second stage classifier, it might have been expected that the detection accuracy of the hierarchical solution should drop by some significant fraction. However, the Brute-Force attack shows a marginal increase in detection accuracy as shown in the figure. This is because the detection accuracy shown is a combination of results from stages one and two. The marginal increase in overall detection accuracy demonstrates additional potential of this solution with this attack type. To present these results in further perspective regarding the hierarchical solution; for a flow duration cutoff of 0.012s (calculated in Section 5.6.1), there is a approximately 1000 times increase in prediction rate and a marginal improvement in detection accuracy. It is difficult to determine the reason for the increase in accuracy for the hierarchical model; however, it may be attributed to the fact that there are now two machine learning models, with the second stage highly tuned to filter out just the much smaller number of flows that fail accurate detection in the first stage.

6.4.2 DoS Attack

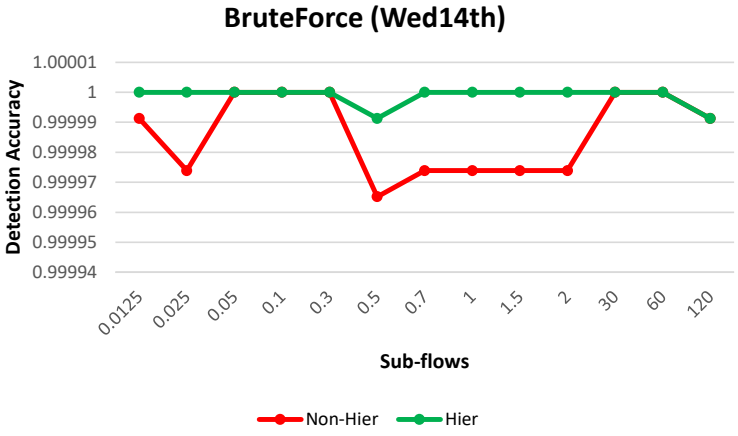
For the DoS attack as shown in Figure 6.4, less than 2% of total traffic is sent to the second stage across all sub-flows. Early stages of flow show about 1.75% of traffic below the probability



(a) Percentage of traffic below threshold for 2nd stage

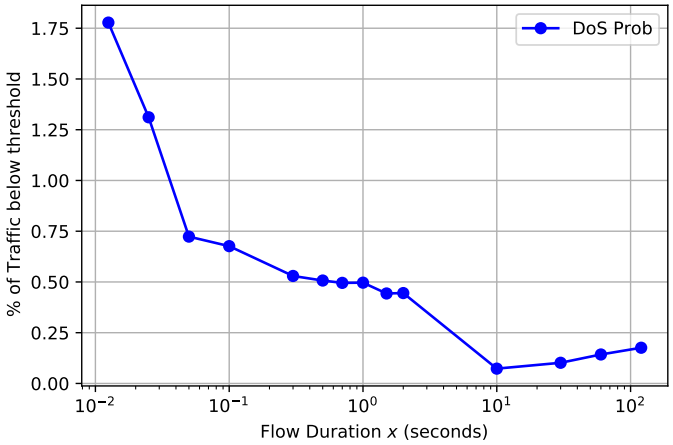


(b) Prediction Rate in flows per second

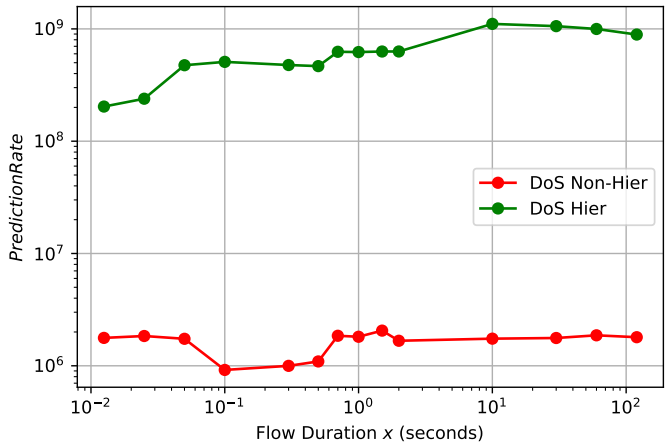


(c) Detection Accuracy

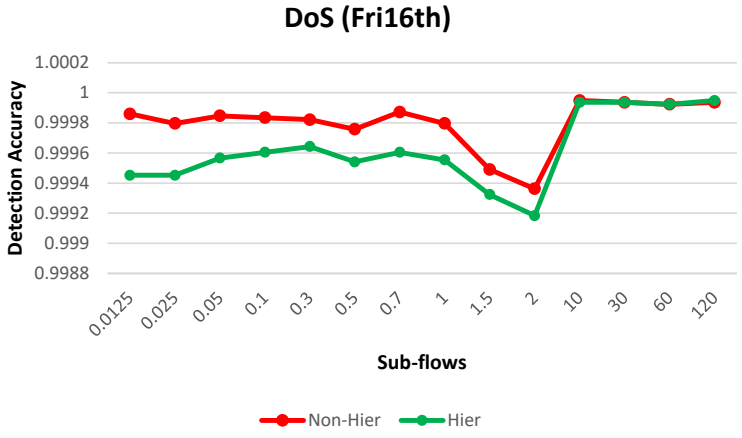
Figure 6.3: Efficiency Results for BruteForce



(a) Percentage of traffic below threshold for 2nd stage



(b) Prediction Rate in flows per second



(c) Detection Accuracy

Figure 6.4: Efficiency Results for DoS

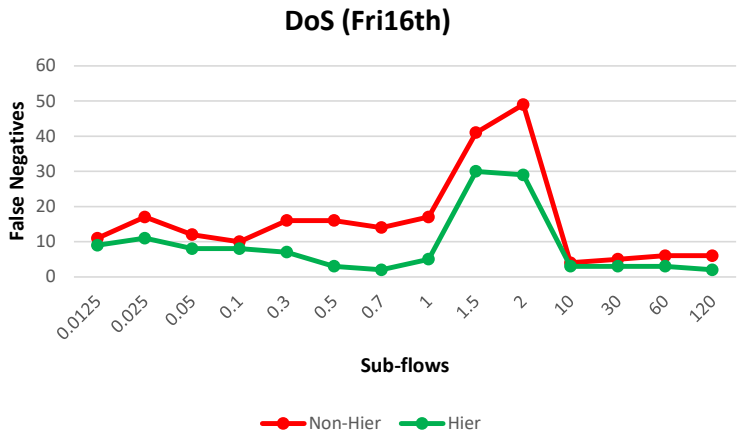


Figure 6.5: Reduction in False Negatives using Hierarchical Solution without Probability Option

threshold. This results in approximately 100 times increase in prediction rate at the early stages of the flow duration. There is a gradual decrease in percentage of traffic sent to the second stage as sub-flow times increase. At latter sub-flow times, there is slightly less than 1000 times increase in prediction rate. Based on the hierarchical solution, for a flow duration cutoff of 1.8s (approx 2s) (calculated in Section 5.6.2), 0.4% of traffic is sent to the second stage classifier with close to 1000 times increase in prediction rate. Interestingly, there is a marginal reduction in overall accuracy of the hierarchical solution upon implementing the probabilistic option. However, results from Section 5.6.2 show an overall reduction in number of false negatives upon implementing the hierarchical solution which clearly indicated an increase in overall accuracy. The relevant portion of the result from Section 5.6.2 is shown in Figure 6.5.

However, the marginal decrease in overall detection accuracy shown in Figure 6.4 may be further investigated by optimising the probability threshold value. This has not been done in this work due to limitation in computation resources as mentioned in Section 6.3. Despite the slight reduction in overall accuracy, the probabilistic option of the hierarchical solution still shows significant potential in efficient malicious packet detection in SDN networks.

6.4.3 DDoS Attack

The DDoS-LOIC attack shows a reduction in traffic of between 8 to 10% across the sub-flows. This results in approximately 10 times increase in prediction rate across sub-flow times. This results as shown in Figure 6.6 shows that there is no significant change in overall detection accuracy. Reference to the hierarchical solution, the flow duration cutoff of 2s gives the approximate responses already stated. The DDoS-HOIC results shown in Figure 6.7 show the same trend except

for percentage of traffic sent to stage two which is as high as 26% for lower sub-flow times and approximately 15% at higher sub-flow times. For flow duration cutoff of 0.0239s (as calculated in Section 5.6.3), approximately 23% of traffic is sent to stage two. The DDoS-HOIC was the most challenging of the attack types to be investigated. The fact that as much as 26% of the traffic had to be sent to the second stage reflects the fact that the first stage classifier was less certain on the classification and required greater power from the second stage classifier. Looking into the attack for the DDoS-HOIC [132], the reason for this is that DDoS-HOIC has the capability to change its behaviour on each request through "fuzzing" whereby the attack parameters are changed a little on each request. Consequently, it is harder to distinguish the attack or benign classification from only the OpenFlow features exposed at the first stage.

6.4.4 Web Attack

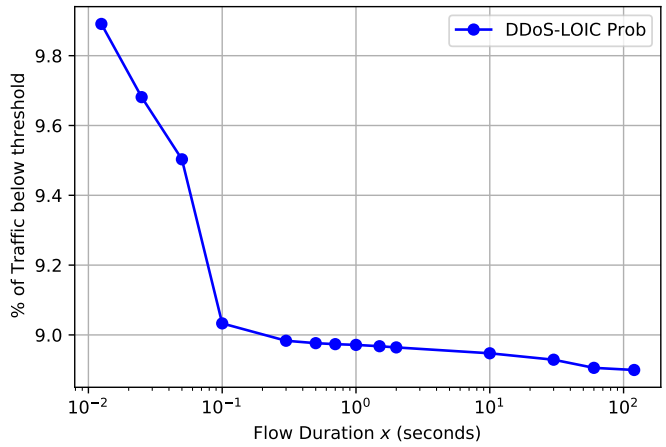
The two types of web attacks shown in Figures 6.8 and 6.9 give very similar results. For both attack traffic, less than 1% of the traffic is sent to the second stage classifier, resulting in 1000 speed up in prediction rate. Also there is negligible difference in overall system accuracy. The flow duration cutoff of both attacks give return the average responses stated already.

6.4.5 Infiltration Attack

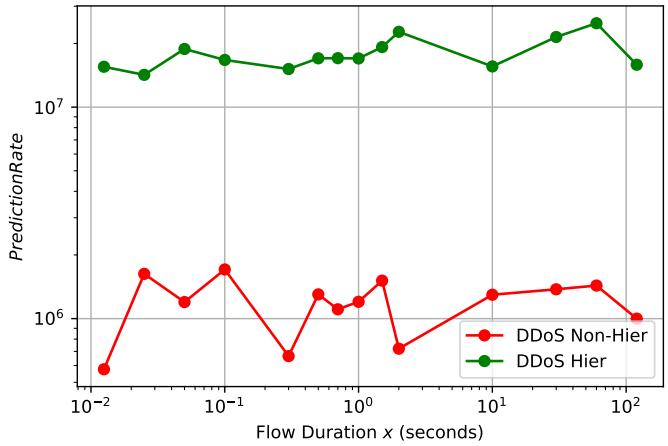
The infiltration attack shows a similar trend with less than 1% of total traffic sent to classifier 2 across all sub-flows. This results in approximately 1000 times increase in prediction rate with an insignificant change in overall detection accuracy. Results for the infiltration attack is shown Figure 6.10. The flow duration cutoff of 0.5s is well within the approximate range of response stated.

6.4.6 BoT Attack

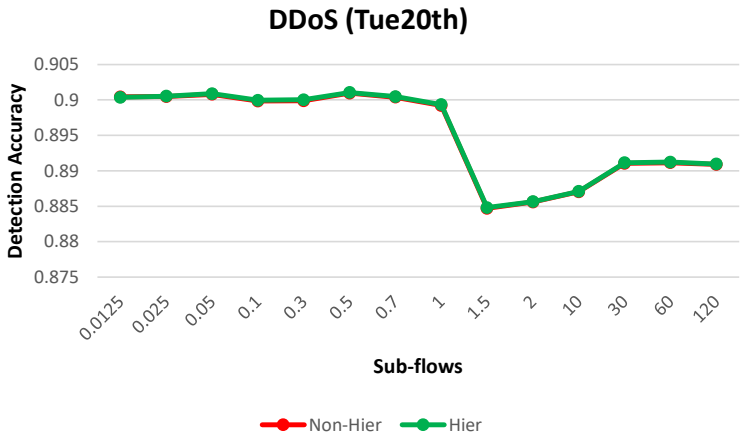
The probabilistic results for BoT attack as seen in Figure 6.11 shows a significant reduction in traffic with approximately between 0.1% to 1.2% of the traffic sent to the second stage across all sub-flow times. Similarly, this results in approximately 1000 times increase in prediction rate across all sub-flow times with a negligible change in overall detection accuracy. For this attack, as seen in Section 5.6.6, the flow duration cutoff is unavailable, but results here show strong potential with the probabilistic approach.



(a) Percentage of traffic below threshold for 2nd stage

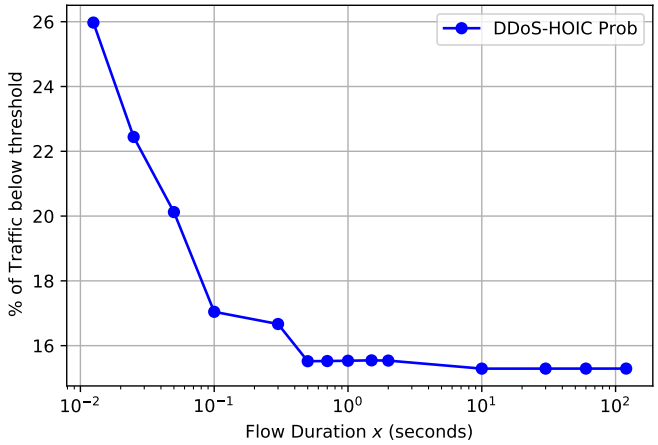


(b) Prediction Rate in flows per second

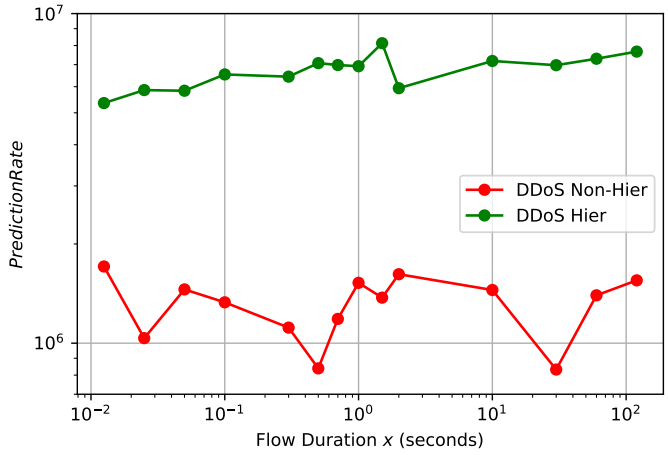


(c) Detection Accuracy

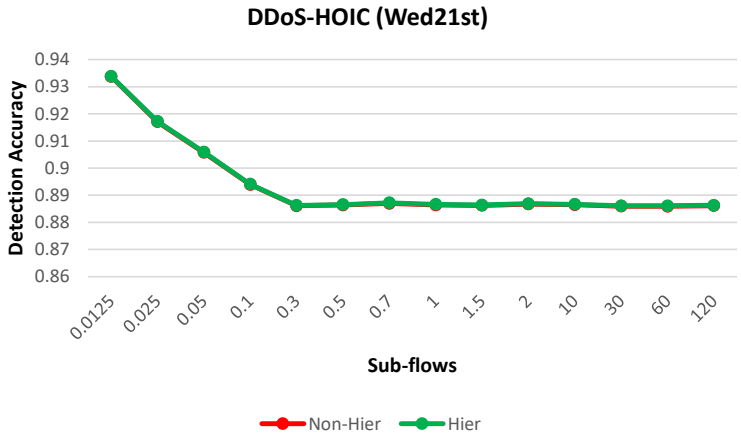
Figure 6.6: Detection Efficiency Results for DDoS-HoIC



(a) Percentage of traffic below threshold for 2nd stage

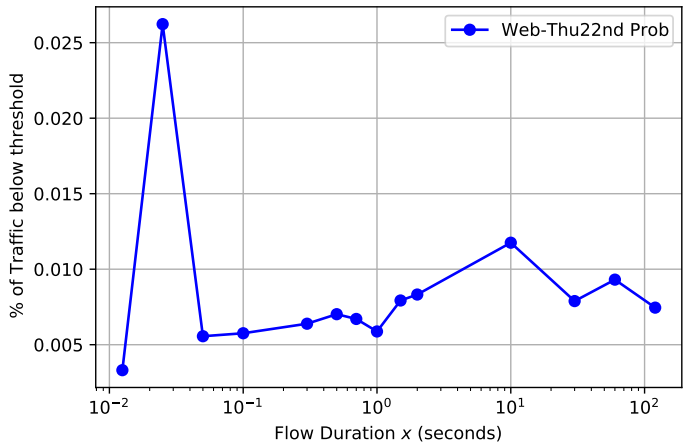


(b) Prediction Rate in flows per second

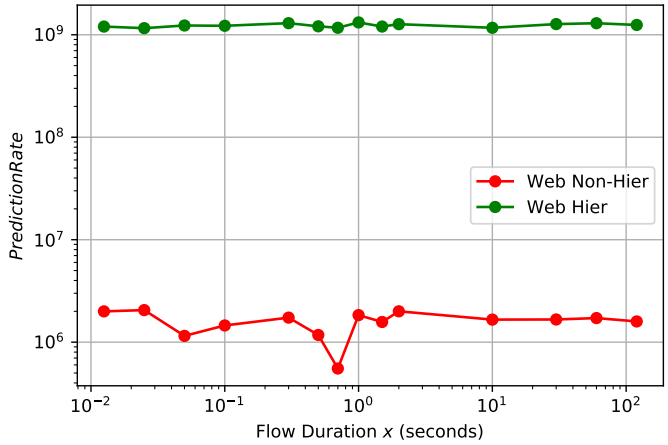


(c) Detection Efficiency Results for DDoS

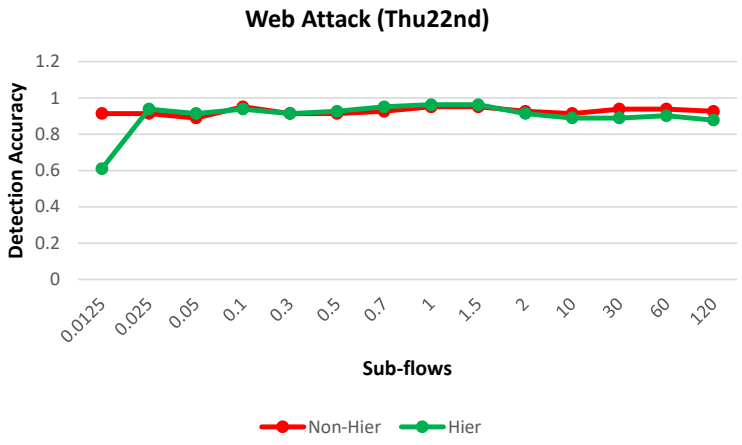
Figure 6.7: Detection Efficiency Results for DDoS-HOIC



(a) Percentage of traffic below threshold for 2nd stage

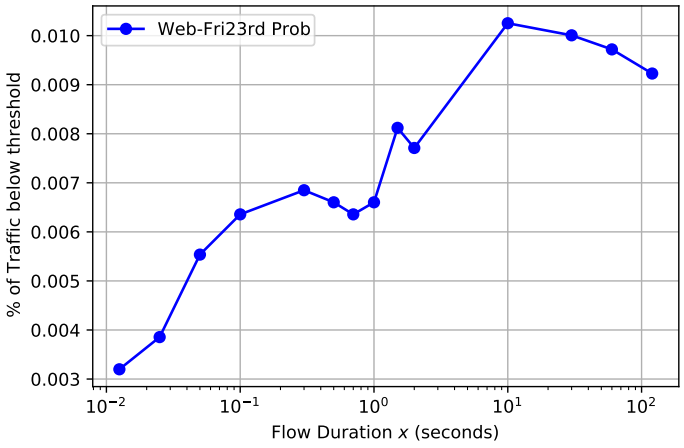


(b) Prediction Rate in flows per second

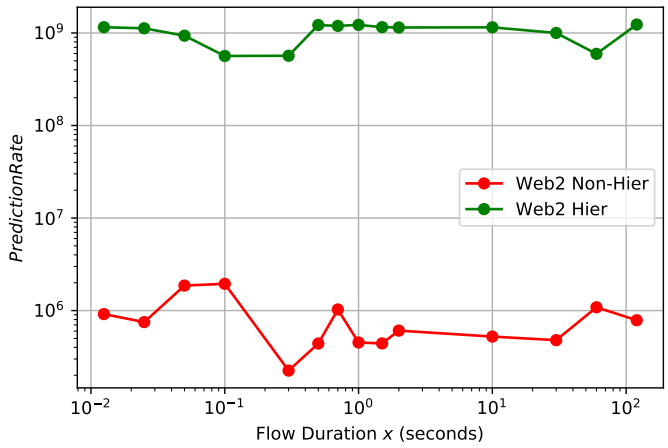


(c) Detection Accuracy

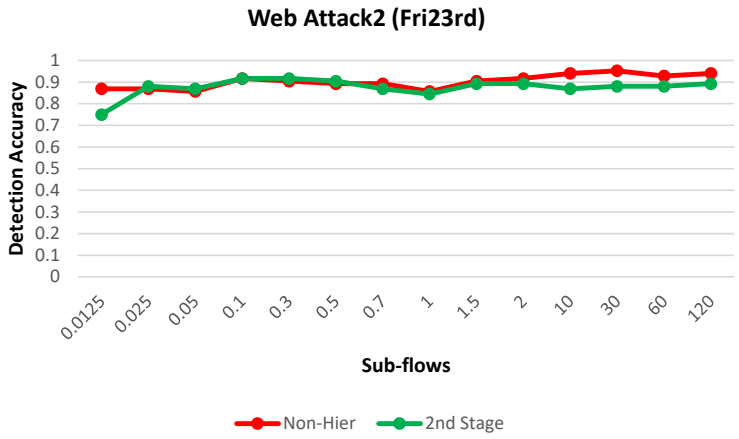
Figure 6.8: Detection Efficiency Results for WEB Attack



(a) Percentage of traffic below threshold for 2nd stage

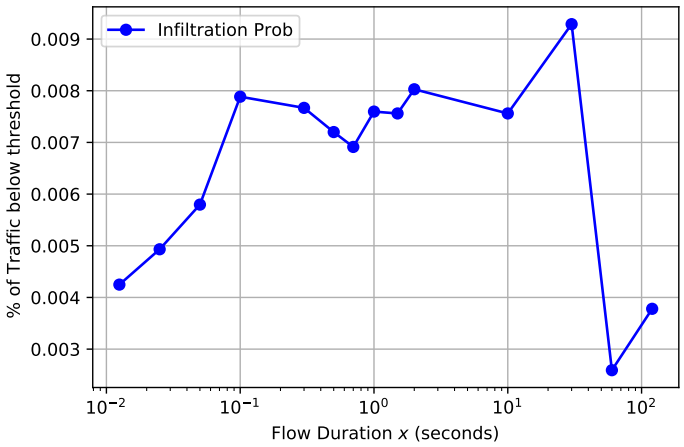


(b) Prediction Rate in flows per second

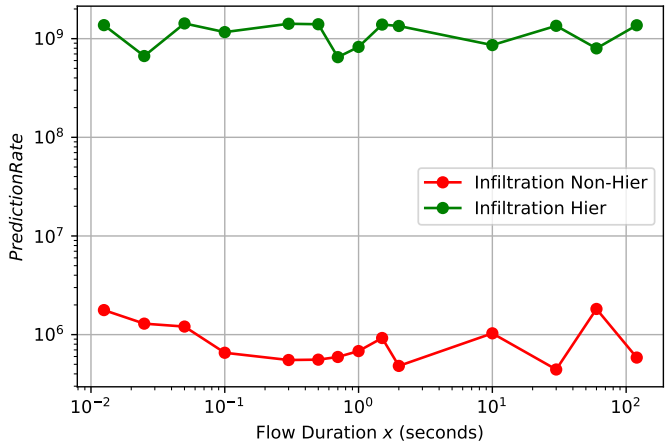


(c) Detection Accuracy

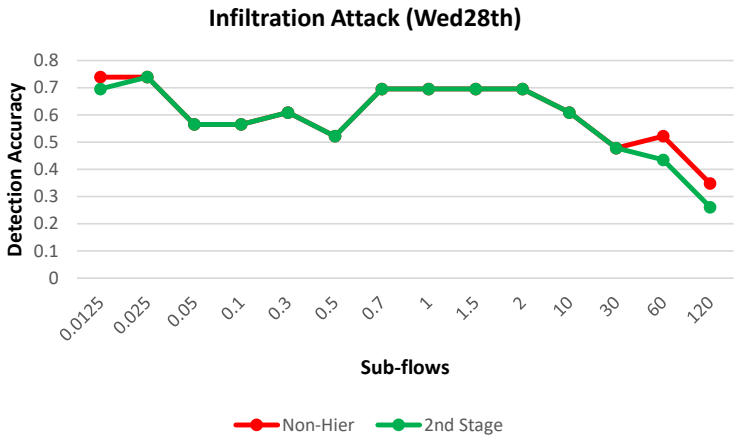
Figure 6.9: Detection Efficiency Results for WEB Attack



(a) Percentage of traffic below threshold for 2nd stage

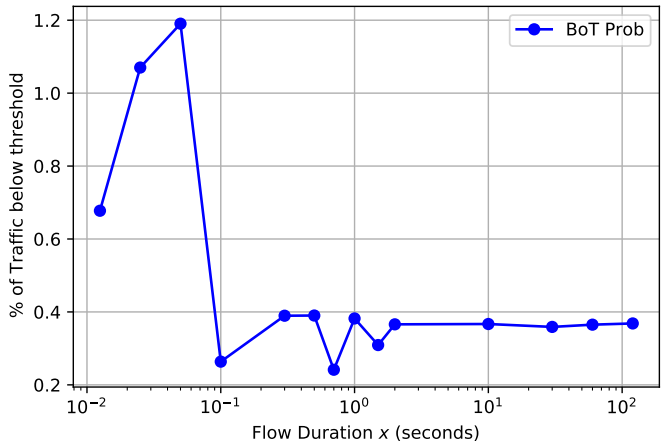


(b) Prediction Rate in flows per second

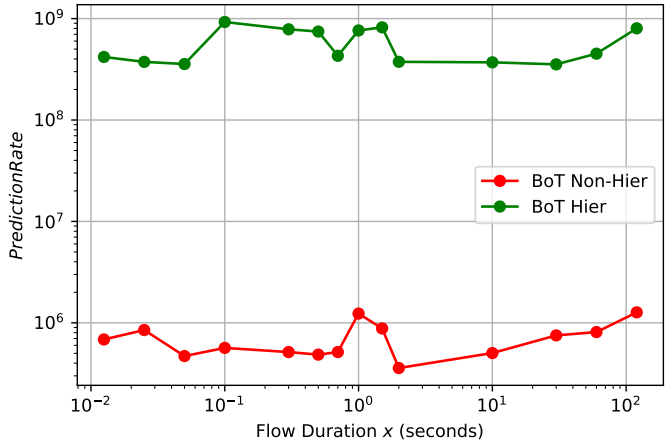


(c) Detection Accuracy

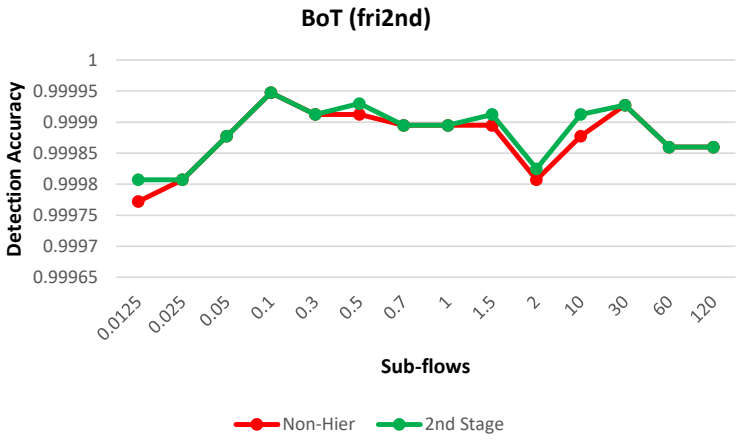
Figure 6.10: Detection Efficiency Results for Infiltration Attack



(a) Percentage of traffic below threshold for 2nd stage



(b) Prediction Rate in flows per second



(c) Detection Accuracy

Figure 6.11: Detection Efficiency Results for BoT Attack

6.5 Summary of Results for Probabilistic Classification

Table 6.3 gives a condensed presentation of the results obtained in this chapter. The summary table shows overall performance across all sub-flow times. System performance specific to flow duration cutoff of the hierarchical model has been discussed in the chapter. As noted from the DDoS_WED21ST data (DDoS HOIC attack), the results depend on the attack type and it is important that the system is tuned as new attacks become known.

Table 6.3: Efficiency of Hierarchical Solution with Probabilistic Method

Attack Type	**Approximate % of Traffic to Edge**	**Approximate Increase in Prediction Rate**	**Approximate Change in Overall Acuraccy**
BRUTE_WED14TH	<1	1000	Marginal increase
DOS_FRI16TH	**<2**	100-1000	Marginal decrease
DDOS_TUE20TH	<10	10	Insignificant
DDOS_WED21ST	15-26	<10	Insignificant
WEB_THU22ND	<1	1000	Insignificant
WEB_FRI23RD	<1	1000	Insignificant
INFILTRATION_WED28TH	<1	1000	Insignificant
BOT_FRI2ND	<1.2	1000	Insignificant

In addition, some results show a marginal increase in detection accuracy when implementing the hierarchical solution. This is rather unexpected given that the second stage classifier processes significantly fewer number of packets which means less information available to the machine learning algorithm. One result which shows this marginal increase is the Brute-Force attack as seen in the table. For big data, high volume communications characterised by modern network applications and services, these marginal difference may be significant. This shows potential for improvement with further investigation and research into the hierarchical SDN security solution. One result which shows relative decrease in detection accuracy is the DoS attack shown in Figure 6.4 and also seen in Table 6.3. In reality, this is to be expected since the second stage classifier only deals with a significantly fewer number of packets for traffic classification. However, as explained in Section 6.4, the categorical classification done in Chapter 5 indicates an increase in accuracy for DoS attack. It is important to note that in Chapter 5, there is consistent improvement in accuracy from the non-hierarchical implementation to the hierarchical implementation. Whereas, in the probabilistic classification carried out in this chapter, there has been no significant change in accuracy except for Bruet-Force and DoS which showed marginal increase and decrease

respectively.

6.6 Chapter Summary

The main contribution of this chapter is to demonstrate the improvement in efficiency between traditional non-hierarchical intrusion detection methods and the hierarchical SDN solution. The efficiency has been demonstrated in terms of attack prediction rate and overall attack detection accuracy.

The results in this chapter show an overall reduction in traffic to the edge, when implementing the hierarchical solution, which results in increased prediction rate at the edge. This indicates that there is a significant reduction in effort at the edge of the network. Despite the significant reduction in effort, the hierarchical solution gives similar detection accuracy as the traditional non-hierarchical approach. In this regard, the results emphasise the difference in accuracy and not necessarily the level of accuracy. Work done and results obtained in this chapter demonstrate a significant reduction in time and effort required to obtain reasonable attack detection in SDN. Without the hierarchical solution, every packet will be processed by the intrusion detection system (IDS). With the hierarchical solution, only a portion of the packets are processed at the edge. Also, implementing only the first stage is not sufficient in terms of overall accuracy.

Chapter 7

Conclusion and Analysis

7.1 Analysis and Limitations

The goal of the work done in this thesis is aimed at answering the research question as stated in Chapter 1: given the computationally intensive nature of machine learning based IDS, can we leverage on the SDN architecture to provide an efficient malicious packet detection solution at the edge of the network? Chapter 4 provided supporting evidence that a hierarchical design might provide a solution although it only focused on what could be achieved in a single classifier. In particular Figure 4.36 showed that there is potential for reducing the traffic volume at an edge classifier, although the overall system had not yet been investigated. Consequently, in Chapter 5, the operation of the proposed hierarchical solution is first presented. Table 5.16 showed a summary of the results. Given a variety of different evaluation metrics it is seen that the hierarchical solution shows good potential for DoS and DDoS attacks. Other attacks show good performance in some metrics but not in some others. For example, the infiltration attack shows good flow duration cutoff percent and attack flows completed, but returns a poor detection cutoff. Chapter 6 demonstrated more concrete evidence regarding the savings in traffic volume processed at the edge. Table 6.3 showed the hierarchical solution demonstrates a reduction in traffic to the edge by approximately 74-90% for DDoS attacks while the other attacks show a reduction of over 98% of traffic.

Generally, both SDN security and machine learning based IDS are still considered to be growing areas of research. There are lots of challenges with integrating machine learning into network security. However, we believe that SDN provides a unique platform to introduce machine learning into network security. This work has demonstrated this conviction using a novel approach: a

lightweight machine learning classifier at the controller combined with an edge classifier that would have to deal with much reduced intrusion detection activities. This potentially frees up resources on the server for other network services and applications. This is because in practice, the edge classifier is most likely to be a server blade which may incorporate several other network based services as is common with network function virtualisation at the edge [133] [134].

Work done in this thesis represents significant improvement compared to previous works in machine-learning based network security. Previous works as discussed in Sections 2.10 and 5.2 have not considered the sub-flow approach to machine learning based traffic classification with respect to network security. For example, works such as [1] [99] [92] [97] [14] [98] [127] [128] [129] [100] [130] [131] and many more have investigated machine learning based network security but they all share this limitation. Most of these works have utilised popular datasets like the KDD99 and the CICIDS2017 datasets but none of these works have re-engineered the datasets in order to utilise a sub-flow approach. Work done in this thesis has utilised a sub-flow approach which is an improvement to previous works and is a better reflection of realistic scenarios. For example, if the whole flow is considered, as is the case with all works seen by this author, then there is no opportunity for detecting the attack before it is finished; this is vital if detection is to be used for protection. This work has investigated the likely detection time for the range of attacks, for some it can be detected before the flow is complete, for example DDoS LOIC can be successfully detected within typically 3% of the flows completed, whereas this increases to 68% for Brute Force attacks if an aggressive flow-cut off criteria is used (see Table 5.16). Alternatively, very good results are achieved if the flows are allowed to fully complete i.e., without using a sub-flow cut-off for detection as implicitly assumed by previous works. In addition, other works do not consider the processor intensive nature of machine learning methods and their proposed solutions would require the detection models to process every packet or flow in the dataset. However, work done in this thesis has addressed this limitation and has demonstrated a more efficient approach to machine learning based network security.

In a practical deployment of this solution, it is believed that the detection accuracy at the controller (first stage) would be poorer than the results suggest due to the fact that the traffic and attacks would have greater variation than in the dataset that was used. This makes the intervention of the edge classifier more critical. This work has demonstrated that working in tandem with the central classifier (1st stage classifier), the edge classifier can be made to work more efficiently when compared to a flat traditional architecture.

As expected, there are limitations to this work due to constraints in experimental procedures. Some of the limitations include the need to optimise the detection probability threshold and other model parameters. The fact that this has to be done with respect to different attack types and for each sub-flow point would require a more extensive engineering process for practical deployment than is available within this study. In practice also, the feature selection process at the edge would have to be optimised to suit the particular traffic and attacks. In addition, the hierarchical solution which is based on a supervised learning approach would have the limitation of its inability to reliably detect unknown attacks. In addition, the quality of the dataset contributes to the experimental limitation of this work. In reality, very high quality (maybe real world) IDS datasets are very difficult to obtain. Also, the re-engineering process carried out on the dataset in this work is bound to create some slight modifications to the original data. In a real deployment, a combination of different detection mechanisms maybe required to effectively deal with the complex nature of cyber attacks in modern networks. Consequently, the limitations of this work as mentioned, are being suggested as items of future work.

7.2 Future Work

In consequent of the limitations discussed in Section 7.1, the following areas are suggested as possible future work:

- Procedures and understanding from this work can be applied to hybrid intrusion detection systems since hybrid detection systems combine advantages of both methods. For example by combining the supervised techniques with an unsupervised approach with the latter working to detect anomalies that are potential "uknownn" attacks. Additionally, the approach could be combined with existing rule based approaches [135] [136]. As with these papers, these techniques could be implemented in a similar hierarchical approach.

- Investigate the overall network performance of the hierarchical solution as regards other areas outside of security. For example exploring latency, throughput, quality of service and any other parameters that may be affected by the addition of additional processing at the edge.

- This work has considered a generic enterprise network. There would be differences if it was deployed in other types of network. For example, an IoT network might have very little processing at the edge such that it might not be feasible to carry out much processing, but

on the other hand the data rate is also lower at the edge in such a network. It would be interesting to look at optimising the design of the hierarchy to suit such a network.

- As stated in Chapter 1, this work does not investigate machine learning iteself, but rather the application of machine learning. Although a number of models where explored in Chapter 4, an important body of work would be to look at optimising the machine learning itself, for example by exploring further models and optimising the parameters of the models that were used in this work. An interesting approach would be to make use of *federated learning* [137] which is a recent approach to distributed machine learning. Indeed this thesis can be classified as using federating learning. However, it would be useful to go further than suggested in this thesis by jointly optimising the machine learning based upon both edge and central classifiers. The thesis optimises the edge classifier based upon the output of the central classifier, but maybe the central classifier could be optimised based upon the edge classifier and achieve truly federated learning? Furthermore, coordinated attacks across a number of edge points could be detected through a combination of edge intelligence and using correlation approaches in a central classifier.

- Investigate the regular update of detection models to suit the changing nature of the attacks that approach a given network. This may require techniques such as reinforcement learning and using human input and general network monitoring to prompt such systems to retrain.

- Investigate the impact or adaptability of this work towards the recently developed networking concept of intelligent multi-edge computing (or intelligent edge environment) [138] [139]. This is the idea that services are going to move to the edge of the network by default. It should be noted that this concept does not invalidate work done in this thesis. As a matter of fact, this concept of multi edge computing makes the work done in this thesis even more relevant. This is because moving more services to the edge places more demand on edge resources, hence an efficient edge intrusion detection system as proposed in this work becomes more relevant. Hence, as future work, the solution presented in this thesis will look to be integrated in the incoming intelligent edge environment.

- There is a possibility that some of the concepts introduced in this work can be built into the SDN architecture in the future (or influences the next generation of SDN). This is more so because SDN is still very much in the infant stages of its development and it is believed that there will be significant developments and improvements in SDN technology with increasing

research over time. For example, it may be possible to build hardware that can be better suitable to this solution or possibly come up with additional fields in the SDN flow table to assist in the implementation of this solution.

- Also as future work, analytical models to look at different attacks based upon this work can be developed.

In conclusion this work has introduced a novel hierarchical machine learning approach that can be optimised across different sub-flow durations and drastically reduce the processing required at an edge classifier. This is important to bring realistic machine learning based intrusion detection to practical networks.

Appendices

Appendix A

Feature Selection Using Mean Decrease Accuracy

Table A.1: First 6 Features for BOTNET

Mean Decrease Accuracy		
BOTNET		
	Feature	**Importance**
1	Flow Duration	1.8635
2	Bwd IAT Std	1.6302
3	Bwd IAT Min	0.6158
4	Fwd Packets/s	0.46
5	Init_Win_bytes_forward	0.3783
6	Init_Win_bytes_backward	0.2791
	accuracy = 0.9718	

Table A.2: First 6 Features for Portscan

Mean Decrease Accuracy		
	PORTSCAN	
	Feature	**Importance**
1	Flow Bytes/s	0.2839
2	Total Length of Fwd Packets	0.1788
3	Fwd Packet Length Max	0.1721
4	ACK Flag Count	0.1652
5	Subflow Fwd Bytes	0.1472
6	Packet Length Mean	0.1043
	accuracy = 0.9998	

Table A.3: First 6 Features for Patator

	Mean Decrease Accuracy	
	PATATOR	
	Feature	**Importance**
1	Bwd Header Length	1.4608
2	Max Packet Length	0.5846
3	Fwd IAT Min	0.2784
4	Init_Win_bytes_forward	0.2645
5	Packet Length Std	0.1987
6	Packet Length Variance	0.1985
	accuracy = 0.9967	

Appendix B

Procedures for IP Packet Processing

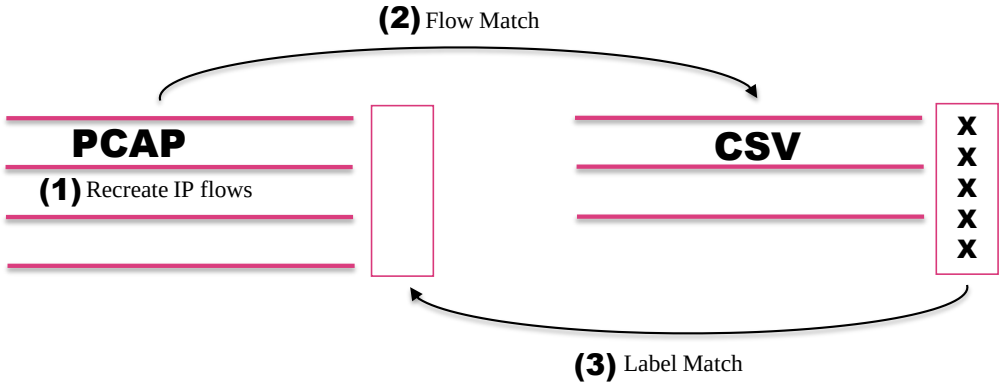


Figure B.1: Reverse Engineering of IP Packets

saddr:192.168.10.50, daddr:192.168.10.3, pfwd:20 prev:10 bfwd:5070 brev:1240 mbfwd:253 mbrev:124 mflen:455 fiatt:64.569859 fiats:0.000000 biatt:64.569515 biats:8.026636 biatmin:0.000003 fldur:129.139374 inwinf:7540 inwinb:2078 tpkts:30 tbytes:6310

saddr:192.168.10.50, daddr:192.168.10.3, pfwd:20 prev:10 bfwd:5070 brev:3680 mbfwd:253 mbrev:368 mflen:455 fiatt:64.569694 fiats:0.000000 biatt:64.569386 biats:8.026619 biatmin:0.000004 fldur:129.139080 inwinf:20238 inwinb:2078 tpkts:30 tbytes:8750

Figure B.2: Sample Flow Statistics from PCAP

Appendix C

List of Re-Engineered Flow Features

s/n	Flow Feature	Description
1	Flow ID ('FlowID')	A five-tuple flow identifier comprising of source and destination IP, source and destination port numbers, and protocol
2	Source IP Address ('SourceIP')	Source IP address of the flow
3	Destination IP Address ('DestinationIP')	Destination IP address of the flow
4	Source Port Number ('SrcPort')	Source port number of the flow
5	Destination Port Number ('DstPort')	Destination port number of the flow
6	Time Stamp ('Timestamp')	The instant in time the first packet of flow was captured
7	Total Forward Packet ('FwdPkts')	Total number of packets in the forward flow direction
8	Total Backward Packet ('BwdPkts')	Total number of packets in the backward flow direction
9	Flow Duration ('FlowDur')	Duration of the flow in seconds
10	Forward Packet Length Mean ('FwdPktLenMean')	Mean packet size (in bytes) in forward direction
11	Backward Packet Length Mean ('BwdPktLenMean')	Mean packet size (in bytes) in backward direction
12	Forward Packet Length Standard ('FwdPktLenStd')	Standard deviation of packet size in the forward direction
13	Backward Packet Length Standard ('BwdPktLenStd')	Standard deviation of packet size in the backward direction
14	Forward Packet Length Minimum ('FwdPktLenMin')	Minimum packet size (in bytes) in the forward direction
15	Backward Packet Length Minimum ('BwdPktLenMin')	Minimum packet size (in bytes) in the backward direction
16	Forward Packet Length Maximum ('FwdPktLenMax')	Maximum packet size (in bytes) in the forward direction
17	Backward Packet Length Maximum ('BwdPktLenMax')	Maximum packet size (in bytes) in the backward direction
18	Forward Packet Length Total 'FwdPktLenTot',	Total packet size (in bytes) in the forward direction
19	Backward Packet Length Total ('BwdPktLenTot')	Total packet size (in bytes) in the backward direction
20	Forward Inter Arrival Time Mean('FwdIATMean')	Mean time between two packets in forward direction of flow
21	Backward Inter Arrival Time Mean ('BwdIATMean')	Mean time between two packets in backward direction of flow
22	Forward Inter Arrival Time Standard ('FwdIATStd')	Standard deviation of time between two packets in forward direction of flow
23	Backward Inter Arrival Time Standard ('BwdIATStd')	Standard deviation of time between two packets in backward direction of flow
24	Forward Inter Arrival Time Minimum ('FwdIATMin')	Minimum time between two packets in forward direction of flow
25	Backward Inter Arrival Time Minimum ('BwdIATMin')	Minimum time between two packets in backward direction of flow
26	Forward Inter Arrival Time Maximum ('FwdIATMax')	Maximum time between two packets in forward direction of flow
27	Backward Inter Arrival Time Maximum ('BwdIATMax')	Maximum time between two packets in backward direction of flow
28	Forward Inter Arrival Time Total ('FwdIATTot')	Total time between two packets in forward direction of flow
29	Backward Inter Arrival Time Total ('BwdIATTot')	Total time between two packets in backward direction of flow
30	Flow Bytes Per second ('FlowBytes/s')	Number of bytes per second per flow
31	Flow Packets Per Second ('FlowPkts/s')	Number of packets per second per flow
32	Forward Initial Window Byte ('FwdInitWin')	Initial window byte size in the forward direction
33	Backward Initial Window Byte ('BwdInitWin')	Initial window byte size in the backward direction
34	Forward Packets Per Second ('FwdPkts/s')	Number of packets per second in the forward direction
35	Backward Packets Per Second ('BwdPkts/s')	Number of packets per second in the backward direction
36	Packet Length Minimum ('PktLenMin')	Minimum packet size (in bytes) in flow
37	Packet Length Maximum ('PktLenMax')	Maximum packet size (in bytes) in flow
38	Packet Length Mean ('PktLenMean')	Mean packet size (in bytes) in flow
39	Packet Length Variance ('PktLenVar')	Packet length variance in flow
40	Packet Length Total ('PktLenTot')	Total packet length (in bytes) in flow
41	Packet Length Standard ('PktLenStd')	Standard deviation of packet lentgth in flow
42	Flow Inter Arrival Time Mean ('FlowIatMean')	Mean time between two packets in flow
43	Flow Inter Arrival Time Variance ('FlowIatVar')	Variance time between two packets in flow
44	Flow Inter Arrival Time Standard ('FlowIatStd')	Standard deviation time between two packets in a flow
45	Flow Inter Arrival Time Minimum ('FlowIatMin')	Minimum time between two packets in a flow
46	Flow Inter Arrival Time Maximum ('FlowIatMax')	Maximum time between two packets in a flow
47	Label ('Label')	The class or state of the flow either benign or malicious

Figure C.1: CICIDS2017 Dataset Re-Engineered Traffic Flow Features

Appendix D

Sanity Check Results for Re-engineered Data

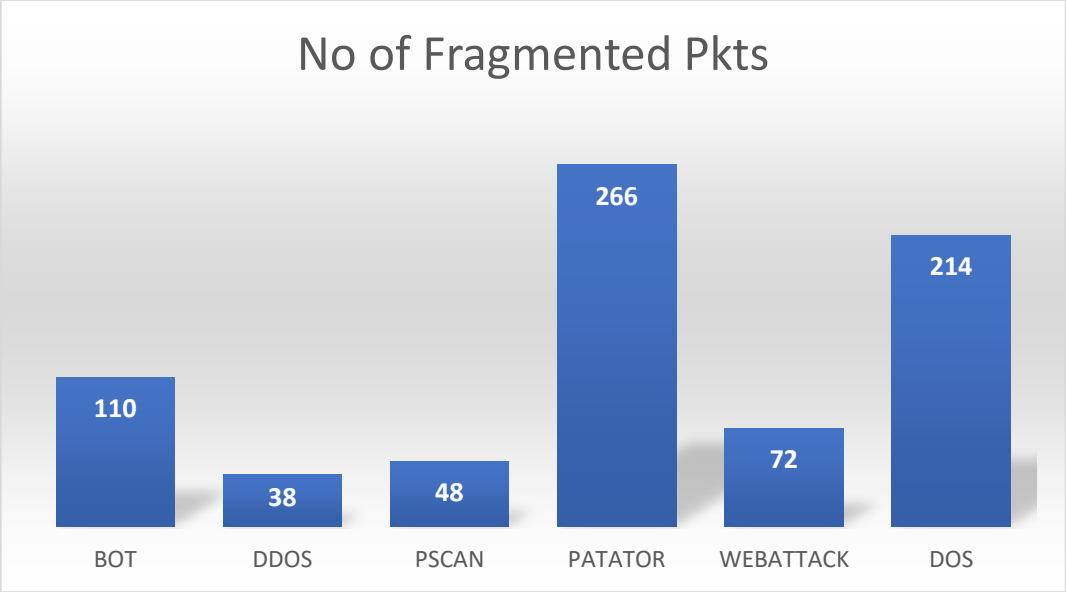


Figure D.1: Fragmented Packets in CICIDS2017 Dataset

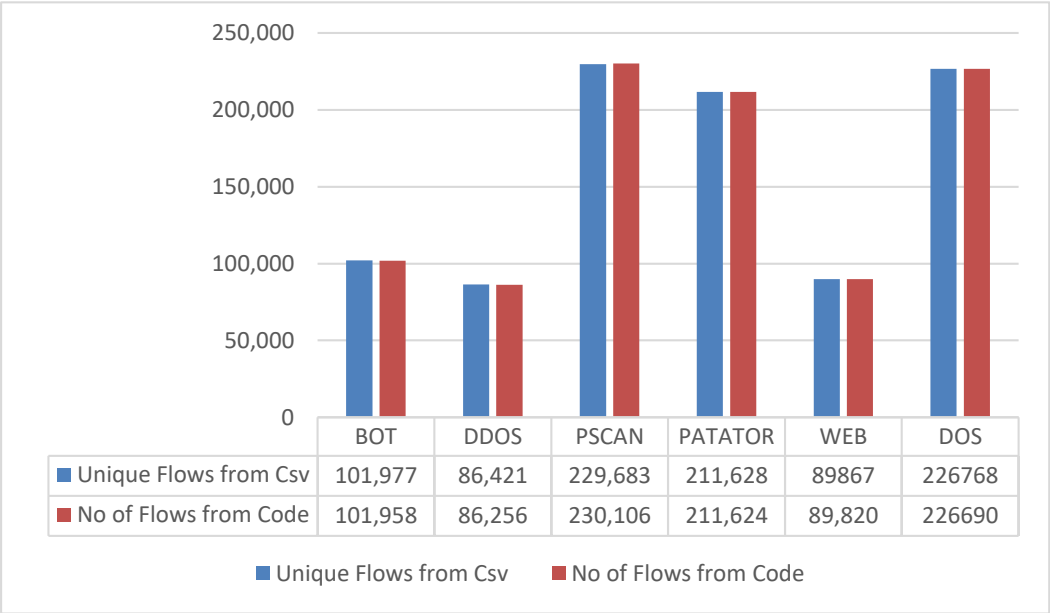


Figure D.2: Unique Flows from Csv vs Total Flows from Code

Appendix E

ECDC Curves for Re-Engineered Data 2017

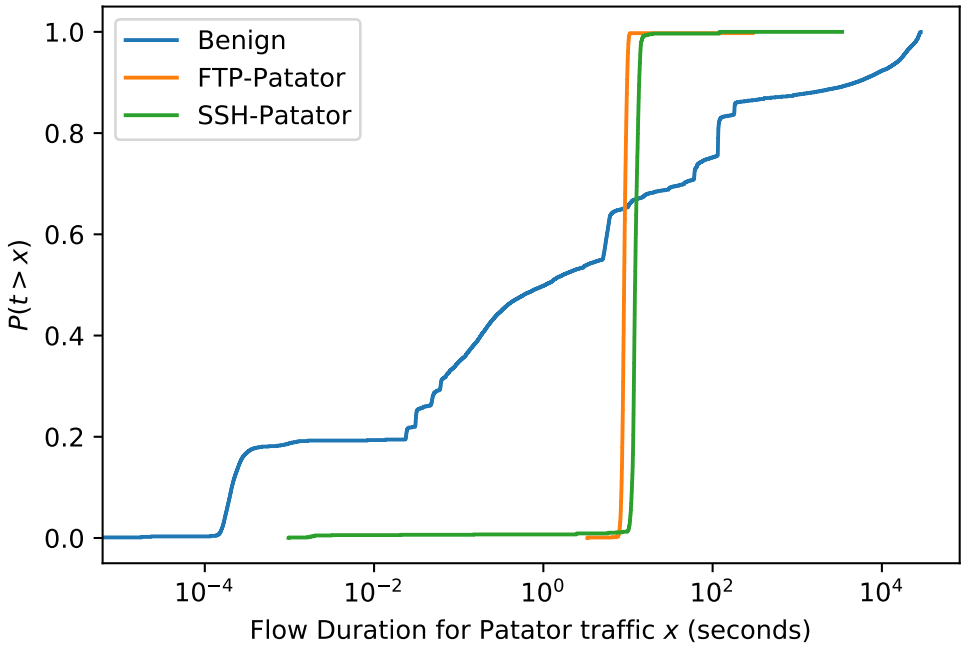


Figure E.1: eCDC Curve for Patator Traffic

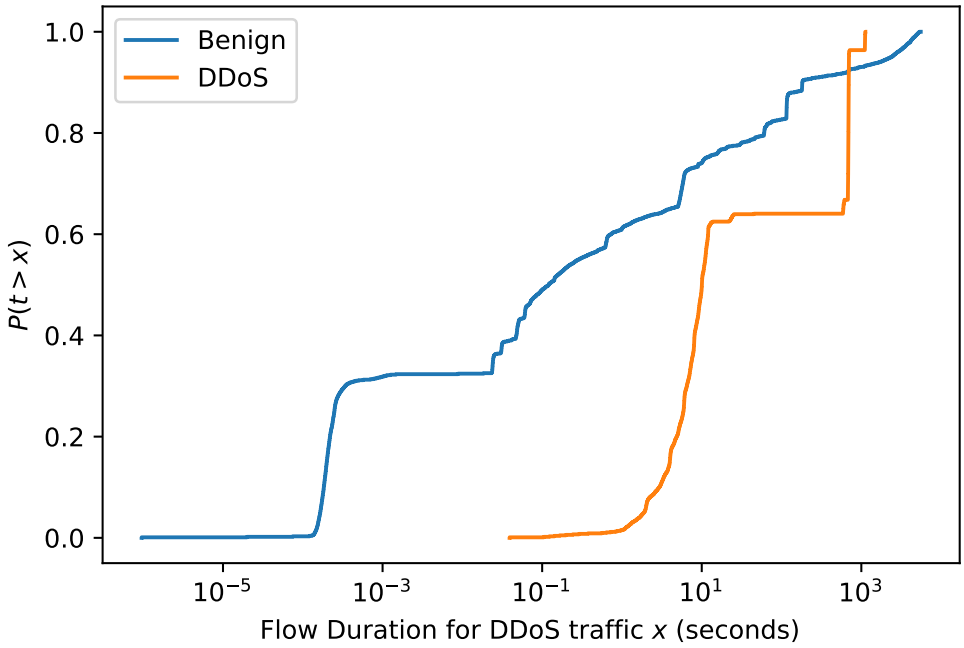


Figure E.2: eCDC Curve for Ddos Traffic

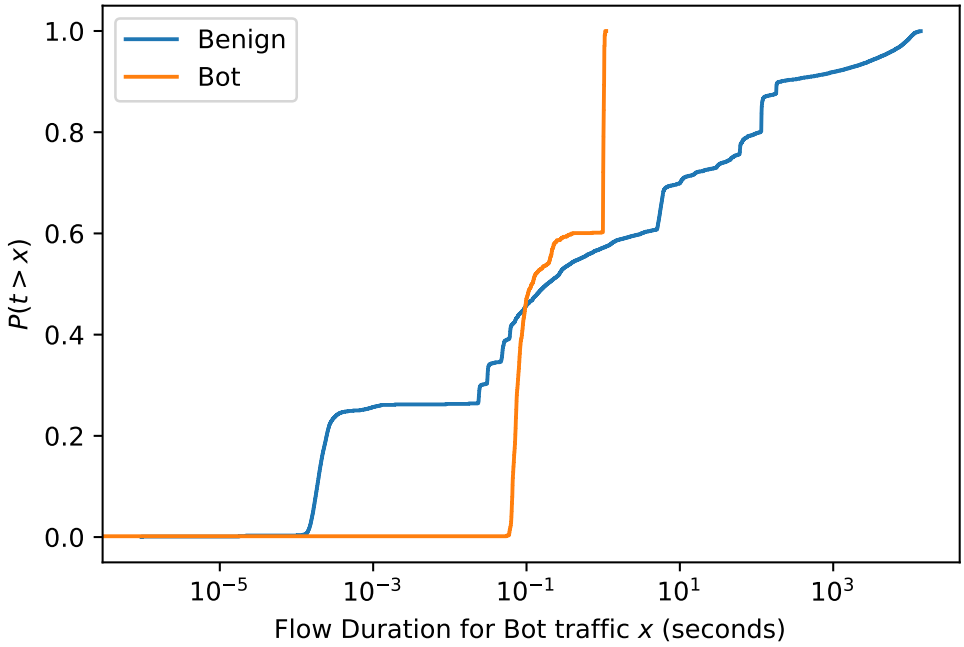


Figure E.3: eCDC Curve for Bot Traffic

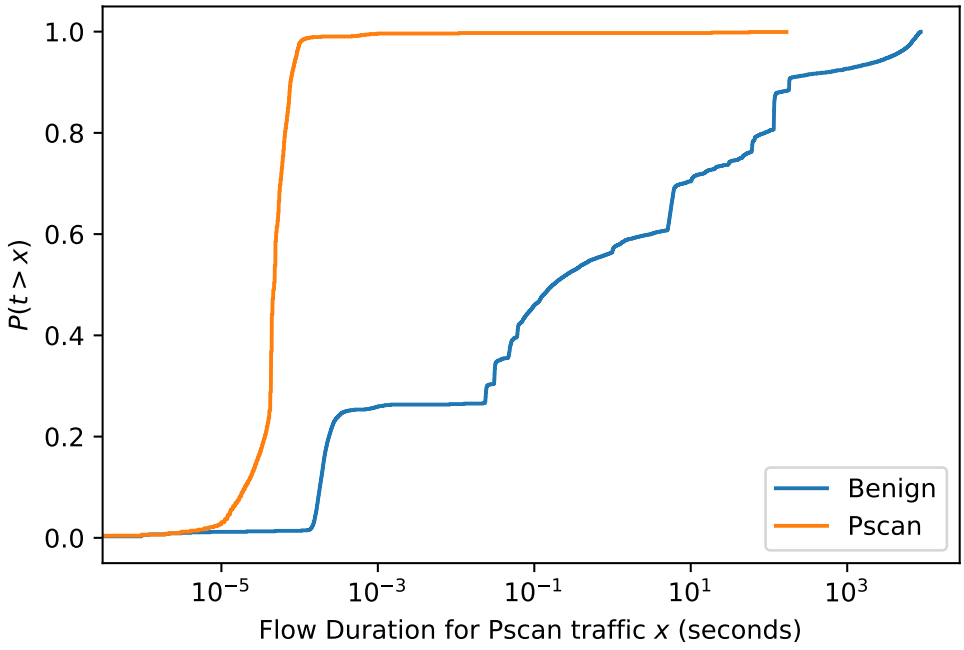


Figure E.4: eCDC Curve for Pscan Traffic

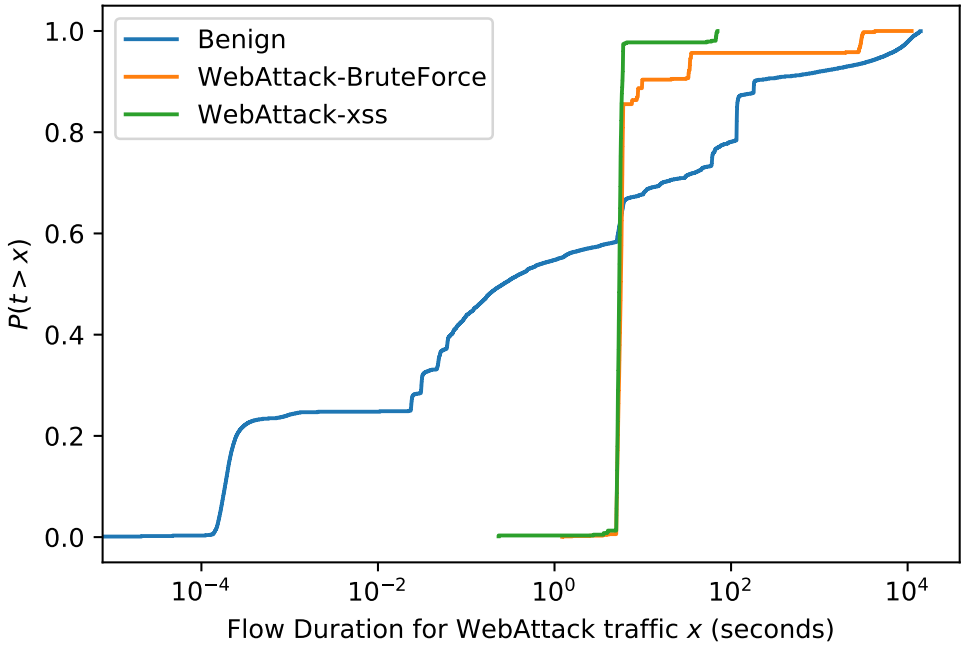


Figure E.5: eCDC Curve for WebAttack Traffic

Appendix F

Machine Learning with Scikit Learn

The intrusion detection module of this work is based on a Machine Learning model. The Machine Learning procedures of this work is implemented using the Scikit Learn library in Python. The problem is a classification problem. Figure F.1 gives a sample output of the Machine Learning process in Scikit Learn illustrating the relevant classification metrics as used in this work. Figure F.2 gives further variations of the output as used in this work. The train and predict times representing the anomaly detection module execution time are also extracted from this process and used in the analysis provided in subsequent Chapters.

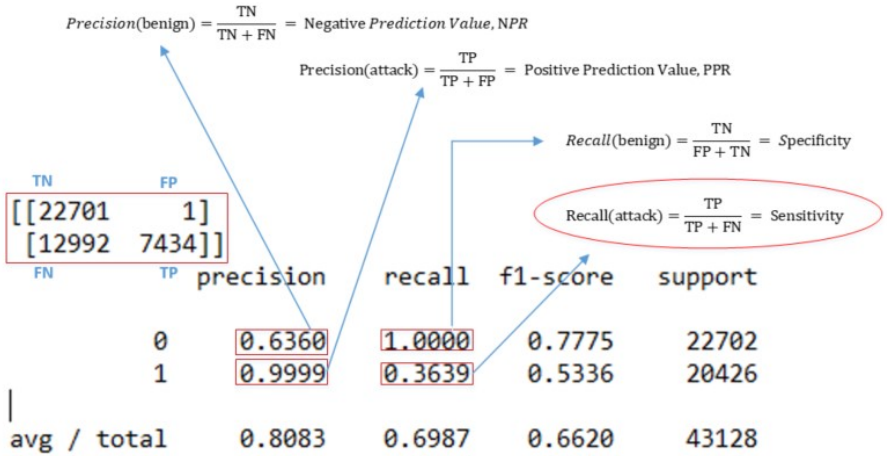


Figure F.1: Machine Learning Output Parameters using Scikit Learn

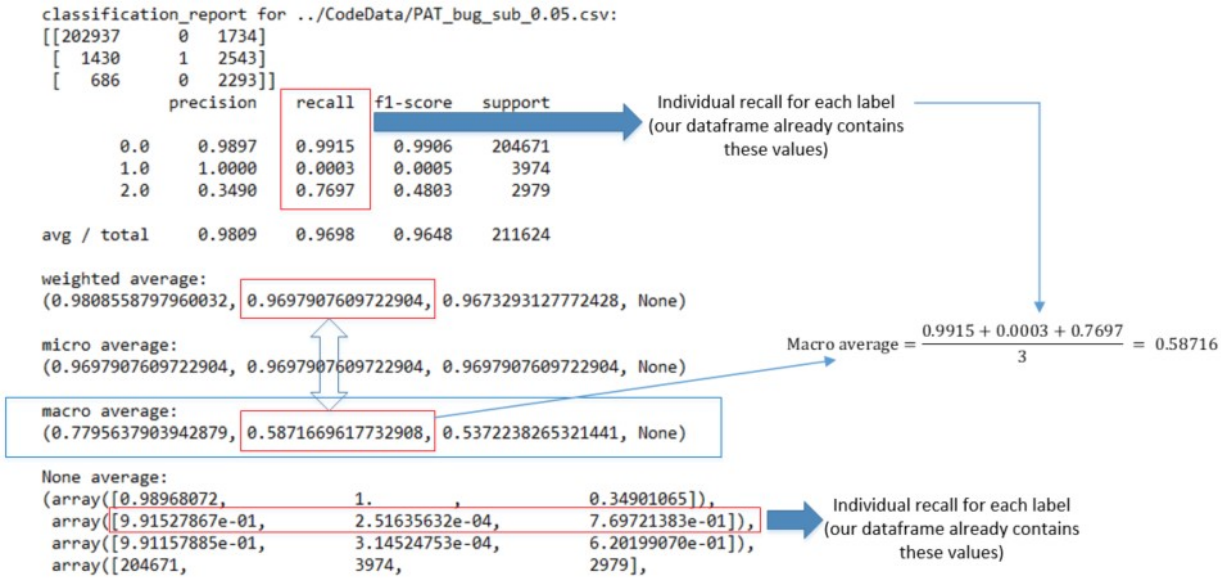


Figure F.2: Extended Machine Learning Output using Scikit Learn

Bibliography

[1] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, no. Cic, 2018, pp. 108–116.

[2] M. Rahouti, K. Xiong, Y. Xin, S. K. Jagatheesaperumal, M. Ayyash, and M. Shaheed, "Sdn security review: Threat taxonomy, implications, and open challenges," *IEEE Access*, vol. 10, pp. 45 820–45 854, 2022.

[3] P. D. Kreutz, M.V. Ramos, "Software-defined networking : A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, 2015.

[4] ONF, "Openflow switch specification," https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, March 2015.

[5] A. Markoborodov, Y. Skobtsova, and D. Volkanov, "Representation of the openflow switch flow table," in *2020 International Scientific and Technical Conference Modern Computer Network Technologies (MoNeTeC)*, 2020, pp. 1–7.

[6] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 686–728, 2019.

[7] S. Prathibha, J. Bino, M. T. Ahammed, C. Das, S. R. Oion, S. Ghosh, and M. Afroj, "Detection methods for software defined networking intrusions (sdn)," in *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2022, pp. 1–6.

[8] M. U. Farooq, M. Rashid, F. Azam, Y. Rasheed, M. W. Anwar, and Z. Shahid, "A model-driven framework for the prevention of dos attacks in software defined networking (sdn)," in *2021 IEEE International Systems Conference (SysCon)*, 2021, pp. 1–7.

[9] B. Daneshmand and T. A. Le, "Software-defined networking: A new approach to fifth generation networks – security issues and challenges ahead," in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2022, pp. 307–313.

[10] L. Fawcett, S. Scott-Hayward, M. Broadbent, A. Wright, and N. Race, "Tennison: A distributed SDN framework for scalable network security," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2805–2818, 2018.

[11] L. Gil, "Stop layer 7 DDoS attacks with multilayered security," https://blogs.oracle.com/cloud-infrastructure/post/stop-layer-7-ddos-attacks-with-multilayered-security, March 2019.

[12] M. Wang, W. Fu, X. He, S. Hao, and X. Wu, "A survey on large-scale machine learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.

[13] E. Dinc, M. Vondra, and C. Cavdar, "Total cost of ownership optimization for direct air-to-ground communication networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10 157–10 172, 2021.

[14] S. Gangadhar and J. P. Sterbenz, "Machine learning aided traffic tolerance to improve resilience for software defined networks," *Proceedings of 2017 9th International Workshop on Resilient Networks Design and Modeling, RNDM 2017*, pp. 1–7, 2017.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] H. Suryotrisongko and Y. Musashi, "Review of cybersecurity research topics, taxonomy and challenges: Interdisciplinary perspective," in *2019 IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, 2019, pp. 162–167.

[17] O. Durmus, A. Varol, and N. Varol, "Infrastructure requirements for cybersecurity," in *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, 2019, pp. 1–5.

[18] S. S. Tirumala, M. R. Valluri, and G. Babu, "A survey on cybersecurity awareness concerns, practices and conceptual measures," in *2019 International Conference on Computer Communication and Informatics (ICCCI)*, 2019, pp. 1–6.

[19] "Weekly threat report 1st april 2022," https://www.ncsc.gov.uk/report/weekly-threat-report-1st-april-2022#section_1, accessed: 2022-05-30.

[20] "Cisco got hit... and immediately took control of the story," https://www.databreaches.net/cisco-got-hit-and-immediately-took-control-of-the-story/, accessed: 2022-05-30.

[21] "Cisco event response: Corporate network security incident," https://tools.cisco.com/security/center/resources/corp_network_security_incident, accessed: 2022-05-30.

[22] "Finland's parliament hit with cyberattack following us move to admit the country to nato," https://thehill.com/policy/technology/3595917-finlands-parliament-hit-with-cyberattack-following-us-move-to-admit-the-country-to-nato/, accessed: 2022-08-20.

[23] Cisco, "Enterprise networking, security, and automation v7.0," https://contenthub.netacad.com/ensa?lng=en, Feb. 2020.

[24] W. Zhang, Q. Yang, and Y. Geng, "A survey of anomaly detection methods in networks," in *2009 International Symposium on Computer Network and Multimedia Technology*, 2009, pp. 1–3.

[25] Y. K. Vani and Krishnamurthy, "Survey anomaly detection in network using big data analytics," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017, pp. 3366–3369.

[26] M. FRUSTACI, P. PACE, G. ALOI, and G. FORTINO, "Evaluating critical security issues of the IoT world: Present and future challenges," *IEEE Internet of Things Journal*, vol. 4662, no. c, 2017.

[27] A. A. Baybulatov and V. G. Promyslov, "Cybersecurity assessment using delay from backlog bound calculation," in *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*, 2020, pp. 1–6.

[28] S. V. Thakare and D. V. Gore, "Comparative study of CIA and revised-CIA algorithm," in *2014 Fourth International Conference on Communication Systems and Network Technologies*, 2014, pp. 713–718.

[29] A. Punia, D. Gupta, and S. Jaiswal, "A perspective on available security techniques in iot," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2017, pp. 1553–1559.

[30] J. Luxemburk, K. Hynek, and T. Čejka, "Detection of https brute-force attacks with packet-level feature set," in *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*, 2021, pp. 0114–0122.

[31] M. M. Raikar and S. M. Meena, "Ssh brute force attack mitigation in internet of things (IoT) network : An edge device security measure," in *2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC)*, 2021, pp. 72–77.

[32] A. S. Hussainy, M. A. Khalifa, A. Elsayed, A. Hussien, and M. A. Razek, "Deep learning toward preventing web attacks," in *2022 5th International Conference on Computing and Informatics (ICCI)*, 2022, pp. 280–285.

[33] D. M. Brandão Lent, M. P. Novaes, L. F. Carvalho, J. Lloret, J. J. P. C. Rodrigues, and M. L. Proença, "A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks," *IEEE Access*, vol. 10, pp. 73 229–73 242, 2022.

[34] M. Korczynski, L. Janowski, and A. Duda, "An accurate sampling scheme for detecting SYN flooding attacks and portscans," in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–5.

[35] R. Rivest, "The MD5 Message-Digest Algorithm," Internet Requests for Comments, MIT Laboratory for Computer Science and RSA Data Security, Inc, RFC 1654, April 1992. [Online]. Available: https://www.ietf.org/rfc/rfc1321.txt

[36] D. Eastlake and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," Internet Requests for Comments, Internet Engineering Task Force (IETF), RFC 1654, April 1992. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6234

[37] M. Firdhous and N. A. Hussien, "Data security implementations in cloud computing: A critical review," in *2018 3rd International Conference on Information Technology Research (ICITR)*, 2018, pp. 1–5.

[38] M. Ren, Y. Tian, S. Kong, D. Zhou, and D. Li, "An detection algorithm for ARP man-in-the-middle attack based on data packet forwarding behavior characteristics," in *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 1599–1604.

[39] A. Borkar, A. Donode, and A. Kumari, "A survey on intrusion detection system (ids) and internal intrusion detection and protection system (IIDPS)," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*, 2017, pp. 949–953.

[40] A. Sharma and A. Bhasin, "Critical investigation of denial of service and distributed denial of service models and tools," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018, pp. 546–550.

[41] V. Zlomislić, K. Fertalj, and V. Sruk, "Denial of service attacks: An overview," in *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, June 2014, pp. 1–6.

[42] S. Alzahrani and L. Hong, "Detection of distributed denial of service (DDoS) attacks using artificial intelligence on cloud," in *2018 IEEE World Congress on Services (SERVICES)*, July 2018, pp. 35–36.

[43] F. Hussain, S. G. Abbas, I. M. Pires, S. Tanveer, U. U. Fayyaz, N. M. Garcia, G. A. Shah, and F. Shahzad, "A two-fold machine learning approach to prevent and detect IoT botnet attacks," *IEEE Access*, vol. 9, pp. 163 412–163 430, 2021.

[44] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2242–2270, Fourthquarter 2015.

[45] A. Aljuhani, "Machine learning approaches for combating distributed denial of service attacks in modern networking environments," *IEEE Access*, vol. 9, pp. 42 236–42 264, 2021.

[46] R. Vishwakarma and A. K. Jain, "A honeypot with machine learning based detection framework for defending IoT based botnet DDoS attacks," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 1019–1024.

[47] Y. Soupionis and T. Benoist, "Cyber-physical testbed — the impact of cyber attacks and the human factor," in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 326–331.

[48] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912860700062X

[49] S. Niksefat, P. Kaghazgaran, and B. Sadeghiyan, "Privacy issues in intrusion detection systems: A taxonomy, survey and future directions," *Computer Science Review*, vol. 25, pp. 69–78, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013716301204

[50] T. Girdler and V. G. Vassilakis, "Implementing an intrusion detection and prevention system using software-defined networking: Defending against ARP spoofing attacks and blacklisted mac addresses," *Computers & Electrical Engineering*, vol. 90, p. 106990, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621000203

[51] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 393–430, 2019.

[52] M. Nadeem, A. Arshad, S. Riaz, S. S. Band, and A. Mosavi, "Intercept the cloud network from brute force and DDoS attacks via intrusion detection and prevention system," *IEEE Access*, vol. 9, pp. 152 300–152 309, 2021.

[53] O. M. Surakhi, A. M. García, M. Jamoos, and M. Y. Alkhanafseh, "A comprehensive survey for machine learning and deep learning applications for detecting intrusion detection," in *2021 22nd International Arab Conference on Information Technology (ACIT)*, 2021, pp. 1–13.

[54] O. Lifandali and N. Abghour, "Deep learning methods applied to intrusion detection: Survey, taxonomy and challenges," in *2021 International Conference on Decision Aid Sciences and Application (DASA)*, 2021, pp. 1035–1044.

[55] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home IoT using openflow," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 2016, pp. 147–156.

[56] D. Kwon, H. Kim, J. Kim, S. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Computing*, vol. 22, pp. 949–961, 2017.

[57] J. Ashraf and S. Latif, "Handling intrusion and DDoS attacks in software defined networks using machine learning techniques," in *2014 National Software Engineering Conference*, 2014, pp. 55–60.

[58] M. Ahmed, A. N. Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *J. Netw. Comput. Appl.*, vol. 60, pp. 19–31, 2016.

[59] M. E. Villa-Pérez, M. A. Álvarez Carmona, O. Loyola-González, M. A. Medina-Pérez, J. C. Velazco-Rossell, and K.-K. R. Choo, "Semi-supervised anomaly detection algorithms: A comparative summary and future research directions," *Knowledge-Based Systems*, vol. 218, p. 106878, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705121001416

[60] IETF, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information," Internet Requests for Comments, Internet Engineering Task Force, RFC 1654, September 2013. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7011#page-7

[61] ——, "Information Model for IP Flow Information Export," Internet Requests for Comments, Internet Engineering Task Force, RFC, January 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5102.html#page-78

[62] "Multicast ping protocol," https://datatracker.ietf.org/doc/rfc6450/, accessed: 2021-01-30.

[63] A. B. James Broad, "Traceroute command," https://www.sciencedirect.com/topics/computer-science/traceroute-command, accessed: 2022-02-23.

[64] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014.

[65] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037–2064, Fourthquarter 2014.

[66] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of IP flow-based intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 343–356, Third 2010.

[67] IETF, "Requirements for IP Flow Information Export (IPFIX)," Internet Requests for Comments, Internet Engineering Task Force, RFC, October 2004. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3917.html

[68] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras, "Towards real-time intrusion detection for netflow and IPFIX," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 227–234.

[69] B. Trammell and E. Boschi, "An introduction to ip flow information export (IPFIX)," *IEEE Communications Magazine*, vol. 49, no. 4, pp. 89–95, 2011.

[70] IETF, "Architecture for IP Flow Information Export," Internet Requests for Comments, Internet Engineering Task Force, RFC, March 2009. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5470.html

[71] ——, "Evaluation of Candidate Protocols for IP Flow Information Export (IPFIX)," Internet Requests for Comments, Internet Engineering Task Force, RFC, October 2004. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3955.html

[72] Cisco, "Netflow version 9," https://www.cisco.com/c/en/us/products/ios-nx-os-software/netflow-version-9/index.html, May 2013.

[73] "What's software-defined networking (SDN)? - sdxcentral." [Online]. Available: https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/

[74] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: State of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128614002588

[75] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.

[76] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[77] A. Prakash and R. Priyadarshini, "An intelligent software defined network controller for preventing distributed denial of service attack," *2018 Second International Conference on*

Inventive Communication and Computational Technologies (ICICCT), no. Icicct, pp. 585–589, 2018.

[78] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," *Proceedings - 2017 7th International Conference on Emerging Security Technologies, EST 2017*, pp. 138–143, 2017.

[79] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *Computer Communication Review*, vol. 38, pp. 69–74, 04 2008.

[80] X. You, Y. Feng, and K. Sakurai, "Packet in message based ddos attack detection in SDN network using openflow," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, 2017, pp. 522–528.

[81] ONF, "OpenFlow Switch Specification, version 1.5.1." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[82] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

[83] 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. [Online]. Available: http://ieeexplore.ieee.org/document/7226783/

[84] S. Shi, J. Li, H. Wu, Y. Ren, and J. Zhi, "Efm: An edge-computing-oriented forwarding mechanism for information-centric networks," in *2020 3rd International Conference on Hot Information-Centric Networking (HotICN)*, 2020, pp. 154–159.

[85] SecurityInfoWatch.com, "A brief history of machine learning in cybersecurity," https://www.securityinfowatch.com/cybersecurity/article/21114214/a-brief-history-of-machine-learning-in-cybersecurity, accessed: 2022-05-09.

[86] M. Kiran and A. Chhabra, "Understanding flows in high-speed scientific networks: A netflow data study," *Future Generation Computer Systems*, vol. 94, pp. 72–79, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18302322

[87] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolu-

tion, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, 05 2018.

[88] "Internet assigned numbers authority. IANA," https://www.iana.org/.

[89] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, 2012.

[90] Z.-H. Zhou, *Machine Learning.* Springer Singapore, 2021. [Online]. Available: https://doi.org/10.1007%2F978-981-15-1967-3

[91] M. Kubat, *Ambitions and Goals of Machine Learning.* Cham: Springer International Publishing, 2021, pp. 1–15. [Online]. Available: https://doi.org/10.1007/978-3-030-81935-4_1

[92] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Systems with Applications*, vol. 41, no. 4, Part 2, pp. 1690–1700, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417413006878

[93] R. Vrána and J. Kořenek, "Efficient acceleration of decision tree algorithms for encrypted network traffic analysis," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2021, pp. 115–118.

[94] S. Lakshminarasimman, S. Ruswin, and K. Sundarakantham, "Detecting DDoS attacks using decision tree algorithm," in *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, 2017, pp. 1–6.

[95] B. Prashant, "Random forest classifier tutorial," https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial/notebook, accessed: 2021-10-30.

[96] ——, "A practical guide to implementing a random forest classifier in python," https://towardsdatascience.com/a-practical-guide-to-implementing-a-random-forest-classifier-in-python-979988d8a263, accessed: 2021-11-25.

[97] S. Nanda, F. Zafari, C. Decusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in SDN using machine learning approach," *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2016*, pp. 167–172, 2017.

[98] C. Song, Y. Park, K. Golani, Y. Kim, K. Bhatt, and K. Goswami, "Machine-learning based threat-aware system in software defined networks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1–9.

[99] N. Bakhareva, A. Shukhman, A. Matveev, P. Polezhaev, Y. Ushakov, and L. Legashev, "Attack detection in enterprise networks by machine learning methods," in *2019 International Russian Automation Conference (RusAutoCon)*, 2019, pp. 1–6.

[100] L. Barki, A. Shidling, N. Meti, D. G. Narayan, and M. M. Mulla, "Detection of distributed denial of service attacks in software defined networks," in *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2016, pp. 2576–2581.

[101] R. U. Rasool, U. Ashraf, K. Ahmed, H. Wang, W. Rafique, and Z. Anwar, "Cyberpulse: A machine learning based link flooding attack mitigation system for software defined networks," *IEEE Access*, vol. 7, pp. 34 885–34 899, 2019.

[102] C. Brown, A. Cowperthwaite, A. Hijazi, and A. Somayaji, "Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–7.

[103] J. O. Nehinbe, "A simple method for improving intrusion detections in corporate networks," in *Information Security and Digital Forensics*, D. Weerasinghe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 111–122.

[104] M. Xie and J. Hu, "Evaluating host-based anomaly detection systems: A preliminary analysis of ADFA-LD," in *2013 6th International Congress on Image and Signal Processing (CISP)*, vol. 03, 2013, pp. 1711–1716.

[105] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset benchmark dataset," *Software Networking*, no. July, 2017.

[106] "Intrusion detection evaluation dataset (CIC-IDS2017)," https://www.unb.ca/cic/datasets/ids-2017.html, accessed: 2018-05-30.

[107] "CSE-CIC-IDS2018 on AWS," https://www.unb.ca/cic/datasets/ids-2018.html, accessed: 2020-09-30.

[108] K. S. Bhosale, M. Nenova, and G. Iliev, "The distributed denial of service attacks (DDoS) prevention mechanisms on application layer," in *2017 13th International Conference on*

Advanced Technologies, Systems and Services in Telecommunications (TELSIKS), Oct 2017, pp. 136–139.

[109] M. Vidhya, "Efficient classification of portscan attacks using support vector machine," in *2013 International Conference on Green High Performance Computing (ICGHPC)*, March 2013, pp. 1–5.

[110] P. A. Sonewar and S. D. Thosar, "Detection of SQL injection and XSS attacks in three tier web applications," in *2016 International Conference on Computing Communication Control and automation (ICCUBEA)*, Aug 2016, pp. 1–4.

[111] "A gentle introduction to vector space models," https://machinelearningmastery.com/a-gentle-introduction-to-vector-space-models/, accessed: 2021-04-26.

[112] F. Bornemann, *Eigenvalue Problems*. Cham: Springer International Publishing, 2018, pp. 75–97. [Online]. Available: https://doi.org/10.1007/978-3-319-74222-9_5

[113] "PCAP capture file format," https://tools.ietf.org/id/draft-gharris-opsawg-pcap-00.html, accessed: 2019-09-30.

[114] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[115] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[116] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[117] scikit-learn.org, "sklearn.preprocessing.OrdinalEncode." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html

[118] A. H. Lashkari, "cicflowmeter."

[119] Canadian Institute for Cybersecurity, "Applications — research — canadian institute for cybersecurity — UNB." [Online]. Available: https://www.unb.ca/cic/research/applications.html

[120] N. Spolaˆ, E. Alvares, M. Carolina, and H. Diana, "A comparison of multi-label feature selection methods using the problem a comparison of multi-label feature selection methods using the problem transformation approach," *Electronic Notes in Theoretical Computer Science*, vol. 292, no. May 2014, pp. 135–151, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2013.02.010

[121] S. Sheen and R. Rajesh, "Network intrusion detection using feature selection and decision tree classifier," in *TENCON 2008 - 2008 IEEE Region 10 Conference*, Nov 2008, pp. 1–4.

[122] "Selecting good features – part III: random forests — diving into data." [Online]. Available: https://blog.datadive.net/selecting-good-features-part-iii-random-forests/

[123] H. Han, X. Guo, and H. Yu, "Variable selection using mean decrease accuracy and mean decrease gini based on random forest," *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, pp. 219–224, 2017.

[124] S. Seetharaman, "Openflow/sdn tutorial OFC/NFOEC," in *OFC/NFOEC*, 2012, pp. 1–52.

[125] Z. Cheng, C. Chen, X. Qiu, and H. Xie, "An improved KNN classification algorithm based on sampling," in *Proceedings of the Advances in Materials, Machinery, Electrical Engineering (AMMEE 2017)*. Atlantis Press, 2017/06, pp. 220–225. [Online]. Available: https://doi.org/10.2991/ammee-17.2017.45

[126] A. Luque, M. Mazzoleni, A. Carrasco, and A. Ferramosca, "Visualizing classification results: Confusion star and confusion gear," *IEEE Access*, vol. 10, pp. 1659–1677, 2022.

[127] M. Subramanian, K. S. Vadivel, and S. R, "Performance evaluation of deep learning models in detection of distributed denial of service attacks," in *2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 2021, pp. 652–658.

[128] W. Al-gethami and A. Aljuhani, "Detection of http attacks using machine learning," in *2022 2nd International Conference on Computing and Information Technology (ICCIT)*, 2022, pp. 344–348.

[129] P. Xiao, W. Qu, H. Qi, Y. Xu, and Z. Li, "An efficient elephant flow detection with cost-sensitive in SDN," in *2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, 2015, pp. 24–28.

[130] B. Kumar Joshi, N. Joshi, and M. Chandra Joshi, "Early detection of distributed denial of service attack in era of software-defined network," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*, 2018, pp. 1–3.

[131] K. Sudar, M. Beulah, P. Deepalakshmi, P. Nagaraj, and P. Chinnasamy, "Detection of distributed denial of service attacks in SDN using machine learning techniques," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021, pp. 1–5.

[132] S. Black and Y. Kim, "An overview on detection and prevention of application layer DDoS attacks," in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0791–0800.

[133] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[134] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Network functions virtualization: The long road to commercial deployments," *IEEE Access*, vol. 7, pp. 60 439–60 464, 2019.

[135] S. Jin, Y. Jiang, and J. Peng, "Intrusion detection system enhanced by hierarchical bidirectional fuzzy rule interpolation," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 6–10.

[136] Z. S. Malek, B. Trivedi, and A. Shah, "User behavior pattern -signature based intrusion detection," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2020, pp. 549–552.

[137] R. Yu and P. Li, "Toward resource-efficient federated learning in mobile edge computing," *IEEE Network*, vol. 35, no. 1, pp. 148–155, 2021.

[138] P. Hu and W. Chen, "Software-defined edge computing (sdec): Principles, open system architecture and challenges," in *2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications,*

Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2019, pp. 8–16.

[139] Y. Wang, M. Tang, S. Zhou, G. Tan, Z. Zhang, and J. Zhan, "Performance analysis of heterogeneous mobile edge computing networks with multi-core server," in *2020 IEEE 20th International Conference on Communication Technology (ICCT)*, 2020, pp. 1540–1545.