Intelligent Traffic Classification for Detecting DDoS Attacks using SDN/OpenFlow

by

Jarrod N. Bakker

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Engineering
in Network Engineering.

Victoria University of Wellington
2017

Abstract

Distributed denial of service (DDoS) attacks utilise many attacking entities to prevent legitimate use of a resource via consumption. Detecting these attacks is often difficult when using a traditional networking paradigm as network information and control are not centralised. Software-Defined Networking is a recent paradigm that centralises network control, thus improving the ability to gather network information. Traffic classification techniques can leverage the gathered data to detect DDoS attacks. This thesis utilises *nmeta2*, a SDN-based traffic classification architecture, to study the effectiveness of machine learning methods to detect DDoS attacks. These methods are evaluated on a physical network testbed to demonstrate their application during a DDoS attack scenario.

Acknowledgments

I dedicate this work to my parents, Michael and Carol Bakker. This thesis not only embodies countless hours of research and study, it is symbolic of the encouragement and support that they have provided throughout my life.

I thank my partner Molly MacKenzie. You listened and supported me throughout the highs and lows of this research. You kept me sane, even though it may not have appeared that way.

I thank my supervisor, Professor Winston Seah, for his guidance and wisdom within the domain of engineering research. I also thank Dr Bryan Ng for selflessly offering advice when asked. Matthew Hayes, whose original research I expanded upon with this thesis, also provided insight into the principles and context behind the larger *nmeta* project.

To my friends and fellow students, Jordan Ansell and Alexander Deng, I valued your willingness to listen to my ideas, even when you were approached at the most inconvenient of times.

Finally, I would like to thank InternetNZ for their financial support of this research.

Contents

Chapter 1

Introduction

Distributed denial of service (DDoS) attacks utilise many attacking entities to prevent legitimate use of a resource via consumption [1]. The motivations for carrying out DDoS attacks vary but the main motivating factor is to cause damage to the victim. However, this may also be accompanied by personal reasons, prestige, material gain or political reasons [1]. The disruptive nature of DDoS attacks means that the infrastructure used to forward the malicious traffic is often affected as well. Therefore DDoS attacks often result in collateral damage.

The debilitating potential of DDoS attacks has increased with the advent of the Internet of things (IoT). IoT has been a disruptive agent within the domain of computer networks as objects such as fridges and security cameras are being given the capability to connect to the Internet. This has unfortunately drawn the attention of malicious parties. The world has been repeatedly shown that IoT devices are vulnerable to being used as a platform to perform distributed denial of service (DDoS) attacks. A DDoS attack utilises many attacking entities to prevent legitimate use of a resource via consumption [1].

IoT-driven DDoS attacks became notorious in 2016 with the advent of the Mirai botnet. KrebsOnSecurity became the target of one of the largest ever DDoS attacks on September 20, 2016 [2]. The attack flooded the website with around 620 Gbps of traffic. Akamai, who provides KrebsOnSecurity with services to mitigate such attacks, stated that the September 20 attack surpassed a 363 Gbps DDoS attack which they had seen earlier that year.

The source of the attack on KrebsOnSecurity was initially unknown. However it was later found that the source was the Mirai botnet. The size of the Mirai botnet was estimated to be around 500000 to 550000 devices at the time [3]. We live in a reality where hundreds of thousands of devices can be recruited to perform massive attacks.

Mirkovic and Reiher attribute the difficultly in detecting and mitigating DDoS attacks to the lack of collocation in regards to network intelligence and resources [1]. They go on to say that this deficit makes the enforcement of global security policies challenging as networks are run independently of one another. Two years after their research was published, Casado *et al.* presented *SANE* [4]. *SANE* embodied a networking architecture that promised to increase visibility and control in networks by separating the control and data planes in network forwarding elements such as switches and routers. Further development in this area led to OpenFlow [5].

OpenFlow is an implementation of a networking architecture known as Software-Defined Networking (SDN). SDN is characterised by the separation of the control and data planes. The control-plane is responsible for deciding how traffic is forwarded through a network and is typically realised by a logically centralised controller. The data-plane is responsible for forwarding network traffic, specifically packets, between devices by following the behaviour specified by the controller. OpenFlow manages network traffic through the definition of flow table entries, which are stored in a switch's flow table. These entries describe how packets with matching characteristics should be handled by a switch [6].

The role of the controller is to manage the behaviour of a network. In OpenFlow, the switches within the network are given instructions on how to forward packets via flow table entries and they can provide the controller with information as required. The centralisation of intelligence means that a device such as a switch can be enriched with information that has been gathered from other switches. The Open Networking Foundation (ONF) augment the separated control and data planes with a third, as seen in Figure 1.1. This model contains a higher level where business applications communicate with network services, the network services then enforce the desired behaviours on the infrastructure.

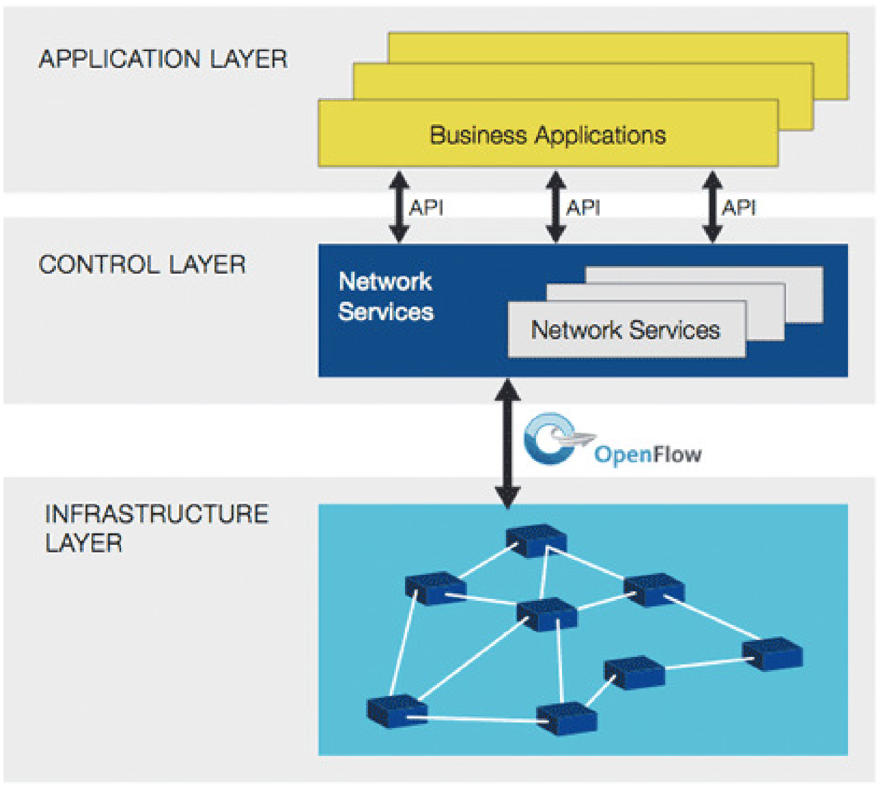

Figure 1.1: The ONF separate the SDN approach into three layers. Image sourced from [7].

The ability to gather information from network forwarding elements into a centralised location makes SDN a candidate for traffic classification. Traffic classification is a process whereby network traffic is classified to help with the management of: network resources, network security and quality of service (QoS) [8–11]. By determining the nature of traffic within a network, network operators can better respond to extreme changes in traffic behaviour.

SDN and OpenFlow have been used in conjunction with traffic classification techniques in the past to detect DDoS attacks [12–16]. Traffic classification and DDoS attack detection may seem like different topics at first but they share a common trait: the determination of the nature of traffic. As such, DDoS attack detection can be viewed as a special case of traffic classification. Earlier research has not used scalable approaches for detecting attacks [9, 12, 15, 17] and has relied on anomaly detection techniques [15–17] that have a tendency to misclassify normal traffic as malicious [1]. Recent research has shown how scalable traffic classification can be performed with SDN but has not focussed on improving classification performance by exploring alternative techniques and their effectiveness.

1.1 Research Problem

There is a process involved in applying classification techniques. The problem at hand must be broken down and understood, typically asking the following questions: What traffic patterns or artefacts are being classified? What features do they exhibit? Do these features make the traffic artefacts distinguishable from background traffic? By addressing these questions, a classifier can be developed and applied for the desired scenario.

Classification is a well understood domain within mathematics and computer science. Tools such as *Scikit-learn*[1], a machine learning package for the Python language, make this domain accessible for programmers but they do not typically focus on applying classification methods within the domain of network traffic classification. This can make the process of

[1]Scikit-learn homepage: `http://scikit-learn.org/stable/`

applying classification techniques seem daunting for those who are new to network traffic classification. Understanding the classifier selection process is key to addressing some of the issues found with current approaches to DDoS attack detection. Chapter 2 explores classifier performance statistics such as detection rate and false positive rate in an accessible manner.

A networking context means that the magnitudes of classifier performance statistics such as true and false positive rates (both of which are covered in Chapter 2) must be interpreted differently. For example, a network traffic classifier designed to detect DDoS attacks should avoid misclassifying non-malicious traffic. Such instances result in innocent flows being flagged which is unacceptable behaviour in modern networks. This makes DDoS attack detection intolerant to misclassifications.

Traffic classifiers also need to scale with fluctuating volumes of network traffic. Existing work was leveraged to address this need as the purpose of this research was not to assess the scalability of traffic classification. This problem has been explored by Hayes [18], who leveraged SDN to present a scalable traffic classification architecture called *nmeta2*. This research leveraged the flexibility offered by *nmeta2* to support user defined (also known as *custom*) classifiers.

This thesis presents a study into the application of intelligent classification techniques using a SDN/OpenFlow-based traffic classifier for security applications. A selection of classifiers were evaluated on a physical network testbed using a DDoS attack dataset to determine their effectiveness.

1.2 Research Objectives

The research objectives of this thesis were to:

1. Propose a methodology for selecting statistical classifiers to detect DDoS attacks.

2. Use SDN to determine the effectiveness of statistical classifiers within a network environment.

1.3 Research Tasks

The following tasks needed to be met to address the research problem and the research objectives:

1. Survey traffic classification approaches and related work within the domain of DDoS attack detection.

2. Determine a selection of classification methods for detecting DDoS attack traffic in an off-line environment; i.e. not connected to a live/evaluation network.

3. Integrate the selected classifiers with the *nmeta2* system.

4. Evaluate the selected classifiers on a network testbed environment (on-line).

1.4 Contributions

The contributions of this thesis are aimed at the field of network traffic classification within SDN. Given the increased interest in using machine learning methods in various domains of science and engineering, it was important that the information within this thesis be accessible to network engineers who may have little to no experience with machine learning. The contributions can be broken down into the following areas:

1. An investigation into the application of statistical classifiers for DDoS attack detection.

2. An evaluation method that considers various aspects of statistical classifiers in networks such as classifier prediction performance, classifier execution time performance and the effect a classifier has on network traffic.

3. Evaluation results that have been obtained from a physical network testbed as opposed to a virtualised environment. Virtualised environments tend to be more popular within SDN research that concerns traffic classification [9,12,14,16–19].

1.5 Thesis Structure

The thesis has the following structure. Chapter 2 provides necessary background information and discusses related work. Chapter 3 presents a study into various statistical classification methods and examines their effectiveness for deployment on a physical network testbed. Chapter 4 explains the modifications that were made to the *nmeta2* system to support statistical classifiers. Chapter 5 determines the effectiveness of statistical classification within an SDN/OpenFlow environment by evaluating a selection of classifiers on a physical network testbed. Chapter 6 closes the main body of the thesis by presenting the final conclusions and proposing future work. The attached appendices provide statistical metadata on the dataset that was used and extra results from the experiments in Chapter 3.

Chapter 2

Background and Related Work

The detection of DDoS attacks can be approached as a classification problem. Classification is a well understood domain that has been leveraged within networks to perform traffic classification. This chapter provides background information on traffic classification, DDoS attack detection and related work within SDN.

2.1 Classification

Classification is the procedure of assigning pre-defined classes to a continuing set of unseen cases based on observed attributes or features [20]. A *class* is a label that describes an object within a context. Trees for instance may be classed as being *deciduous* or *evergreen*. A *case* refers to an object (a tree in the previous example) whose class is currently known, this is in contrast to an *unseen case* whose class is not currently known. The attributes or *features* of an object are used to determine its class. This section covers some of the language and core topics within the domain of classification. This language will be used throughout this thesis.

2.1.1 Classification Approaches

Michie *et al.* describe three types of classification: statistical, machine learning and neural networks [20]. Statistical classification uses an underlying probability model to calculate the probability of a case belonging to a class. Given a case, its class can be determined through knowledge of its attributes. Background knowledge can be used to inform the classification process and human intervention can be used to modify the variables as needed.

Machine learning uses logical or binary operations to perform an automatic computation. The decision process followed by a machine learning approach can often be followed by a human. This allows observers to gain insight into the decision process. Background knowledge may be used in a similar fashion to statistical approaches, but human intervention is unorthodox unlike in statistical approaches.

Neural networks combine aspects of statistical and machine learning approaches. A neural network is constructed from interconnected nodes, where each node produces a non-linear function of its input. A node's input may come from other nodes or from raw data. Neural networks have been compared to the behaviour of networks of neurons within the brain.

2.1.2 Classifier Training

A classifier uses a classification method and a set of features to form a model by training on data. We will refer to a classifier as a combination of a classification method and a set of features because of this relationship. Training (or learning) is a process where a method takes data as an input to generate a model. This model can then be used to classify cases by making predictions. Several learning approaches exist but two will be described for simplicity.

Supervised learning is an approach where training data has known classes (also referred to as *labels* [20–22]. Cases are then assigned a class from the training data when a prediction is made. Unsupervised learning assumes that the labels for the classes are unknown and methods that use this approach cluster cases together based on some measure of similarity [11, 21, 22]. The use of unsupervised methods is often referred to as *clustering* as a result.

It is difficult to difficult if supervised or unsupervised learning is better than the other. Methods that utilise unsupervised learning infer classes when clustering [20]. The clusters themselves have no label, therefore the types of objects within a system cannot be easily known. Supervised learning however utilises labelled data. The ground truth offered by labelled datasets has demonstrated use within the area of anomaly detection, especially when evaluating detection methods [23]. As such, the classifiers referred to in the rest of this thesis use supervised learning methods.

2.1.3 Binary and Multiclass Classification

Classifiers assign a class from at least two options. Classifiers that predict a class based on two options are known as binary classifiers. Multiclass classifiers have the capability to predict a class based on more than two options. This is not to say that multiclass classifiers are better than binary classifiers. The nature of the classification problem defines how many classes are necessary.

2.1.4 Binary Classifier Performance

The predictions made by classifiers can be collected to form a suite of performance statistics. These statistics provide insight into how successfully a classifier is able to classify cases. This must be done in reference to a set of *testing data* where the true class of each case is known. Therefore the actual and predicted cases can be compared.

The preliminary data needed before the statistics can be calculated will be explained first. Following this, the formulae for the statistics will be provided alongside a short description [24]. It is important to note that the statistics provided below refer to binary classifiers as the scope of the classification problem fits within the domain. This will be justified in Section 2.3. Two classes, A and A', will be used below where A represents a positive prediction and A' represents a negative prediction.

Predicted Class vs. Actual Class

- True Positive: The actual class of a case was A and the predicted class was A. This represents a successful prediction.
- True Negative: The actual class of a case was A' and the predicted class was A'. This represents a successful prediction.
- False Positive: The actual class of a case was A' and the predicted class was A. This represents an unsuccessful prediction.
- False Negative: The actual class of a case was A and the predicted class was A'. This represents an unsuccessful prediction.

Performance Statistics

The true positive rate (or recall) is the ratio of successful predictions made to cases of class A. This will be referred to as the detection rate (DR) throughout the rest of the thesis as positive predictions will concern the detection of malicious traffic. The formula is shown in equation 2.1:

$$DR = \frac{TP}{TP + FN} \,. \tag{2.1}$$

The false positive rate (FPR) is the ratio of unsuccessful predictions made to cases of class A'. The formula is shown in equation 2.2:

$$FPR = \frac{FP}{TN + FP} \,. \tag{2.2}$$

Accuracy is the ratio of successful predictions made to both classes. The formula is shown in equation 2.3:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \,. \tag{2.3}$$

Precision (or positive predictive value) is the ratio of correct predictions made for class A. The formula is shown in equation 2.4:

$$Precision = \frac{TP}{TP + FP} \,. \tag{2.4}$$

The f-measure statistic (or F1 score) considers both the DR and precision of a classifier to measure its quality. The forumla is shown in equation 2.5:

$$f\text{-}measure = 2 \times \frac{DR \times Precision}{DR + Precision} \,. \tag{2.5}$$

The performance statistics provided above will be used and referenced throughout this thesis. Note that other performance statistics exist within the domain of classification. The selection of performance statistics above sufficed for this research.

2.2 Traffic Classification Techniques

Traffic classification concerns the application of classification techniques on network traffic. The goal of traffic classification is to improve the management of: network resources, network security and QoS [8–11]. Traffic classification mechanisms classify traffic by collecting data on packets, unidirectional flows or bidirectional flows as information passes through a network.

A unidirectional flow consists of traffic sent from one host to another. It is defined using a network five-tuple (or just simply *five-tuple*) which constist of a source and destination IP address, a transport layer protocol, and source and destination port numbers (if applicable). A bidirectional flow extends the notion of a unidirectional flow by considering the traffic sent in both directions between hosts.

Techniques used to classify traffic include static classification, identity classification, deep packet inspection (DPI) and machine learning. These techniques do not necessarily follow the definition of classification where a learning process is used. Instead traffic classification aims to identify the class of traffic using the methods below.

2.2.1 Static

Static classification uses information contained within protocol headers to classify traffic. A subset of this is known as port-based classification, where TCP and UDP port numbers are used to identify applications [22]. This technique however is not resilient as some applications used dynamically allocated ports or ports that are not registered with the Internet Assigned Numbers Authority (IANA). This technique is often the fastest and simplist traffic classification technique [25].

2.2.2 Identity

Identity classification utilises techniques to determine the identity of a device so that a specific policy can be applied to the identity [26]. Identity is loosely defined within the context of network traffic. It could refer to identities obtained from MAC addresses, 802.1X or *Active Directory* for instance. Identity classification techniques that utilise 802.1X or *Active Directory* require identities to be verified before the respective policy can be applied. Communication with an external service is necessary to verify the identity as a result. This can increase classification time especially if a person is required to provide authentication credentials.

2.2.3 Payload Inspection

Payload inspection techniques look beyond the network five-tuple and details in other packet headers to determine the class of a packet or flow. Deep packet inspection (DPI) is a popular approach as it typically provides the greatest accuracy. This accuracy comes at a high cost however as packet payloads must be loaded into memory before being processed.

Alcock and Nelson proposed *libprotoident*, a mechanism that performs payload inspection by only reading the first four bytes of a packet's payload [27]. Their research suggests that a high level of accuracy can be achieved even when the amount of information used to classify packets is limited. Their experiments showed that *libprotoident* was successful in classifying twenty-five of the twenty-eight applications used for testing. The second most accurate classifier, the DPI mechanism nDPI, successfully classified twenty-three applications in comparison.

Payload inspection techniques, regardless of the depth of inspection, must be kept up-to-date. Extensive knowledge of applications and services is required as updates can change the semantics of their respective flows. Furthermore, proprietary protocols and encryption can essentially defeat efforts to classify traffic as the data within each packet becomes obfuscated [22].

2.2.4 Machine Learning

Machine learning approaches utilise a learning process using training data. The notion of training was covered earlier in Section 2.1.2 within the context of classification. This similarity with classification means that statistical classification techniques can be thought of a particular kind of machine learning approach.

The popularity of machine learning in recent years has seen the coupling of flow statistics with machine learning approaches to classify network traffic [11]. This approach is not new despite its recent interest. It has been seen in published work to address intrusion detection as early as 1994 [28].

Machine learning has significant advantages over static and payload inspection based techniques. The development of new approaches has been motivated by limitations in previous efforts. For instance, static classifiers are ineffective as they assume that applications use known port numbers that do not change. Payload inspection address this by examining the payload of each packet, encryption however defeats this approach [11]. Machine learning is the next step in the evolution of traffic classification as it looks beyond these features.

2.3 DDoS Attack Detection

DDoS attacks were defined in Chapter 1 as an attempt to prevent legitimate use of a resource by using numerous attacking entities to consume that available resource [1]. The disruption of resources can be debilitating to organisations, as customers expect to use reliable services. Furthermore, the ease at which an attack can be launched at a target makes the detection of DDoS attacks an important topic.

2.3.1 DDoS Attack Description

DDoS attacks are made possible by the Internet. Forwarding elements within the Internet forward packets towards their destination with little to no consideration into the behaviour of the sender or the receiver. For this reason Mirkovic and Reiher make the observation that “the Internet [was] not designed to police traffic” [1].

Mirkovic and Reiher also presented a taxonomy on the forms of DDoS attacks. The most simple form of DDoS attack is arguably a flooding attack. A flooding attack is characterised by a victim being sent tremendous volumes of traffic to consume resources. The second type requires the attacking party to send malformed packets to machines that cause applications to freeze or reboot. The third type requires the attacking party to gain privileged access to devices within the victim's network. These devices are then used against the victim until the target resources become unavailable.

DDoS attacks can be described and modelled mathematically. Xiang and Li presented a model based on the interactions between the attacking and defending parties. Keeping within the scope of this thesis, they defined a DDoS attack as a battle of resources. An attacking party is successful if they can prevent legitimate resource use and the defender is successful if they can filter enough attack traffic such that legitimate resource use is maintained [29].

2.3.2 Detection Approaches

Mirkovic and Reiher's taxonomy on DDoS attacks also identified three approaches to detecting DDoS attacks. These were: pattern detection, third-party detection and anomaly detection.

Pattern Detection

Pattern detection uses principles similar to virus detection mechanisms. Signatures of known attacks are used to identify DDoS attacks as traffic passes through a network. Static classification is an example of pattern detection where a signature contains a combination of IP addresses and port numbers. *Snort*[1] is an example of an intrusion detection system (IDS) that uses signatures to detect malicious traffic. Pattern detection approaches are not flexible as they must be updated regularly to learn new attacks.

[1]Snort homepage: `https://www.snort.org/`

Third-party Detection

Third-party detection relies on external parties to detect and inform networks of the occurrence of an attack. A trace-back mechanism is an example of a third-party detection approach that aims to identify the origin of a DDoS attack [30]. The disadvantage of relying on a third-party however is that the detection signal must travel to the victim network. As well as incurring transmission delay, the signal must make it to the victim network which may not be possible during a DDoS attack. Alternatively, a separate network may be used for signalling but this comes at a greater monetary cost to the victim.

Anomaly Detection

Anomaly detection requires the creation of a model of normal traffic behaviour. Flows are compared to the model as traffic passes through a network. Formally speaking, anomalies are patterns in data that do not conform to expected patterns of behaviour [21]. Such patterns manifest themselves as outliers and they can be used to determine the state of a system or entity. Anomaly detection can also be used in other domains to find faulty equipment in critical systems.

It is important to distinguish anomaly detection from two similar processes that handle outliers in data: noise removal and novelty detection. Noise removal is the process of removing unwanted data points before data analysis is performed. Novelty detection aims to identify previously unobserved behaviour, which is then typically included in the model of "normal" traffic behaviour.

Anomaly detection approaches tend to excel in detecting previous unknown attacks. Despite this, they have a tendency to misclassify non-malicious traffic (false positives). This can inconvenient to legitimate users of a resource as they may be flagged and treated as an attacking party. This can be attributed to the absence of knowledge that anomaly detections algorithms have regarding malicious traffic. Anomaly detection algorithms are only designed to identify cases that fall outside the bounds of normal behaviour. Unlike statistical classifiers, for instance, they have no knowledge of the characteristics of malicious traffic.

2.4 Related Work

The logically centralised control-plane offered by SDN has made it a popular platform for detecting DDoS attacks and performing traffic classification in general. This shows its versatility within the contexts of network management, security and QoS. Table 2.1 presents a selection of previous research concerning DDoS attack detection and traffic classification using SDN. These examples are described in more detail below.

Braga *et al.* [12] address the difficulties of distinguishing legitimate traffic from DDoS attack traffic. Their solution can be broken down into two parts: the utilisation of the NOX controller and a Self-Organising Maps (SOM) algorithm. NOX was used to provide a programmatic interface to handle the collection of switch information. Therefore the OpenFlow protocol was only used to collect statistics on flow table entries. SOM is an unsupervised artificial neural network. An advantage of SOM is its ability to transform n-dimensional data into a one or two-dimensional map or grid to use in the neuron selection process. This approach was evaluated in a virtualised environment.

Mehdi *et al.* [17] investigated three anomaly detection approaches within an SDN environment. The methods, rate-limiting, entropy and NETAD, were run on the controller. The anomalies used during their evaluation were a TCP portscan, a TCP SYN-flood and a UDP flood. The entropy and NETAD methods did not perform as well as the rate-limiting method. The cost of having a 100% DR was having a FPR of at least 50% for those two methods. This approach was evaluated in a virtualised environment.

Qian *et al.* [10] classify HTTP traffic within a 3G mobile network data-plane. Their solution is unique compared to previous work as they utilise signatures based on HTTP headers instead of just categories based on port numbers. As with other solutions, this classification was performed on the controller. The evaluation environment was not specified.

Ng *et al.* [9] developed a SDN traffic classification platform called *nmeta*. This was not developed to classify a particular kind of traffic. Instead, it allows the network operator to define their own classifiers to apply to traffic. Traffic can be classified using static, identity or machine learning techniques. This solution classifies traffic on the OpenFlow controller. However their results revealed that this approach does not scale. This approach was evaluated in a virtualised environment.

Hayes [18] developed *nmeta* further to address the performance and scalability concerns that were identified. This resulted in a distributed platform called *nmeta2*. This approach differs from its predecessor as traffic is no longer classified on the controller. A separate application called the Data Plane Auxiliary Engine (DPAE) is forwarded traffic from switches to perform classification. Results are forwarded to the controller via a dedicated connection over the control-plane. Hayes found that this distributed approach does scale. Similarly to *nmeta*, traffic can be classified using static, identity or machine learning techniques. Unlike *nmeta*, it supports payload inspection techniques. This approach was evaluated in a virtualised environment.

Lim *et al.* [14] presented a detection architecture where the DDoS detection mechanism was deployed on the victim server. Their mechanism has the ability to communicate with the controller via a secure channel so that defensive measures can be taken. Details on the detection mechanism itself were not provided. This approach was evaluated in a virtualised environment.

Giotis *et al.* [15] presented a controller-based DDoS attack detection mechanism that utilises sFlow to collect flow statistics from switches. They found that the use of sFlow resulted in significantly less traffic being sent over the control-plane compared to a native OpenFlow approach where the controller makes flow-statistics requests to a switch. The mechanism was configured to gather data from switches every 30 seconds, which they claim is representative of nearly real-time detection. The detection mechanism was evaluated using entropy and TRW-CB separately. When using entropy, the DR reached 100% but the FPR ranged from 23% to 39.3%. The high FPR is characteristic of anomaly detection approaches. This approach was evaluated on a physical network testbed.

Wang *et al*. [16] implemented an entropy-based DDoS attack detection mechanism by modifying the Open vSwitch software switch. The OpenFlow table pipeline was modified to count the number of packets received within a predefined time period. Their choice to detect attacks closer to the data-plane was motivated by the desire to reduce overheads introduced by classifying traffic on the controller. This limited the complexity of their detection algorithm as they were constrained by the processing capabilities of a switch. As such, they used an entropy-based algorithm which they claim is computationally lightweight. Their entropy-based detection algorithm resulted in a DR of 100% and FPR of 25%. This approach was evaluated in a virtualised environment.

Lin *et al*. [19] used an SDN traffic classification architecture to detect SYN flooding and web application attacks. Compared to other proposed solutions, this solution moves intelligence closer to the data plane. Extensions to the OpenFlow protocol were made to facilitate a two tier architecture. At tier one, a classification module on a switch inspects TCP/IP and application headers. Failing a classification at tier one, traffic is sent to tier two where it is subject to DPI on a network function virtualisation (NFV) module. Their work found that separating the classification function into a NFV module reduced the amount of traffic sent from the data-plane to the control-plane by 99.95% when classifying HTTP packets with a layer 7 load balancer. The published work only assessed the performance overheads on the controller and did not consider the accuracy of the distributed classification mechanism. This approach was evaluated in a virtualised environment.

The related work explored above demonstrates variety in several ways. First, SDN/OpenFlow has been combined with classification techniques to classify network in various scenarios. Although the use-case of traffic classification in this thesis is DDoS attacks, it is important to show that SDN traffic classification can be used in other scenarios.

The use of anomaly detection techniques such as information entropy are popular choices for detecting DDoS attacks. Mechanisms that used this however suffered from a high FPR. False positives are near impossible to avoid but they cannot be ignored. The law of truly large numbers means that even a FPR of 5%, which may be considered to be small in other domains, can result in large amounts of traffic being misclassified. Networks are environments where the total number of flows increase over time. Therefore new methods should utilise techniques such as machine learning that balance both DR and FPR.

Classification mechanisms traditionally operate on the controller. More recent research has seen this task be moved closer to the switches in the data-plane. Lin *et al*. suggest that the processing overheads imposed by classifiers and detection algorithms on a controller can be reliably removed by moving the function to another device [19]. This move is well justified as it has been shown to be scalable as well as necessary.

SDN-based traffic classification platforms are typically evaluated in virtualised environments. There is room for concern that such an approach does not accurately capture how a system would work in the real world despite being the convenience of the environment. Future research needs to utilise hardware as such experiences can bring about events that cannot be easily represented in software.

Author	Year	Traffic Type	Location	Method	Environment
Braga *et al.* [12]	2010	DDoS	Controller	SOM	Virtualised
Mehdi *et al.* [17]	2011	Anomaly	Controller	Rate-limiting, Entropy, NETAD	Virtualised
Qian *et al.* [10]	2013	HTTP	Controller	BLINC	Not stated
Ng *et al.* [9]	2014	Any	Controller	Static (primarily)	Virtualised
Lim *et al.* [14]	2014	DDoS	At victim server	Not stated	Virtualised
Giotis *et al.* [15]	2014	DDoS	Controller	Entropy and TRW-CB	Physical
Wang *et al.* [16]	2015	DDoS	OpenFlow edge switch	Entropy	Virtualised
Lin *et al.* [19]	2015	SYN Flood & Web attacks	Switch and NFV	Packet header analysis	Virtualised
Hayes [18]	2016	Any	DPAE	Static (primarily)	Virtualised

Table 2.1: Table illustrating contributions of previous research. Location refers to where the classification of traffic occurs and method refers to the method used to classify traffic.

2.5 Chapter Summary

This chapter covered background material within the domain of traffic classification, explored approaches for detecting DDoS attacks and showed how SDN has been used as a classification platform specific to detecting DDoS attacks. The language specific to classification will be throughout the rest of this thesis.

This research utilises *nmeta2* to perform statistical classification on traffic to detect DDoS attacks. The *nmeta2* platform demonstrates scalability by classifying traffic on the data-plane. Furthermore, it grants flexibility to network operators by allowing custom classifiers to be integrated with the DPAE. Statistical classification is preferred over anomaly detection approaches as the latter has been shown to misclassify traffic significantly.

The remaining chapters will: demonstrate how classifiers were chosen to be evaluated on a physical network testbed, provide details on how the chosen classifiers were integrated with *nmeta2*, evaluate the chosen classifiers using the ISCX 2012 DDoS dataset on hardware, and present the final conclusions and future work.

Chapter 3

Classifier Selection

This chapter presents an investigation for selecting three statistical classifiers capable of detecting DDoS attacks for deployment on a physical network testbed. The investigation starts with the exploration of various statistical classification methods. These are then combined with network traffic features to form classifiers and are tested against a DDoS attack dataset provided by the Information Security Centre of Excellence (ISCX) at the University of New Brunswick (UNB) in two off-line experiments to determine their effectiveness. As defined earlier, a *classifier* refers to the combination of a classification method and a set of features.

3.1 Statistical Classification Methods

Chapter 2 presented a case for using supervised methods (classification) over unsupervised methods (clustering). This section describes seven different supervised methods. Attributes that distinguish the methods from one another will be mentioned without exploring the theory behind each method.

The methods that are described were not chosen through a well-defined selection process. They were chosen as they are well-known and established methods that have been used to solve classification problems inside and outside the domain of networking. This mirrors the purpose of this thesis: to apply existing methods to a SDN environment, not to explore and present a new method.

The purpose of exploring the distinguishing attributes of each classifier was to aid the preliminary classifier selection process. Knowledge of such attributes was used to explain the results of the off-line classification experiments and select three classifiers to be deployed on a physical network testbed for the on-line classification experiments. Details of the on-line experiments will be covered in Chapter 5.

The use of n below denotes the number of features used by a classifier.

3.1.1 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) generates a linear hyperplane to separate classes within an n-dimension coordinate system using Bayes' Theorem [20, 31]. As a result, the features being used for classification must contain continuous data. LDA addresses binary classification problems by using a single hyperplane to divide the coordinate system into two areas. Multiple hyperplanes can be used to classify three or more classes for addressing multiclass classification problems. Further discussion on this topic will not be provided as this thesis does not concern multiclass classification.

LDA uses R.A. Fisher's assumption that the probability densities of the two classes are Gaussian and have equal covariance matrices [31]. By extension, equal covariance matrices suggest that the classes are linearly separable. It is also important to note that a linear hyperplane is not flexible [32].

Thapngam *et al.* [33] applied LDA to detect DDoS attacks on web servers. Classification was performed on the web server itself by sampling the arrival rate of incoming traffic. Pearson's Correlation Coefficient and Shannon Entropy (information entropy) were used to calculate the dependency and predictability of traffic. The calculated values were used as features for classification.

3.1.2 Quadratic Discriminant Analysis

Similarly to LDA, Quadratic Discriminant Analysis (QDA) uses a hyperplane within an n-dimension coordinate system to separate classes. The hyperplane that is generated however is described using a quadratic function. QDA is preferred over LDA when the covariance matrices for the two classes are not equal [31]. The use of a quadratic decision boundary makes this method more flexible than LDA whilst still being able to support continuous data.

Roughan *et al.* [34] evaluated various QDA classifiers to classify traffic for QoS policy enforcement. They evaluated the QDA-based classifiers alongside LDA and k-nearest neighbour-based classifiers by using the same sets of features. Interesting enough, they discovered that the QDA-based classifiers performed worse compared to the other classifiers. As with any classification problem, these results are particular to the problem being solved and are not reflective of the QDA method as a whole.

3.1.3 Support Vector Machine

The Support Vector Machine (SVM) method can be viewed as an improvement on the LDA and QDA methods mentioned previously [31]. These improvements relate to the positioning of the hyperplane and the ability to more efficiently operate in high dimensional spaces. The position of the hyperplane affects the classifier's ability to handle cases where the values for the corresponding features are close to the decision boundary, these points can also be referred to as noise or boundary cases. Bias can be introduced if the hyperplane is too close to the training data points belonging to one class, this may result in the misclassification of noise.

The SVM method utilises *support vectors* to improve the positioning of the hyperplane. These support vectors, the namesake of the method, are a collection of training data cases that typically make up a small percentage of all training cases. The position of the hyperplane is optimised by maximising the distance between the support vectors and the hyperplane itself; this distance is also referred to as the margin. Once the margin has been maximised across all support vectors, the hyperplane can be described as an *optimal separating hyperplane* or *maximal margin classifier*. The process of fitting the hyperplane is a quadratic optimisation problem which can result in the training time lasting on the order of minutes when large datasets are used [35]. Therefore the use of this method may be undesirable despite the promise of an optimal separating hyperplane.

The performance of a SVM classifier is also dependent on the dimensionality of the model being created. The dimensionality of the coordinate system used in the model is equal to the number of features being used, similarly to LDA and QDA. Therefore it follows that a training set with n features results in a n-dimensional model. Increasing the number of features increases the complexity of calculating an optimal separating hyperplane.

The SVM method uses what is known as the *kernel trick* to address the issue mentioned above. Unlike traditions methods where calculations are performed in high dimensional *feature space*, the kernel trick utilises kernel functions to avoid performing calculations in such high dimensional spaces. This has the effect of reducing the complexity of the computational task. The use of kernels also makes SVMs flexible as different kernels can be selected without changing the mechanics behind the method. Examples of kernels include linear, quadratic or radial basis function (RBF) [31]. Further discussion of the SVM method and the kernel trick is outside the scope of this thesis.

Yang *et al.* [36] apply a SVM classifier with a RBF kernel to detect network intrusions. The network intrusions were sampled from the KDD-Cup99 dataset from the MIT Lincoln Laboratory. They augmented their detection mechanism by using a particle swarm algorithm to optimise the kernel coefficient and penalty parameters to the SVM classifier.

3.1.4 k-Nearest Neighbours

The k-nearest neighbours (KNN) method is quite possibly one of the easier classification methods to understand. This is in part due to its simplicity: unseen cases are classified based on the class of neighbouring cases [37]. To expand on this further, the process consists of two steps:

1. Find k training instances that are **closest** to the unseen case.
2. Take the most commonly occurring class from the k neighbours as the class for the unseen case.

The hyperplane-based methods explored above suffer from a lack of flexibility. Regardless of the function used to generate the decision boundary, the fixed nature of the boundary means that cases may be misclassified if they appear to be too similar to cases of another classes. In comparison, the simple principle used by KNN is effective in situations where the decision boundary contains irregularities [38].

This method has some caveats despite its simplicity and apparent benefits. The first is the definition of the word **closest** seen above in step 1. Closeness implies a measure of *distance* between neighbours, the definition of which alters the behaviour of a classifier. Examples of distance include: Euclidean Distance, Manhattan Distance and maximum dimension distance. The use of some distances (such as Euclidean) can result in large feature values swamping smaller ones, thus reducing the number of features that matter. Normalising values can be used to overcome this problem if required [37].

The second caveat regards the learning/training process of the method. Unlike the methods that were mentioned previously, the k-nearest neighbours method learns as predictions are made, this is also known as lazy learning [39]. Furthermore, state is not kept between predictions. This can result in undesirably long prediction times when larger training sets are used as distances must be calculated for each sample. Despite this, the initialisation time of classifiers that use this method is very small (typically zero) as a model does not need to derived before classifications can be performed.

Su [40] used the KNN method to detect Denial of Service (DoS) attacks. This was combined with a genetic algorithm for selecting features and determining weights for each of the distances. The research suggested that cases could be classified using the presented KNN classifier within tens of milliseconds. Su concluded that this method could be deployed to perform real-time classification.

3.1.5 Naive Bayes

The Naive Bayes method utilises Bayes' Theorem to classify unseen cases [37]. In short, Bayes' Theorem describes how to calculate the probability of an event occurring given the occurrence of another event. This scenario, also known as conditional probability, can be calculated using the formula in equation 3.1:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \, . \tag{3.1}$$

The rest of the exploration of the Naive Bayes classifier assumes that the reader understands the aforementioned principle of conditional probability.

The Naive Bayes method learns by using the training data to compute a probability model. This model facilitates the calculation of multiple posterior probabilities. A posterior probability is a conditional probability that is assigned to an event once relevant evidence has been considered. The values of features for unseen cases would be considered as evidence within the context of classification. The posterior probabilities that are calculated represent the likelihood of an unseen case belonging to each class. The class that is assigned to an unseen case comes from the largest posterior probability that was calculated.

It is important to mention two assumptions behind the Naive Bayes method. The first assumption is that the features contain categorical values. This can be handled by discretising continuous values, this can however lead to over-sensitivity. The cut-off points for ranges of continuous values means that values which fall just above the range get treated differently to values that fall just below.

The second assumption of the method is that the features of cases are considered to be independently distributed. This can be an issue within the domain of computer networking as traffic features such as and byte and packet counts are dependent on one another. The number of bytes sent in one direction increases linearly over time with the number of packets that are sent in that direction. Despite this, the Naive Bayes method has been shown to perform well even if this assumption is broken [41].

Toulouse *et al.* [42] use the Naive Bayes method alongside an average consensus algorithm to detect DDoS attacks. Posterior probabilities are calculated in parallel across several detection modules. The modules reach an average consensus for each posterior probability by agreeing on the average sum of calculated probabilities. The consensus is then used to classify traffic.

3.1.6 Decision Tree

The Decision Tree method classifies unseen cases by traversing a decision tree [37]. Decision Trees are created by finding the best split among all features. Categorical values are split into the possible values that a feature can take. Continuous values are split by determining a split value for data *less than or equal to* and data *greater than*.

Top-Down Induction of Decision Trees (TDIDT), also known as recursive partitioning, is a popular algorithm for creating decision trees. This algorithm is underspecified however as the method for determining the split in values is not given. For instance, the underspecification of TDIDT can result in over-complex trees that do not generalise well due to over-fitting. It is also worth noting that TDIDT relies on the Adequacy Condition. This requires cases with the same values for all attributes to belong to the same class.

This method also exhibits the following attributes. The prediction time is log(T), where is T is the number of training cases. This comes as a result of use of a tree data structure in the underlying model. The method is also unstable. This means that small variations in training data can result in completely different trees or in other words: a completely different classifier [43].

Wu *et al.* [44] present a DDoS defence mechanism that uses a Decision Tree-based classifier to detect DDoS attacks. They go beyond binary classification and utilise the Decision Tree method to perform multiclass classification. In particular, they class traffic as either being: normal, a TCP SYN flooding attack, a UDP flooding attack or an ICMP flooding attack.

3.1.7 Random Forest

The Random Forest method is similar to the Decision Tree method in that a decision tree is used to classify cases. The key difference is that the Random Forest method utilises randomness during the construction of the tree to mitigate over-fitting and improve accuracy [45]. The randomness manifests during the splitting of a node in the tree construction process. In contrast to splitting on the the best split among all features, this method chooses the best split across a random subset of features.

The Random Forest method has been used extensively to solve classification problems. Its success within these problems strongly suggests that this method is robust. For instance, research conducted by Fernandez-

Delgado *et al.* found that Random Forest classifiers were the first and second best in regards to accuracy when evaluated against 121 datasets [46]. These datasets are said to "Represent the whole UCI data base".

Singh *et al.* [47] applied the Random Forest method to detect botnets. This method was chosen as the authors desired a method that had a high prediction accuracy and could handle a large number of features. Furthermore, they utilised Apache Hadoop to distribute the prediction task across a cluster. The prediction time when using the cluster ranged between 5 to 30 seconds.

3.1.8 Classification Method Summary

Tables 3.1, 3.2 and 3.3 summarise the key properties of each classification method that was described above.

Classifier	Key Properties
LDA	• Linear decision boundary. Results in a lack of flexibility. • Assumes Gaussian density with same covariance matrices (Fisher's assumption). • Requires continuous data.
QDA	• Quadratic decision boundary. More flexible than LDA. • Assumes unequal covariance matrices, unlike LDA. • Requires continuous data.

Table 3.1: Key properties of the seven candidate classifiers - part 1.

Classifier	Key Properties
SVM	• Flexible decision boundary granted by the ability to change the kernel. • Utilises the *kernel trick*. • Training an SVM is a quadratic optimisation problem. • Requires continuous data.
KNN	• Unseen cases classified as the majority class from neighbouring cases. • Effective in situations where the decision boundary is irregular. • Is a lazy learner. • Can handle both categorical and continuous data.
Naive Bayes	• Classifications are made using Bayes Theorem. • Performs well even if the assumption of feature independence is broken. • Can handle both categorical and continuous (must be discretised) data.
Decision Tree	• Classifies using a decision tree created using TDIDT. • TDIDT relies on the Adequacy Condition and may create trees that over-fit. • Can handle both categorical and continuous data.

Table 3.2: Key properties of the seven candidate classifiers - part 2.

Classifier	Key Properties
Random Forest	• Classifies using a decision tree that utilises randomness during the training phase. • Designed to mitigated over-fitting. • Can handle both categorical and continuous data.

Table 3.3: Key properties of the seven candidate classifiers - part 3.

3.2 Classification Method Implementation

Section 3.1 presented the principles of seven supervised classification methods. Implementations of the methods were not provided as it was not necessary for this research. A machine learning library was used instead. This section provides details on the library that was used and the parameters used by the methods.

3.2.1 Scikit-learn

Scikit-learn is an open source machine learning library/package for the Python programming language. The package contains implementations of various algorithms for performing classification, regression and clustering tasks. Scikit-learn's implementation of classification methods in the Python language means that classifiers can be easily integrated with *nmeta2*; which is also written in Python. Table 3.4 below explicitly states the classes used from within the Scikit-learn package to perform classification.

Classifier	Scikit-learn Class
LDA	discriminant_analysis.LinearDiscriminantAnalysis
QDA	discriminant_analysis.QuadraticDiscriminantAnalysis
SVM	svm.SVC
KNN	neighbors.KNeighborsClassifier
Naive Bayes	naive_bayes.GaussianNB
Decision Tree	tree.DecisionTreeClassifier
Random Forest	ensemble.RandomForestClassifier

Table 3.4: Scikit-learn classes used for each classification method. Note that the 'sklearn.' prefix has been removed from each class name.

3.2.2 Classification Method Parameters

Most of the classification methods offered by Scikit-learn allow users to tune their performance through various parameters. These were typically configured to the default values set by the package. Parameter tuning was not deemed necessary as the research problem being addressed concerned whether or not it is feasible to deploy statistical classifiers within an OpenFlow/SDN environment. Therefore it was more beneficial to investigate a wider variety of methods rather than picking one or two methods and tuning their parameters.

The parameter values are shown below in Tables 3.5 and 3.6 to clearly illustrate the configuration of each method. Deviations from the default values are highlighted and explained where necessary.

Classifier	Parameters
LDA	*solver*='svd', *shrinkage*=None, *priors*=None, *n_components*=None, *store_covariance*=False, tol=0.0001
QDA	*priors*=None, *reg_param*=0.0

Table 3.5: Scikit-learn classification method parameters - part 1.

Classifier	**Parameters**
SVM	*C*=1.0, *kernel*='rbf', *degree*=3, *gamma*='auto', *coef0*=0.0, *shrinking*=True, *probability*=False, *tol*=0.001, *cache_size*=200, *class_weight*=None, *verbose*=False, *max_iter*=-1, *decision_function_shape*=None, *random_state*=None
KNN	*n_neighbors*=5, *weights*='uniform', *algorithm*='auto', *leaf_size*=30, *p*=2, *metric*='minkowski', *metric_params*=None, *n_jobs*=1. Note that *algorithm* was set to 'kd_tree' for the on-line evaluation of the classifiers to decrease the prediction time by storing the training cases in a k-d tree.
Naive Bayes	*priors*=None
Decision Tree	*criterion*='gini', *splitter*='best', *max_depth*=None, *min_samples_split*=2, *min_samples_leaf*=1, *min_weight_fraction_leaf*=0.0, *max_features*=None, *random_state*=None, *max_leaf_nodes*=None, *min_impurity_split*=1e-07, *class_weight*=None, *presort*=False
Random Forest	*n_estimators*=10, *criterion*='gini', *max_depth*=None, *min_samples_split*=2, *min_samples_leaf*=1, *min_weight_fraction_leaf*=0.0, *max_features*='auto', *max_leaf_nodes*=None, *min_impurity_split*=1e-07, *bootstrap*=True, *oob_score*=False, *n_jobs*=1, *random_state*=None, *verbose*=0, *warm_start*=False, *class_weight*=None

Table 3.6: Scikit-learn classification method parameters - part 2.

3.3 Datasets

Data is required to train any kind of algorithm that makes use of prior knowledge. Datasets are particularly useful in this regard as they typically provide data specific to a context. There resides a couple of problems however regarding datasets for use in network traffic classification. Their creation is expensive and their availability limited [21,22]. Researchers often respond by evaluating their work against private datasets, this serves to compound the problem of comparing classifiers [48].

Datasets are ideally supplied with labels for all cases. The presence of labels for all cases means that supervised learning methods can be used. They supply ground truth, a necessary ingredient when evaluating classifiers.

A popular dataset is the Center for Applied Internet Data Analysis (CAIDA) DDoS Attack 2007 dataset [49]. This dataset contains a packet trace (in a PCAP file) containing malicious flows from a DDoS attack. The authors of the dataset do not guarantee that the non-malicious flows have been completely removed from the trace. As such, researchers need to use their own background traffic in place of the non-malicious flows. These factors make this dataset less than desirable as ground truth is difficult to define.

ISCX provides a dataset for evaluating intrusion detection mechanisms [50]. This dataset contains traces on various kinds of attacks including an HTTP denial of service (DoS), an HTTP GET DDoS attack performed by an IRC botnet and a brute force SSH attack. Each scenario is supplied with a PCAP file containing both malicious and non-malicious flows, as well as a set of XML files that provide metadata on each flow. The metadata: allows each flow to be identified via a five-tuple, provides some statistical information on the flow and indicates if the flow was part of the attack or not. This dataset will referred to as the ISCX 2012 DDoS dataset.

3.4 Initial Classifier Selection Experiment

This section presents the results from the initial classifier selection experiment. This experiment was conducted in order to determine the effectiveness of a set of classifiers when evaluated against network data in an off-line scenario. These classifiers will be referred to as the *first generation classifiers* from now on. The rest of the section will explore the components of the classifiers used, the experimental setup and the findings.

3.4.1 First Generation Classifiers

The first generation classifiers consisted of the seven classification methods explored in Section 3.1 and six sets of features. Each set of features (or feature sets) consisted of two statistical features of a flow of network traffic. Most of these features were standard network traffic statistics; the remaining features were derived from the standard traffic statistics.

Features

Table 3.7 lists the individual features used to build the feature sets. The *source* refers to the host that sent the first packet within the flow; typically the client. The *destination* is the recipient of the first packet; typically the server. It is assumed that packets or bytes sent by the source are received by the destination and vice versa.

It is interesting to note that two of the features in Table 3.7 utilise a base 10 logarithm. As the number of the bytes sent by a host can grow very large[1], a logarithm can be used to reduce the magnitude of such values, some of which may manifest as outliers. The hypothesis was that by transforming features in this way, a more accurate classifier may be produced.

[1]Appendix A contains relevant feature information relating to the ISCX dataset used in this research.

Feature	**Description**
totalSourceBytes	The number of bytes sent by the source.
log(totalSourceBytes)	The base 10 logarithm of totalSource-Bytes.
totalDestinationBytes	The number of bytes sent by the destination.
totalSourcePackets	The number of packets sent by the source.
FlowDuration	The duration of a bidirectional flow in seconds.
log(FlowDuration)	The base 10 logarithm of FlowDuration.
SourceBytes-per-Packet	The number of bytes sent from the source divided by the number of packets sent by the source.
DestinationBytes-per-Packet	The number of bytes sent from the destination divided by the number of packets sent by the destination.

Table 3.7: Features used to build feature sets for the first generation classifiers.

These features describe two statistical properties of traffic: the amount of information sent in one direction and the duration of a connection. These properties can be used to paint a picture that illustrates the standard behaviour of a host. Deviations from standard behaviour may be indicative of anomalous behaviour, such as a DDoS attack. The nature of DDoS attack is to consume resources to the point of exhaustion. These events result in deviations from standard behaviour, which are assumed to leave footprints in the aforementioned statistical features of traffic. The footprint left behind is dependent on the attack being performed. A flooding attack for instance would result in a large volume of traffic being sent over a shorter period of time compared to a normal session.

Feature Sets

Six feature sets were composed using the features from Table 3.7. Each feature set combined the different statistical properties of network traffic (as seen in Table 3.7) to form a different set of predictors to be used by the classification methods. The feature sets only contained two features, which is not large by any means. However this was acceptable as the initial classifier selection experiment was exploratory.

The feature sets were:

1. totalSourceBytes and totalSourcePackets
2. totalSourceBytes and totalDestinationBytes
3. totalSourceBytes and FlowDuration
4. totalSourceBytes and log(FlowDuration)
5. log(totalSourceBytes) and FlowDuration
6. SourceBytes-per-Packet and DestinationBytes-per-Packet.

3.4.2 Experimental Setup

The first generation classifiers were tested in a test harness implemented in Python[2]. Such a program facilitates the implementation and testing of classifiers within an off-line environment. Several testing parameters could be tuned depending on the requirements of the desired experiment. These parameters were: the number of folds to split the dataset into, a seed for a random number generator and the number of repeated experiments.

The first generation classifiers utilised the Scikit-learn package mentioned in Section 3.2.1 to implement the classifiers in Section 3.4.1. The classifiers were trained with and tested upon data from the ISCX 2012 DDoS dataset. This data was split into training and testing sets using

[2]`https://github.com/bakkerjarr/NetTrafClassificationExploration`

the k-fold cross validation technique. This technique has advantages over hold-out validation; a technique where the dataset is simply split into two arbitrarily-sized sets. For instance, hold-out validation can result in overfitting [51]. The Scikit-learn package offers functions for splitting datasets for k-fold cross validation. More specifically, this was done using the *StratifiedKFold* object to shuffle the data, then stratify it to maintain the same proportion of classes across all folds.

The data used for training and testing originated from the bidirectional flow summaries stored in XML formatted files. The elements contained within the file identify bidirectional flows using a network 5-tuple and 'DateTime' labels for when the flow started and finished. Extra information included within each flow record was pertinent to the experiments: flow statistics detailing the quantities of bytes and packets sent in either direction, and a tag that indicates if the flow was part of a DDoS attack or can be considered to be normal.

Each classifier was subjected to numerous trials in order to get a clearer idea of its prediction performance. The number of folds used was 30, however this was conducted in an unorthodox fashion in order to accommodate classifiers utilising the SVM method. In k-fold cross validation, $k - 1$ folds are traditionally used to train the classifier with 1 fold being used for testing. Due to the size of the dataset however (571698 samples), training a SVM-based classifier with 29 folds (each with roughly 19056 flows) results in an unreasonably long training time. This is due to the SVM training algorithm being a quadratic optimisation problem. As a result, the training and testing sets were switched for all experiments. To be explicit: 1 fold was used for training and 29 folds were used for testing.

Each experiment was repeated ten times to increase the number of results for evaluation. The random seed was incremented by one after each experiment. As the random seed was utilised by the *StratifiedKFold* object for the shuffling process, there were 10 different sets of 30 folds for training and testing. Taking into consideration the cross validation process and the repeated experiments, each classifier was subjected to 300 classification trials.

3.4.3 Results and Discussion

The results of the initial classifier selection experiment were used to inform a classifier selection process. For the purposes of this experiment, the mean detection rate (DR) and mean false positive rate (FPR) was calculated for each classifier. The rest of this section will use tables 3.8, 3.10, 3.11 and 3.12 for the discussion of the results. These tables present a summary of the results; see Appendix B for the DR and FPRs for all classifiers. The tables below use the following abbreviations:

- tSB: totalSourceBytes
- ltSB: log(totalSourceBytes)
- tDB: totalDestinationBytes
- tSP: totalSourcePackets
- FD: FlowDuration
- lFD: log(FlowDuration)
- SBpP: SourceBytes-per-Packet
- DBpP: DestinationBytes-per-Packet

Mean DR

Ranking the best classifiers in terms of mean DR yields interesting patterns. Table 3.8 shows that the log(totalSourceBytes) and FlowDuration feature set resulted in four of the highest DRs. To a lesser extent the QDA method also performed well as the first and fifth highest DR ranked classifiers used this method.

Method	Features	Mean DR	Mean FPR
QDA	ltSB and FD	0.990011046	0.022104852
LDA	ltSB and FD	0.989999724	0.029018682
Naive Bayes	ltSB and FD	0.989364833	0.021982912
SVM	ltSB and FD	0.988362117	0.012553594
QDA	tSB and tDB	0.983063906	0.013685196

Table 3.8: Five best performing first generation classifiers, sorted by mean DR.

The success of the log(totalSourceBytes) and FlowDuration feature set is an interesting result. This may be explained by exploring the spread of the values for the totalSourceBytes and log(totalSourceBytes) features. Table 3.9 shows that the difference between the minimum and maximum values for the log(totalSourceBytes) feature is considerably smaller when compared to the totalSourceBytes feature.

The interquartile range (IQR = first quartile (Q_1) - third quartile (Q_3)) for each feature shows a similar pattern. The IQR for the totalSourceBytes feature is 1643 compared to 0.185 for log(totalSourceBytes). Noting that the IQR is a measure of spread, the variability for the log(totalSourceBytes) is smaller. This suggests that logarithm transformations accentuate the trend of DDoS attack flows that have similar flow characteristics by minimising the effects of outliers. It must be noted that this assumption is specific to the ISCX 2012 DDoS dataset.

	Min.	Q_1	Median	Mean	Q_3	Max.
totalSourceBytes	64	8063	8729	8842	9706	380100
log(totalSourceBytes)	4.159	8.995	9.074	9.063	9.180	12.850

Table 3.9: Summary statistics for the totalSourceBytes and log(totalSourceBytes) features for ISCX attack flows.

Table 3.10 displays the bottom five classifiers when sorted by mean DR.

Ignoring the fact that the least successful classifier in regards to detecting attacks was LDA with the totalSourceBytes and FlowDuration feature set, it is clear that the LDA method formed the poorest classifiers regardless. This result can be explained by the inflexibility of LDA's linear decision boundary. One cannot assume that variations in features between two classes are linearly separable.

Method	**Features**	**Mean DR**	**Mean FPR**
LDA	tSB and FD	0.0000253	0.002416255
LDA	tSB and lFD	0.0000431	0.00185139
LDA	SBpP and DBpP	0.0061397	0.007521567
LDA	tSB and tDB	0.085453263	0.00489996
LDA	tSB and tSP	0.335905144	0.007711583

Table 3.10: Five worst performing first generation classifiers, sorted by mean DR.

Mean FPR

Table 3.11 ranks the best classifiers in terms of mean FPR with the smaller values representing a higher rank. Ignoring the methods and features behind each classifier, one observes that these classifiers have significantly poorer mean DRs compared to the classifiers in Table 3.8. This suggests that a classifier that rarely misclassifies normal traffic is not necessary well suited to detecting attacks. The second apparent observation is that the SVM method resulted in classifiers with low mean FPRs.

Method	Features	Mean DR	Mean FPR
SVM	tSB and tDB	0.750116539	0.001744573
LDA	tSB and lFD	0.0000431	0.00185139
SVM	tSB and FD	0.74577407	0.001861962
SVM	tSB and tSP	0.762044205	0.001903781
SVM	tSB and lFD	0.763057881	0.002086601

Table 3.11: Five best performing first generation classifiers, sorted by mean FPR.

The final table, Table 3.12 which ranks the worst classifiers in terms of mean FPR, contains two common denominators. Firstly, three of the five classifiers with the highest FPR used the Naive Bayes method. Secondly, the log(totalSourceBytes) and FlowDuration feature set had a similar result with it being present in three of the five classifiers in the same bracket.

It is interesting to note the presence of the log(totalSourceBytes) and FlowDuration feature set in this table as it was also predominant in Table 3.8. It appears that this feature could adequately separate normal and attack flows enough for the correct identification of attacks, however it still lead to misclassifications. This suggests that a third feature may be necessary to augment the feature set to separate cases close to the decision boundary. Finally, the QDA-based classifier had the third highest FPR yet it also had the highest DR as seen in Table 3.8.

Method	Features	Mean DR	Mean FPR
LDA	ltSB and FD	0.989999724	0.029018682
Naive Bayes	tSB and tSP	0.962315858	0.024197488
QDA	ltSB and FD	0.990011046	0.022104852
Naive Bayes	ltSB and FD	0.989364833	0.021982912
Naive Bayes	tSB and FD	0.962605386	0.020798816

Table 3.12: Five worst performing first generation classifiers, sorted by mean FPR.

3.4.4 Method and Feature Set Vetting

Classification methods and feature sets were vetted using the results that were presented above. In particular, the DR and FPR was used alongside the attributes of the classifications methods. It is important to consider such attributes within the context of the research problem at hand, as none of the tables contained a classifier that used the KNN, Random Forest or Decision Tree methods. Appendix B shows that they did not perform poorly and these methods were still subject to a vetting process like the remaining methods.

The list below presents the methods and feature sets that were selected along with justification:

- QDA - Two QDA-based classifiers ranked in the top five in regards to mean DR.
- SVM - SVM-based methods exhibited instances of both low FPR and high DR.
- KNN - Despite not receiving a ranking, this classifier offers operational advantages in regards to the absence of training time.
- Naive Bayes - Four Naive Bayes-based classifiers ranked in the top ten in regards to mean DR and one of these ranked third. Furthermore, the assumption of feature independence can be broken [41]. This is useful as network traffic features such as flow duration and the amount of traffic sent in a flow are typically dependent.
- Random Forest - Despite the fact that classifiers that utilised this method did not rank highly, the formulation of the method suggests that larger feature sets are needed. This method is also designed to handle outliers and noise [45].
- tSB and tDB - This feature set was adopted by the fifth best classifier in terms of mean DR. This feature set can be used to capture the

asymmetry in the amount of traffic between hosts during a DDoS flooding attack.

- tSB and FD - Although this feature set did not perform as well in the experiment compared to the ltSB and FD feature set, work by Braga *et al.* suggests that considering the number of packets as well as the number of bytes may improve performance [12].
- ltSB and FD - Four of the five top classifiers in terms of mean DR used this feature set.

The list below presents the methods and feature sets that were not selected along with justification:

- LDA - Five of the six LDA-based classifiers ranked in the bottom five in regards to mean DR.
- Decision Tree - We cannot guarantee that the Adequacy Condition will hold with network data.
- tSB and tSP - This feature set contains features that are redundant. Roughly speaking, the number of packets in unidirectional flow is linearly proportional to the the number of bytes in said flow. This relationship holds if you assume that all packets adhere to the standard length of a packet[3].
- tSB and lFD - Logging the FlowDuration feature did not improve the mean DR in the experiments. There was some success in terms of mean FPR as two of the five lowest mean FPRs used this feature set. However, those two classifiers exhibited poor mean DR (0.000043 and 0.76).
- SBpP and DBpP - Poorer mean DR compared to other features sets. The derived nature of the features within did not appear to improve results.

[3]The maximum transmission unit of an Ethernet frame is typically 1500 bytes.

3.5 Second Classifier Selection Experiment

This section presents the results from the second classifier selection experiment. This experiment used the results from initial classifier selection experiment to develop and test a second set of classifiers in an off-line scenario. The purpose of this experiment was to select three classifiers to deploy on a physical network testbed for on-line experimentation. The rest of the section will explore the components of the classifiers used, the experimental setup and the findings.

3.5.1 Second Generation Classifiers

The second generation classifiers consisted of five classification methods and five feature sets. Unlike the first generation classifiers, the feature sets used for the second generation classifiers contained more than two features. The classification methods used here were vetted as a result of the initial classifier selection experiment. These are listed below as a reminder:

1. QDA
2. SVM
3. KNN
4. Naive Bayes
5. Random Forest

Features

The features used to form the new feature sets are shown in Table 3.13 below. The new features are labelled with a $*$.

Feature	Description
totalSourceBytes	The number of bytes sent by the source.
log(totalSourceBytes)	The base 10 logarithm of totalSourceBytes.
totalDestinationBytes	The number of bytes sent by the destination.
totalSourcePackets	The number of packets sent by the source.
∗log(totalSourcePackets)	The base 10 logarithm of totalSourcePackets.
∗totalDestinationPackets	The number of packets sent by the destination.
FlowDuration	The duration of a bidirectional flow in seconds.

Table 3.13: Features used to build feature sets for the second generation classifiers.

Feature Sets

Five feature sets were built using the features table Table 3.13. The feature sets used for the first generation classifiers were made explicitly for exploratory purposes. The feature sets seen below however were made using the lessons learnt from the initial classifier selection experiment. Therefore, the feature sets below include justification for their formation as well as the feature set that it was based on.

1. totalSourceBytes, totalSourcePackets and FlowDuration - Based on totalSourceBytes and FlowDuration feature set. Braga *et al.* use both packets and bytes of a flow as they argue that packets in a DDoS flooding attacks have smaller payloads than packets from normal flows of traffic [12].

2. log(totalSourceBytes), totalSourcePackets and FlowDuration - Based on log(totalSourceBytes) and FlowDuration feature set. The log(total

SourceBytes) and FlowDuration feature set performed well in the initial experiment so it was augmented with the totalSourcePackets feature using the justification above.

3. log(totalSourceBytes), log(totalSourcePackets) and FlowDuration - Based on log(totalSourceBytes) and FlowDuration feature set. Taking the base 10 logarithm of the totalSourceBytes feature improved the detection rate in the initial experiment. The decision was made to take the base 10 logarithm of the totalSourcePackets feature to investigate if a similar effect can be found.

4. totalSourceBytes, totalDestinationBytes and FlowDuration - Based on totalSourceBytes and totalDestinationBytes feature set. DDoS attacks can be characterised by a particular volume of traffic sent over a period of time. The FlowDuration feature was appended following this reasoning.

5. totalSourceBytes, totalSourcePackets, totalDestinationBytes, totalDestinationPackets and FlowDuration - Based on totalSourceBytes and totalDestinationBytes feature set. This feature set was devised to explore how a large feature set would affect the performance of each classification method.

3.5.2 Experimental Setup

The second generation classifiers were tested in a test harness implemented in Python[4]. This bares resemblance to the experimental setup used during the initial classifier selection experiment. In fact, the same number of repeats and the same folds were used to test the second generation classifiers. The only difference lies in the classifiers that were used for this experiment. Refer to Section 3.4.2 for more information on the setup of these experiments.

[4]https://github.com/bakkerjarr/NetTrafClassificationExploration

3.5.3 Results and Discussion

The results of the second classifier selection experiment were used to select classifiers to be evaluated in an on-line networking environment. For the purposes of this experiment, the mean DR, mean FPR and mean f-measure statistics were calculated for each classifier. The inclusion of the f-measure statistic was for the purpose of selecting classifiers; this will be explored in more detail later on. The rest of this section will use tables 3.14, 3.15, 3.16 and 3.17 for the discussion of the results. These tables present a summary of the results; see Appendix C for the DRs, FPRs and f-measures for all classifiers. The tables below use the following abbreviations:

- tSB: totalSourceBytes
- ltSB: log(totalSourceBytes)
- tDB: totalDestinationBytes
- tSP: totalSourcePackets
- ltSP: log(totalSourcePackets)
- tDP: totalDestinationPackets
- FD: FlowDuration

Mean DR

Table 3.14 sorts the results for the second generation classifiers in descending order with regards to mean DR. The most prevalent feature set contained the ltSB, ltSP and FD features. This was used by the Naive Bayes, QDA and SVM methods.

The five feature sets shown in Table 3.14 are based on the log(totalSourceBytes) and FlowDuration feature set from the initial classifier selection experiment. The purpose of appending an extra feature to the feature set was to observe changes in the detection rate of classifiers.

The second generation classifier with the fifth highest mean DR (SVM with ltSB, ltSP and FD) had a mean DR of 0.9893688. This classifier has a higher mean DR than the third highest first generation classifier when they are ranked by mean DR; Naive Bayes with ltSB and FD, and a mean DR of 0.989364833. These results indicate an increase in detection performance when the feature set is augmented.

Method	Features	Mean DR	Mean FPR
Naive Bayes	ltSB, ltSP and FD	0.990108069	0.029333478
QDA	ltSB, tSP and FD	0.990000645	0.027850928
QDA	ltSB, ltSP and FD	0.989906291	0.014991284
Naive Bayes	ltSB, tSP and FD	0.989756706	0.024335124
SVM	ltSB, ltSP and FD	0.9893688	0.012388389

Table 3.14: Five best performing second generation classifiers, sorted by mean DR.

Table 3.15 sorts the worst results for the second generation classifiers in ascending order with regards to mean DR. These results rank more favourably compared to the five worst first generation classifiers depicted in Table 3.10, as only three of the bottom five second generation classifiers have a mean DR smaller than 90%. Besides this, the most interesting feature of the table below is that four classifiers utilised the SVM method.

Method	Features	Mean DR	Mean FPR
SVM	tSB, tDB and FD	0.725160172	0.001547747
SVM	tSB, tSP, tDB, tDP and FD	0.726826225	0.001583319
SVM	tSB, tSP and FD	0.738530663	0.001656662
SVM	ltSB, tSP and FD	0.929109403	0.004674993
KNN	ltSB, tSP and FD	0.935423086	0.005043936

Table 3.15: Five worst performing second generation classifiers, sorted by mean DR.

Mean FPR

Table 3.16 sorts the results for the second generation classifiers in descending order with regards to mean FPR. The most common denominator is the SVM method. The third SVM-based classifier (with the tSB, tSP and FD) had a mean FPR of 0.001656662, this result is smaller and therefore better than the first generation classifier with the smallest mean FPR; SVM with tSB and tDB, and a mean FPR of 0.001744573. Similarly to the mean DR, it appears that the inclusion of more traffic features can be used to improve mean FPR.

It is important to note the trade-off between mean FPR and DR despite these improvements. The first three classifiers in Table 3.16 (showing low mean FPR) were also the first three classifiers in Table 3.15 (showing low mean DR). This shows that there is a trade off between classifiers that can reliably detect attacks and those that tend to not misclassify normal traffic.

Method	Features	Mean DR	Mean FPR
SVM	tSB, tDB and FD	0.725160172	0.001547747
SVM	tSB, tSP, tDB, tDP and FD	0.726826225	0.001583319
SVM	tSB, tSP and FD	0.738530663	0.001656662
Random Forest	tSB, tSP, tDB, tDP and FD	0.948062485	0.002653073
Random Forest	tSB, tDB and FD	0.947373016	0.002765957

Table 3.16: Five best performing second generation classifiers, sorted by mean FPR.

Table 3.17 sorts the results for the second generation classifiers in ascending order with regards to mean FPR. The most obvious pattern is that four Naive Bayes-based classifiers had poor mean FPRs compared to the other classifiers. Strangely, the least effective classifiers in terms of mean FPR was also the most effective in terms of mean DR. This was followed by the second and third least effective classifiers in terms of mean FPR being ranked second and fourth most effective in terms of mean DR respectively. This result reiterates the trade-off between the DR and FPR of a classifier. It highlights the need for a method of selecting classifier that considers both true and false positives.

Method	Features	Mean DR	Mean FPR
Naive Bayes	ltSB, ltSP and FD	0.990108069	0.029333478
QDA	ltSB, tSP and FD	0.990000645	0.027850928
Naive Bayes	ltSB, tSP and FD	0.989756706	0.024335124
Naive Bayes	tSB, tSP and FD	0.962656107	0.022301342
Naive Bayes	tSB,tSP, tDB, tDP and FD	0.988954563	0.019051931

Table 3.17: Five worst performing second generation classifiers, sorted by mean FPR.

3.5.4 Final Classifier Selection

The vetting of feature sets and classification methods for the initial classifier selection experiment utilised knowledge of the methods and the results from the experiment. At this point however, it is necessary to utilise a different selection method for classifiers to be deployed within a networking environment. The results from the second classifier selection experiment showed that three classifiers ranked within the top five mean DR were also ranked within the five worst mean FPR. Interestingly enough, the three worst classifiers in terms of mean TPR were also the three best when ranked by mean FPR.

This situation poses the question: How does one choose a classifier based on its performance? The results have shown that selecting classifiers based on their DR or FPR leads to trade-offs in either metric. Considering other measures such as precision and recall may be a solution however there is still the issue of deciding how to weight each measure against each other.

The method used to select three classifiers involved ranking the second generation classifiers by their mean f-measure results. This choice was made in part to avoid bias and trade-offs when using the mean DR and FPR of the classifiers. With this being said, it is fair to note that because the f-measure statistic is an algebraic formula, it subject to favouring classifiers with particular values for precision and recall (also known as DR). Despite this, the f-measure statistic considers other measures of classifier performance in a way that is well understood within the domain of statistics.

The f-measure statistic has been used to compare and rank the performance of classifiers within the domain of network traffic classification [48]. The context of the referenced research, which manifests itself in an application named NeTraMark, is important to note. The purpose of this thesis was not to contribute to research within the domain of machine learning. Instead, the target for the contributions lie within the domain of DDoS attack detection and network traffic classification using SDN. For this reason it sufficed to use a well understood metric such as f-measure as an optimal solution was not needed.

Table 3.18 ranks the second generation classifiers by their f-measure statistics. Selecting the three best performing classifiers from the table below would have resulted in three classifiers that use the Random Forest method. Observing the table further shows that the methods tend to cluster to one another. As a result, the three best performing classifiers with *unique* methods were chosen. This ensured that different methods could be evaluated on the physical network testbed to obtain more breadth in the results.

Using the described selection process against the results in Table 3.18 yields the following three classifiers:

- Random Forest with the totalSourceBytes, totalSourcePackets, totalDestinationBytes, totalDestinationPackets and FlowDuration feature set.

- k-Nearest Neighbours with the totalSourceBytes, totalDestination-

Bytes and FlowDuration feature set.

- Support Vector Machine with the log(totalSourceBytes), totalSourcePackets and FlowDuration feature set.

Method	Features	f-measure
Random Forest	tSB, tSP, tDB, tDP and FD	0.954795172
Random Forest	tSB, tDB and FD	0.953657985
Random Forest	tSB, tSP and FD	0.94864049
Random Forest	ltSB, tSP and FD	0.948592306
Random Forest	ltSB, ltSP and FD	0.948583216
KNN	tSB, tDB and FD	0.948387955
KNN	tSB, tSP, tDB, tDP and FD	0.948385248
KNN	ltSB, ltSP and FD	0.943749165
KNN	ltSB, tSP and FD	0.932025381
SVM	ltSB, tSP and FD	0.931058739
KNN	tSB, tSP and FD	0.921951716
QDA	tSB, tSP, tDB, tDP and FD	0.915662191
SVM	ltSB, ltSP and FD	0.913524809
QDA	tSB, tSP and FD	0.911904559
QDA	tSB, tDB and FD	0.911523632
QDA	ltSB, ltSP and FD	0.89844222
Naive Bayes	tSB, tDB and FD	0.884021726
Naive Bayes	tSB, tSP, tDB, tDP and FD	0.878745778
Naive Bayes	tSB, tSP and FD	0.863597597
Naive Bayes	ltSB, tSP and FD	0.847633059
SVM	tSB, tSP and FD	0.838201025
SVM	tSB, tSP, tDB, tDP and FD	0.830928575
QDA	ltSB, tSP and FD	0.830602444
SVM	tSB, tDB and FD	0.83005504
Naive Bayes	ltSB, ltSP and FD	0.822862165

Table 3.18: Second generation classifiers sorted by f-measure.

3.6 Chapter Summary

This chapter presented an investigation into various statistical classifiers. The purpose of the investigation was to determine three classifiers that could be deployed on a physical network testbed to detect DDoS attacks from within the ISCX 2012 DDoS dataset. The investigation consisted of two off-line experiments where the results were used to inform selection making processes. As a result of the second selection making process, the following classifiers were selected to be integrated with *nmeta2* and deployed on a physical network testbed:

- Random Forest with the totalSourceBytes, totalSourcePackets, totalDestinationBytes, totalDestinationPackets and FlowDuration feature set.
- k-Nearest Neighbours with the totalSourceBytes, totalDestinationBytes and FlowDuration feature set.
- Support Vector Machine with the log(totalSourceBytes), totalSourcePackets and FlowDuration feature set.

Chapter 4

nmeta2 Classifier Integration

This chapter presents the modifications that were made to the *nmeta2* system to support the detection of DDoS attacks using the statistical classifiers that were selected at the end of Chapter 3. Key components of the *nmeta2* system that were leveraged will be explored. Details on the modifications that were made will then be described.

This chapter does not set out to justify the design decisions that were made by following the code line-by-line. It instead paves the way between the classifier selection process seen in Chapter 3 and the subsequent evaluation of the classifiers on a physical network testbed. The latter is presented in Chapter 5.

The modified *nmeta2* system provided a platform for evaluating the statistical classifiers in a network environment using SDN and OpenFlow. More specifically, the evaluation set out to test the classifiers in two situations. In the first situation, the DPAE needed to gather flow information by processing each packet that was forwarded to a switch on the data plane. In the second situation, the DPAE needed to periodically sample packets from the switch in order to gather flow information. This chapter shows how the modifications to the *nmeta2* system realise this.

4.1 Key nmeta2 System Components

The *nmeta2* system consists of two separate applications. The first application, simply called *nmeta2*, is a Ryu controller application that manages OpenFlow 1.3 compliant switches. The second application, called *nmeta2dpae*, is designed to run on a machine that is separate from the controller and perform traffic classification on bidirectional flows. Both applications communicate with one another to configure policies and relay classification results. The system architecture for *nmeta2* is illustrated in Figure 4.1. The relationships described in sections 4.1.1 and 4.1.2 are depicted in this figure. The rest of this section will not explore the internals of each application in depth, instead it will cover a selection of points that aid the understanding of the modifications that were made to the system.

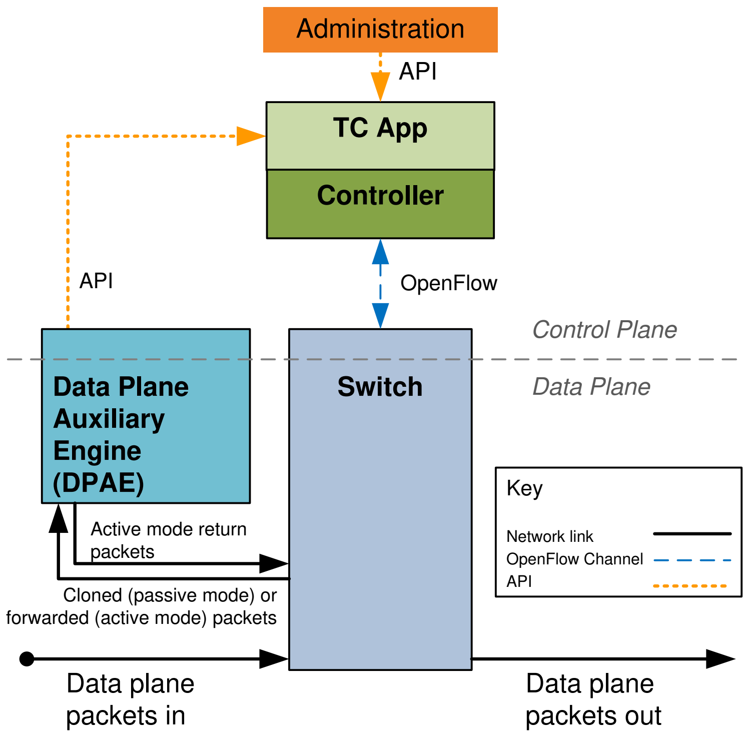

Figure 4.1: System architecture of the *nmeta2* traffic classifier (TC). Diagram sourced from [18].

4.1.1 nmeta2 Components

This application is designed to run on the Ryu controller. First and foremost, it manages OpenFlow 1.3 compliant switches that have connected to the controller by configuring their respective flow tables. This is used to forward packets through a network, be it to another switch, a host, the controller or to a DPAE.

The second responsibility of the *nmeta2* controller application is to manage one or more DPAE. Configuration information is sent to a DPAE to inform it of its expected behaviour as part of a connection protocol. The controller and DPAE then maintain connectivity throughout the DPAE's uptime. This is necessary for the controller to maintain global knowledge of the classifications that have been made by one or more DPAE.

The relationship between the controller and DPAE is important for the management of network traffic. Once classifications have been made by the DPAE and the controller has been informed, QoS treatment may be enforced on the switches for example. The controller also has the ability to prevent flows from being sent to the DPAE when requested. This feature is useful as it allows the network operator to determine how much traffic can be sent to the DPAE for classification.

4.1.2 nmeta2dpae Components

This application is designed to be executed on a host that has the sole purpose of classifying network traffic. Switches forward packets to the DPAE host to be parsed and classified. Classification results are then sent to the controller. The relationship between the controller and DPAE applications is discussed above.

The controller can instruct a DPAE to operate in one of two modes. While in *active* mode, all data plane traffic that arrives at a switch is forwarded to a DPAE instead of being of being forwarded to its destination. Once a DPAE has processed a packet, it gets sent back to the switch where it is then forwarded towards its destination. In contrast, *passive* mode informs the DPAE not to send packets back to the switch once they have been processed. This occurs because the packet is cloned after it arrives at the switch. The switch forwards one copy to its destination and the other to the DPAE.

The DPAE maintains a database on the host during its operation. The Flow Classification in Progress (FCIP) database is used to store information on bidirectional flows of traffic. When the DPAE processes a packet, it harvests information pertaining to its source and destination and collects statistics such as byte and packet counts. This information is associated with a matching entry within the FCIP database. This information can be used to provide context to the packet and the flow that it is part of. The current DPAE implementation uses MongoDB to store FCIP database information.

4.2 Modifications to nmeta2

This section presents the changes made to the *nmeta2* code to support the chosen statistical classifiers[1]. The changes will be addressed by focusing on two key areas: flow treatment advice support and flow suppression. Incidentally, both modifications were made to the FlowTable class within the *nmeta2* file `switch_abstraction.py`.

4.2.1 Flow Treatment Advice Support

Flow treatment advice is a process where the controller deploys flow table entries to enforce flow treatment policies in response to a classification

[1]The forked repository containing the modifications used in this work can be found at `https://github.com/bakkerjarr/nmeta2/tree/jnb_ME`. The original implementation made by Matthew Haytes can be found at `https://github.com/mattjhayes/nmeta2`.

made by a DPAE. Flow treatment policies utilise QoS mechanisms within OpenFlow-enabled switches. This may only be performed on flows that are specified with the `main_policy.yaml` file. Non-matching flows still follow the flow treatment advice process but they do not receive QoS treatment.

The flow treatment advice process in the original implementation of *nmeta2* only supported TCP flows encapsulated by IPv4. This was extended to support flows that utilised ICMP and UDP, both encapsulated by IPv4. Modifying the logic within the `add_fe_tt_advised` method allowed the controller to support ICMP and UDP flows. It was necessary to modify this functionality even though traffic treatment was not used explicitly to handle DDoS attacks.

4.2.2 Flow Suppression

Flow suppression is a feature of *nmeta2* where the DPAE can make a request for packets matching a specific flow to no longer be forwarded to the DPAE for classification. The ability to temporarily prevent flows traffic from being forwarded to the DPAE can be utilised to implement packet sampling on a flow-level. The `add_fe_tcf_suppress` method required modifications to support this.

The `add_fe_tcf_suppress` method required similar modifications to the `add_fe_tt_advised` method mentioned previously. The original implementation of the method was only able to deploy flow table entries to suppress TCP flows encapsulated in IPv4. The logic within the function was modified to support ICMP and UDP as well.

Flow-level packet sampling was implemented by modifying the timeout behaviour for suppressed flows. The original implementation suppressed flows with flow table entries that utilised the idle_timeout field. This field is a feature of the OpenFlow 1.3 protocol that is used to specify the length of time (in seconds) before a flow table entry is removed if no matching packets are received [6]. This is not appropriate for sampling as the sampling rate would be at the mercy of the flow. In other words, the sampling rate would be non-deterministic.

The OpenFlow 1.3 protocol offers the hard_timeout field as an alternative option for flow table entry timeouts. This field indicates the maximum amount of time (in seconds) that a flow table entry can remain in a switch before it is removed. Changing the implementation to utilise the hard_timeout field instead meant that a deterministic sampling rate could be specified. This rate can be modified by changing a parameter in *nmeta2*'s `config.yaml` file: `suppress_idle_timeout`. The unit for this configuration parameter is seconds.

4.3 Modifications to nmeta2 DPAE

This section presents the changes made to the *nmeta2* DPAE code to support the chosen statistical classifiers[2]. The changes will be addressed by focusing on two key areas: the harvesting of bidirectional flow information and the integration of the classifiers themselves.

4.3.1 Flow Information Gathering

Section 4.1.2 described the purpose of the FCIP database. To recap, flow information is collected on a per-packet basis and is stored to a matching bidirectional flow entry in the FCIP database. It is possible to harvest flow information from this database during the classification process. The process of gathering flow information needed to be modified to collect the desired information for the three statistical classifiers.

[2]The forked repository containing the modifications used in this work can be found at `https://github.com/bakkerjarr/nmeta2dpae/tree/jnb_ME`. The original implementation made by Matthew Hayes can be found at `https://github.com/mattjhayes/nmeta2dpae`.

The original DPAE implementation only collected information on TCP flows encapsulated by IPv4. Directionality of a flow is computed by observing the TCP flags, however this determination may not be accurate as the process relies on capturing packets during the TCP three-way handshake. IP addresses are labelled as either *server* or *client* once a determination is made. The information that was originally collected and stored by the DPAE is displayed in the following list:

- *hash* - A hash that uniquely identified the bidirectional flow.
- *ip_A* - The IPv4 address of one of the hosts.
- *ip_B* - The IPv4 address of the other host.
- *port_A* - The TCP port number used by the host identified by *ip_A*.
- *port_B* - The TCP port number used by the host identified by *ip_B*.
- *proto* - The transport protocol (in this case TCP).
- *finalised* - Integer (0 or 1) that can be used to indicate whether or not this flow has been classified.
- *packet_count* - Integer counting the number of packets sent in either direction for this flow.
- *packet_timestamps* - List of packet timestamps. Updated until the flow is finalised.
- *tcp_flags* - List of TCP flags. Updated until the flow is finalised.
- *packet_lengths* - List of the sizes of the packets in bytes. Updated until the flow is finalised.
- *client* - IPv4 address of the host that was determined to be the client.
- *server* - IPv4 address of the host that was determined to be the server.

- *packet_directions* - Direction of the packet within the flow. Updated until the flow is finalised.
- *verified_direction* - Describes how the directionality of the flow was determined.
- *suppressed* - Number of packets in the flow at the time when the controller was instructed to stop forwarding packets within the flow to the DPAE.

This flow data was not broad enough to support the three statistical classifiers. Support for flows that use the ICMP and UDP transport protocols, and the collection of additional flow statistics was needed.

In order for the DPAE to collect flow statistics for flows that did not use TCP, specific flow classes were made for ICMP and UDP flows. This was supported through the creation of a parent flow class. This abstraction contained the general properties and features for a flow that can be shared. The support for more transport protocols was mirrored in the FCIP database by creating specific MongoDB collections for ICMP, TCP and UDP flows.

The three statistical classifiers each need to be aware of the amount of traffic sent in either direction of a bidirectional flow and the duration of the flow. The amount of traffic refers to the total number of packets and the sum of the packet lengths in bytes. The DPAE was originally built to monitor the amount of traffic for the entire flow, with no regard for separate statistics for each direction. Packet timestamps were only collected until a flow was noted as being finalised.

None of the existing flow statistic fields were modified to support the classifiers. New fields were created instead to ensure that existing functionalities were not damaged. Four fields were created to record the number of packets and bytes sent in either direction of a flow. As the directionality of a flow is determined when the first packet is processed by the

DPAE, the relative direction of each subsequent packet need only be determined and the respective counts updated.

Flow duration was not calculated and stored in the FCIP database directly. Instead, a field was created to store the timestamp of the most recent packet within the flow. As the timestamp of the first packet in a flow is recorded, the duration of a flow can be calculated by taking the difference of the latest and first timestamps.

Each new flow statistics field is updated for every packet that is processed by the DPAE. To be explicit, the new fields are:

- *latest_timestamp* - The timestamp of the most recent packet in the flow.
- *total_pkt_len_A* - Total number of bytes sent by the host identified by *ip_A*.
- *total_pkt_cnt_A* - Total number of packets sent by the host identified by *ip_A*.
- *total_pkt_len_B* - Total number of bytes sent by the host identified by *ip_B*.
- *total_pkt_cnt_B* - Total number of packets sent by the host identified by *ip_B*.

4.3.2 Classifier Integration

The DPAE offers functionality whereby custom classifiers can be implemented in Python without the need to include the class explicitly in the code. Once a class that represents a desired classifier has been implemented, the DPAE can be instructed (via the *nmeta2* controller application) to instantiate the classifier when the application is loaded. Scikit-learn classes that implement the Random Forest, KNN and SVM classification methods were encapsulated in separate classes that obey the syntax of custom classifiers. Using this methodology, the classifiers are created in the *TC* class[3] during application runtime.

Classifier Initialisation

The statistical classifiers begin their initialisation processes when they are instantiated. As mentioned previously, statistical classifiers must undergo a training or learning phase before classifications can be made. The initialisation process not only includes the training phase of a classifier, but also the necessary steps to load and prepare training data before training may begin. The implementation of the classifiers for the DPAE required the following Python packages:

- lxml - For loading and parsing the training data that was stored in XML formatted files.
- numpy - For creating and handling sklearn-compliant data structures.
- sklearn - For utilising Scikit-learn classification methods.

Classifier Predictions

The statistical classifiers predict the class of a bidirectional flow each time a matching packet is encountered. The features used by the classifiers have a notion of a source and destination which roughly map to the notions of client and server that are used by the DPAE. These notions typically represent who initiated a session, namely the source or client.

Given the DDoS attack context of the dataset being used, one cannot assume that an attack is commenced by the party who initiated the connection. Therefore when a classifier makes a prediction, the total number of bytes sent in the current direction will be used as the source bytes. This is done regardless of the label assigned to the direction of a flow.

[3]This class is contained within the file `tc.py`.

Consider hosts A and B, a flow that was initialised by host A to host B and the selected KNN classifier as an example. At some point in time, host B sends a packet to host A. For the flow to be classified by the KNN classifier data referring to the totalSourceBytes, totalDestinationBytes and FlowDuration is needed. As host A initiated the flow one may be tempted to use the host A's byte count for the totalSourceBytes feature. Given the aforementioned assumption however, this may not result in an attack being detected. Therefore we use host B's byte count for the totalSourceBytes feature and host A's byte count for the totalDestinationBytes feature.

Flow information is harvested for use in the classifiers using the objects instantiated from the flow classes mentioned earlier. An alternative approach would be to use an interface for performing data mining operations. This approach looks nice as it reduces the amount of code within the flow classes and may facilitate the caching of data in the future. However, this functionality is made redundant by flow classes. The necessary information for the classifiers is obtained when the packet is processed which already results in a connection being made to the MongoDB database. Creating another class to mine data when required would only result in more connections being made thus increasing the packet processing time.

4.4 Chapter Summary

This chapter presented the modifications that were made to the *nmeta2* system to support the three chosen statistical classifiers. The controller was extended to handle flow treatment advice messages from the DPAE that contained information on ICMP and UDP flows encapsulated by IPv4. Modifications were also made to support flow-level packet sampling.

The DPAE was extended to gather information on ICMP and UDP flows encapsulated by IPv4. Further extensions were made to gather more flow statistics as per the requirements of each statistical classifier. More importantly, the statistical classifiers were integrated into the DPAE application by leveraging the Scikit-learn Python package.

Chapter 5

Evaluation

This chapter presents an evaluation of the statistical classifiers that were selected in Chapter 3 using the modified *nmeta2* system shown in Chapter 4. The suitability of each statistical classifier when being deployed within a SDN/OpenFlow environment is assessed. This evaluation assesses the prediction performance, the execution performance and the DPAE host performance for each classifier. The experiments performed in the environment are described as on-line experiments, this is in contrast to the off-line experiments as seen in Chapter 3.

To reiterate, the chosen classifiers were:

- Random Forest with the totalSourceBytes, totalSourcePackets, totalDestinationBytes, totalDestinationPackets and FlowDuration feature set.

- k-Nearest Neighbours with the totalSourceBytes, totalDestinationBytes and FlowDuration feature set.

- Support Vector Machine with the log(totalSourceBytes), totalSourcePackets and FlowDuration feature set.

These will be referred to by the classification method (Random Forest, KNN and SVM) for the rest of this chapter.

5.1 Evaluation Method

This section describes the methodology used for the evaluation below. The network environment and tools used during the experiments are described. The scope of the measurements that were taken are also discussed. Finally, appropriate *nmeta2* configuration details are provided.

5.1.1 Network Environment

The testbed used for the evaluation was built using a hardware-based environment. Figure 5.1 illustrates the network topology and provides a high-level overview of the traffic sent between each device. The network can be sliced in two. The first slice was used for network traffic sent over the data-plane and the second for the control-plane and test control network.

The data-plane consisted of three hosts and a switch. A host was configured to replay network traffic (traffic source), a host configured to receive the traffic (sink), a host configured to run the *nmeta2* DPAE application (DPAE) and an OpenFlow 1.3 compliant Allied Telesis switch (AT-x930-28GSTX).

The control-plane and test control network connected all of the devices using a layer 2 Ethernet switch. This facilitated communication between the controller, switch and DPAE for the purposes of the *nmeta2* system, as well as the orchestration of experiments. This network also consisted of a host that utilised Ansible playbooks[1] to orchestra each experiment and a host that was configured to run the *nmeta2* controller application on Ryu.

[1]Documentation for Ansible can be found at `http://docs.ansible.com/ansible/`.

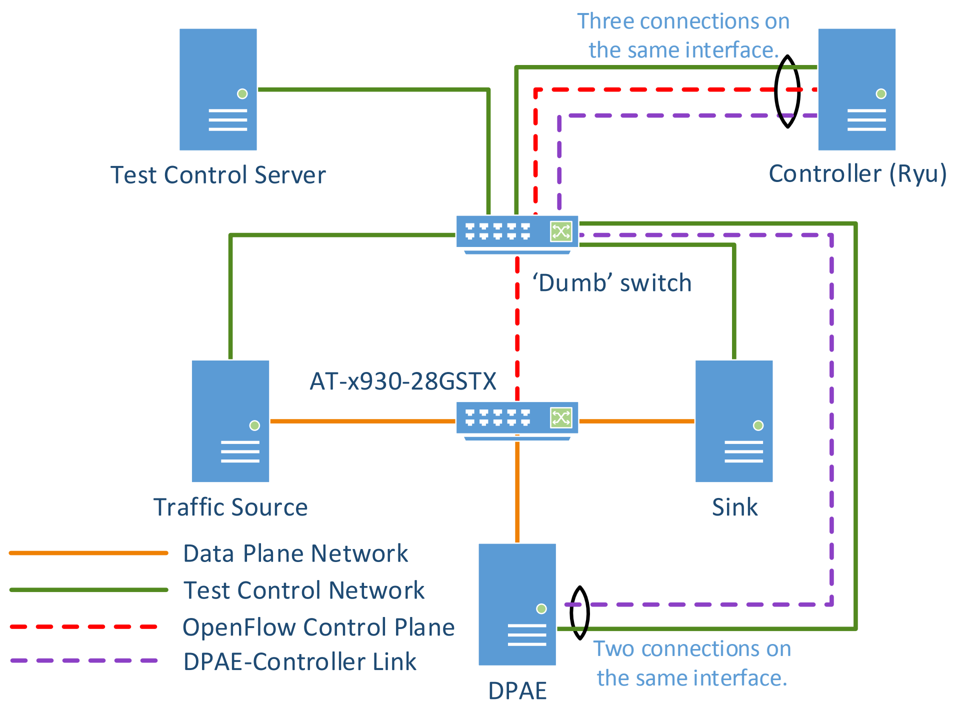

Figure 5.1: Physical network testbed topology used for the on-line experiments.

The following list contains information on each device within the network:

- Allied Telesis: AT-x930-28GSTX, OpenFlow 1.3 compliant switch.
- Allied Telesis: AT-FSW708, layer 2 Ethernet switch.
- Test Control Server: Raspberry Pi 3 running Raspbian GNU/Linux 8.0 (Jessie).
- Controller: Dell OptiPlex 9010 running Ubuntu 16.04 LTS (64-bit). Intel i7-3770 CPU (3.40 GHz $\times$ 8). 2 $\times$ 4 GB DDR3 RAM (1600 MHz each).

- Traffic Source: Dell OptiPlex 9010 running Ubuntu 16.04 LTS (64-bit). Intel i7-3770 CPU (3.40 GHz $\times$ 8). 2 $\times$ 4 GB DDR3 RAM (1600 MHz each).
- Sink: Dell OptiPlex 9010 running Ubuntu 16.04 LTS (64-bit). Intel i7-3770 CPU (3.40 GHz $\times$ 8). 2 $\times$ 4 GB DDR3 RAM (1600 MHz each).
- DPAE: Dell OptiPlex 9020 running Ubuntu 16.04 LTS (64-bit). Intel i7-4790 CPU (3.60 GHz $\times$ 8). 2 $\times$ 4 GB DDR3 RAM (1600 MHz each).

5.1.2 Classifier-DPAE Scenarios

Classifier-DPAE scenarios refer to the combinations of classifier and DPAE mode that were used as the experiments were running. Two DPAE modes were used: active and passive with flow-level packet sampling. The three classifiers and a control experiment were tested with the two DPAE modes. The control experiment did not use a classifier and therefore performed no statistical classification. This was used for establishing baselines for execution and DPAE host performance. In summary, there were eight classifier-DPAE scenarios.

5.1.3 Cross-validation

The experiments for selecting classifiers had the benefit of being able to be completed within a reasonable amount of time. The use of bidirectional flow metadata (counts of bytes and packets sent in either direction etc) instead of packet data greatly decreased the running-time of experiments. This allowed experiments to be conducted using 30-fold cross validation and be repeated 10 times. Roughly speaking, a set of such experiments where seven different classification methods need to be tested with six different sets of features could be be completed in two to three days. This produces 12600 sets of results.

The above benefit does not hold when testing classifiers using data on a live network. The ISCX 2012 DDoS dataset includes a PCAP file containing captured packets over a 24 hour period. Using the same rigour as before with the 24 hour packet trace would take 300 hours for a single classifier (one classification method with one set of features). Therefore it is necessary to reduce the number of folds and repeats for experiments when using the traffic data in the PCAP file.

The classifier selection experiments were re-run for the three chosen classifiers to investigate other options. The following three scenarios were tested against each classifier: 10-fold cross validation repeated 10 times, 5-fold cross validation repeated 10 times and 5-fold cross validation with no repeats. The purpose of these experiments was to investigate if there were significant variations in the coefficient of variation (COV) of the f-measure for each classifier when the number of folds and repeats were varied. As the flow metadata used for classifier selection represents the flows contained with the PCAP file, conclusions on classifier performance can be considered to be equivalent for both off-line and on-line classifier testing.

Table 5.1 shows use that there is no significant variation between experiments with the same classifier and a different number of folds when we round COV to two decimal places. Furthermore, reducing the number of repeats from ten to one does not adversely affect the COV. As a result, the tests on the physical testbed will be conducted using 5-fold cross validation with no repeats.

Classifier	k-fold	No. Experiments	COV f-measure
Random Forest	5	1	0.00099966701644763
Random Forest	5	10	0.00065650435919871
Random Forest	10	10	0.00082674768864716
Random Forest	30	10	0.00205295057699385
K Nearest Neighbours	5	1	0.00317414815060351
K Nearest Neighbours	5	10	0.00459777310723121
K Nearest Neighbours	10	10	0.00431762721635929
K Nearest Neighbours	30	10	0.00453013494114403
SVM	5	1	0.00125285766453559
SVM	5	10	0.00108220751797692
SVM	10	10	0.00118310590013836
SVM	30	10	0.00242391553407141

Table 5.1: COV of the f-measure for the chosen classifiers with differing values for k-fold and experiment repeats.

5.1.4 ISCX Dataset Replay

The ISCX 2012 DDoS dataset is a collection of network data that contains both DDoS (attack) and background (normal) network traffic. The network data is presented in a PCAP file (captured over a 24 hour period: 24GB in size, 34983042 packets) and a collection of three XML files summarising the flows within the PCAP file. One can assume that the network data in PCAP and XML files are statistically equivalent as the XML files have been derived using information from the PCAP file. Figure 5.2 provides an illustration for the relationship between the PCAP file and the collective XML data.

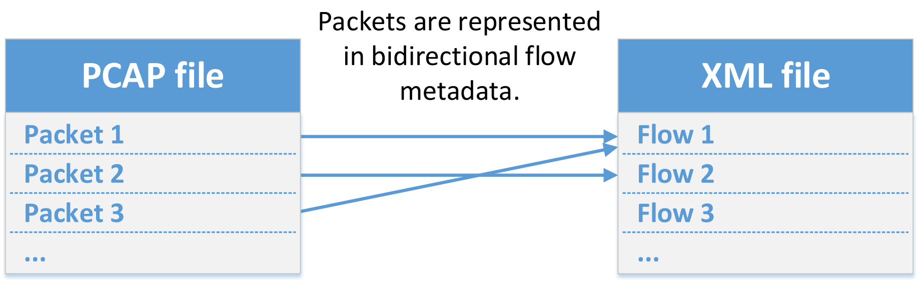

Figure 5.2: Relationship between PCAP and XML data within the ISCX 2012 DDoS dataset.

Flows within the XML files are bidirectional flows. The summarises within the XML files contain information such as: the IP addresses, port numbers (if applicable), timestamp of when the flow started and stopped, number of packets sent in either direction and the number of bytes sent in either direction.

The classifiers that have been implemented train their models using data derived from the XML flow summary files. Each classifier has no knowledge of an individual packet as a result. Instead, the classifiers discriminate classes of traffic based on the state of a bidirectional flow.

When evaluating the accuracy of a classifier it is commonplace to divide an entire dataset into training and testing sets. This means that the XML flow summaries need to be divided up into training and testing sets. It could follow that the PCAP file also be cloned and filtered such that each clone contained packets belonging to flows within the various testing sets. Instead of processing the 24GB PCAP file into different testing sets, the PCAP file can simply be replayed in its entirety.

Prior to each test we know what flows are part of the testing set. Therefore once a test has ended and the results have been collected, the data points that refer to flows within the training set can be removed. This leaves behind the data for flows within the testing set; which is data that would have otherwise been collected had the PCAP file been split earlier. Furthermore, as the flows that are used to train the classifiers are not formed from random packet samples from the PCAP file, selectively removing the results associated with training flows will not alter the distribution of the results. The results have merely been sanitised of data points that would have resulted in overfitting.

5.1.5 Intended Measurements

The measurements that were taken are described below. The scope of each measurement is also described. This was done for the three classifiers as well as a control experiment (no classifier). The control experiment presents a baseline for comparison for all measurements except the prediction performance and classifier initialisation time.

Classifier Prediction Performance

The prediction results made for each flow needed to be recorded and collected during the replay of the PCAP file. Once collected, the number of true positives, true negatives, false positives and false negatives were determined. Theses quantities allowed the mean DR, mean FPR and mean f-measure to be calculated for each classifier.

Classifier Initialisation Time

A distinction needed to be made in regards to the measurement of classifier training time. This could either refer to the time taken for the *nmeta2* DPAE class containing a classifier to train or the method used by the Scikit-learn module to train a model i.e. <classifier>.fit(). In order to get an idea of the impact a machine learning classifier has on a network environment, it is important to consider the time that it takes to load training data from file. Therefore the time that was measured was the time taken for a classifier to be in a state where it could start making predictions, i.e. its initialisation time.

Packet Processing Time

Similar to the measurement of classifier training time, a distinction needed to made with respect to the packet processing time. The measurement could either consider the time taken for a Scikit-learn module to make a prediction or for the *nmeta2* DPAE class containing a classifier to train. The latter was assumed given that the focus of this research was to use machine learning classifiers to detect DDoS attacks using SDN. As a result, the measurement not only considered the time taken for a classifier to make a prediction but also the time taken for a classifier to prepare network data for use in a classifier.

DPAE Host Performance

The DPAE host was monitored throughout each of the experiments. MOSP[2] is a program for monitoring the performance of a machine. Information such as the CPU utilisation and the amount of traffic sent and received by the host was recorded to a file. This information was gathered so that further insight into the performance of the classifiers could be made. MOSP was configured with a fetching interval of 1 second and was run throughout the duration of each experiment.

5.1.6 Data Pre-processing

The ISCX 2012 DDoS dataset PCAP file contains packets with the MAC addresses of the machines used within the capture environment. As the testbed being used for the evaluation of the classifiers does not contain those machines, it is necessary to pre-process the PCAP data such that it is compatible with the classifier evaluation testbed. The PCAP file was replayed using *tracereplay*[3] with the traffic being directed to the sink.

[2]Code can be found at `https://github.com/mattjhayes/mosp`.

[3]This program is part of *libtrace* which can be found at `https://github.com/wanduow/libtrace`.

The destination MAC addresses of the packets within the PCAP file were changed to the MAC address of the sink's data-plane network interface card (NIC). This served to direct the packets to a termination point. This does not change the statistical distribution of each bidirectional flow within the PCAP file. Care was taken to not change any packets with the broadcast destination MAC address as this would have altered the behaviour of flows. Furthermore, there was no issue of conflicting IP addresses as the subnet for the data-plane of the testbed does not match any private IP address subnet within the data set [50].

5.1.7 Measurement Tools

Relevant tools that were used to take readings are mentioned below.

Execution Time

The *nmeta2* system is built following Python version 2. One method of measuring execution time of a process would involve the use of the *timeit* module [52]. This module can be used to measure the execution time of functions. This function was not appropriate for use in these experiments as there was no need to measure the time spent on lines of code within the functions of interest. As a result it was necessary for to measure execution time using the *time* module [53].

The *time* module offers two functions for fetching the current time: `time.time()` and `time.clock()`. The return value of `time.time()` is the number of seconds since the epoch expressed as a floating point number. This representation of time can also be referred to as 'wall-clock' time. The return value of `time.clock()` differs from operating system to operating system. On Unix operating systems it returns the current processor time in seconds as a floating point number. It does not return the processor time in seconds on Windows operating systems however. It instead returns the elapsed number of wall clock seconds since the first call to the function. It is important to note that the precision of the value returned by `time.clock()` is higher than the return value from `time.time()`.

Another option is the use of the *cProfile* module [54]. This module utilises C code to profile Python code. The generated profile contains statistics describing how often parts of a program were executed and for how long. The module can be used to profile an entire program or sections of code within the program. The *cProfile* module was too heavyweight for the purposes of this evaluation as there was no need to measure the time spent in individual functions within a section of code.

The choice of timing method was motivated by the setting of this research. The measurements concerning the execution time of the training and prediction phases of classification should consider the stress of the machine running the system. Such measurements reflect the effects of receiving traffic, especially DDoS attack traffic, on machine learning-based statistical classifiers. Therefore the chosen method of measuring execution time was `time.time()`. In contrast, the `time.clock()` function only measures the time that the code was running on the processor. It does not capture any moments when the program was interrupted by the operating system to perform other tasks due to the arrival network traffic. Despite this, this second method was also used as a point of comparison.

Prediction Results

The DPAE distinguishes bidirectional flows by using a five-tuple. Using this definition, the bidirectional flows within the dataset change their behaviour over time. Traffic with a particular five-tuple may be non-malicious at one moment and malicious at a later time. This demonstrates the reuse of addresses within a five-tuple.

Each prediction result was written to file. Attached to each result was the time the prediction was made, the five-tuple of the flow and the execution times of the prediction. As the PCAP file contains 34983042 packets, one would expect 34983042 predictions to have been made at the most when running the DPAE in active mode. The prediction results were correlated with the flow summaries within the XML files for each fold to calculate the prediction performance statistics.

5.1.8 nmeta2 System Configuration

The *nmeta2* controller and DPAE applications allow the network operator to configure their behaviour. This is enabled through YAML formatted configuration files. The configurations used during the evaluation experiments are given below.

nmeta2 Controller Application

The file *main_policy.yaml* describes the classification behaviour for the entire *nmeta2* system. A network operator can specify what information is collected through the definition of policies. The following settings were changed from the original values as pulled from GitHub:

1. Identity information was not collected by setting the *arp*, *lldp*, *dns* and *dhcp* fields to 0.

2. A port set was created to specify that traffic only need be classified if it arrives on the switch port of the load generator.

3. A traffic classification rule set was created to specify that all traffic should be classified against the implemented classifiers.

4. A traffic classification policy was created to apply the previously created traffic classification rule set on the aforementioned port set and set the DPAE mode to either *active* or *passive*.

The file *config.yaml* is used to configure the *nmeta2* controller application. For the purposes of the experiments, the only item that was edited was the *suppress_idle_timeout* field during the experiments where the DPAE was run in passive mode. This field denotes the suppression time for sending traffic to a DPAE from a switch. This was used to implement flow-level packet sampling. This was set to 1 during the passive mode experiments. This value was set to 0 during the active mode experiments as all traffic was being forwarded to the DPAE from the OpenFlow switch.

The choice of 1 second for the flow suppression time was chosen as it was the smallest value. This provided the DPAE with the maximum amount of information from a sampling configuration. However this will result in the largest amount of control plane traffic between the controller and DPAE for a sampling configuration (compare with value set to 5 seconds for instance). Despite this, 1 second was chosen in order to improve the likelihood of better classifier performance.

nmeta2 DPAE

The file *config.yaml* is used to configure the *nmeta2* DPAE application. The following settings were changed from the original values as pulled from GitHub:

1. The field *nmeta_controller_address* was set using URL syntax to point to an HTTP endpoint on the *nmeta2* controller application.

2. The field *sniff_if_names* was set to the interface on the DPAE machine connected to the switch.

3. The field *console_log_enabled* was set to the 0 to disable console logging during experiments. This style of logging was not necessary as the experiments were orchestrated using Python scripts and Ansible playbooks. This did not disable logging to syslog.

Default parameters were used for each classification method as the purpose of the research was not to optimise classifier parameters. The only parameter that was changed was the "algorithm" parameter for the KNN classifier. This was set to "kd_tree" and has the effect of storing the training data in a k-d tree. This option promises to decrease the prediction time. The leaf size was left at the default value.

Other Items

After each fold was finished, the MongoDB logfile located at `/var/log/mongodb/mongodb.log` was deleted. Allowing this file to grow too big prevented test data from being collected as the available space on the filesystem was exhausted.

5.2 Measurement Hypotheses

This section presents the hypotheses for the results that were used to evaluate the classifiers.

5.2.1 Classifier Prediction Performance

The expectation was for the ranking of each classifier in terms of f-measure to match the ranking of the three classifiers in the second classifier selection experiment. The places Random Forest classifier first, followed by the KNN and SVM classifiers. The values for each f-measure value were not expected to match between the selection experiments and the hardware testbed experiments. These values were expected to be lower in fact, as the classifiers in the selection experiments were provided with perfect information. That is, information that is not subject to variation caused by networking equipment.

A Receiver Operator Characteristics (ROC) curve is an evaluation tool for assessing the performance of a classifier [55]. The DR is plotted against the FPR for various classifier experiments where the parameters of the classifier are changed. The resulting curve depicts the performance of the classifier as it is tuned. This tool was not appropriate for this evaluation as the parameter that was changed for each classifier was the the DPAE mode, of which there were two. This was too few parameter changes for a ROC curve to be used effectively.

5.2.2 Classifier Execution Time Performance

Hypotheses must be given for the classifier initialisation and packet processing times. These are given below.

Classifier Initialisation Time

The initialisation time for each statistical classifier was expected to be dependent on the method behind each classifier. It can be assumed that the time taken to load the training data from file will consistent for each classifier as the files were all the same size. The KNN classifier should have the lowest initialisation time as it is a lazy learner. The SVM classifier should have the highest initialisation time as the method needs to perform several calculations using support vectors to find an optimally placed decision boundary.

Packet Processing Time

The first expectation was that the mean packet processing time for the control experiments (i.e. when using the empty classifier) was going to be smaller than the scenarios where an actual classifier was being used. The KNN classifier was expected to exhibit the highest mean packet processing time as it is a lazy learner. In terms of the shortest mean packet processing time for the classifier, the belief was that the SVM classifier will perform better than the Random Forest classifier. This is because the Random Forest prediction method involves a search through a tree-based data structure to determine classes.

5.3 Classifier Prediction Performance

This section discusses the results regarding the prediction performance of the three statistical classifiers. The results will first be presented with comparisons being made to the respective hypothesis (Section 5.2.1) and the results of the second classifier selection experiment. This is followed by a closer examination into the predictions.

5.3.1 Prediction Results

The prediction performance measures used for the second classifier selection experiment were also used for on-line experiments. The DR and FPR was calculated for each classifier alongside the f-measure to rank the classifiers. Table 5.2 presents the results for each classifier and includes both DPAE modes.

Classifier	DPAE	Mean DR	Mean FPR	Mean f-measure
SVM	Active	0.144378318	0.002040893	0.021384715
Random Forest	Active	0.003937928	0.002657403	0.000594661
SVM	Passive	0.000749199	0.002534000	0.000114200
KNN	Passive	0.000700116	0.000119214	0.000104720
KNN	Active	0.000273643	0.000092519	0.000040693
Random Forest	Passive	0.000053412	0.002272015	0.000007984

Table 5.2: Prediction statistics for the deployed statistical classifiers - ranked by mean f-measure.

The first hypothesis for the prediction results stated that the f-measure ranking would follow the ranking provided at the end of the second classifier selection experiment. This order was Random Forest, KNN and SVM. Table 5.2 shows that was not the case regardless of the DPAE mode. The active mode experiments yielded the order of SVM, Random Forest and

KNN. The passive mode experiments yielded the order of SVM, KNN, Random Forest.

The second hypothesis for these results stated that the f-measure values would be lower than their counterparts in the second classifier selection experiment. This hypothesis is confirmed by the results in Table 5.2. It is worth pointing out that the difference in the orders of magnitude is larger than one might expect. The f-measure statistics used to rank the classifiers during the selection process suggested that the classifiers should have yielded promising results.

The mean DR for all classifiers was also low. The highest mean DR was given by the SVM classifier when the DPAE was run in active mode. This value suggests that it could accurately identify roughly 14% of the malicious flows. The mean FPR results showed more promise as lower values indicate better performance in this context. The smallest mean FPR was given by the KNN classifier when the DPAE was run in active mode.

The results suggest that having the DPAE in active will produce more palatable results. Of the six classifier-DPAE scenarios, two of the active-based classifiers were in the top three when ranking the classifiers by mean f-measure. Furthermore, the lowest mean FPR was given by the KNN classifier when the DPAE was run in active mode. The disadvantage of using the DPAE in passive mode with packet sampling is that predictions are made with less information. Active mode puts the DPAE into a position where it has a closer relationship with the flows of traffic. Thus the DPAE is closer to having perfect information, similar to the off-line experiments used for classifier selection.

5.3.2 Prediction Accuracy

The prediction results presented in Table 5.2 show that the classifiers were not successful in detecting the malicious flows in a DDoS attack. This conclusion was made on the basis of the f-measure, however the results were also examined from a different point of view: accuracy. Table 5.3 ranks the classifiers by their accuracy.

Classifier	DPAE	Mean DR	Mean FPR	Mean Accuracy
SVM	Active	0.144378318	0.002040893	0.93468575
KNN	Active	0.000273643	0.000092519	0.92575702
KNN	Passive	0.000700116	0.000119214	0.92521195
Random Forest	Passive	0.000053412	0.002272015	0.92318754
Random Forest	Active	0.003937928	0.002657403	0.92252053
SVM	Passive	0.000749199	0.002534000	0.92159593

Table 5.3: Prediction statistics for the deployed statistical classifiers - ranked by mean accuracy.

Each classifier was roughly 92% to 93% accurate which was in itself was a very promising result. However such measurements should be taken with a grain of salt. Equation 2.3 showed how to calculate accuracy using the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Immediately one can see that the calculated value is governed by the TP and TN terms in the numerator. This can be used to explain the high accuracy of the classifiers despite their low f-measure.

The high accuracy was attributed to the low mean FPR. The low mean DR for each classifiers informs us that the number of TP predictions must have been smaller than the number of FN predictions. Similarly, the low mean FPR informs us that the number of TN predictions must have been larger than the number of FP predictions. Using these two pieces of information, one can infer that the high accuracy was due to a large number of TN predictions. This tells us that the classifiers were successful in correctly identifying non-malicious flows of traffic.

5.3.3 Examining the Collected Results

The results and observations presented in sections 5.3.1 and 5.3.2 paint a fairly bleak picture. The classifiers were not successful in detecting malicious flows within an live network environment. Despite this they can still be considered to be accurate as they appeared to correctly identify the non-malicious flows of traffic. Table 5.4 contains the mean prediction quantities (flows) which can be used to explain the aforementioned phenomena.

Classifier	DPAE	Mean TP	Mean TN	Mean FP	Mean FN	Mean N/A
Random Forest	Active	118	366873.4	975	29844.4	59547.6
KNN	Active	8.2	373966	34.6	29957	53392.6
SVM	Active	4326	373540.4	764.4	25637	53090.6
Random Forest	Passive	1.6	370154.8	842.6	29955.4	56404
KNN	Passive	20.8	367232.2	43.8	29642.6	60419
SVM	Passive	22.2	358570.2	910.6	29595.4	68260

Table 5.4: Mean prediction quantities (flows) for each statistical classifier - True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN) and No Answer (N/A).

Each classifier produced a large amount of TN predictions on average. When this is compared to the mean number of FP predictions we can understand why the mean FPR for each classifier was low. A similar comparison can be made with the mean number of TP and FN predictions. The significantly lower number of TP predictions resulted in a low mean DR. This is very apparent when the Random Forest classifier with the passive DPAE results are considered. The packet sampling approach resulted in 1.6 malicious flows being detected on average.

Included in Table 5.4 is a column named "Mean N/A". This column presents the mean number of flows that did not receive a prediction. The reasons for this will be discussed in Section 5.5.1. The worrying large number of flows that did not receive predictions is problematic for the prediction quantities. However this can be used to explain why the mean number of TP predictions was so low. It is possible that flows that should have been identified as malicious did not receive any prediction value at all. It is important to note that at this stage of the evaluation, this is only speculation.

5.3.4 Classifier Prediction Performance Summary

The three statistical classifiers produced results that defied the expectations given by the second classifier selection experiment. The low mean f-measure across all classifier-DPAE scenarios makes it difficult to determine what classifier performed the best. Despite this, the classifier-DPAE scenario with the highest f-measure was the SVM classifier with the DPAE configured in active mode.

The low mean f-measure statistics are matched with equally low mean DRs. However it is important to consider each classifier's mean FPR as well, which got no larger than roughly 0.27%. The accuracy of each classifier fell within the range of 92-93%. While being satisfactory results, it is important to not let this paint a false picture of the predictive performance of each classifier. The last point to note is the sizeable amount of flows that did not receive a prediction value.

5.4 Classifier Execution Performance

This section discusses the results regarding the execution performance of the three statistical classifiers. Three areas were explored to address this: the initialisation time of each classifier, the packet processing time of each classifier and the number of packets processed by each classifier. The second and third measurements were also collected for control experiments where the DPAE was not configured to perform any statistical classification. The resulting impact that statistical classification has on the DPAE portion can then be understood. The hypotheses for each set of results (Section 5.2.2) will be addressed where appropriate.

5.4.1 Classifier Initialisation Time

Section 5.1.5 defined classifier initialisation time as the time taken for a classifier to be in a state where it can make predictions. This includes the time taken for training data to be loaded from file as well as the time taken for the classifier to train. These results are displayed in Table 5.5.

Classifier	**DPAE**	**Mean Proc. Time (s)**	**Mean Wall Time (s)**
Random Forest	Active	5.6848638	6.258123016
KNN	Active	5.448394	6.004147053
SVM	Active	443.2090188	443.9620652
Random Forest	Passive	5.7031904	6.177779818
KNN	Passive	5.4808608	6.127974558
SVM	Passive	443.1051298	443.8416832

Table 5.5: Mean statistical classifier initialisation times - Processor (Proc.) and Wall time.

The hypothesis regarding classifier initialisation times successfully predicted what classifiers would be the quickest and the slowest. Both instances of the KNN classifier were the quickest to initialise within their respective DPAE mode scenarios. Furthermore, this holds across both the mean processor and wall time columns. This result is unsurprising as the KNN method is a lazy learner. Each KNN classifier was followed closely by the Random Forest classifiers. The SVM classifier took the longest to initialise, taking roughly 7 minutes and 23 seconds.

The difference between the mean processor and wall times for each classifier-DPAE scenario is fairly consistent. Some of this difference can be accounted for by the order in which time was recorded; processor then wall. These results suggest that the DPAE host was able to consistently concentrate on the initialisation process without being interrupted by other processes.

It is well understood that modern operating systems (OS) share processor time among numerous processes. A significant difference between the mean processor time and the mean wall time would indicate that modern OSs are not be suitable for use within a networking environment. An alternative option might involve the use of dedicated hardware running a unikernel; however, this would cost more to develop in a temporal and monetary sense.

5.4.2 Packet Processing Time

Section 5.1.5 defined packet processing time as the time taken for a classifier to gather the required information to make a prediction and then make the prediction itself. These results are displayed in Table 5.6.

Classifier	**DPAE**	**Mean Proc. Time (s)**	**Mean Wall Time (s)**
No classifier	Active	0.000512707	0.000974115
Random Forest	Active	0.004153738	0.004366805
KNN	Active	0.001187487	0.001447127
SVM	Active	0.000788909	0.001097702
No classifier	Passive	0.000480256	0.001844048
Random Forest	Passive	0.004201095	0.004400715
KNN	Passive	0.001138495	0.002002197
SVM	Passive	0.000740066	0.002011115

Table 5.6: Mean packet processing times - Processor (Proc.) and Wall time.

Three hypotheses were made regarding the packet processing times and two of these were confirmed. Firstly, the mean packet processing time when no classifiers were used was smaller than when a classifier was used. Secondly, the SVM-based classifiers had smaller packet processing times on average compared to the Random Forest-based classifiers.

The third hypothesis incorrectly predicted that the KNN-based classifiers would have the highest mean packet processing time. This was made on the assumption that a lazy learner would exhibit longer processing times.

A strange observation that can be made concerns the difference in times between the different DPAE scenarios. The mean processor time for each classifier scenario was higher when the DPAE was in active mode, except for the Random Forest classifier. This fits the assumption that an active DPAE is put under more stress as it must process all traffic being forwarded to its neighbouring switch. However, the same increase was not observed when using mean wall time. It is counter-intuitive that an active DPAE would increase the time spent on a processor instead of the time from the perspective of the OS.

The packet processing results for the Random Forest classifier indicate that it is the least suitable for deployment. Compared to the control experiment, the mean processor time increased by a factor of 10 (or 1000 microseconds). The SVM classifiers only increased the mean processor times by 200 microseconds. The longer packet processing time for the Random Forest classifier could be attributed to two things. The Random Forest classifier that was selected utilised a feature set that contained five features whereas the other two classifiers utilised three. In terms of packet processing time, a larger feature set means that more time must be spent preparing the required data before a classification can be made. Also, the Random Forest method relies on a search through a tree-like data structure.

5.4.3 Number of Predictions

The purpose of counting the number of predictions made by each classifier was to check for possible packet loss. This can be done by comparing the number of predictions that were made against the number of packets contained within the dataset's PCAP file. The current implementation of the DPAE means that they will never be equal however as the DPAE does not subject all packets to the entire classification pipeline. Packets that are part of DNS, DHCP, LLDP or ARP-based flows are not processed by custom classifiers. Baselines were provided by running the DPAE with no classifier. This provided the expected number of packets that could be processed by the classification pipeline.

Another way of detecting packet loss involves counting the packets that are received by the sink. This could be done as all of the packets sent on the data-plane should be forwarded to the sink, regardless of their type. Table 5.7 displays that the mean number of predictions that were made and the mean number of packets that were received by the sink for each classifier-DPAE scenario.

Classifier	**DPAE**	**Mean Pred.**	**Mean S. Pkt.**
No classifier	Active	22913939.4	24326964.6
Random Forest	Active	13182770.6	16099466.8
KNN	Active	21781835.4	23817711.8
SVM	Active	22798603	24116786.6
No classifier	Passive	22648513.2	20003418
Random Forest	Passive	13361685.6	20212835.4
KNN	Passive	24862596.8	19517048.2
SVM	Passive	23788578.2	19477950.4

Table 5.7: Mean statistical classifier predictions - mean number of predictions (Pred.) and mean number of packets received by the sink (S. Pkt.).

Key Features of the Results

The most glaring feature of these results is that the quantities are significantly less than the number of packets in the dataset's PCAP file. The largest mean number of packets received by the sink (24326964.6) was roughly 10 million fewer packets than the 34983042 contained within the PCAP file. Fewer predictions were expected to be made as not all packets were going to be subjected to statistical classification. Regardless, the lack of packets and predictions would offer a possible explanation for the poor prediction performance.

The second most glaring feature of the results is that three of the four passive-based scenarios had sinks that received fewer packets on average than the mean number of predictions that were made. An explanation for this can be given using the behaviour of the flow table pipeline used by *nmeta2*. The key principle for this explanation is that the last table in the pipeline is used for forwarding packets to their destinations and the preceding tables are used for handling packets before they are classified. One of these tables is used for copying packets, one copy to be forwarded to the DPAE and the other to be sent through the rest of the pipeline. The fact that the latter packet is still being processed means that it is possible for packets to be lost if the switch's buffer becomes full.

The above argument does not explain why the three classifiers made more predictions on average when flow-level packet sampling was used. Logic would dictate that packet sampling should result in fewer predictions being made, assuming that the packet arrival rate in each flow is less than the packet sampling rate.

This phenomena can be explained as follows. Assume that the mean number of predictions made for each classifier when the DPAE is in active mode is the processing capacity for the system. We know that there was a sizeable amount of packets that did not receive a prediction and logic dictates that sampling should result in fewer predictions being made. However it is possible the number of packets that are sampled from the switch is larger than the packet processing rate of the DPAE. This means that one would never observe fewer predictions when sampling traffic from the switch.

Other Observations

The order of both columns in Table 5.7 for the active DPAE scenario matches the order of both packet processing time columns in Table 5.6 for the same scenario. The same observation cannot be made for the passive scenario. By putting the DPAE into passive mode and giving the switch the responsibility to sample packets, the expected results have changed. The cause of this cannot be concluded given there are two contributing factors. However, the results suggest that switch-based packet sampling is not desirable. All traffic should ideally be forwarded to the DPAE for classification where sampling decisions can be made away from the data-plane.

The average number of predictions made by the Random Forest classifier is still less than the average number of packets received by the sink within the passive DPAE scenario. This is in contrast to the other set of experiments within the same DPAE scenario. This is to be expected as the mean packet processing times for the Random Forest method were higher than the other classifiers. Therefore it is unsurprising that fewer predictions were made.

5.4.4 Classifier Execution Performance Summary

The classifier execution results show that classifiers with shorter initialisation periods may not be desirable. The SVM classifier, despite having the longest initialisation time, had the least significant impact on the average packet processing time. These results also provide an insight into the reasons why some flows did not receive predictions. The average number of predictions that were made in each classifier-DPAE scenario was significantly lower than the number packets in the dataset's PCAP file.

5.5 DPAE Host Performance

This section examines the performance impact the statistical classifiers had on the DPAE host in terms of the amount of traffic sent and received by the DPAE host. Analysing this information provides details into why the classifiers produced the results seen in sections 5.3 and 5.4 as it shows how different traffic conditions can effect the classification process. In this case: traffic arriving at a *normal* rate and traffic from a DDoS attack.

The results consider the mean and maximum values. The mean is typically used to determine what one could expect on average. This does not adequately capture the extreme ends of the scale. Examining the maximum values over a period of time shows how the worse case scenario may change. This is important for researchers and network operators as it helps in understanding the upper bounds of a system, an understanding which may be used to build and maintain a resilient network.

5.5.1 DPAE Host NIC Impact

The DPAE acts as a sponge. By observing how it absorbs and releases packets one can draw conclusions regarding its performance. The DPAE used in these experiments was setup with one NIC connected to an OpenFlow switch and another connected to the control plane network (see Figure 5.1). The NIC connected to the OpenFlow switch receives packets from the switch to be classified. This interface is also used to forward packets back to the switch when the DPAE is in active mode. The NIC connected to the control plane network (abbreviated to contr.) is used by the DPAE to send classification information to the controller via a TCP session. Monitoring both NICs shows how the data and control planes are effected by statistical classifiers.

No Classifier

The first set of results to be explored concern the control experiments. By observing the DPAE when no statistical classification is being performed, the performance baseline can be established. Figures 5.3, 5.4, 5.5 and 5.6 will be referenced.

Figure 5.3 clearly shows the impact a DDoS attack has on the average number of packets received by a switch. It also highlights a discrepancy between the traffic received by the DPAE and the traffic forwarded back to the switch. Monitoring the traffic forwarded from the DPAE to the switch is important as it reveals how many packets can be processed by the DPAE. Active mode is distinguishable by its responsibility to forward packets back to the switch once they have been processed. Therefore we can conclude that the DPAE can process most packets in an active manner on average when no statistical classifiers are being run.

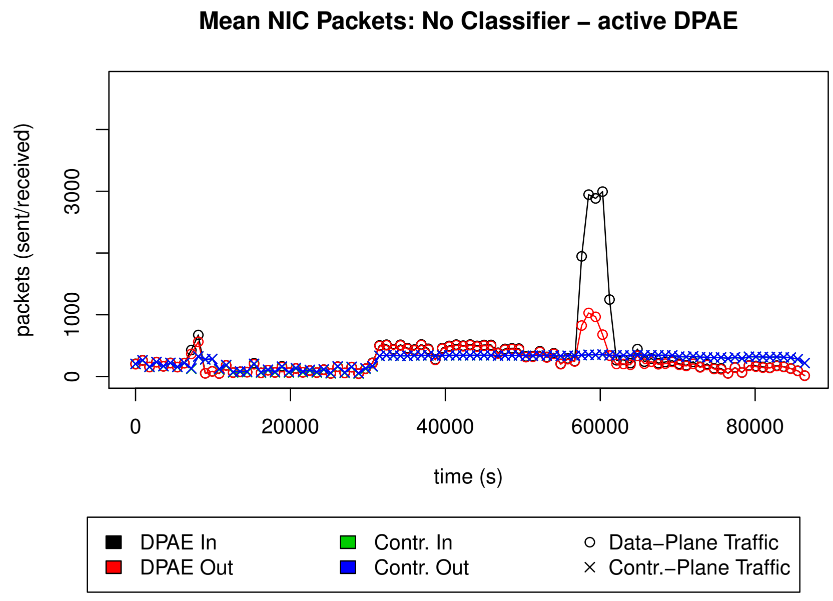

Figure 5.3: Active DPAE: Mean number of packets sent and received by the DPAE host during the experiments with no statistical classification.

Figure 5.4 shows the same discrepancy but on more occasions. This shows that the maximum amount of packets that can processed by the DPAE is around 1500 per second (as the readings were taken every second). This figure also accentuates the discrepancy mentioned in Figure 5.3. This represents a significant amount of packet loss on the DPAE. This accounts for the smaller number of predictions that were made and number of packets that were on received by the sink.

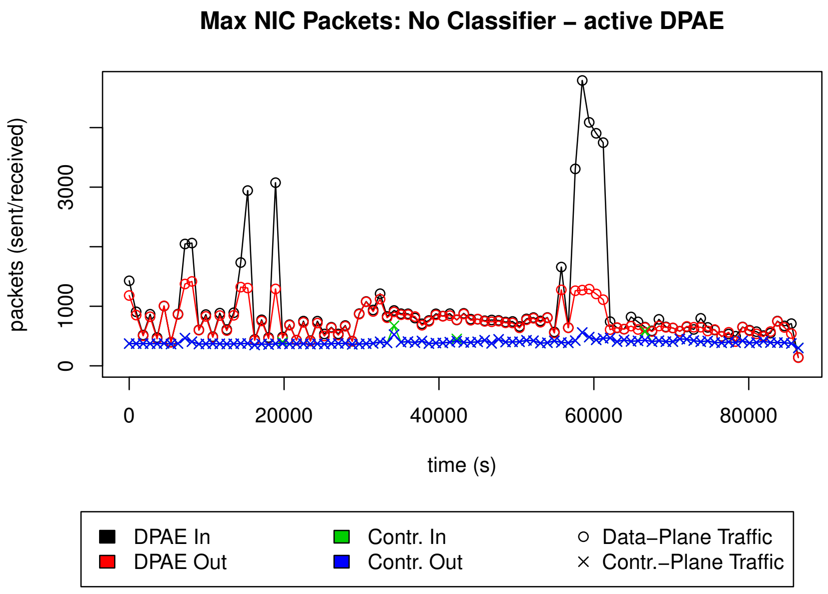

Figure 5.4: Active DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with no statistical classification.

Figures 5.5 and 5.6 contain the results for the control experiment with the DPAE was in passive mode. The data-plane traffic does not show the same spikes that were seen during the active DPAE experiments. Packet sampling from the switch has reduced the spikes of traffic received from the data-plane on average. In contrast, the amount of traffic increases at around 30000 seconds and plateaus.

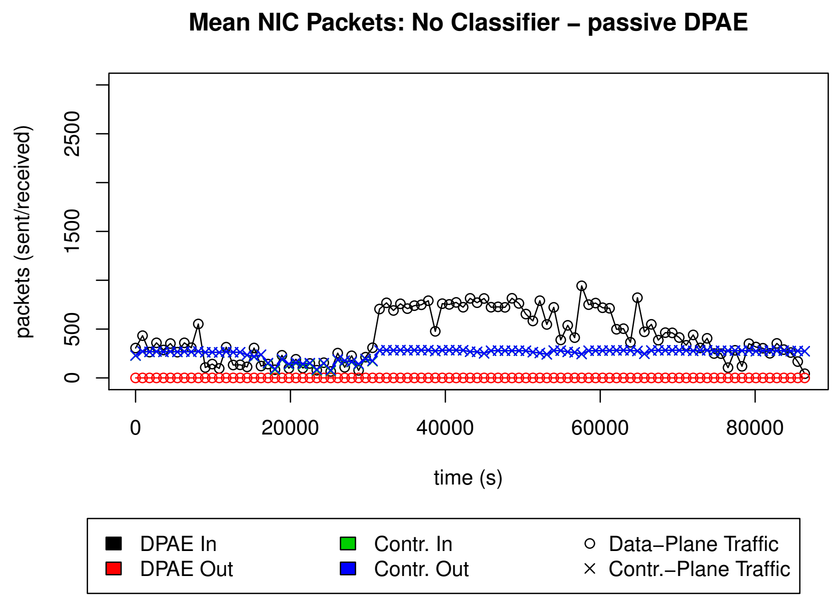

Figure 5.5: Passive DPAE: Mean number of packets sent and received by the DPAE host during the experiments with no statistical classification.

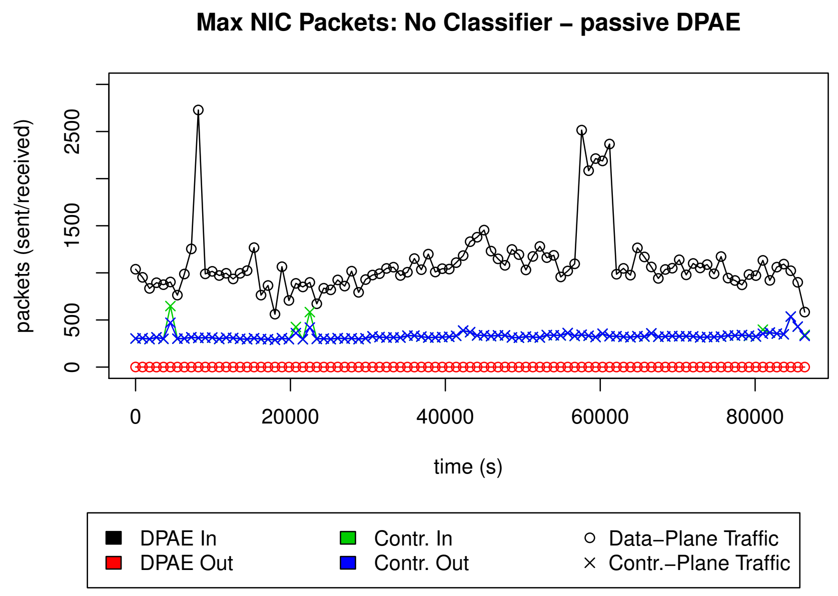

Figure 5.6: Passive DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with no statistical classification.

Packet sampling has affected the amount of traffic received from the data-plane during the DDoS attack at 60000 seconds. Figure 5.6 shows that the maximum amount of packets received at this time approached 2500 whereas it was greater than 4500 when the DPAE was in active mode. The amount of packets received during the passive scenario at this time is still greater than the maximum amount of packets that the DPAE processed during the active experiments. Therefore packet loss would still have been occurring even though Figures 5.5 and 5.6 provide no direct indication.

Figures 5.3, 5.4, 5.5 and 5.6 all show the same features in regards to the traffic sent on the control plane. The first feature is that the traffic sent from the DPAE to the controller does not increase once it has plateaued. This is a promising result as it suggests that if the data-plane imposes a significant load on the DPAE, the control-plane will not suffer adverse effects. The results from the control experiments suggest that the OpenFlow controller will not be flooded with traffic from the DPAE.

The second feature is that the number of packets sent on the control-plane is typically equal to the number of packets received from the control-plane. This is expected as the DPAE communicates with the conroller using TCP. Therefore each packet sent by one host must be acknowledged by the other.

Random Forest Classifier

Figures 5.7, 5.8, 5.9 and 5.10 show the traffic received and sent by the DPAE host when the Random Forest classifier was being run. These figures share many similar features to the figures depicting the control experiment when no classification was being performed.

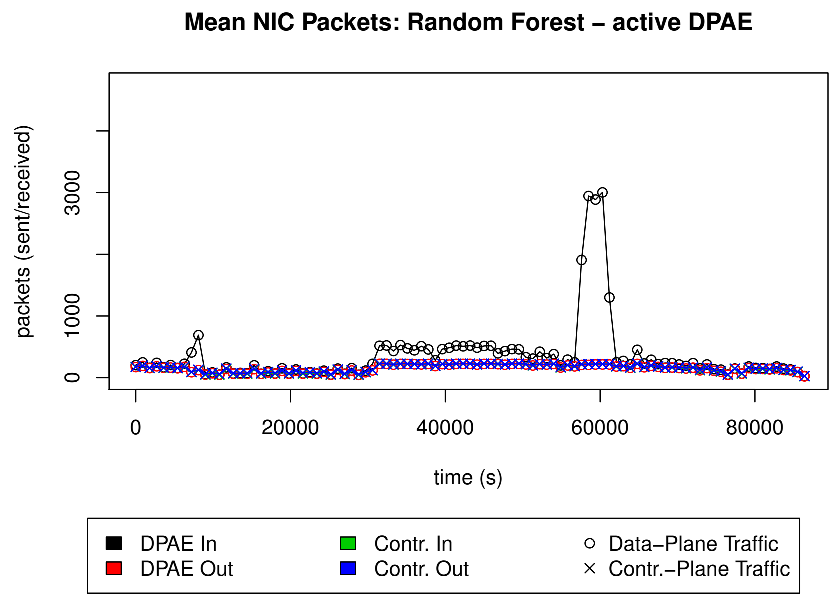

Figure 5.7: Active DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the Random Forest classifier.

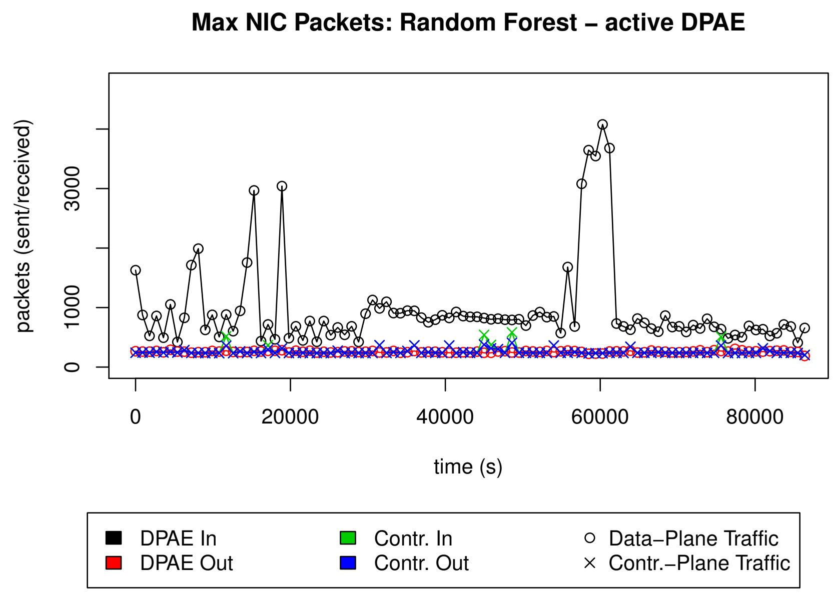

Figure 5.8: Active DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the Random Forest classifier.

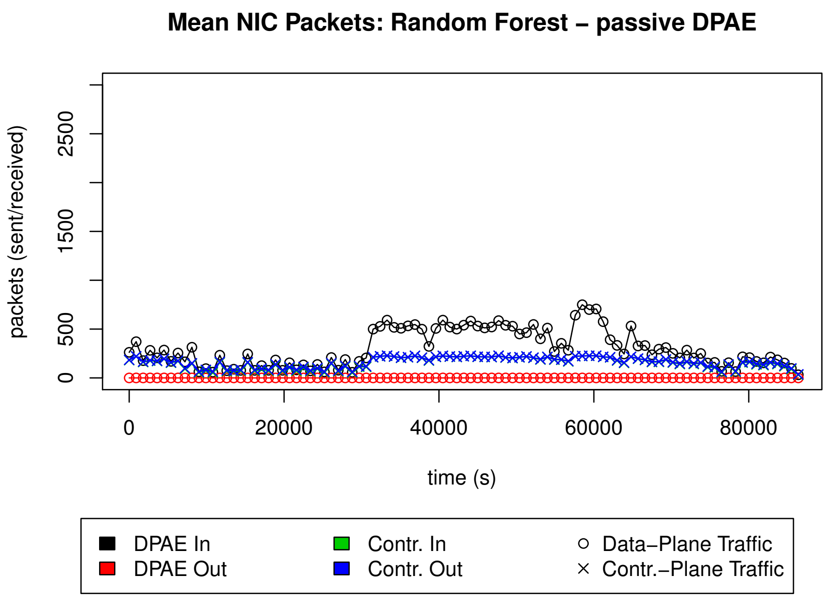

Figure 5.9: Passive DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the Random Forest classifier.

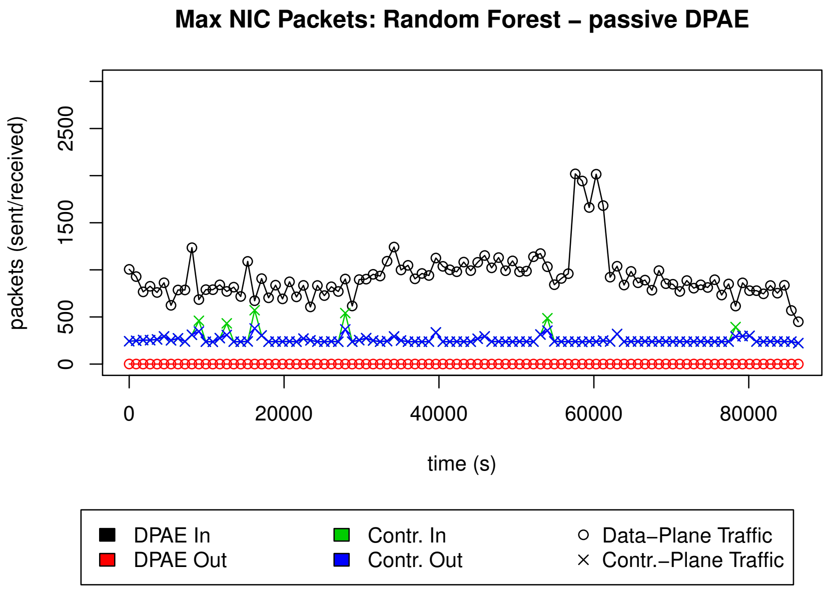

Figure 5.10: Passive DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the Random Forest classifier.

The main difference when considering the figures for the active experiments however is the positions of the line for the number of packets sent from the DPAE back to the switch. Figures 5.3 and 5.4 showed that this line matched closely with the number of packets received from the switch the majority of the time. Figures 5.7 and 5.8 do not show this however.

The discrepancy between the two quantities means that the Random Forest classifier was unable to process all of the packets from the data-plane as fewer packets were being forwarded back to the switch. The average packet processing time for the Random Forest classifier was higher than the processing time for the control experiment. The longer processing time would result in fewer packets being processed per second and by extension fewer packets being forwarded back to the switch. This provides an explanation as to why the Random Forest classifier made fewer predictions compared to the other classifier-DPAE scenarios.

A strange feature within the Figure 5.10 is the absence of the spike seen at around 10000 seconds during the control experiment. This instance can be considered to be anomalous with the sampling behaviour when the Random Forest classifier is being used.

KNN Classifier

Figures 5.11, 5.12, 5.13 and 5.14 show the traffic received and sent by the DPAE host when the KNN classifier was being run. These figures largely contain results that one would expect after having seen the results for the control experiment. The results for the Random Forest classifier showed that variation can be found in the amount of traffic sent back from the DPAE when it is in active mode. The results that will be discussed below indicate why the KNN classifiers resulted in more flows being classified compared to the Random Forest classifiers.

The discussion of the NIC results for the Random Forest classifier identified the difference between the traffic sent and received by the DPAE on the data-plane compared to the control experiment. The results for the

KNN classifier still do not compare too favourably to the control experiment however they are considerably better than the results for the Random Forest classifier. Figure 5.11 illustrates that the KNN classifier with the active DPAE was able to forward packets back to the switch on average.

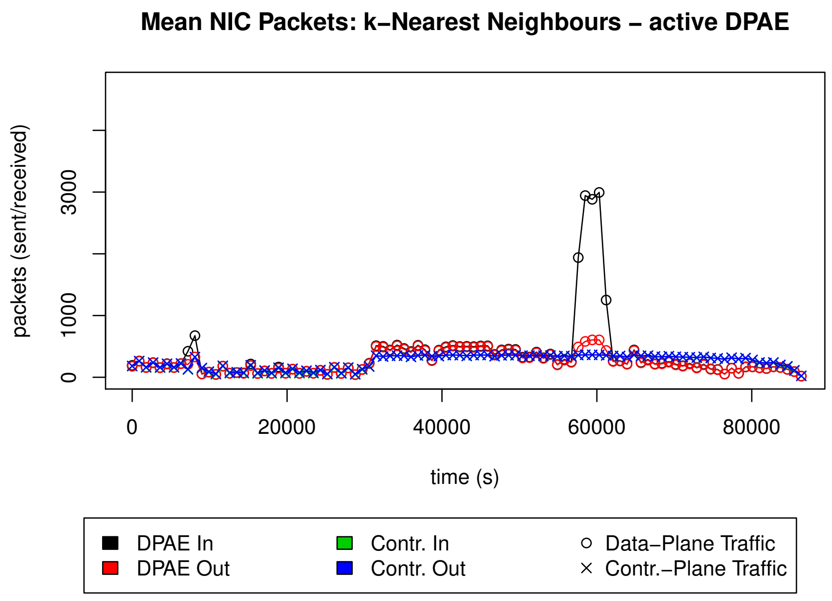

Figure 5.11: Active DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the KNN classifier.

The improvement presented by the KNN classifier is still far from optimal. The best examples that back this claim can be seen in Figure 5.12. As stated earlier, the maximum amount of traffic forwarded from the switch when the KNN classifier was running closely matches that during the control experiment. This is expected as the amount of traffic forwarded from the switch is independent of the classifier when the DPAE is in active mode in the context of this evaluation. The amount of packets sent from the DPAE to the switch during the time period 0 to 20000 seconds does not follow the amount of packets that were received as closely as in the control experiment (see Figure 5.4).

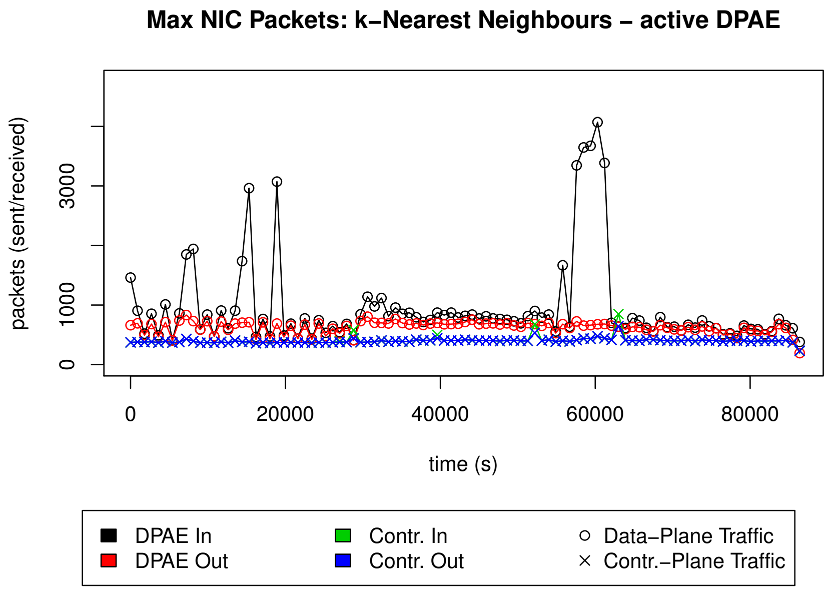

Figure 5.12: Active DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the KNN classifier.

It has been mentioned before that a discrepancy between the traffic sent to a DPAE in active mode and the traffic sent back from the same DPAE represents packet loss and fewer predictions being made. The results for the KNN classifier when the DPAE was in active mode show how statistical classification can impact negatively on network traffic. This is rather apparent at 60000 seconds in Figure 5.12 as the maximum amount of traffic being forwarded from the DPAE to the switch does not increase with the amount of traffic being sent to it for classification.

Figure 5.13 does not show anything that is worth discussing as it shows a very similar graph to what has been seen prior.

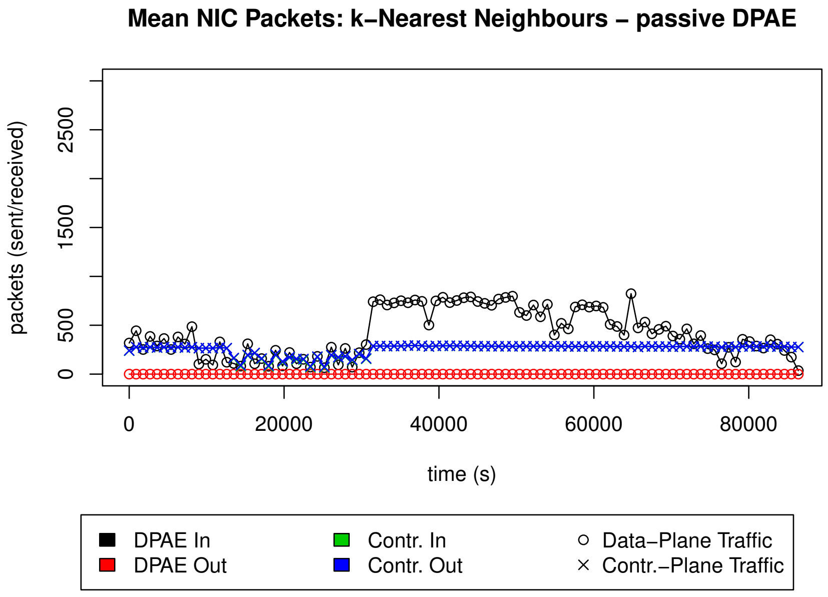

Figure 5.13: Passive DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the KNN classifier.

Figure 5.14 however does contain some features that are distinguishable. The first large spike in traffic just before 10000 seconds was present in the control experiment graph (Figure 5.6) but not in the Random Forest classifier graph (Figure 5.10). The presence, or lack thereof, of this feature is peculiar. Furthermore, the spike in traffic at 60000 second is smaller that the spike seen in the aforementioned graphs. This will be discussed in further detail when the results for the SVM classifier are examined.

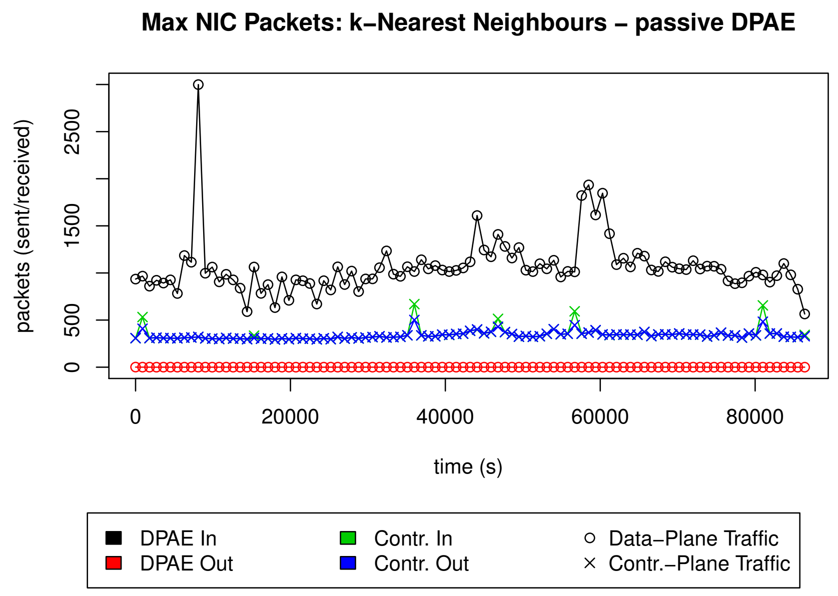

Figure 5.14: Passive DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the KNN classifier.

SVM Classifier

Figures 5.15, 5.16, 5.17 and 5.18 show the traffic received and sent by the DPAE host when the SVM classifier was being run.

Discussion concerning the Random Forest and KNN classifiers when the DPAE was in active mode referred to the discrepancy between the traffic received and sent by the DPAE on the data-plane. Figures 5.15 and 5.16 reveal that the discrepancy for the SVM classifier most closely follows the discrepancy seen in the control experiment.

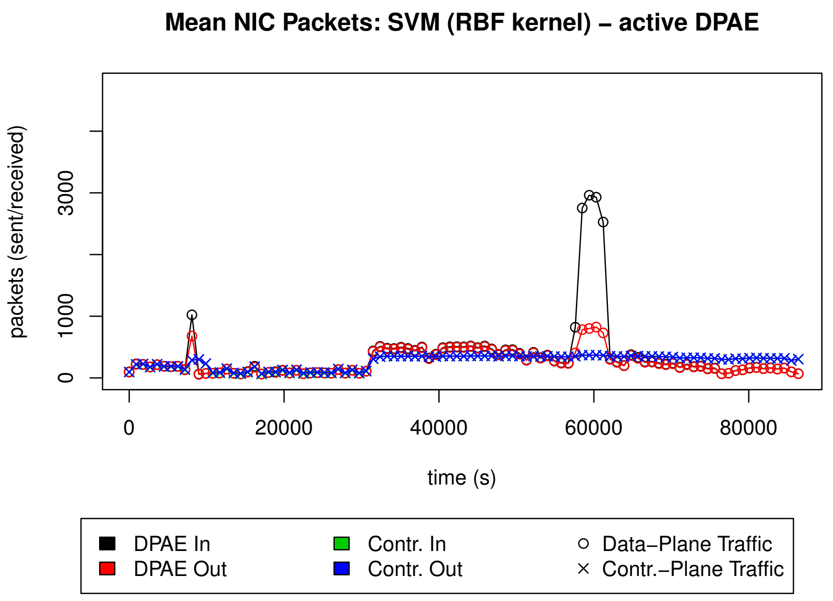

Figure 5.15: Active DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the SVM classifier.

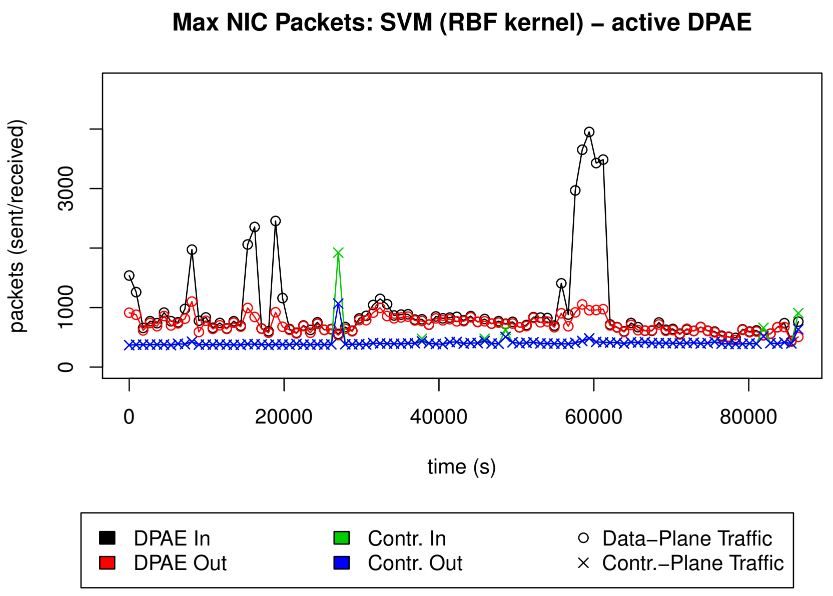

Figure 5.16: Active DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the SVM classifier.

The discrepancy in Figures 5.15 and 5.16 can be interpreted as less packet loss and therefore a larger quantity of packets being processed for classification. Table 5.7 showed that the combination of the SVM classifier and DPAE in active mode resulted in highest average number of packets being received by the sink, out of the three classifiers. This classifier-DPAE scenario also had the highest mean DR and the fewest number of flows that did not receive a prediction.

Figure 5.17 is very similar to the other figures of its kind as it shows very similar traffic patterns. Figure 5.18 however contains a significant anomaly that also occurs in Figure 5.16. What is being referred to specifically is the spikes of control-plane traffic. This is strange as it is seen when the DPAE is either in active or passive, albeit at different times. The largest volume of traffic in both instances was sent by the DPAE itself. It is unlikely that the anomaly was caused by the DPAE as it is configured to use the associated NIC for *nmeta2* related tasks only. These tasks produce low volumes of traffic and are typically messages used to inform the controller of a classification result.

The spike in traffic seen at around 60000 seconds in Figure 5.18 has been seen before in Figures 5.6, 5.10 and 5.14. The magnitude of the spike however has been different each time. Although it is not necessarily significant, it does show the variability in the quantity of packets received from the data-plane when packet sampling is conducted at the switch.

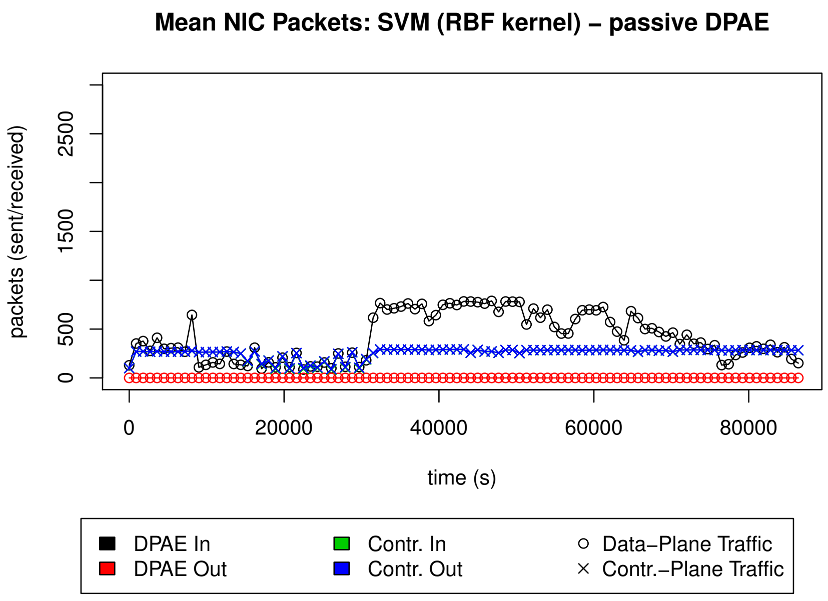

Figure 5.17: Passive DPAE: Mean number of packets sent and received by the DPAE host during the experiments with the SVM classifier.

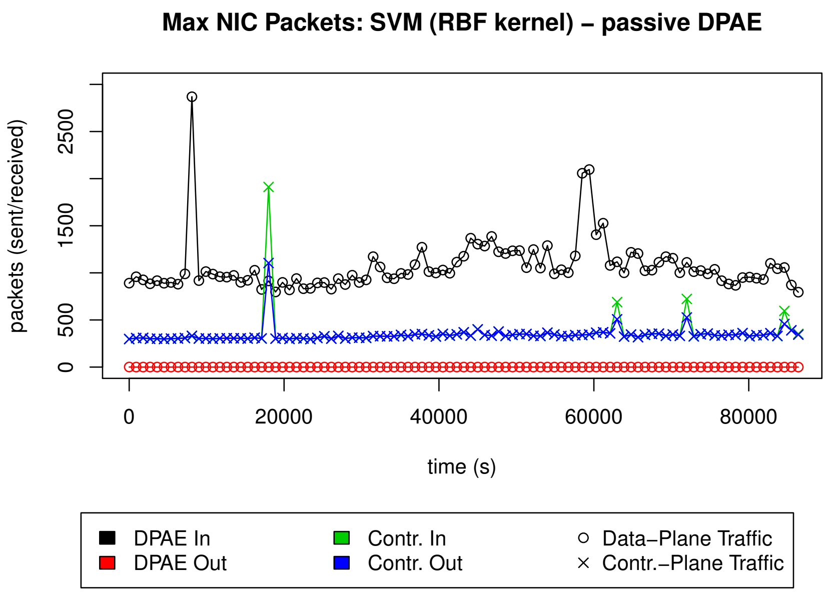

Figure 5.18: Passive DPAE: Maximum number of packets sent and received by the DPAE host during the experiments with the SVM classifier.

5.5.2 DPAE Host Performance Summary

This section showed how the quantity of packets sent and received by the DPAE host changed over time for the eight classifier-DPAE scenarios. The results show that the characteristics of the scenario affect the packets sent between the switch and the DPAE, and the traffic between the DPAE and the controller to a lesser effect. For instance the Random Forest classifier run alongside an active DPAE will result in less traffic being forwarded back to the switch than the SVM classifier with the same DPAE mode.

The results concerning the active DPAE more importantly illustrate the varying degrees of packet loss between the three classifiers and the controller. As expected, the classifiers with shorter average packet processing times also had smaller discrepancies between the traffic sent and received by the DPAE to the data-plane. These results explain why each of the classifiers within this DPAE scenario processed as many packets as they did.

5.6 Final Results Discussion

Sections 5.3, 5.4 and 5.5 explored different areas of classifier performance. This section uses these findings in conjunction with one another to discuss the effectiveness of the statistical classifiers.

5.6.1 General Remarks

Section 5.4.2 showed that using statistical classifiers to classify traffic increases the packet processing overhead. Furthermore, the classifiers with higher overheads processed fewer packets on average during the course of the experiments. Section 5.3.3 showed that a significant number of flows did not receive a predicted value. It follows that packet loss over contiguous periods of time would result in flows not receiving predictions.

Figures concerning the active DPAE scenario in Section 5.5 also revealed instances of packet loss during the experiments. This was shown through the difference in traffic that was received and sent back to the switch from the DPAE, which was especially apparent at 60000 seconds. It fits to reason that the majority of malicious flows within the dataset would have been sent around that time period in the experiment. The DPAE's inability to process that volume of traffic would mean that flows would not have received predictions. As a result, the number of positive predictions that could have been made were reduced.

The quantity of positive predictions should have been high given the performance of the classifiers in Chapter 3. Packet loss within the network meant that this quantity was low enough for the mean DR of each classifier to be less than satisfactory. It is important to remember that the very nature of networking hardware and software means that packets will be lost. The DDoS attack within the dataset was able to exploit this feature which impacted negatively on each classifier's ability to process traffic.

Earlier in this chapter we justified why the data within the XML files and PCAP file were statistically equivalent. As a result, one would expect the difference between the prediction results in this chapter with the those in Chapter 3 to be smaller than it was. The volume of data summarised within the XML files can be assumed to have been sent through the network during the experiments. The statistical feature that would have been subject to variability however would have been time. The duration of each flow could be modified by latency introduced at the switch or at the DPAE itself. This variability could quite possibly have modified the statistics that were collected by the DPAE and used by the classifiers.

5.6.2 Best Performing Classifier

Networks change how one must approach classification problems. The investigations in Chapter 3 very much follow traditional approaches to classification problems within different scientific domains. The Random Forest method has shown versatility when evaluated against datasets representing the UCI data base [46]. This evaluation suggests that the method may not be suitable for deployment within a live network environment.

The results for the SVM classifier demonstrate a couple of desirable qualities. The SVM classifier had the shortest average packet processing time of the three classifiers when the DPAE was in active (processor time and wall time) and passive mode (processor time only). This is important when the DPAE is in active mode as a shorter packet processing time reduces the end-to-end delay of a packet and means that fewer packets will be dropped by the DPAE.

It terms of prediction performance, the SVM classifiers performed better compared to the other classifiers within their respective DPAE scenarios. When used alongside the DPAE in active mode, the SVM classifier had the highest mean f-measure and was also the most accurate. When used alongside the DPAE in passive mode with packet sampling, the SVM classifier had the highest mean f-measure but its accuracy was the lowest.

The initialisation time for the SVM classifier (greater than 7 minutes) dwarfed the initialisation times for the Random Forest and KNN classifiers. This was expected however as it is well understood that the training algorithm for the SVM method is a quadratic optimisation problem. Classifier initialisation time may be an important factor when deciding to deploy a machine learning method within a network environment. The results suggest that if this time is of no concern, then a classifier that utilises the SVM method may be appropriate for deployment.

5.6.3 DPAE Suitability

An objective of this thesis was not to determine if the DPAE would be suitable for statistical classification or detecting DDoS attacks. The results however do provide some insights into this topic that are worth covering.

The two DPAE scenarios used were active mode and passive mode with flow-level packet sampling. The active mode scenario generally resulted in better classification results, both in terms of f-measure and accuracy. A disadvantage of the active mode however was that the DPAE introduced packet loss which was measured at the sink and observed at the DPAE. Packet loss was also measured at the sink when passive mode was used; however, this cannot be observed at the DPAE as it does not forward packets back to the switch.

Having the DPAE in active mode would be the most desirable of the two scenarios mentioned above. This is because it resulted in more desirable prediction results. An alternative approach would be to use passive mode without sampling packets from the switch. This would mean that overheads caused by statistical classifiers at the DPAE would not result in packets being lost on their way to their destination. This does not consider any overhead at the switch caused by the cloning of packets. Furthermore, packet sampling could then be performed on the DPAE if necessary.

Each figure in Section 5.5.1 shows the same trend in regard to the traffic sent between the DPAE and controller. The results suggest that regardless of the volume of traffic on the data-plane or the DPAE mode, the DPAE will not flood the controller with traffic most of the time. Despite the anomalies that were observed, the results confirm the scalability of the DPAE shown by Hayes [18].

5.6.4 Future Optimisations

Some future optimisations may provide some improvements to results in the future. The DPAE does not treat flows with the same five-tuple but initiated at different times as different flows. Modifying this mechanism of the DPAE would prevent statistical data from different flows from being aggregated. Another optimisation would involve using a different classifier for the scenario where packet sampling is used. Statistical classifiers that take less flow information into account would be more appropriate in this scenario.

5.7 Chapter Summary

This chapter evaluated three statistical classifiers on a physical network testbed. These classifiers did not classify traffic nearly as successfully as when they were tested using off-line data. It is important to note that the poor classification of traffic concerns the classifiers' abilities to detect malicious flows within the dataset. Each classifier was successful in identifying non-malicious flows.

The results do offer hope however. Looking beyond the prediction results of the classifiers, it appears that it is plausible to deploy particular statistical classifiers into networking environments because of their packet processing times. The classifier that performed best in this regard was the SVM classifier.

Chapter 6

Conclusion

This chapter presents the final concluding remarks for the thesis. Future work is presented following these remarks.

6.1 Final Conclusions and Discussion

This thesis has shown how statistical classification can be deployed using SDN to detect DDoS attacks. Three classifiers were selected in an off-line environment to be integrated with *nmeta2*. These were then evaluated on a physical network testbed by replaying a DDoS attack scenario.

An SVM classifier combined with the DPAE in active mode provided the highest f-measure and accuracy. Using the DPAE in a situation where no packet sampling was being performed proved to be advantageous. A greater volume of network traffic information typically resulted in more attacks being detected.

The experiments also showed that the SVM classifier had the shortest packet processing time on average. This was reflected in the deficit between the amount of traffic received from a switch to the DPAE and the amount of traffic sent back. This information is important to consider when deploying machine learning techniques in networks. The methods that were investigated in Chapter 3 utilise processes with ranging computational costs. Chapter 5 showed that weighing the cost between initialisation and packet processing times is just as important as classifier accuracy.

Anomaly detection approaches utilised in previous research demonstrated a tendency to misclassify non-malicious traffic despite having a high DR. For instance, Giotis *et al.* used information entropy to detect DDoS attacks with a DR of 100% but a FPR ranging between 23% and 39.3%. Ignoring the low mean DRs of the classifiers that were evaluated on the physical testbed, which can attributed to packet loss during the DDoS attack itself, the mean FPRs were all smaller than 0.3%. The highest FPR experienced during the second classifier selection experiment was no larger than 3%. These results suggest that statistical classification approaches can be used to reduce the number of misclassified non-malicious flows.

The use of SDN potentially enables us to discern if performance is caused by the control and/or data planes. The packet loss experienced by the DPAE during the DDoS attack suggests that further improvements to the data-plane are necessary. By better handling traffic during a DDoS attack, more information can be gathered thus improving the chances of determining the offending flows. Making such conclusions using a traditional networking paradigm is difficult as the control and data planes have a tighter coupling.

Statistical classification can be deployed using SDN to classify traffic. Careful consideration must be made to pick classifiers that result in the smallest possible packet processing overhead. While the classifiers did not demonstrate a high DR, results did suggest that particular statistical classification methods can classify network traffic under normal conditions using the *nmeta2* architecture. Under a DDoS attack scenario however, nothing is safe.

6.2 Future Work

Future work will explore using the DPAE in other network security research problems involving SDN. Bakker *et al.* [56] presented a network-wide firewall using SDN/OpenFlow. Their solution was limited to filtering traffic in a stateless manner however. Stateless firewalls are vulnerable to attacks where a malicious host masquerades as a web server and sends harmful traffic to host from outside a network. A stateful firewall can prevent this by only allowing traffic into a network if the connection was initiated from a host inside the network. The DPAE could be leveraged to monitor the state of TCP connections and inform a SDN firewall of connections that have been initialised and completed. This could perform the role of a stateful SDN firewall. It would be important to monitor TCP flow performance as part of the evaluation of the architecture.

Bibliography

[1] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *SIGCOMM Computer Communication Review*, vol. 34, pp. 39–53, Apr. 2004.

[2] KrebsOnSecurity, "KrebsOnSecurity Hit With Record DDoS," 2016. `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/` (Accessed on 20/09/2016).

[3] SDxCentral, "IoT Botnet To Blame for Big DDoS Attack," 2016. `https://www.sdxcentral.com/articles/news/iot-botnet-blame-big-ddos-attack/2016/10/` (Accessed on 02/11/2016).

[4] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, (Berkeley, CA, USA), USENIX Association, 2006.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[6] Open Networking Foundation, "OpenFlow Switch Specification - Version 1.3.5," Mar. 2015. Retrieved 17 March, 2016, from `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.5.pdf`.

[7] Open Networking Foundation, "Software-defined networking (sdn) definition," 2017. `https://www.opennetworking.org/sdn-resources/sdn-definition` (Accessed on 08/02/2017).

[8] M. Crotti, F. Gringoli, P. Pelosato, and L. Salgarelli, "A statistical approach to IP-level classification of network traffic," in *IEEE International Conference on Communications, 2006. ICC '06.*, vol. 1, pp. 170–176, IEEE, June 2006.

[9] B. Ng, M. Hayes, and W. K. G. Seah, "Developing a Traffic Classification Platform for Enterprise Networks with SDN: Experiences & Lessons Learned.," in *2015 IFIP Networking Conference (IFIP Networking)*, pp. 1–9, IEEE, May 2015.

[10] L. Qian, B. Wu, R. Zhang, W. Zhang, and M. Luo, "Characterization of 3G Data-Plane Traffic and Application towards Centralized Control and Management for Software Defined Networking," in *2013 IEEE International Congress on Big Data (BigData Congress)*, pp. 278–285, IEEE, June 2013.

[11] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust Network Traffic Classification," *IEEE/ACM Transactions on Networking*, vol. 23, pp. 1257–1270, Aug. 2015.

[12] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *2010 IEEE 35th Conference on Local Computer Networks (LCN)*, (Denver, CO, USA), pp. 408–415, IEEE, Oct. 2010.

[13] R. T. Kokila, S. T. Selvi, and K. Govindarajan, "DDoS Detection and Analysis in SDN-based Environment Using Support Vector Machine Classifier," in *2014 Sixth International Conference on Advanced Computing (ICoAC)*, (Chennai, India), pp. 205–210, IEEE, Dec. 2014.

[14] S. Lim, J.-I. Ha, H. Kim, Y. Kim, and S. Yang, "A SDN-Oriented DDoS Blocking Scheme for Botnet-Based Attacks," in *2014 Sixth International Conf on Ubiquitous and Future Networks (ICUFN)*, pp. 63–68, IEEE, July 2014.

[15] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments," *Computer Networks*, vol. 62, pp. 122–136, Apr. 2014.

[16] R. Wang, Z. Jia, and L. Ju, "An Entropy-Based Distributed DDoS Detection Mechanism in Software-Defined Networking," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, pp. 310–317, IEEE, Aug. 2015.

[17] S. Mehdi, J. Khalid, and S. Khayam, "Revisiting Traffic Anomaly Detection Using Software Defined Networking," in *Recent Advances in Intrusion Detection* (R. Sommer, D. Balzarotti, and G. Maier, eds.), vol. 6961 of *Lecture Notes in Computer Science*, pp. 161–180, Springer Berlin Heidelberg, 2011.

[18] M. J. Hayes, "Scalability and Performance Considerations for Traffic Classification in Software-Defined Networks," MSc thesis, School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand, 2016.

[19] Y.-D. Lin, P.-C. Lin, C.-H. Yeh, Y.-C. Wang, and Y.-C. Lai, "An Extended SDN Architecture for Network Function Virtualization with a Case Study on Intrusion Prevention," *IEEE Network*, vol. 29, pp. 48–53, May 2015.

[20] Michie, D. and Spiegelhalter, D.J. and Taylor, C.C., *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, pp. 1–58, July 2009.

[22] T. T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, pp. 56–76, Oct. 2008.

[23] H. Ringberg, M. Roughan, and J. Rexford, "The Need for Simulation in Evaluating Anomaly Detectors," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 55–59, Jan. 2008.

[24] B. Kolo, *Binary and Multiclass Classification*. Weatherford Press, 1st ed., 2011.

[25] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, pp. 35–40, January 2012.

[26] M. Hayes, "Traffic Classification in Enterprise Networks with the Era of IoT," COMP489 Report, Victoria University of Wellington, Nov. 2014.

[27] S. Alcock and R. Nelson, "Measuring the Accuracy of Open-Source Payload-Based Traffic Classifiers Using Popular Internet Applications," in *2013 IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops)*, (Sydney, Australia), pp. 956–963, IEEE, Oct. 2013.

[28] J. Frank, "Artificial Intelligence and Intrusion Detection: Current and Future Directions," in *Proceedings of the 17th National Computer Security Conference*, (Baltimore, Maryland, USA), 1994.

[29] Y. Xiang and Z. Li, "An Analytical Model for DDoS Attacks and Defense," in *International Multi-Conference on Computing in the Global Information Technology, 2006. ICCGI '06.*, (Bucharest, Romania), pp. 66–66, 1-3 Aug 2006.

[30] V. A. Foroushani and A. N. Zincir-Heywood, "Deterministic and Authenticated Flow Marking for IP Traceback," in *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, (Barcelona, Spain), pp. 397–404, March 2013.

[31] A. J. Izenman, *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning*. Springer, New York, 2008.

[32] scikit-learn developers, "1.2. Linear and Quadratic Discriminant Analysis," 2014. `http://scikit-learn.org/stable/modules/lda_qda.html` (Accessed on 16/08/2016).

[33] T. Thapngam, S. Yu, and W. Zhou, "DDoS discrimination by Linear Discriminant Analysis (LDA)," in *2012 International Conference on Computing, Networking and Communications (ICNC)*, pp. 532–536, Jan 2012.

[34] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for qos: A statistical signature-based approach to ip traffic classification," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, IMC '04, (Taormina, Sicily, Italy), pp. 135–148, ACM, 2004.

[35] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, (Pittsburgh, Pennsylvania, USA), pp. 144–152, 1992.

[36] Q. Yang, H. Fu, and T. Zhu, "An Optimization Method for Parameters of SVM in Network Intrusion Detection System," in *2016 Interna-*

tional Conference on Distributed Computing in Sensor Systems (DCOSS), (Washington, DC, USA), pp. 136–142, May 2016.

[37] M. Bramer, *Principles of Data Mining*. Undergraduate Topics in Computer Science, Springer London, 2nd ed., 2013.

[38] scikit-learn developers, "1.6. Nearest Neighbors," 2014. `http://scikit-learn.org/stable/modules/neighbors.html` (Accessed on 09/08/2016).

[39] L. Jiang, Z. Cai, D. Wang, and S. Jiang, "Survey of Improving K-Nearest-Neighbor for Classification," in *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007.*, vol. 1, (Haikou, Hainan, China), pp. 679–683, Aug 2007.

[40] M.-Y. Su, "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3492 – 3498, 2011.

[41] H. Zhang, "The Optimality of Naive Bayes," in *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference*, (Miami Beach, FL, USA), pp. 562–567, 2004.

[42] M. Toulouse, B. Q. Minh, and P. Curtis, "A Consensus Based Network Intrusion Detection System," in *2015 5th International Conference on IT Convergence and Security (ICITCS)*, (Kuala Lumpur, Malaysia), pp. 1–6, Aug 2015.

[43] scikit-learn developers, "1.10. Decision Trees," 2014. `http://scikit-learn.org/stable/modules/tree.html` (Accessed on 09/08/2016).

[44] Y. C. Wu, H. R. Tseng, W. Yang, and R. H. Jan, "DDoS Detection and Traceback with Decision Tree and Grey Relational Analysis," in *2009*

Third International Conference on Multimedia and Ubiquitous Engineering, (Qingdao, China), pp. 306–314, June 2009.

[45] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[46] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?," *Journal of Machine Learning Research*, vol. 15, pp. 3133–3181, Jan. 2014.

[47] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests," *Information Sciences*, vol. 278, pp. 488 – 497, 2014.

[48] S. Lee, H. Kim, D. Barman, S. Lee, C.-k. Kim, T. Kwon, and Y. Choi, "NeTraMark: A Network Traffic Classification Benchmark," *SIGCOMM Comput. Commun. Rev.*, vol. 41, pp. 22–30, Jan. 2011.

[49] "The CAIDA "DDoS Attack 2007" Dataset." `http://www.caida.org/data/passive/ddos-20070804\_dataset.xml`.

[50] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & Security*, vol. 31, pp. 357 – 374, 2012.

[51] S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," in *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pp. 78–83, Feb 2016.

[52] Python Software Foundation, "26.6. timeit measure execution time of small code snippets python 2.7.12 documentation," 2016. Retrieved 20 November, 2016, from `https://docs.python.org/2/library/timeit.html`.

[53] Python Software Foundation, “15.3. time time access and conversions python 2.7.12 documentation.html,” 2016. Retrieved 20 November, 2016, from `https://docs.python.org/2/library/time.html`.

[54] Python Software Foundation, “26.4. the python profilers python 2.7.12 documentation,” 2016. Retrieved 20 November, 2016, from `https://docs.python.org/2/library/profile.html`.

[55] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*. Springer, 2008.

[56] J. N. Bakker, I. Welch, and W. K. G. Seah, “Network-wide Virtual Firewall using SDN/OpenFlow,” in *2016 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, (Palo Alto, CA, USA), 2016.

Appendix A

Problem Space Investigation for Features

This appendix contains information on bidirectional flow features in the ISCX 2012 DDoS dataset. This dataset contains 571698 labelled bidirectional flows. Table A.1 contains information on features that were derived directly from the dataset provided. Tables A.2 and A.3 contain information on features that need to be derived using the features from Table A.1.

All tables show that all features, raw and derived, are arguably continuous variable types. This point may be contested as the features relating to packet and byte counts for flows can only be expressed as discrete integers. However, due to the large range of values that these statistical features encompass, it makes it impractical for them to be considered *categorical*. Further discretisation into ranges of values would be necessary for the categorical type to hold but this is a lossy process.

Feature	**Properties**
totalSourceBytes	• Value type: **Continuous**, Integer, Unimodal • Min: 0 Median: 477 Max: 763277598
totalDestinationBytes	• Value type: **Continuous**, Integer, Unimodal • Min: 0 Median: 1691 Max: 176755516
totalSourcePackets	• Value type: **Continuous**, Integer, Bimodal • Min: 0 Median: 6 Max: 514794
totalDestinationPackets	• Value type: **Continuous**, Integer, Bimodal • Min: 0 Median: 6 Max: 538076
startDateTime	• Value type: **Continuous**, DateTime with the format ”%Y-%m-%dT%H:%M:%S” e.g. 2010-06-04T21:54:57
stopDateTime	• Value type: **Continuous**, DateTime with the format ”%Y-%m-%dT%H:%M:%S” e.g. 2010-06-04T21:54:57

Table A.1: Properties of the features within the ISCX 2012 DDoS Dataset.

Feature	**Properties**
FlowDuration	• Derived from: stopDateTime$-$startDateTime • Value type: **Continuous**, Integer, Unimodal
log(FlowDuration)	• Derived from: natural logarithm of stopDateTime$-$startDateTime • Value type: **Continuous**, Integer, Multimodal • Note: As log(0) is not defined, I interpreted $\log(0) = 0$.
log(totalSourceBytes)	• Derived from: natural logarithm of totalSourceBytes • Value type: **Continuous**, Integer, Multimodal • Note: As log(0) is not defined, I interpreted $\log(0) = 0$.
SourceBytes-per-Packet	• Derived from: $\frac{totalSourceBytes}{totalSourcePackets}$ • Value type: **Continuous**, Integer, Multimodal

Table A.2: Features derived from features within Table A.1 - part 1.

Feature	**Properties**
DestinationBytes-per-Packet	• Derived from: $\frac{totalDestinationBytes}{totalDestinationPackets}$ • Value type: **Continuous**, Integer, Multimodal

Table A.3: Features derived from features within Table A.1 - part 2.

Appendix B

Initial Classifier Selection Experiment Results

This appendix contains the complete list of results for the initial classifier selection experiment. Separate tables are used for each method. The following abbreviations are used:

- tSB: totalSourceBytes
- ltSB: log(totalSourceBytes)
- tDB: totalDestinationBytes
- tSP: totalSourcePackets
- FD: FlowDuration
- lFD: log(FlowDuration)
- SBpP: SourceBytes-per-Packet
- DBpP: DestinationBytes-per-Packet

B.1 LDA

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.989999724	0.029018682
SBpP and DBpP	0.0061397	0.007521567
tSB and FD	0.0000253	0.002416255
tSB and lFD	0.0000431	0.00185139
tSB and tDB	0.085453263	0.00489996
tSB and tSP	0.335905144	0.007711583

Table B.1: Mean Detection and False Positive Rates for LDA-based first generation classifiers.

B.2 QDA

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.990011046	0.022104852
SBpP and DBpP	0.75763138	0.005886225
tSB and FD	0.962506798	0.014008363
tSB and lFD	0.922515947	0.017442291
tSB and tDB	0.983063906	0.013685196
tSB and tSP	0.975312122	0.014930263

Table B.2: Mean Detection and False Positive Rates for QDA-based first generation classifiers.

B.3 SVM (RBF kernel)

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.988362117	0.012553594
SBpP and DBpP	0.928522653	0.007447488
tSB and FD	0.746	0.001861962
tSB and lFD	0.763	0.002086601
tSB and tDB	0.750116539	0.001744573
tSB and tSP	0.762044205	0.001903781

Table B.3: Mean Detection and False Positive Rates for SVM-based first generation classifiers.

B.4 k-Nearest Neighbours

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.941112196	0.004893325
SBpP and DBpP	0.938639201	0.005284846
tSB and FD	0.93536111	0.007139069
tSB and lFD	0.920191745	0.007936338
tSB and tDB	0.94632216	0.003572972
tSB and tSP	0.933408686	0.007366625

Table B.4: Mean Detection and False Positive Rates for KNN-based first generation classifiers.

B.5 Naive Bayes

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.989364833	0.021982912
SBpP and DBpP	0.398054629	0.005845593
tSB and FD	0.962605386	0.020798816
tSB and lFD	0.92243702	0.019069295
tSB and tDB	0.975657684	0.015039642
tSB and tSP	0.962315858	0.024197488

Table B.5: Mean Detection and False Positive Rates for Naive Bayes-based first generation classifiers.

B.6 Decision Tree

Feature Set	Mean DR	Mean FPR
ltSB and FD	0.930443237	0.004166076
SBpP and DBpP	0.936057959	0.004597989
tSB and FD	0.930418475	0.004179437
tSB and lFD	0.929958674	0.004132092
tSB and tDB	0.943267767	0.003760251
tSB and tSP	0.935239892	0.004232693

Table B.6: Mean Detection and False Positive Rates for Decision Tree-based first generation classifiers.

B.7 Random Forest

Feature Set	**Mean DR**	**Mean FPR**
ltSB and FD	0.933274492	0.00399865
SBpP and DBpP	0.939866433	0.003926908
tSB and FD	0.932887318	0.003989646
tSB and lFD	0.932399254	0.003949279
tSB and tDB	0.945090213	0.003108191
tSB and tSP	0.938066354	0.003907299

Table B.7: Mean Detection and False Positive Rates for Random Forest-based first generation classifiers.

Appendix C

Second Classifier Selection Experiment Results

This appendix contains the complete list of results for the second classifier selection experiment. Separate tables are used for each method. The following abbreviations are used:

- tSB: totalSourceBytes
- ltSB: log(totalSourceBytes)
- tDB: totalDestinationBytes
- tSP: totalSourcePackets
- ltSP: log(totalSourcePackets)
- tDP: totalDestinationPackets
- FD: FlowDuration

C.1 QDA

Feature Set	Mean DR	Mean FPR	f-measure
tSB, tSP and FD	0.987526269	0.012491378	0.911904559
ltSB, tSP and FD	0.990000645	0.027850928	0.830602444
ltSB, ltSP and FD	0.989906291	0.014991284	0.89844222
tSB, tDB and FD	0.988821823	0.012680678	0.911523632
tSB, tSP, tDB, tDP and FD	0.988482794	0.011960445	0.915662191

Table C.1: Mean Detection and False Positive Rates, and f-measure for QDA-based second generation classifiers.

C.2 SVM (RBF kernel)

Feature Set	Mean DR	Mean FPR	f-measure
tSB, tSP and FD	0.738530663	0.001656662	0.838201025
ltSB, tSP and FD	0.929109403	0.004674993	0.931058739
ltSB, ltSP and FD	0.9893688	0.012388389	0.913524809
tSB, tDB and FD	0.725160172	0.001547747	0.83005504
tSB, tSP, tDB, tDP and FD	0.726826225	0.001583319	0.830928575

Table C.2: Mean Detection and False Positive Rates, and f-measure for SVM-based second generation classifiers.

C.3 k-Nearest Neighbours

Feature Set	Mean DR	Mean FPR	f-measure
tSB, tSP and FD	0.93788822	0.00678328	0.921951716
ltSB, tSP and FD	0.935423086	0.005043936	0.932025381
ltSB, ltSP and FD	0.944430283	0.004001303	0.943749165
tSB, tDB and FD	0.948361482	0.003619612	0.948387955
tSB, tSP, tDB, tDP and FD	0.948345925	0.003618799	0.948385248

Table C.3: Mean Detection and False Positive Rates, and f-measure for KNN-based second generation classifiers.

C.4 Naive Bayes

Feature Set	Mean DR	Mean FPR	f-measure
tSB, tSP and FD	0.962656107	0.022301342	0.863597597
ltSB, tSP and FD	0.989756706	0.024335124	0.847633059
ltSB, ltSP and FD	0.990108069	0.029333478	0.822862165
tSB, tDB and FD	0.969123479	0.014176686	0.884021726
tSB, tSP, tDB, tDP and FD	0.988954563	0.019051931	0.878745778

Table C.4: Mean Detection and False Positive Rates, and f-measure for Naive Bayes-based second generation classifiers.

C.5 Random Forest

Feature Set	**Mean DR**	**Mean FPR**	**f-measure**
tSB, tSP and FD	0.941803212	0.003069825	0.94864049
ltSB, tSP and FD	0.941674526	0.003067095	0.948592306
ltSB, ltSP and FD	0.941763447	0.003075131	0.948583216
tSB, tDB and FD	0.947373016	0.002765957	0.953657985
tSB, tSP, tDB, tDP and FD	0.948062485	0.002653073	0.954795172

Table C.5: Mean Detection and False Positive Rates, and f-measure for Random Forest-based second generation classifiers.