Developing an Intrusion Detection Model for Distributed Denial of Service Attacks in Cloud Computing

A thesis submitted for the degree of Doctor of Philosophy

by

Roja Ahmadi

Department of Computer Science

College of Engineering, Design and Physical Sciences

Brunel University London

December 2019

ABSTRACT

Distributed Denial of Service (DDoS) attacks are one of the most significant threats to the availability of resources offered using the cloud computing model. One way to counter such attacks is by employing Intrusion Detection Systems which seek to identify attacks so that countermeasures can be deployed. However, owing to changes in the characteristics of the attacks, Intrusion Detection Systems can sometimes fail to accurately detect DDoS attacks. In seeking to improve Intrusion Detection Systems, the scarcity of publicly-available cloud intrusion detection datasets hinders the development of more precise detection models. Moreover, existing public cloud intrusion datasets have several notable issues, including differences in their formats, limited traffic features, and not including all types of DDoS attack. Additionally, research in this area often has a lack of transparency in terms of the structure and processing of the datasets and the resulting models, making it difficult to undertake comparative work. To address the identified issues, the initial stage of this research developed a detection model using a well-established non-cloud dataset, applying a transfer learning approach to assess the performance of the model using one of the limited number of public cloud datasets by remapping the DDoS attack types between the two datasets. The accuracy of the model was high on the non-cloud dataset; but, when it was applied to the cloud dataset, its accuracy fell significantly owing to the different structures of the two datasets and the limited common feature set. To address the identified issues, the obtained result of the first stage motivated this research to develop an emulated cloud intrusion detection dataset, with a broad range of features and the same structure as the existing cloud dataset. Using different classifiers, two detection models were created, and the generated cloud dataset was used to analyse their performance in a novel way by using different time intervals, or ’slices’. The results showed a general increase in the accuracy of the detection models as the time interval increased. To further explore the relationships between features over time, cross-correlation analysis was used to identify when the most significant correlations occurred between feature pairs at different time lags. The analysis showed that the highest frequencies of the ‘most significant correlation values’ occurred at the 0-1 and 7-9 second time periods, but it did not show a correlation between these frequencies and the accuracy of the models across the time. The research reported in this thesis has led to four contributions to the field. The first contribution lies in the novel application of transfer learning to build a detection model. The second contribution is a practical contribution to the creation of a new cloud-based dataset. The third contribution centres on the use of a novel approach to the analysis of the generated dataset using different time intervals as the unit of analysis and comparisons of the accuracy of the model when applied to them. The fourth contribution is in the provision of a clear and transparent process for generating an emulated cloud-based dataset and undertaking systematic analysis of it.

DEDICATION

This thesis is dedicated to my beloved parents, my brother Roozbeh and my sister Rosa for their unconditional love, endless support and constant encouragement. Thank you for your limitless faith in my abilities. I love you so much.

تقديم به پدر، مادر، روزبه و رزا عزيزتر از جانم براي عشق بي قيد و شرط، پشتيباني و تشويق بي پايانشان. دوستتون دارم

ACKNOWLEDGEMENTS

I would like to express my appreciation to all the people who have supported me while completing this thesis. First and foremost, I remain forever thankful to my supervisor, Professor Rob Macredie, for his professionalism, insight, critiques, patience, detailed and prompt reviews. This journey would have never been completed without his generous support. No words can express my gratitude for all the endless care he has provided me, particularly during the most challenging time of my PhD journey. It is my hope that the knowledge I have acquired from him will continue to impact my future endeavors personally and professionally. Once again, thank you very much.

I would also like to extend my gratitude to my second supervisor, Dr. Allan Tucker, as he has provided valuable advice and support in all possible ways during my research.

I would also like to thank Dr Armin Kashefi, Dr Faris Alwzinani and Dr Alaa Marshan, who have always been great friends to me and have supported me throughout my research. I would also like to offer my warmest thanks to all my friends – near and far – who believed in me, particularly, Mashael and Lilly who assisted me through the dark moments of coding and data analysis.

Finally, I would like to thank all my wonderful families in Iran and the supportive people at the Brunel University London's Department of Computer Science, especially Ela Heaney, who has always been understanding and has made sure that PhD researchers have a conducive academic environment.

DECLARATION

- The following paper has been published as a direct result of the research presented in this thesis:
- Ahmadi, R., Macredie, R. D. and Tucker, A (2018). Intrusion Detection Using Transfer Learning in Machine Learning Classifiers Between Non-cloud and Cloud Datasets. *Lecture Notes in Computer Science* (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 11314 LNCS: 556-566.

ABBREVIATIONS

SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
CSA	Cloud Security Alliance
ENISA	European Network and Information Security
NIST	National Institute of Standards and Technology
DDoS	Distributed Denial of Service Attacks
IDS	Intrusion Detection Systems
CIDD	Cloud Intrusion Detection Dataset
NSL-KDD	National Security Laboratory – Knowledge Discovery Dataset
OSI	Open System Interconnect
TCP/IP	Transmission Control Protocol/Internet Protocol
MAC	Media Access Control
ICMP	Internet Control Message Protocol
IPV4	Internet Protocol Version 4
IPV6	Internet Protocol Version 6
TTL	The Time to Live
PoD	Ping of Death
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
SYN-ACK	Synchronize-acknowledge
Land	Local Area Network Denial
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
FTP	File Transfer Protocol
CAIDA	The Centre for Applied Internet Data Analysis
IMPCAT	Information Marketplace for Policy and Analysis of Cyber-risk & Trust
ADFA	Australian Defense Force Academy
LBNL	Lawrence Berkeley National Laboratory

DARPA	Defense Advanced Research Project Agency
U2R	User to Root attacks
R2L	Remote to Local attacks
ML	Machine Learning
SVM	Support Vector Machines
Eucalyptus	Elastic Utility Computing Architecture for Linking Your Program to Useful System
API	Application-Programming Interface
VMs	virtual machines
CCF	Cross-correlation function
KPPS	Kwiatkowski–Phillips–Schmidt–Shin
ADF	Augmented Dicky-Fuller

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 : Introduction

1.1 Introduction

In the last few years, Cloud Computing has gained popularity as a novel distributed computing paradigm, offering a pay-as-go-based model through on-demand access to systems and services which has the benefits of scalability afforded by shared computing resources (e.g., servers, networks, applications, virtual infrastructure) (Paxton, 2016). These computing resources are delivered over the internet in three different types of service – Software as a Service (SaaS); Platform as a Service (PaaS); and Infrastructure as a Service (IaaS) – and can be deployed as private, public, community cloud or hybrid cloud configurations (Kumar, Lal, & Sharma, 2016). A popular example of cloud services is the Google App suite, which comprises a collection of a web-based applications (the Gmail email service; the Google Talk instant messaging service; and the Google Calendar scheduling/time management service) and some of Google's productivity tools (the Google Docs word processor; the Google Sheets spreadsheet application; and Google Slides presentation program).

This innovative model provides remarkable benefits to the computing domain as a result of minimizing upfront investment in IT infrastructure and software licensing, replacing them with cost effective on-demand services. Clients no longer need to have their own servers; they can access cloud services in a scalable form without managing the complex underlying technology infrastructure on which it relies. These advantages have motivated various industries to adopt the cloud computing model and move their applications and daily operations into the cloud. Indeed, according to a recent Flexera (2019) survey, 94 percent of respondents use cloud services in some way.

Though there is noticeable growth in adoption driven by the many advantages of the cloud computing model, the availability of cloud computing services is a significant concern. Because the main function of the cloud is to deliver services in different forms on the Internet, and also to provide secure resources for users, lack of availability (or 'outages') of cloud services can severely impact the workload of enterprises and the experience of customers who use such services.

The Cloud Security Alliance (CSA), the European Network and Information Security (ENISA) and the National Institute of Standards and Technology (NIST) have identified Distributed Denial of Service Attacks (DDoS) as one of the greatest threats to the availability of cloud services and servers on the internet. DDoS attacks are easy to implement and challenging to counter and this type of attack is growing dramatically, causing significant financial damage to organisations through outages to cloud services across the world (NETSCOUT, 2018).

This chapter will explain the broad sense of the research, providing further argument for DDoS attacks being one of the main threats to cloud computing services and providing a high-level view of existing research in this area in order to frame the research presented in this thesis.

The remainder of this chapter is structured as follows. Section 1.2 will introduce the importance and impact of DDoS attacks. Section 1.3 will provide a brief review of existing research into DDoS attacks to introduce the motivation for the research reported in this thesis. Section 1.4 and section 1.5 will then present the research objectives and the research questions. Section 1.6 will explain the structure of the thesis, providing a high-level description of what will be covered in each of the remaining chapters.

1.2 The Importance and Impact of DDoS Attacks

This section will explain DDoS attacks and illustrate their importance by presenting examples of such attacks and their impact on the affected organisations. DDoS attacks seek to make network resources, such as websites and online applications, unavailable to their users. A DDoS attack is typically accomplished by overwhelming the target system with a high amount of unwanted network traffic in an attempt to prevent legitimate users from accessing the target system's services (David and Thomas, 2015).

In most DDoS attacks, the attacker uses multiple, compromised, vulnerable machines, termed ‘zombies’, which are often connected into ‘botnets’. There are different types of DDoS attack, depending on which layer of the network the attacker is targeting; to produce the attacks, the compromised machines are equipped with tools that are capable of generating fake packets and establishing connections with the targeted system which cause the host systems to suffer a range of issues (which will be discussed in detail in Chapter 2). For all of the DDoS attack types, the actions of the zombies/botnet result in the disruption of legitimate users’ access to the resources of the targeted system. In the case of cloud computing environments, DDoS attacks have a more devastating effect than they do on local servers owing to the distributed nature of the cloud (Fernandes et al., 2014). Core features of the cloud, such as multi-tenancy, where the cloud platform is shared by potentially millions of ‘customers’, create potential issues since among the customers there might be malicious tenants whose aim is to cause harm to other legitimate tenants/customers (Hashizume, et al., 2013).

A high number of organisations and the services that they offer, often through websites, are targeted by DDoS attacks, with revenge, financial advance, governmental issues, extortion, reputation damage and competition between cloud providers being potential motivations for DDoS attacks. These issues may become more common, and more acute, for organisations in terms of the disruption that they may cause given the ever-growing reliance on cloud services, but DDoS attacks have been a persistent issue over the last 20 years.

Indeed, the first DDoS attack was identified in 1999 (MIT Technology Review, 2019). By 2000, large organisations, such as Amazon, Yahoo, CNN and eBay, were being targeted by DDoS attacks, causing disruption in their services for a few hours at a time (Long, 2012). The type and nature of DDoS attacks continued to develop with, for example, in 2002, a number of DNS root servers being targeted simultaneously by a DDoS attack (Vixie, Sneeringer and Schleifer, 2012). The scale of DDoS attacks also began to grow as such attacks became more established. In 2007, for example, 10,000 online game servers were targeted by a DDoS attack and, in 2009, GoGrid, a cloud provider, suffered a DDoS attack that resulted in half of its customers being affected (Kawamoto, 2009).

Very high-profile companies have also been targeted by DDoS attacks, including Twitter and Facebook in 2009, and Visa, MasterCard and PayPal in 2010. Large financial services institutions, including major United States banks, have also been subject to DDoS attacks, including Bank of America, Wells Fargo, Bancorp and others who were affected by DDoS attacks in 2012 that caused simultaneous, significant disruptions to their systems, leading to outages (Constantin, 2012).

By 2015, most major providers of cloud services had become targets for DDoS attacks, including Amazon and Rackspace. Attacks are not, though, restricted to large corporate organisations. For example, in the same year, Greatfire.org, an activist site which monitors web sites that are blocked by the Chinese government, was attacked by a heavy DDoS attack which left it facing a bill of $30,000 per day for its use of Amazon EC2 cloud, which scaled up in response to the attack, using more resources to respond to it (Munson, 2015).

Arguably the largest DDoS attack to date took place in 2018, when GitHub, a platform through which developers can manage code, was attacked with (at its peak) an incoming traffic rate of 1.3 terabytes per second and incoming packets numbering 126.9 million per second. At the time of writing, there seems no likelihood of decline in DDoS attacks, with September 2019, for example, seeing Wikipedia disrupted by such an attack.

As well as the inconvenience to an organisation and its customers of suffering a DDoS attack, the average financial loss to an organisation of such an attack is estimated to be $444,000 US (Darwish, et al., 2013; Girma, et al., 2015; Osanaiye, et al., 2016; Somani, et al., 2017), providing user/customer experience and financial imperatives for seeking to better understand and respond to DDoS attacks.

The next section will introduce existing research into DDoS attacks to frame the motivation for the research reported in this thesis.

1.3 Existing Research into DDoS Attacks

In an attempt to address DDoS attacks, a range of techniques have been developed and incorporated into Intrusion Detection Systems (IDS) in cloud environments to provide well-defined security mechanisms to monitor and analyse the behaviour of users and network activities with the aim of detecting possible DDoS attacks and other forms of intrusion. A categorisation of IDS and the techniques used within them is provided by Carlin, Hammoudeh and Aldabbas (2015).

However, IDS systems can fail to accurately detect DDoS attacks, resulting in the creation of false alarms, owing to changes in the characteristics of the attacks (Sari, 2015). IDS systems therefore need to develop over time in order to cope with the evolving nature of DDoS attacks. This leads to on-going work into specific DDoS attack types, and ways in which to accurately identify them, so that IDS may be subsequently improved by incorporating refined, or newly-developed, attack identification techniques/models.

When undertaking research in this area there are, though, several broad issues that, this thesis will argue, require consideration and attention. First, much of the work that is done by industry is obviously commercially sensitive and uses data that are not made available to researchers. As a result, there is a lack of public cloud datasets, which hinders the academic development of models to detect DDoS attacks in cloud environments. As a result of the lack of public cloud datasets, most of the existing academic research uses the currently available non-cloud datasets to develop models to detect DDoS attacks and this may lead to limitations (Sharafaldin, Lashkari and Ghorbani, 2018).

Second, datasets are often structured differently from each other, with some presenting packet-based data and others data that is time-based. This means that comparing detection techniques/models across different datasets can be challenging, if not impossible (Ahmadi, Macredie and Tucker, 2018).

Third, datasets may focus on different attack types (Nadiammai and Hemalatha, 2014), meaning that even if the datasets are publicly-available, they are of restricted value to other researchers since they limit the types of attack which they may be used to explore.

Fourth, datasets often include a limited feature set – that is, they only contain information on an, often small, number of network traffic data features (such as protocol type, byte values, etc.) (Tavallaee, Stakhanova and Ghorbani, 2010). This naturally limits the complexity (and, potentially, the accuracy) of the detection models that may be built using these datasets.

Finally, there is a lack of transparency in some of the research studies in the field which makes it difficult to compare their results with those of other published studies (Shiravi et al., 2011). For example, the way in which datasets have been created and processed prior to creating detection models is often opaque, lacking the details that would be necessary for other researchers to seek to replicate the work and build on, or challenge, it.

Based on these identified issues, the next section will present the research aim and objectives that will be addressed by this thesis.

1.4 Research aim and objectives

To address these broad issues, this thesis aimed to develop an accurate cloud intrusion detection model that might be useful in mitigating DDoS in the cloud environment by discovering the relationship between attack features pairs within the different time frames. As such, the following objectives were established to achieve the overall aim of this research:

- To develop an intrusion detection model using two different datasets through the application of machine learning and transfer learning.
- To generate a cloud intrusion detection dataset including a broad range of network traffic and all types of the DDoS attack.
- To develop intrusion detection models using the generated cloud dataset to analyse the performance of the models and the relationships between attack features within five different time frames.
- To present a transparent approach to the creation of the cloud dataset, the pre-processing steps required, and the development and analysis of the detection models.

The next section will present the research questions associated with these objectives.

1.5 Research Questions

To address the objectives of this thesis, the following set of research questions were framed to guide the research:

1. Can a well-established non-cloud dataset (which includes a range of features and attack types) be used, as part of the application of machine learning and transfer learning, to develop an intrusion detection model that accurately identifies DDoS attack types in one of the few existing cloud datasets?
2. To address differences in the structure of existing intrusion detection datasets, and the variations in the feature and attack types that they contain, can a new cloud dataset be created that has a similar structure to, broadens the feature set of, and includes the same attack types as, the well-established non-cloud dataset used in the phase of the research that addresses research question 1?
3. If such a cloud dataset can be developed, can it be used to create accurate DDoS attack detection models and what effect is there on accuracy if the times periods of analysis are varied (choosing a range of periods from 1 second 'time slices' up to 10 second 'time slices')?
4. Does an analysis of the relationship between pairs of features in the developed cloud dataset help to explain any variations in accuracy across the time periods, found in answering research questions 3?
5. Can the approach to the creation of the cloud dataset, the processing of the dataset prior to the creation of the detection models, the model development, and the analysis of the dataset be presented in a transparent and clear way such that other researchers would be likely to be able to meaningfully compare their results to those reported in this thesis?

1.6 Structure of the thesis

The remainder of the thesis is structured into six further chapters.

Chapter 2 will explain the novelty of the cloud computing 'platform' and its services. It will illustrate several issues that have arisen from the adoption of cloud computing, highlighting DDoS attacks as a critical negative issue in relation to the availability of cloud computing services. A taxonomy of DDoS attacks will be presented and explained in terms of the network model and its layers in order to explain in detail how each attack type operates. The chapter will then review existing approaches in the area of Intrusion Detection Systems (IDS), which aim to detect DDoS attacks in the cloud environment, discussing why IDS systems may fail to accurately detect DDoS attacks. As a result of this discussion, the research gap that this thesis aims to explore will be framed.

Chapter 3 will illustrate the process of developing a detection model which aims to offer high levels of accuracy to detect DDoS attacks. The model was built using a well-established non-cloud dataset (NSL-KDD) and tested with one of the few available cloud datasets (CIDD) through a combination of a machine learning classifiers and transfer learning to remap the DDoS intrusion/attack types (since the two datasets do not contain exactly the same set of DDoS attacks). The results of the analysis will show that the model performed well on the non-cloud dataset, but that its performance was limited in classifying attacks in the CIDD dataset owing to the different structures of the two datasets, the small overlapping feature set and the different attack types that the datasets contained. Chapter 3 will end by arguing that, as a result of the outcomes of this first phase of practical work, there is a need to develop an original cloud-based dataset to be used in the remainder of the research reported in this thesis.

Chapter 4 will first review the different approaches that could have been used to create such a cloud-based intrusion detection dataset, explaining why an emulation approach was chosen in this research. The chapter will present all of the software and hardware specifications that formed the experimental environment. The tools that were used to generate the different types of DDoS attack and normal network traffic that comprised the final dataset (which had a similar structure, comprised a broader but overlapping feature set, and contained the same set of DDoS attack types as the CIDD dataset) will also be described.

Chapter 5 will explain the process that was used to build two detection models (one using Naïve Bayes and the other using Random Forest) with the generated dataset. The chapter will then present an analysis of the models' performance in five selected time intervals by observing the accuracy of each model and identifying feature patterns associated with any misclassifications. Through the analysis, Chapter 5 will show that the performance of the two models generally improved over time, that their best performance was at the 7 second time interval, but that no discernable patterns were found in terms of the different features that made up the misclassification cases. The Chapter will suggest that further analysis of the relationships between the features in the dataset might be useful in seeking to understand the accuracy of the models at the different time intervals.

Chapter 6 will explain the use of cross-correlation analysis, which was applied to the dataset to discover relationships between pairs of features over time, with the aim of understanding the accuracy of the models at the different time intervals. The chapter will explain the approach taken, leading to the identification of the 'highest significant correlation' value between the feature pairs in the generated CCF graphs. Chapter 6 will then present the results of the analysis, which will show that there was a general increase in the frequency of 'highest significant correlation' values between feature pairs as the time intervals increased, but that there was not a direct relationship between the frequencies and the accuracy at the different time intervals for either model.

Chapter 7 will provide a brief review of each of the chapters in the thesis before presenting answers to the research questions set out in section 1.4. From here, the chapter will frame the overall contributions of the research. Limitations of the work will be identified and discussed and areas for future work will be considered.

1.7 Summary

This chapter has provided an introduction to, and overview of, this research effort. It has highlighted the growing importance of cloud computing and the prevalence and impact of DDoS attacks in cloud computing environments. Further, it has introduced research into DDoS attack detection in cloud computing, illustrating the challenges for existing IDS systems in detecting DDoS attacks in cloud environments. The chapter has introduced a number of broad issues that are important when considering research in this area and used them to frame a set of research questions that the remainder of the thesis will seek to address. Finally, this chapter has explained the structure of the thesis, providing a brief summary of each of the remaining chapters.

Chapter 2 : Cloud Computing and Distributed Denial of Service Attacks

2.1 Introduction

A definition of cloud computing, its essential characteristics and its different models of deployment and services, will be explained in this chapter to highlight the advantages that cloud computing has brought to the computing domain. The chapter will then describe number of issues that have arisen as a result of adopting cloud computing. It will review these issues and then focus on the security threats that represent the most critical issues in the cloud, focusing on what is argued to be the most important security issue: Distributed Denial of Service (DDoS) attacks. The chapter will explain why there is this critical problem in the cloud and introduce a taxonomy of DDOS attacks, discussing the network model and its layers to highlight the structural issues in the protocols at the layers that are exploited by the different attack types, before analyzing the associated intrusion detection systems. This will lead to the identification of weaknesses with the current detection systems, framing the research gap on which this thesis will focus.

The chapter is structured as follows. Section 2.2 will introduce basic cloud computing concepts to set the context for the area of interest. Section 2.3 will explore the major security issues in cloud computing before arguing, in section 2.4 that DDoS attacks represents an important security issue. Section 2.5 will provide a taxonomy of DDoS attacks, explaining them in terms of the network model and its layers in order to provide a detailed understanding of how each attack type operates. This will lead to a discussion of systems that seek to identify DDoS attacks in the cloud (called cloud intrusion detection systems) of and issues with them, which subsequently frames the research gap, proposed in section 2.7.

2.2 Basic Cloud Computing Concepts

This section will explain the growth in the adoption and use of cloud computing, so it is important to understand the main characteristics and the core infrastructure of cloud computing that has driven this growth. The chapter will then focus on security issues that are associated with, or arise from, these features of cloud computing, setting the scene for the focus of this thesis on a specific type of security threat (DDoS attacks).

According to the National Institute of Standards and Technology (NIST), "cloud computing is a model for enabling convenient, on–demand network access to a shared pool of configurable computing resources (e.g., network, server, storage, application, and services) that can be rapidly provisioned and released with minimal management effort of five essential characteristics, three service models, and four deployment models" (Mell and Grance, 2011, p.2). Figure 2.1 captures the fundamental characteristics of cloud computing, and presents the different types of cloud service and deployment model. Each area will be explained in detail in the remainder of this section.

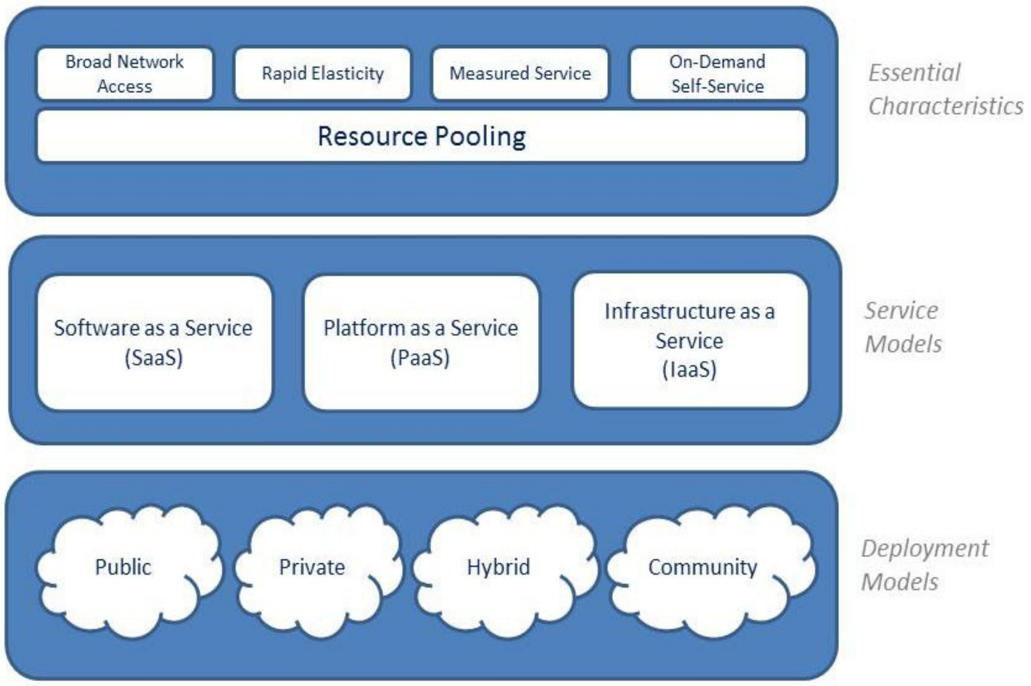


Figure 2.1 Essential characteristics of cloud computing (Vaquero et al., 2009)

As Figure 2.1 shows, in the top layer, cloud computing makes use of five essential characteristics. The 'broad network access' characteristic is the capability of providing all the computing resources that are available over the network, accessed through standard mechanisms that promote use by heterogeneous thin- or thick-client platforms (e.g., smart mobile phones, laptops, and PDAs). The 'rapid elasticity' characteristic is the capability of quickly and elastically managing computing resources, in some instances automatically, to rapidly scale out and promptly release resources to quickly scale in. There are almost unlimited available computational resources/capabilities for provisioning to customers/clients that can be purchased at any time and in almost any quantity. The 'measured service' characteristic offers a metering capability for the resources used, based on a pay-as-you-go basis. It enables cloud systems to automatically govern and enhance resource usage, and provides transparency for both provider and customer/client. The 'on-demand, self-service' characteristic is the capability of providing almost unlimited computing resources, such as network and storage, as required, without any interaction with a service provider. Finally, at this layer, the 'resource-pooling' characteristic enables a service provider to pool all the computing resources to serve many customers/clients based on a multi-tenancy model. It also dynamically allocates or reallocates the computing resources according to customer's/client's demands.

As the second layer of Figure 2.1 shows, the five fundamental characteristics of cloud computing are implemented in three different types of service model that are offered to organisations and individuals on a pay-as-you-go basis: Software as a Service (SaaS); Platform as a Service (PaaS); and Infrastructure as a Service (IaaS).

In the SaaS model, the cloud provider offers an application that is hosted on the cloud infrastructure as a service to customers/clients. This level of service is accessible from various client devices through a thin interface, such as a web browser (e.g., web-based email) or dedicated apps. The clients are free of managing or controlling any aspect of the underlying cloud infrastructure, including storage, servers, operating system and network. Dropbox is one of the examples of this model, allowing users to share files and folders easily (Shahzad, 2014).

In the PaaS model, a cloud provider offers a computing platform for developers, including the operating system, programming languages, database, web servers, etc. These services allow clients to deploy, manage and test their own applications without the need to install any software on their local machine and without the need to manage and control the underlying cloud infrastructure. Amazon Web Service (AWS), Windows Azure, and Google App Engine are examples of PaaS offerings (Samimi, Teimouri and Mukhtar, 2016).

In the Infrastructure as a Service (IaaS) model, a cloud provider delivers virtualized computing resources such as virtual machines, network and storage. At this service level, clients also do not manage or govern the underlying cloud infrastructure, but they do have limited control of some networking components. Microsoft Azure and EC2 are examples of IaaS offerings (Al Morsy, Grundy and Müller, 2010).

As the last layer of Figure 2.1 shows, the three levels of the cloud computing service delivery are commonly deployed in four different models: Public cloud; Private cloud; Hybrid cloud; and Community cloud.

Public cloud platforms are those that are made available by a particular organisation (who also host the service) for general public use or for the use of a large number of industry groups, such as government organisations and businesses (Zhang, He and Liu, 2008). In contrast, in the Private cloud deployment model, the cloud infrastructure is implemented only for a single organisation's use. This type of deployment might be managed by third party organisation (Zissis and Lekkas, 2012). In the Hybrid cloud deployment model, the cloud infrastructure comprises two or more different cloud infrastructure types (private, community, or public) that stay as unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (Alam, Pandey and Rautaray, 2015). In the Community cloud deployment model, cloud infrastructure is shared by a specific community of organisations that have a shared concern. The infrastructure may be managed by third party organisation (Akherfi, Gerndt and Harroud, 2016).

This section has introduced the service delivery types and deployment models associated with cloud computing to provide a broad view of cloud computing. The security issues on which this research will focus can affect all of the deployment models and service deliveries types, and are significant, not least because of the growth in cloud computing, which will be addressed in the next section.

2.2.1 Growth in migration to cloud computing

This subsection will describe how the novel characteristics of cloud computing have brought advantages to the computing domain, helping to explain why there has been such a growth in the migration to the cloud computing. This subsection will then describe the most common issues experienced when adopting cloud computing because it is also important to appreciate the problems that may be faced in order to be able to address them so that this model of computing can be effectively utilized.

The on-demand characteristic of cloud computing leads to the optimization of computing resources. The resource pooling characteristic provides faster access to almost all computing resources. The measured-service characteristic of cloud computing, based on the pay-as-you-go pricing model, results in minimizing the customer's/client's upfront costs of IT infrastructure. Moreover, this paradigm considerably reduces the operational cost for organisations and improves business agility by offering highly scalable and available software and hardware infrastructure as a result of the broad network access and rapid elasticity characteristics. This is reflected in Forrester's research, which suggests that "80% of cloud adopters believe that it helps their organisation to reduce IT cost" (RapidValue Solutions, 2015). The scale of interest in cloud computing is captured by Right Scale, who report that "The global cloud computing market will grow more than $241 billion by 2020" (RapidValue Solutions, 2015).

The appearance of novel cloud computing platforms represents a critical change in the way in which Information Technology (IT) services are designed, developed, deployed, scaled, updated, paid for, and maintained. The novel characteristics of cloud computing, providing the three service models and four deployment models introduced in section 2.2), have brought substantial advantages to the computing domain. All of these advantages of cloud computing have led to it gaining significant popularity and being broadly adopted in different industries and academia (Changchit and Chuchuen, 2018).

Despite there being a noticeable growth in adoption driven by the benefits of the cloud computing model, migration has created a number of issues related to the technological, environmental and organisational aspects of cloud computing adoption. According to relevant literature, the most important issues relate to data security, availability and data lock-in (Morgan and Conboy, 2013).

Data security is the top concern of companies considering moving their data and computing resources to the cloud. This may be because they are not fully aware of how security at all levels can be achieved when their data are held away from their premises (Nedev, 2014). Availability related to services and resources is another risk when using the cloud-computing model. In the case of an outage, caused by network or system failure or by cyber-attacks, the unavailability of services would critically affect the organisation's and/or their operation overall (Behl and Behl, 2012). Data lock-in issues refer to problems associated with the lack of compatibility among existing cloud technologies and may result in the dependency of customers on their current cloud vendor's products and services, meaning that they would be unable to change suppliers without (often significant) switching costs (Nedev, 2014).

Consequently, to effectively utilize the innovations offered by cloud computing, these issues need to be taken into consideration. This research, therefore, aims to focus on specific aspects of the main concern associated with the adoption of cloud computing: security issues.

2.3 Cloud Computing Security

This section will describe the major security issues in cloud computing in order argue what this research sees as the most crucial security issue, before developing approaches that seek to address it though practical work.

Although section 2.2.1 has already presented the core characteristics associated with cloud computing, various service delivery and deployment models of cloud computing provide enhanced and optimized services to customer/clients. They introduce new security issues in addition to the existing, inherent security issues of conventional IT infrastructure. These security issues seek to address the vulnerabilities and potential threats to the infrastructure and characteristics of cloud computing that are subject to various types of cyber-attack (Gonzalez, et al., 2012). A vulnerability is defined as a weakness of the cloud computing model that can be exploited by a cyber attacker to gain access to cloud resources (Modi, et al., 2013). A threat in cloud computing is a potential factor that causes serious damage to the cloud resources. A cyber-attack is an action that exploits the vulnerabilities of the cloud infrastructure, resulting in disrupting and compromising cloud resources (Somani, et al., 2015). The security issues in cloud computing environments that have been ranked as the top concerns in relation to its adoption are multi-tenancy, insecure APIs, account hijacking, shared technology, data breaches and distributed denial of service attacks (Morgan and Conboy, 2013).

The security issue associated with multi-tenancy in cloud computing is about its underlying infrastructure technology, which based on a multi-tenancy model. For example, in a cloud environment various operating systems will be running on a host operating system. In this kind of configuration, an attacker can compromise the host operating system to gain access to all of the other operating systems (Modi et al., 2013).

Security issues related to Insecure Application Programming Interfaces (APIs) reflect the critical role of APIs in cloud computing architecture, as users employ APIs to manage and interact with cloud services (Wagner, 2015). Unavailable or insecure APIs have a direct effect on the accessibility of cloud resources. Therefore, APIs must protect properly against any malicious attempt to access the resources to which they relate. That this is a significant problem is reflected in the fact that: "In 2015, the US Internal Revenue Service (IRS) exposed over 300,000 records via a vulnerable API" (DarkReading, 2012).

Account hijacking represents a security issue where individual or organisational accounts are 'stolen' to impersonate the account owner. Account hijacking is not a new security issue, but the advance of cloud computing amplifies the issue's impact. F or example, if an attacker gains access to the user's credentials, it can result in eavesdropping on user activities and transactions, manipulating data and redirecting a user to malicious sites (Gonzalez et al., 2012). In 2014, for example, Amazon Web Services (AWS) was compromised when it was unable to guard the administrative console with multi-factor authentication. As result, all of the company's assets were 'cracked' (Bourne, 2014).

Security issues related to shared technology refer to shared infrastructure and application scalability in cloud computing. For example, an underlying component of the cloud, that supports the cloud service delivery model, may not be strongly designed to support the multi-tenancy nature of cloud architectures (Gonzalez et al., 2012), making it subject to shared technology vulnerabilities. This issue creates risk in the cloud environment because it can potentially disturb the whole cloud system at once.

The data breach security issue is about confidential data that may be released as a result of unauthorized access. A data breach can result from application vulnerabilities, poor security mechanisms, and human error. This security issue is not unique to cloud computing – traditional network infrastructures are subject to the same risk; but the cloud's use of shared resources and the highly-accessible nature of cloud computing services make it a more attractive target for attackers (Cloud Security Alliance, 2016). The scale of such breaches, which are seemingly quite common, can be significant. For example, the UK-based TalkTalk telecommunications company had a security breach incident that resulted in the extraction of four million clients' personal banking information (Gibbs, 2015).

The Distributed Denial-of-Service (DDoS) attack is one of the most significant security issues for, and powerful threats to, cloud computing (Sabahi, 2011). This type of attack aims to saturate the network with a high volume of unwanted traffic to prevent users from accessing services and applications. In a cloud computing environment, because of the distributed nature of the cloud and its nature as a shared-resource platform, the impact of DDoS attacks is greater than in traditional network infrastructures. DDoS typically takes advantage of the characteristic of cloud computing in order to launch the attacks (Gonzalez et al., 2012). The impact of such attacks can be high for organisations. For example, a DDoS attack on the Amazon cloud infrastructure caused a service outage of over 19 hours (Metz, 2009).

This section has introduced and explained the top security issues in cloud computing to provide the broad security context in which this research effort is located. This research will focus on security issues related to DDoS attacks as the rate of DDoS attacks is increasing in cloud computing environments and it is ranked as one of the top security issues (Modi *et al.*, 2013).

2.4 Distributed Denial of Service Attacks

This section will introduce the DDoS attack and its process before explaining why the rate of DDoS attacks is increasing in cloud computing. This is important because it is imperative to understand what is different about cloud configurations that cause this type of attack to be more successful in this type of environment.

As has already been explained, a DDoS attack aims to render unavailable all computing resource and to disturb the offered services in different ways (which will be explained in more detail in section 2.5). Figure 2.2 presents the process of a DDoS attack.

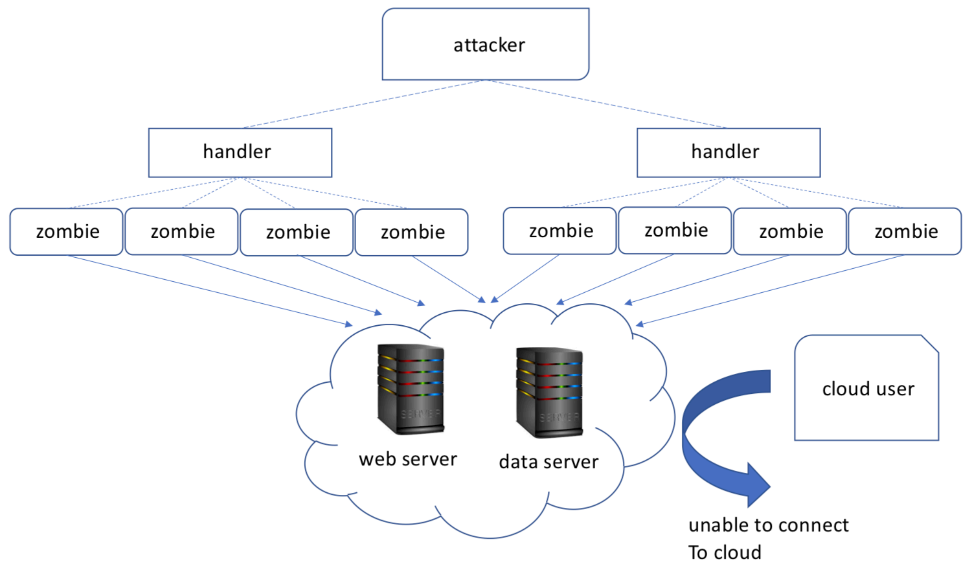


Figure 2.2 The DDoS attack process (Somani et al., 2015)

In the DDoS attack process, an attacker can be one or more person. A handler is (one of) the attacker's machine(s), also called masters or botnet controllers, and is capable of monitoring multiple zombies/botnets. Zombies (botnets) are compromised machines that are running a program that is responsible for sending a stream of traffic to the targeted system (also known as the victim). The attacker scans the network with various network scanning tools to identify vulnerabilities. Afterwards, through the exploited vulnerabilities, the attacker can gain access to the victim's machines (a web server or a data server in Figure 2.2) and compromise them with the specific attack program, which is called malware (Catteddu and Hogben, 2009).

The rate of DDoS attacks is growing in cloud computing environments (Bhushan and Gupta, 2018). As already noted, the main features and characteristics of cloud computing (see section 2.2) and the related security issues in the cloud (see section 2.3), and vulnerabilities of the Internet in general, are all factors that contribute to the occurrence of DDoS attacks in cloud computing (Beitollahi and Deconinck, 2012). Moreover, one of the major reasons for the growth of DDoS attacks is the development of the botnet. The on-demand characteristic of cloud computing contributes to enabling the deployment of powerful botnets for DDoS attacks, whereas in traditional network infrastructures it is quite complex to compromise many machines in a short period of time and use them as botnets to launch a DDoS attack (Santanna et al., 2015).

The broad network access and elasticity characteristics of the cloud create an attractive environment for the attacker to produce more frequent, large, and complex DDoS attacks to overwhelm the victim network, as attackers take advantage of the cloud's high-speed broad network and ability to deploy botnets (Adhianto et al., 2010).

The cloud's resource pooling characteristic, based on virtualization and multi-tenancy technologies, provides opportunities for an attacker to use a virtual machine (VM) to build more botnets for performing an attack, which uses less memory and other storage capacity with lower costs. In addition, in a multi-tenancy environment, an attack against one of customer is an attack against all customers in the targeted cloud system (Stillwell et al., 2010).

Finally, the measured service characteristic in the cloud, which is based on the pay-as-you-go model, has the potential to be exploited by an attacker in a way that consumes more cloud resources. This can result in financial losses and service downtime for a victim (Somani et al., 2015). For example, Greatfire.org was attacked by a heavy DDOS attack leading it to accumulate a high bill of $30,000 per day on Amazon's EC2 cloud (Munson, 2015).

Having introduced the general idea of a DDoS attack, and explained how the cloud environment may be more vulnerable to such attacks, the next section will consider different types of DDoS attack.

2.5 A Taxonomy of DDoS Attacks

This section will introduce the main ways that DDoS attacks might happen in a network. It will also present a taxonomy of DDoS attacks to show the similarities and differences between the different attack types. To understand the attack types, it is important first to understand the network communication model and related protocols associated with each layer of the network to appreciate how data is transferred over the network according to a set of defined protocols. Then, the functionality of each protocol in relation to each layer of network model will be explained; as the protocols are introduced, reference will be made to how some features of these protocols might be vulnerable in relation to the occurrence of specific DDoS attack types related to each layer of network model in cloud computing network environments.

The Open System Interconnect (OSI) model and Transmission Control Protocol/Internet Protocol (TCP/IP) models are the most popular network models in use today. The four layers of the TCP/IP network model map onto to the seven layers of the OSI model, showing that they closely correspond. In this research, it was decided to use the TCP/IP network model because it is a more generic model to describe the different layers of the network and the associated protocols than the OSI model.

The TCP/IP network model is one of the most common abstract models of networking or Internet communications; it explains the operation of a network in terms of how hardware (computers) can connect and operate across it (Zaman and Karray, 2009). This model, often referred to as the TCP/IP suite or stack, specifies how data must be packaged, addressed, broadcast, directed and received over the network by defining a set of rules for each layer of the model (Juniper, 2017). Figure 2.3 presents the model and shows that it is organized into four layers, with each layer having a set of communication protocols that have their own functionality. Moreover, according to Osanaiye, Choo and Dlodlo (2016), the different DDoS attacks can be grouped according to which network model layer that they target (see Figure 2.4).

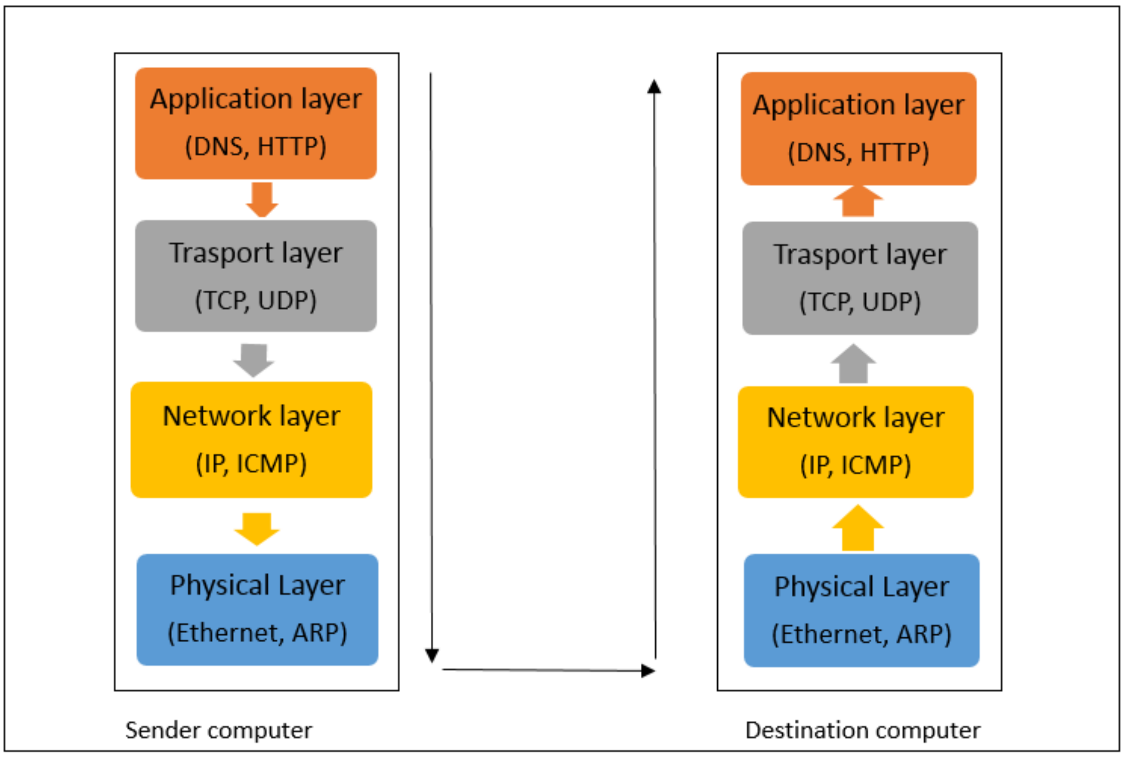


Figure 2.3 The TCP/IP network model (Lakshman and Madhow, 1997)

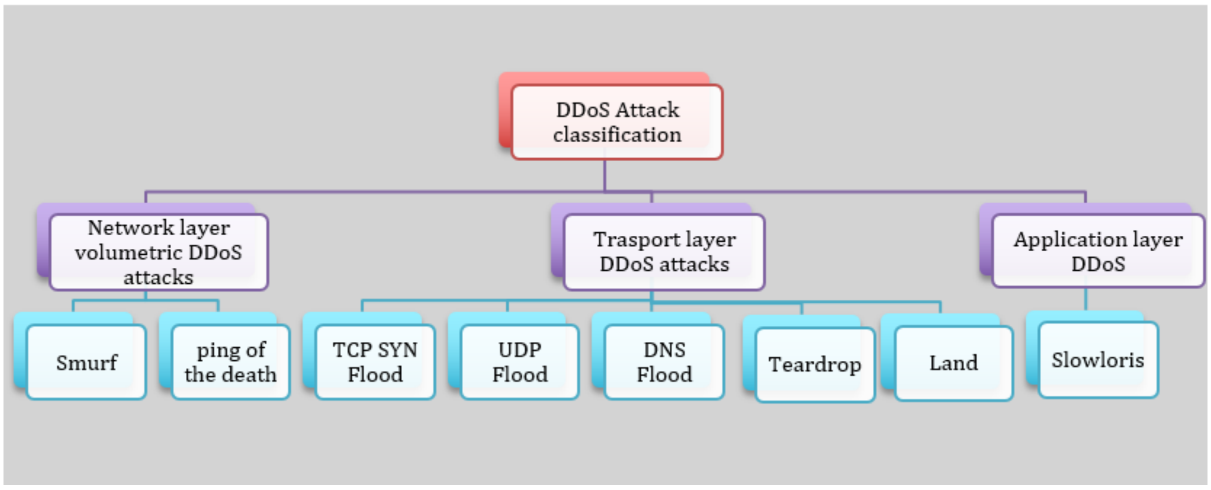


Figure 2.4 A taxonomy of DDoS attack types in a cloud computing, organized by the network layer that they target (Osanaiye, Choo and Dlodlo, 2016)

For example, when a user (the ‘sender computer’ in Figure 2.3) sends a request to access services from the cloud server (the ‘destination computer’ in Figure 2.3), the request travels from the top (application) layer to the lowest (physical) layer to be received by the cloud server (as the arrow shows in Figure 2.3). Then, it moves from the lower level to the top layer of the destination computer to be processed. The request takes the form of packets in the network. A packet is a unit of data that follows a specific format to be transmitted on a network (Unuth, 2017). At the lowest level in the model (see Figure 2.3), the physical layer (also called data link layer) is responsible for interconnecting host computers in the network, framing (converting the raw bits to groups of bits, called packets, and adding a link layer header to create a frame) and physical addressing (adding a hardware-level address by translating the addressing used in the Internet Protocol layer to link layer addresses, such as Media Access Control (MAC) addresses) (Incapsula, 2017). The physical layer is not the focus of this research and is not directly relevant to our consideration of DDoS attacks so will not be considered in further detail.

2.5.1 The network layer: protocols and attack types

The second layer up, the network layer, checks the routing of the data over the Internet. Internet Protocol (IP) and Internet Control Message Protocol (ICMP) are the main protocols associated with this layer (Cobb, 2017). IP is responsible for routing and addressing a packet from a sender host machine to a destination host machine over the network (Techopedia, 2017). ICMP is used as an error reporting protocol for network devices such as routers (which forward the data between devices on the network). ICMP generates and sends error messages or operational information to the source IP address (sender) when the network is unable to deliver the sent IP packets, or a request cannot be fulfilled. Returning to our example, if the desired cloud services are not available, an ICMP message is used to deliver the information to the sender that "the service is unavailable" or that "the host is unreachable" (Cha and Kim, 2011). The ICMP protocol is therefore mostly used for diagnostic purposes.

Figure 2.5 illustrates the format of the IP protocol header for Internet Protocol Version 4 (IPV4) and Internet Protocol Version 6 (IPV6), identifying a packet which contains 14 fields in IPV4 and 8 fields in IPV6. Each of the fields will now be explained, before the ICMP message structure is addressed.

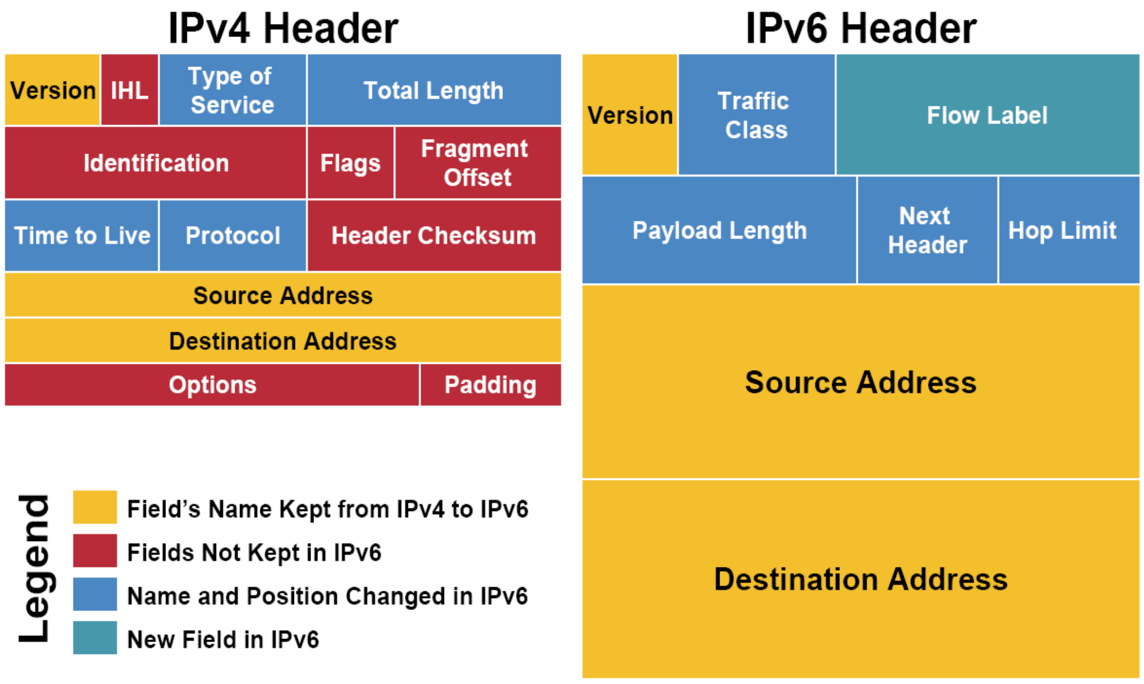


Figure 2.5 The structure of the IPV4 and IPV6 packet header (Al-Shaeya, 2010)

There are some fields in the IPV4 header that correspond directly to IPV6 header fields. In fact, IPV6 has been designed to be basically similar to IPV4, with only minor modifications with regard to protocol details (Comer, 2006 p.563). Figure 2.5 uses different colours to indicate the changes in the fields from IPV4 to IPV6. The Version field (4 bits) in IPV4 identifies the Internet protocol version 4. The Version field in IPV6 contains the same detail, but includes 6 bits for this field. The IHL field (4 bits), defines the packet header size; but in IPV6 this field is replaced by the16-bit Payload Length field that specifies the size of the packet plus the size of header. The Type of Service field in IPV4 (8 bits) specifies the quality of the service, such as delay, throughput, etc.; however, this field has been renamed to the Traffic Class field in IPV6. The Total Length field in IPV4 (16 bits) gives the sum of the size of the actual data being communicated and the size of the packet header, with the maximum size being 65,535 octets (an octet is a unit of digital data that consist of eight bits).

The Identification field in IPV4 (16 bits) includes a specific number which is related to the packet fragment. Fragmentation is a process that breaks packets into smaller pieces (fragments) in order to pass them through a transmission link so that they may then be re-assembled by the destination computer. For example, if an IP packet is fragmented during transmission, all the fragments include the same identification number to specify the original packet to which belong. The Flag field in IPV4 controls the fragmentation process and includes 3 bits. The first bit must be set to 0. The second bit is called the DF (don't fragment) flag, indicating that the packet should not be fragmented. The third bit is called the MF (more fragment) flag, which indicates that there are more fragments to come. The Fragment Offset field in IPV4 contains 13 bits and is usually used in the reassembly of fragmented packets and indicates the position of a fragment in the original fragmented IP packet.

The Time to Live (TTL) field in IPV4 includes 13 bits, which identify the maximum time that a packet can stay on a network and thus prevents a packet from looping in a network indefinitely. The maximum value of TTL is set to 225 (the measurement unit is seconds). Whenever the packet passes through the router, the value is decreased by one; once it reaches zero, the router drops the packet. The TTL field in IPV4 corresponds to the Hop limit field in IPV6. Unlike in IPV4, IPV6 interprets the TTL value as the maximum fixed value of hops that the packets can make before being discarded.

The Protocol field in IPV4 includes 8 bits and specifies the protocol of the higher level to which the packet should be routed. The Header Checksum field in IPV4 indicates the checksum value of the whole header and is used by the destination machine to check for errors in the packet header. The Source Address field in IPV4 contains 32 bits and specifies the source address from which the packets originate/are being sent; this corresponds to the field of source address (128 bits) in IPV6.

The Identification field, Flags field, Fragment Offset field and Header Checksum field have been replaced with the Flow Label field and Next Header field in IPV6. IPV6 uses the information in these fields to associate packets with precise flow and priority.

The Destination Address field in IPV4 contains 32 bits that identify the intermediate or final destination address of the packets; this maps to the Field of Destination Address (128 bits) in IPV6. Finally, the Option field and Padding field in IPV4 are often used for security and control of the packets. IPV6 does not have these fields (Zargar, 2013).

As has already been noted, the second protocol at this layer is ICMP. Figure 2.6 demonstrates the format of an ICMP message that is transmitted as an IP packet over the network. The packet includes the entire IP header that encapsulates the ICMP message. The Type section determines the type of ICMP message that is being sent to the host; the Code section includes more information about the type section; and the Checksum section indicates the error during the transmission process. Therefore, all these three sections of an ICMP packet plus the original IP header identify whether, and if so for what reason, the packet failed to reach its destination.

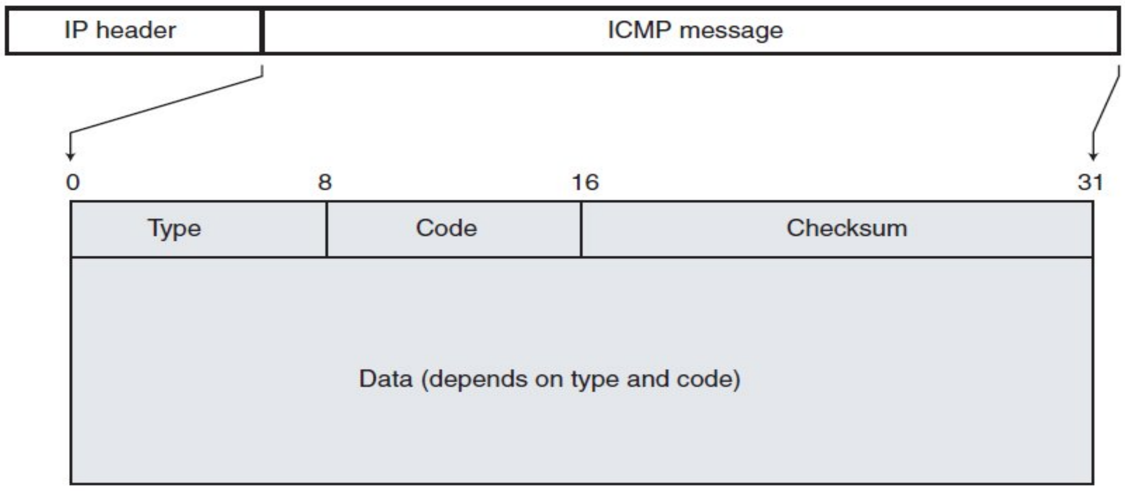


Figure 2.6 The format of an ICMP message

As Figure 2.4 illustrates, the network layer is associated with two types of DDoS attack: Smurf and Ping of Death (PoD).

In Smurf DDoS attacks, the attacker exploits the ICMP protocol. The attacker starts an attack by generating ICMP packets and attaching the 'spoofed' (or false) IP address of the targeted computer (in our earlier example, a cloud server) to a packet which is broadcast to the network using an IP broadcast address. The aim is to solicit echo responses from all the devices on the network. Since routers, through which the packets will usually be directed, often communicate with a high number of devices, such as VMs in the cloud, many responses may be generated. This makes the targeted cloud server expend all its resources handling the requests, and results in a successful DDoS attack in terms of consuming the available bandwidth and leaving the targeted server unable to process legitimate requests (Beitollahi and Deconinck, 2012).

In PoD DDoS attacks, the attacker uses the ‘ping’ function, a utility tool for testing the connectivity of a remote host which operates by generating an ICMP message echo request to the targeted host expecting an echo reply. The attacker sends malformed ‘ping’ packets, which exceed the maximum packet size, to the targeted system. Since a ping packet greater than 65,535 bytes disrupts the Internet Protocol, the targeted system ends up with an over-sized packet, causing buffer overflow and resulting in a crash (Darwish and Capretz, 2013).

2.5.2 The transport layer: protocols and attack types

The Transport layer, the third layer in Figure 2.3, is responsible for providing logical communication between various hosts running application processes (such as cloud servers). These communication services are implemented by two crucial protocols in this layer: The Transmission Control Protocol (TCP); and the User Datagram Protocol (UDP) (Kristof, 2002).

TCP is the main protocol of the network layer. This protocol allows host machines to establish a connection to exchange data packets over a network. It ensures the delivery of data packets to a destination and guarantees that the sequence of the packets is the same by using the times that they were sent (Rouse, 2014). The TCP connection process is known as a “three-way-handshake”. First, a legitimate user sends a request (in the form of packet) to the server to establish a connection by sending a SYN message (a SYN – short for synchronize – message is a communication between two applications) to the server. Next, the server acknowledges the request by sending a SYN-ACK (synchronize-acknowledge) message to the user. Finally, the user replies with an ACK (acknowledge) message that means an acknowledgement of the received data, and the connection is established. Figure 2.7 shows the TCP packet header format.

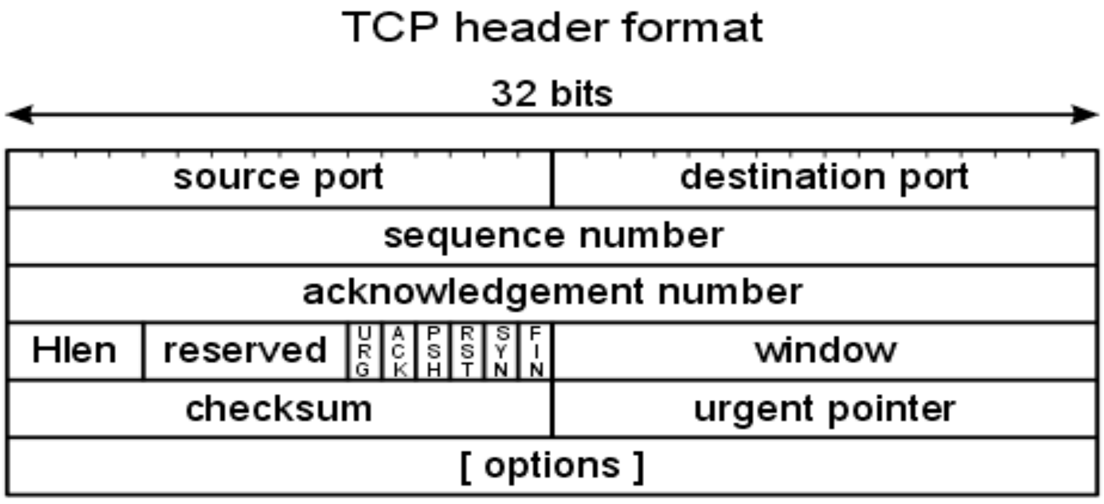


Figure 2.7 The format of a TCP message

The Source Port field (16 bits) identifies the source port number of the sender. The Destination Port (16 bits) specifies the port number of the destination computer. The Sequence Number field (32 bits) includes a random number that specifies the first initial segment number that was sent when establishing the TCP three-way-handshake (explained above). The Acknowledgement Number field (32 bits) includes a sequence number that is generated by the destination computer to inform the sender computer of the expected octet number (Tetz, 2011). The HLEN field (4 bits), also called data offset, identifies the size of the TCP header. Depending on the option selected in the “Option field”, the size may vary. The Reserved field (3 bits) is set to zero and it is for future use.

The Flag Field (6 bits) controls the flow of the data by the numbers of flags; the SYN flag (1 bit) and the FIN flag (1 bit) identify the initiation and termination of the connection, respectively. The ACK flag (1 bit) shows the acknowledgement section of the connection. The SYN flag (1 bit) indicates the sequence number of the synchronized message. The PSH flag (1 bit) is for requesting a push function. The URG flag (1 bit) is for use when the urgent pointer field is valid.

The Windows Size field (16 bits) indicates the size of the buffer set by the destination machine (receiver) in each connection, pointing to the sender machine how much data is expected to be received. The RST flag (1 bit) specifies the end of the connection. The Checksum field (16 bits) specifies the integrity of the TCP segment data, to determine whether any segments have been damaged, and is mostly used by UDP. The Urgent Pointer field is only significant when the URG flag is set.

As has already been mentioned, the other protocol used in the Transport Layer is UDP (User Datagram Protocol). The UDP protocol is a connectionless Internet protocol which provides a mechanism to send the message from an application program on one host to the application program on another host in the cloud environment. Unlike the TCP protocol, the reliability of the packet transmission is not compulsory for this protocol. Ordering the incoming messages, providing feedback to the sender, and having an acknowledgement process are not the responsibilities of the UDP protocol. Therefore, a UDP message may arrive out of order or may be lost. Examples of the use of the UDP protocol are in on-line-chat and online-gaming (Incapsula, 2017). All of these issues make this protocol vulnerable to DDoS attacks.

Figure 2.8 illustrates the format of the UDP packet header, which includes four fields plus data.

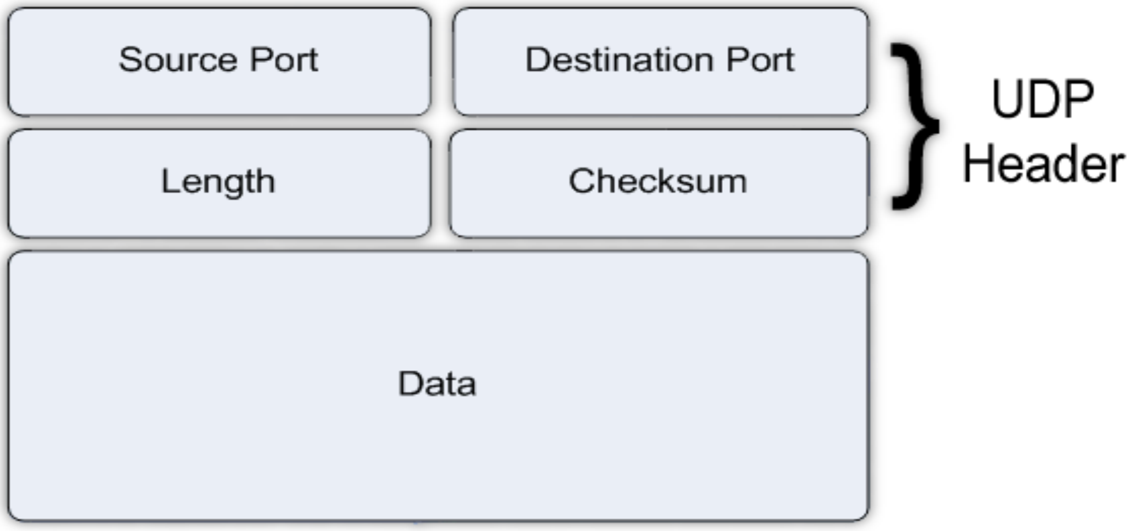


Figure 2.8 The format of a UDP Packet header

The Source Port and Destination Port fields include 16 bits that specify the sender's and receiver's port, respectively. Different ports are allocated to common/well-known services, some are used for registered services and others may be used for any purpose. The Source Port number is an optional field and is usually set to zero if it is not used.

When it used, it identifies the port to which the replied message is expected to be sent. The length field includes the total octets in a UDP packet, and includes the header and the data. The minimum value for this field is eight bits. The checksum field includes 16 bits, is an optional field and is set to zero, which means that it has not been computed.

As Figure 2.4 illustrates, the Transport Layer is associated with five types of DDoS attack: TCP-SYN attack, Teardrop attack and Land attack often exploit the TCP protocol; UDP Flood attack and DNS Flood attack frequently exploit the UDP protocol.

In a TCP-SYN DDoS attack (see Figure 2.9), an attacker exploits the TCP connection (three-way handshake) by first sending frequent SYN packets to the targeted cloud server, often using a fake IP address. Subsequently, a server that is unaware that the attack taking place replies to each connection with a SYN-ACK packet from each open port. As a result, in the third stage, either the cloud server never receives the SYN-ACK, in the case of using a spoofed IP address, or a malicious user never sends the expected ACK. Therefore, the server will wait for acknowledgement of the final ACK, and the connection will be left open. The server frequently receives SYN requests before the connection times out. This leaves a high number of open connections to the server, and results in services being denied for legitimate users and the server crashing (Mirkovic and Reiher, 2004). One example of a successful TCP-SYN flood attack was on Amazon Cloud Web Services in 2011 (RapidValue Solutions, 2015).

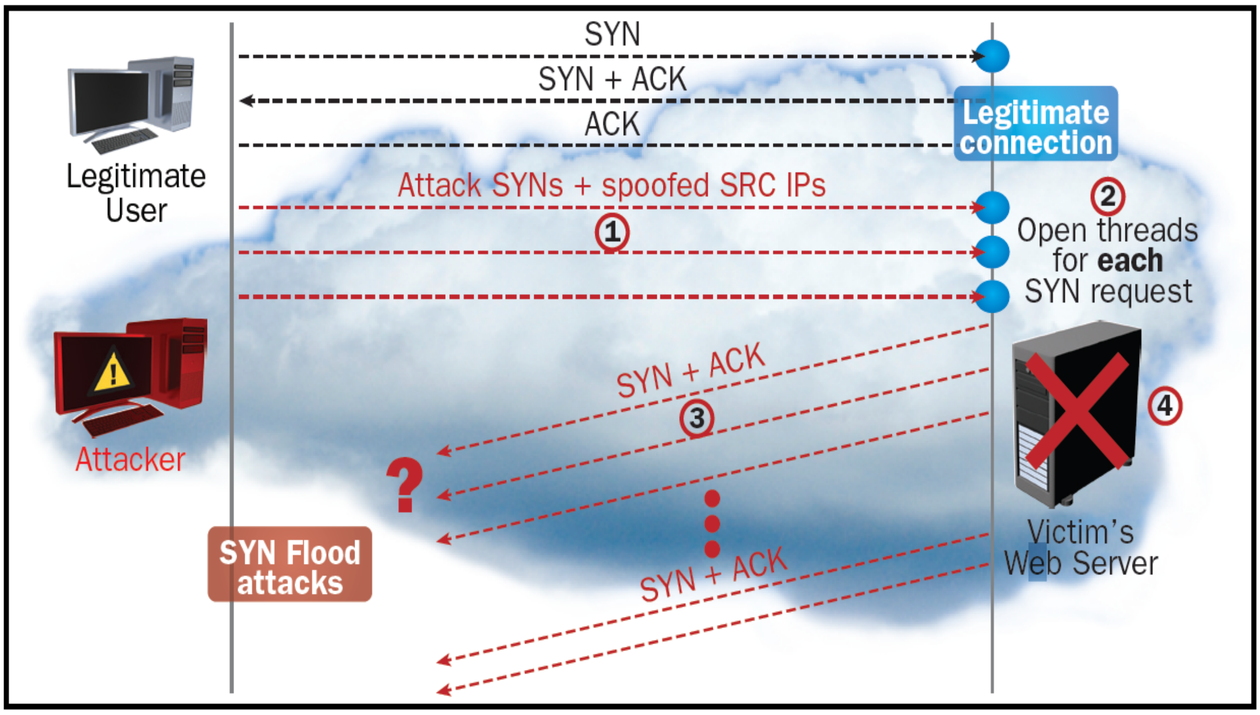


Figure 2.9 TCP-SYN DDoS flooding attacks (Radware, 2013)

In a Teardrop attack, an attacker exploits the data offset field of the TCP header packet, sending malformed TCP packets to the targeted cloud server. Once the victim (a cloud server) has received these packets, owing to the bug (with fragment overlapping offset) in the TCP packets the victim/server will be unable to reassemble these packets (Somani et al., 2015). As result, the packets overlap, and this crashes the system.

In a Local Area Network Denial (Land) attack, an attacker exploits the Source Port and Destination Port fields of the TCP packet header (as explained earlier; see Figure 2.8). The attacker sends spoofed TCP packets to targeted machines and sets these packets in a way such that the host Source Port and host Destination Port are the same (Moradian and Science, 2006). This makes an empty connection which replies to itself until the targeted machines are overwhelmed, with all their resources consumed by the attack.

In UDP Flood attacks, an attacker commonly exploits the protocol's vulnerabilities, such as the unreliability property of UDP, to launch a flooding UDP attack (Incapsula, 2017). An attack is initiated by generating a UDP packet; as the UDP protocol does not have a specific packet format, the attacker makes an oversize packet and fills it with random numbers or text. Afterwards, these UDP packets are sent to many random ports on the targeted cloud server, and then the targeted cloud server checks the associated application on the port that 'listens' to the incoming UDP packet. Once the targeted cloud server observes that the given UDP packets are not associated with the 'listening' application on the port, it responds with an "ICMP destination, unreachable packet" (Osanaiye and Dlodlo, 2015). As a result, UDP packets will fill the reply queues and the cloud server will be unable to respond to legitimate clients, resulting in a denial of service attack.

In a Domain Name System (DNS) Flood attack, an attacker attempts to overwhelm the DNS server. The DNS server is known as the 'phone book of the Internet' and provides useful information about the domain names of all the connected devices on the Internet and translates them to Internet Protocol (IP) addresses (Beal, 2017). It also maps the domain name of services to IP addresses. All Internet-based services, such as cloud computing, and web browsing, rely on DNS. A DNS attack is usually initiated by sending a high number of DNS requests, generating UDP traffic which causes legitimate users to be unable to access to the DNS server (Hope, 2017).

2.5.3 The application layer: protocols and attack types

The Application Layer, the top layer in Figure 2.3, is used to invoke application programs that have access to most of the services on the Internet. This layer, then, interacts with the lower layers, such as the Transport Layer, for the sending and receiving of data. The Hypertext Transfer Protocol (HTTP) is one of the most important protocols associated with the Application Layer. It is responsible for exchanging information and transferring various types of file on the Internet (Cobb, 2017). The responsibility of this protocol is in the way in which messages are configured and sent, and what sort of actions web servers and browsers should take to reply to different commands. For example, when a user enters a web address in a browser, the HTTP protocol directs this request to the web server (Beal, 2017). An HTTP request includes a header, a blank line and the information being sent. Table 2.1 presents the structure of the HTTP header and the meaning of each of its fields. (There are also other protocols associated with this layer such as Simple Mail

Transfer Protocol (SMTP), File Transfer Protocol (FTP)) but HTTP is used here as an illustrative example given its ubiquitous use).

Header	**Meaning**
Content-length	Size of the item in octet
Content-Type	Type of the Item
Content-Encoding	Encoding used for the item
Content-Language	Languages used in item

Table 2.1 The HTTP header

In a Slowloris DDoS attack, an attacker initiates the attack by sending incomplete but legitimate HTTP packets to the targeted cloud web service to open a connection, and then holds the connection open and does not release it before the system reaches the maximum number of allowable open connections. This results in a denial of service attack (Defense, 2015). Unlike the other forms of DDoS attack, Slowloris attacks do not require a large number of requests to be sent to crash the system (Anitha and Malliga, 2013).

2.6 Cloud Intrusion Detection Systems

Having introduced the different DDoS attack types and explained how they relate to the layers of the network model and its protocols, this section will discuss Intrusion Detection Systems (IDS), addressing the main configurations, approaches, techniques and deployment models that are used. It is important to understand how traditional IDS can deal with DDoS attacks in the cloud-computing environment. Additionally, this section will identify the weaknesses of these systems which make it difficult for them to effectively detect DDoS attacks – these weaknesses frame the 'research gap' that this thesis aims to address.

Intrusion covers attempts that aim to compromise the security of different components of a network – in the case of this research the cloud – such as VMs, routers, network bandwidth, and applications. For example, a common intrusion type in the cloud environment is DDoS attacks, as the taxonomy presented in detail in section 2.4.2 has explained (Modi, et al., 2013).

In an attempt to address such attacks, IDS have been implemented in cloud environments to provide well-defined security mechanisms to monitor and analyse the behaviour of users and network activities with the aim of detecting possible intrusions. Where a potential intrusion is identified, the IDS automatically alerts the administrator so that they make take further action (Girma, et al., 2015).

The two main components of an IDS are data collection and data analysis. Data collection is the process of capturing the audit data, such as network traffic packets, data from a network interface, and system calls (requests from the user for action from the operating system). Data analysis is the process of analyzing the audit data using well-known approaches and techniques, such as signature-based intrusion detection, anomaly-based intrusion detection, statistical techniques, data mining techniques and machine learning techniques. Figure 2.10 illustrates the associated approaches, techniques and deployment models used in IDS (Chen, 2014). Each of the elements will be discussed in the remainder of this section.

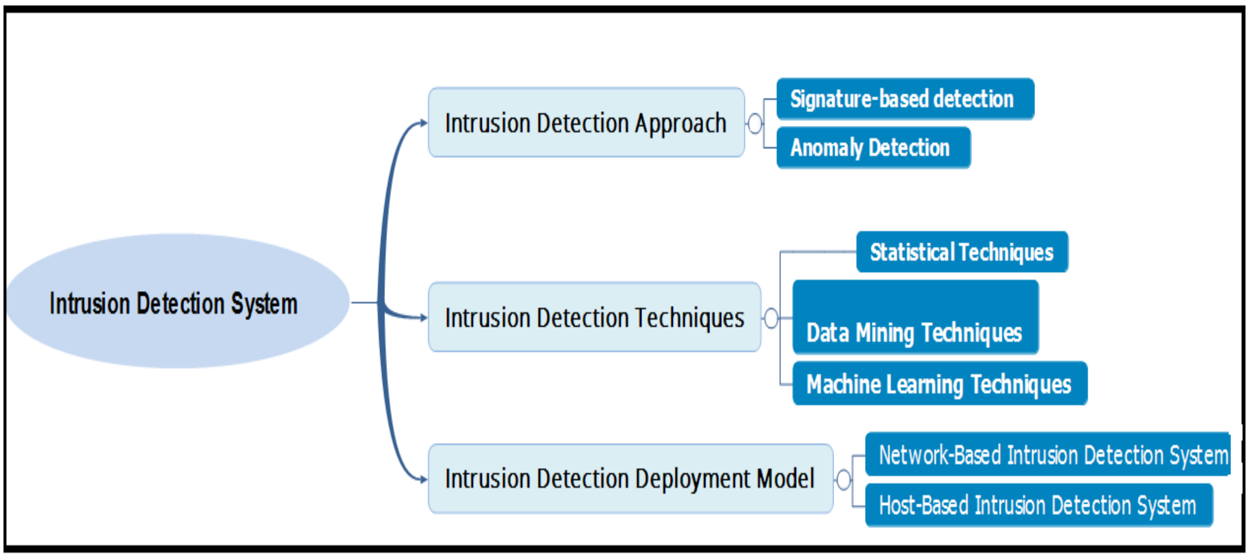

Intrusion Detection System
Intrusion Detection Approach
Signature-based detection
Anomaly Detection
Intrusion Detection Techniques
Statistical Techniques
Data Mining Techniques
Machine Learning Techniques
Intrusion Detection Deployment Model
Network-Based Intrusion Detection System
Host-Based Intrusion Detection System

Figure 2.10 Intrusion detection approaches, techniques and models (Defence, 2015)

There are two main detection approaches for IDS: signature-based intrusion detection; and anomaly-based intrusion detection. The two approaches can be implemented together or separately. Signature-based intrusion detection is also called misuse detection, and its focus is on known attacks. It attempts to build a model with characteristics of the attacks (each characteristic is called a signature or rule). For example, the different types of DDoS attack may have different signatures (the detail of these signatures will be explained in Chapter 3). The signature-based intrusion detection system observes the behaviour of the system or network traffic and compares it with a pre-defined attack pattern model and generates an alarm (comprising information regarding the attack type, and the victim of the system) if the behaviour of the system or network match with existing attack signatures (Meng, Li and Kwok, 2014). As a result, known attacks can be detected through signature-based detection. However, the detection system may not be effective against unknown attacks (Bhuyan, et al., 2014).

In anomaly-based intrusion detection systems, the system attempts to model the normal behaviour of the system. This model includes features that represent user activities and network connections over a time period. For example, returning to Figure 2.3, when a legitimate user or a malicious user (the sender computer) sends a request to the cloud server (the destination computer), this request travels from the top layer of TCP/IP network layer under a set of protocols (as explained in detail in section 2.5) to reach to the cloud server. In the data collection and data analysis phase of an IDS, this traffic can be collected and analyzed using various techniques and approaches (which will be explained in later paragraphs of this section). Afterwards, the detection system compares the observed behaviour of the system with the stored model. Any deviation from the normal behaviour model is considered to be suspicious behaviour, leading the system to raise an alarm with the administrator. As such, anomaly-based IDS would seem to have an advantage over signature-based IDS in detecting unknown attacks. However, they might suffer from false alarm issues, as not all deviations from the normal behaviour model will be attacks (Chen, Chen and Lin, 2010).

As seen in Figure 2.11, there are three main techniques that are used for IDSs, according to analysis of literature in the field (Shelke, Sontakke and Gawande, 2012). The first, which uses statistical techniques, has the capability to model the behaviour of the system, including legitimate and malicious behaviours. The models tend to model a number of related features of either attack or normal behaviour over a specific period of the time. For example, the system can create a profile or a model with the activities of the user during the login session – which may include features such as login time, logout time, the number of TCP connections, the amount and duration of computing resources consumed during the session – captured as statistical distributions. The detection system then creates two profiles: one saves the average value of these features and the second detects an attack if the values exceed standard threshold values (Thatte, Mitra and Heidemann, 2011).

Data mining represents the second type of technique used, and is the process of discovering meaningful patterns in a large dataset using classification, clustering and association rule tools to find useful patterns to help detect intrusion/attacks. The data can be mined from cloud system logs or cloud network connection traffic logs, such as TCP dump data (these log files will be explained in detail in Chapter 3). Moreover, data mining techniques can be applied to model the normal behaviour of the cloud network connection traffic for the system to allow the differentiation of attack traffic from normal traffic (Elshoush and Osman, 2011).

Some machine learning techniques have a capability for learning and enhancing their functionality over the time, and are the third type of technique used. For example, a machine learning focus may be used to build a model of either user behaviour or DDoS attack behaviour that can optimize its functionality in a loop cycle and can adjust its implementation scheme based on feedback information (Patel, Taghavi, Bakhtiyari and Celestino, 2013). The machine learning technique is similar to the data mining technique. However, machine learning uses the data to predict the attack pattern, whereas data mining uses the data to find the attack patterns which may be useful for intrusion detection.

As Figure 2.10 demonstrates, host-based intrusion detection systems and network-based intrusion detection systems are two forms of IDS deployment model that can be implemented based on the defined defensive scope., and they can be configured to be used together or separately in an IDS.

A Host-based Intrusion Detection System (HIDS) is defined as protecting the host from intrusion. In the cloud environment, such systems can be installed on each VM or cloud server that hosts cloud services and resources. HIDS can either use anomaly-based or signature-based detection. HIDS monitors the traffic within VMs and from VMs and analyses the information, which is collected from system calls, detecting possible intrusion through any modification to them (Lo, Huang, and Ku, 2010).

Network-based Intrusion Detection Systems (NIDS) focus on protecting the local network from intrusion. The NIDS detection approach can be also signature-based or anomaly-based. NIDS monitor the traffic within the connected hosts to detect intrusion such as a DDoS attack and protect the cloud services and resources from any possible intrusion (Osanaiye and Dlodlo, 2015).

According to the literature, there are four main metrics that are used to evaluate the performance of IDS techniques: True Positive (TP); True Negative (TN); False Positive (FP); and False Negative (FN) (Om, 2012). A True Positive (TP) is where an anomaly is correctly classified as an intrusion. A True Negative (TN) is where an anomaly is correctly classified as not being intrusion. A False Positive (FP) is where an intrusion is falsely classified as being an intrusion. A False Negative (FN) is where an intrusion is falsely classified as normal behaviour.

2.7 Research Gap

There is a lot of research work in the area of IDS that focuses on improving the accuracy of the intrusion detection system to effectively detect intrusions such as DDoS attacks. For example, Mondal et al. (2017) presented a fuzzy-based mechanism to detect anomalous behaviour of the attacker in the cloud, while Zekri et al. (2017) proposed a DDoS attack detection approach using a decision tree machine learning algorithm. Bhushan and Gupta (2018) presented a statistical approach to detect low rate DDoS attacks in the cloud and Madhupriya, Shalinie and Rajeshwari (2018) proposed a Dimension-Reasoning Local Outlier Factors (DR-LOF) approach to identify the source of DDoS attacks in the cloud. Liu et al. (2018) verified the security policy of cloud computing to detect DDoS attacks by implementing hidden Markov model and neural network algorithms, whileSultana et al. (2019) proposed a Hop Count Filtering (HCF) mechanism to prevent and detect IP spoofing DDoS attacks in the cloud. Gupta and Chaturvedi (2019) introduced a method to detect DDoS attacks using the features of Software Defined Networking (SDN) in the cloud; and recently Velliangiri, Karthikeyan and Vinoth Kumar (2020) developed the Tylor-Elephant Herd Optimisation based Deep Belief Network (TEHO-DBN) algorithm to detect DDoS attacks in the cloud, while Hezavehi and Rahmani (2020) proposed an anomaly-based DDoS attack detection framework to detect attacks using a third-party auditor (TPA).

As mentioned in section 2.6, and reinforced in the previous paragraph, a number of techniques and approaches are used in IDS, most notably signature-based and anomoly-based. However, according to the literature, neither approach is able to effectively detect all types of DDoS attack, leading to false alarms and poor detection rate accuracy (Meng et al., 2014; Dahiya and Gupta, 2020).

For example, although signature-based detection has the advantage of detecting DDoS attacks through matching signatures, it might generate large numbers of true positives as a result of dropping network packets during the monitoring of large volumes of incoming network traffic, affecting its accuracy rate as a result (Luo and Xia, 2014). Moreover, as DDoS attacks are becoming more sophisticated, signature-based techniques have become unable to identify these attacks effectively; because no prior signature of the new attacks exists in the database of the detection system (Andrabi, Pandey and Nazir, 2020). Also, to constantly update the database with the new signatures of the attacks, requires significant effort (Premalatha, 2019).

Additionally, although anomaly-based IDS approaches can overcome the noted limitation of signature-based approaches and mostly detect any types of intrusion, it might be challenging to find relevant features to create a precise profile for normal behaviour of the system or network in the cloud (Mehmood et al., 2013;Singh et al., 2019). For instance, this approach might result in generating a high number of false-positives where the anomalies are a normal new behaviour of the system rather than an intrusion (Khraisat et al., 2019).

Despite the strengths and weaknesses of existing detection systems, the dynamic nature of cloud computing and its specific characteristics (as discussed in section 2.1), and the evolving nature of attack tools, can make detection intrusion of DDoS attacks using traditional IDS systems difficult and can result in them producing false alarms and having poor detection accuracy (Sari, 2015). For example, the novel architecture of the cloud computing platform and its different network structure lead to traditional anomaly-based detection systems becoming less effective in the cloud as the normal pattern of activities of users are different to those seen on conventional network infrastructures (Virupakshar et al., 2020).

In addition, the lack of cloud intrusion datasets hinders the evaluation of intrusion detection systems for the researchers in this area. For instance, to accurately develop anomaly-based detection models, itis required to train the model with the recent behaviuor of attacks. A lack of suitable training data sets is therefore a serious issue. Moreover, the computational cost and the training time required for anomaly-based approaches are both very high and abundant memory is required for the associated data analysis (Hajiheidari et al., 2019).

The range of issues identified highlight the importance of developing accurate models that may be subsequently used as part of IDS to identify DDoS attacks in the cloud. Failure to detect accurately intrusions results in degradation of security services such as data confidentially, integrity and availability in cloud computing environments.

In summary, then, the analysis presented in this chapter has argued the importance of understanding DDoS attacks and being able to develop models that accurately support their identification in cloud-based environments in order to support effective intrusion detection, and have highlighted the criticality of having suitable cloud-based datasets from which to develop, and then on which to test, the models.

2.8 Summary

This chapter has provided information about the novelty of cloud computing, its fundamental characteristics and its architecture that have resulted in it gaining significant popularity in different sectors. It has demonstrated that there has been significant growth in adoption as a result of its characteristics, leading many companies to move their data and operational services to the cloud. At the same time, the chapter has identified several issues that have arisen from the adoption of cloud computing, and has argued that the most important issues relate to security. Based on a review of the identified security issues, the chapter argued the importance of DDoS attacks in this context and of how critical is to mitigate these types of attack. The different types of DDoS attack were then explored, and explained in relation to the TCP/IP network model and its protocols. The chapter concluded by discussing existing approaches to intrusion detection and why it can be difficult to accurately detect DDoS attacks, in preparation for reporting (in Chapter 3) the initial practical work that sought to develop a model to improve DDoS attack detection.

Chapter 3 : Developing an Initial Detection Model

3.1 Introduction

As argued when presenting the research gap in section 2.7, applying existing non-cloud IDS to cloud-based environments may lead to poor detection accuracy and increased false alarms, in the form of false positives or true negatives. Given the focus of this research, it is important to address these issues and understand whether there is any difference between DDoS attack types in cloud and noun-cloud environments that may cause these issues.

In seeking to undertake research in this area, a critical problem is the lack of public cloud datasets, which hinders the development of detection models for attacks in the cloud environment. This chapter will therefore explore an approach to address the identified issues that uses non-cloud-based and cloud-based datasets to seek to address the accuracy and false alarm issues, reporting the process of the development of a detection model. The approach followed will be introduced in section 3.2, with subsequent sections (sections 3.3-3.5) reporting on each phase in detail.

As such, section 3.3 will present the process of selecting the datasets and features that formed the input to the model. The NSL-KDD intrusion detection dataset and Cloud Intrusion Detection Dataset (CIDD), which include what are seen as important features of DDoS attacks in cloud and non-cloud environments respectively, were used to develop the model. A small set of similar features that were present in both datasets was selected as the initial input to the model development activity. This first phase of the practical work was critical, using the overlapping features from the two datasets to construct the base of the model. This then enabled us to determine how to proceed with the next stage of the research as a result of understanding the correspondence between attack types in the two datasets, developed through the activity reported in sections 3.4 and 3.5, and the issues raised as a result of the associated analysis. It is, though, important to note that the two datasets have different structures: the NSL-KDD dataset contains packet-based captured traffic whereas the cloud-based dataset contains network traffic that is aggregated into time intervals. As will be seen, this causes issues which frame later parts of the research (see Chapter 4)

Section 3.4 will then present the process of preparing the datasets to be free of any noise, so that they were ready for the processing task, and will explain the pre-processing that was undertaken on the cloud data to 'unpack' the time-based data into a form that was closer in structure to (but still not the same as) the non-cloud dataset. In addition, this section will describe each stage of developing the model, reporting the use of machine learning techniques to classify the cloud-based attacks, based on learning/training with the non-cloud-based attack dataset (NSL-KDD). This involved training the model first with NSL-KDD dataset to ensure that it was a good fit to the cloud data by looking at its performance using a confusion matrix to assess the correspondence between the actual and predicted DDoS attack types.

Section 3.5 will then report the transfer learning approach that was used, the result of testing the model with the cloud intrusion detection dataset to see whether cloud-based attacks can be predicted based on the training on non-cloud data. To achieve this, it will present the analysis of the results of this first piece of empirical work using sensitivity analysis. Here, though, the confusion matrix will be used to analyse the performance of the proposed model by understanding how the model (which uses non-cloud-based attack types) classifies the types of cloud-based DDoS attacks contained in the cloud-based dataset. Examining the resulting classifications may enable us to understand the correspondence between the attack types in the two datasets. High levels of correspondence between attack types identified by the model and the known cloud-based attack types (contained within the dataset) will be analyzed to understand the relationships between factors that are important in the non-cloud and cloud attacks.

3.2 Approach Taken to Designing the DDoS Attack Detection Model in a Cloud-computing Environment

Figure 3.1 presents a schematic showing each phase of the experimental work aimed at developing an initial DDoS IDS model in a cloud-computing environment. The three phases of the work will be described in detail in sections 3.3-3.5.

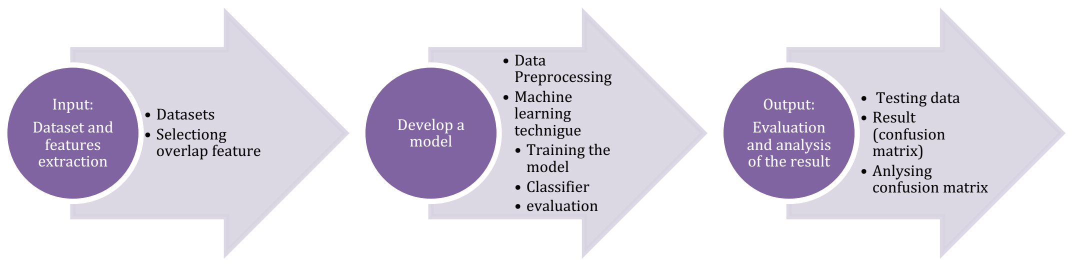

Input:
Dataset and features extraction
• Datasets
• Selectiong overlap feature
Develop a model
• Data Preprocessing
• Machine learning technigue
• Training the model
• Classifier
• evaluation
Output:
Evaluation and analysis of the result
• Testing data
• Result (confusion matrix)
• Anlysing confusion matrix

Figure 3.1 The phases of the experimental process to develop an initial DDoS IDS model in a cloud-computing environment

3.3 Datasets and Feature Extraction

In this first piece of practical work, we wanted to understand whether there was any difference between DDoS attacks in the cloud environment and non-cloud environment that would lead traditional IDS systems to identify false alarms and have poor detection accuracy in the cloud environment. Hence, we decided to develop a model using a traditional network intrusion detection dataset and then see how well this model predicted intrusion (in the form of different types of DDoS attacks) in the cloud environment. To identify an appropriate non-cloud dataset, we investigated almost all of the available network intrusion detection datasets – for reasons of privacy, some network intrusion detection datasets are not easily available. Section 3.3.1 will introduce the datasets and explain and justify the dataset that was chosen to be used in this research. The Cloud Intrusion Detection Dataset (CIDD), used in the third phase of the experiment (see section 3.5) will then be introduced (section 3.3.2) before being compared (in section 3.3.3) to the chosen non-cloud dataset to ensure that there is suitable correspondence between the training (non-cloud) and testing (cloud) dataset considered in this piece of the research.

3.3.1 Current non-cloud network intrusion detection datasets

Table 3.1 presents summary information on the currently available intrusion detection datasets and explains the type of data included within each dataset, states who created each dataset, provides a reference to the relevant published work, and sets out any limitations with the dataset with respect to data type and attack type. Among them, the National Security Laboratory – Knowledge Discovery Dataset (NSL-KDD), provided by the Canadian Institute of Cyber Security of University of New Brunswick (UNB), was determined to be the most appropriate choice for this phase of the research because it contains different types of DDoS attack. More importantly, despite being published in 2009, it remains an effective dataset in the research community for building and/or comparing detection models, with more than 50% of published research work in this area using the dataset for intrusion detection evaluation. Each dataset will be briefly discussed, ending with the NSL-KDD dataset, to further justify the choice that was made.

CAIDA Dataset: The Centre for Applied Internet Data Analysis (CAIDA) dataset considered in this research was developed and published by the Information Marketplace for Policy and Analysis of Cyber-risk & Trust (IMPCAT) repository, which is supported by the US Department of Homeland Security's Science and Technology Directorate. CAIDA provides relevant network security datasets and tools for cyber security researchers, systems evaluator and developers. It also offers a captured network trace, but it does not provide label attack types in the datasets which made it unsuitable for our purposes since attack types had to be identified in order to test the accuracy of the developed model.

UNSW-NB15 Dataset: The University of New South Wales Network-Based 2015 (UNSW-NB15) dataset, generated by the Australian Cyber Security Center, includes network data of DoS attacks and another eight types of network attack. However, as with the CAIDA dataset, the type of DoS attack is not specified in the UNSW-NB15 dataset, making it unsuitable for our purposes.

KYTO, DEFCON, ADFA and LBNL Datasets: The Kyto University dataset, DEFCON dataset, Australian Defense Force Academy (ADFA) dataset and Lawrence Berkeley National Laboratory (LBNL) dataset were generated as datasets for intrusion detection, but the types of intrusion that they cover are not related to DDoS attacks, making them unsuitable for our purposes.

Name of dataset	Data Type	Dataset Creator	Reference	Limitations in relation to data type and attack type
CAIDA dataset	Network traffic	Centre for Applied Internet Data Analysis, University of California, San Diego, USA	Elshoush and Osman, 2011	Unlabelled data
UNSW-NB15	Network traffic	Australian Cyber Security Centre	Moustafa, 2015	Specific types of DoS attack have not been identified
DEFCON	Network traffic	Hacker competition called Capture the Flag (CTF)	Bhuyan, Bhattacharyya and Kalita, 2015	It contains only intrusive traffic without any normal traffic
KYOTO	Network traffic	Honeypot system in Kyoto university	(Song, Takakura and Okabe, 2012)	Specific types of DoS attack have not been identified
ADFA	System call traces	Australian Defence Force Academy	(Datasets, Borisaniya and Patel, 2015)	It related to Host based Intrusion detection system
LBNL	Packet trace	Lawrence Berkeley National Laboratory (LBNL), USA	(Pang *et al.*, 2005)	It does not include malicious traffic
NSL-KDD	Network traffic	Canadian Institute of cyber security of University of New Brunswick	Tavallaee *et al.*, 2009	None

Table 3.1 Contemporary network intrusion detection datasets

The NSL-KDD dataset contains instances of network traffic that contain 41 features and one class attribute, which indicates the type of the network connection traffic (either the attack type or that the data instance represents normal traffic). As such, it is a packet-based dataset. There are four different groups of attack in the dataset: Probe attacks; User to Root attacks (U2R); Remote to Local attacks (R2L); and Denial of Service attacks (DoS). As the focus of this research is on DoS attacks, only the group of records where DoS attacks were signified were selected. The DoS group comprises a number of attack types (as explained in detail in section 2.5): Back; Land; Neptune; Ping of Death; Smurf; and Teardrop.

In the dataset, the 41 features, or attributes of network traffic connection, are classified into three groups of features. The first is the group of basic features related to the attributes of the observed network packet headers (the packet structure is based on the relevant protocol of the network model, as described in section 2.5) (Kayacik, Zincir-Heywood and Heywood, 2005). The second is the group of the features associated with the content of the packet. Finally, the third is the group of network traffic features – statistical information representing all connections to the same destination machine within a two-second timeframe. This experiment focused on the basic feature group since this is where information relevant to DDoS attacks might be identified.

Having introduced the traditional/non-cloud dataset that was selected for the training activity, section 3.3.2 will introduce the cloud intrusion dataset that was used for the subsequent testing phase.

3.3.2 The cloud intrusion detection dataset

Only two public cloud datasets were identified. The first contained data related to masquerade attacks, DoS attacks, U2R attacks and R2L attacks, but the dataset did not contain network packet features. Instead, the dataset included audit parameters that were not compatible with the NSL-KDD Dataset selected for the training activity.

The second dataset contained the same attack types as the NSL-KDD dataset and had been developed to focus on the important features in detecting DDoS attacks. Therefore, it was decided to select this dataset for the testing phase. More detail on the dataset will now be provided.

The CIDD dataset was generated in 2016 from testbed experimental work on cloud DDoS attack detection. The simulated cloud environment that generated the data consisted of six nodes each of which included six virtual machines (VMs). In the scenario of this research that generated the dataset, each VM was allocated a specific role. One VM hosted the cloud server and this was considered the target machine for the attack(s). Some of the VMs were equipped with DDoS attack tools for launching different types of DDoS attack. The rest of the VMs were designed to have a normal cloud user role, such as requesting webpages from the cloud server.

A connection was initiated by sending a request from a user (the user of a VM that might be an attacker or a normal user) to the servers. The request referred to different types of service, such as downloading files, etc. These connections were sequences of TCP packets produced under well-defined protocols, such as the TCP protocol (the protocol for routing application packets to the correct application on the destination computer – see section 2.5). The generated traffic was recorded by the sniffer, a network packet analyzer in the form of software or hardware used to capture the TCP/IP packets over a network. The subsequent outcome of captured traffic included 5274 instances of network TCP/IP connections. Of these instances, 682 records were different types of DDoS attack, such as ICMP Flood, Ping-of-Death, UDP Flood, TCP SYN Flood, TCP LAND, DNS Flood and Slowloris.

A custom-built sniffer was used to extract the desired features from the IP packet header fields. Six distinct features were calculated per incoming IP address for each type of transport layer protocol (TCP, UDP, ICMP and other types), resulting in 24 features in total. Figure 3.2 presents these features and their descriptions. The extracted data were processed in five-second time intervals to create a time-based, rather than packet-based, dataset.

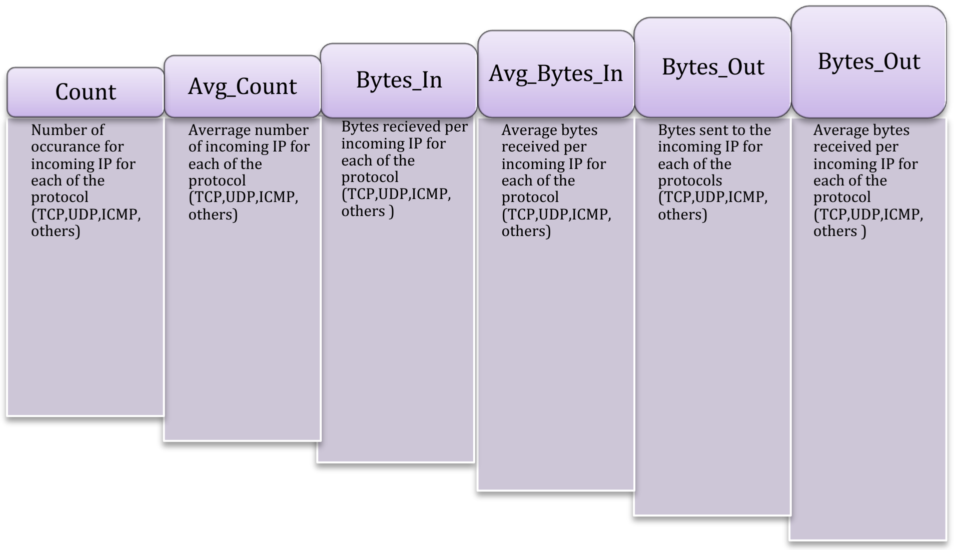

Count
Number of occurance for incoming IP for each of the protocol (TCP,UDP,ICMP, others)
Avg_Count
Averrage number of incoming IP for each of the protocol (TCP,UDP,ICMP, others)
Bytes_In
Bytes recieved per incoming IP for each of the protocol (TCP,UDP,ICMP, others)
Avg_Bytes_In
Average bytes received per incoming IP for each of the protocol (TCP,UDP,ICMP, others)
Bytes_Out
Bytes sent to the incoming IP for each of the protocols (TCP,UDP,ICMP, others)
Bytes_Out
Average bytes received per incoming IP for each of the protocol (TCP,UDP,ICMP, others)

Figure 3.2 Traffic features of the CIDD dataset

3.3.3 Overlapping features between the two datasets

As the previous discussion highlights, the two chosen datasets are structured in different ways: the selected group of features from the NSL-KDD dataset is packet-based whereas the entire CIDD dataset comprises time-based traffic, aggregated into records covering five-second intervals. This is one reason why the datasets have a restricted set of overlapping features, making using them together challenging (and potentially limiting the value of such an approach).

As already noted, owing to the structure of the proposed model (which will be explained in section 3.4), the training set and testing set had to have similar features. Therefore, to be able to train the model with the NSL-KDD dataset and then test it with CIDD dataset we selected features that were common between the two datasets, using them to form the initial, small feature set of the model.

As a result, the initial model was developed based on the three features from the NSL-KDD dataset that were also contained in the CIDD dataset. Figure 3.3 demonstrates the selected features from the NSL-KDD dataset, each of which will be briefly discussed.

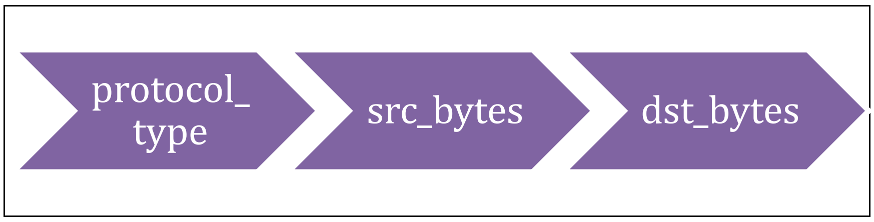


Figure 3.3 The selected features of NSL-KDD dataset

Protocol type identifies the type of connection to which the network traffic belongs. I t also refers to the associated type of protocol of each layer of the network model, as explained in section 2.5, such as UDP, TCP and ICMP, etc. The src_bytes feature of the dataset contains the number of bytes transmitted from the sender computer (source) to the destination computer. The dst_bytes feature contains the number of bytes sent from the destination computer to the source computer.

In the CIDD dataset, the Bytes_In and Bytes_Out features for TCP, UDP and ICMP protocols (therefore also giving the protocol type feature) were selected to subsequently test the model with the CIDD dataset. These features are similar to the three selected from the NSL-KDD dataset. (Bytes_In and Bytes_Out refer to the bytes received from the source to destination, and from destination to source, respectively. As such they correspond to the src_bytes and dst_bytes features). The pre-processing that was undertaken to 'unpack' the time-based data into a form that was closer in structure to (but still not the same as) the non-cloud dataset will be discussed in the next section.

3.4 Developing the Model

This section reports the techniques that were applied to the datasets to prepare them for the processing task (i.e., it reports the data pre-processing). It then describes the machine learning approach, algorithm and tools that were used to develop the detection model. Finally, it presents the initial results of training the model with the NSL-KDD dataset and evaluates the results based on two approaches/measures: cross-validation and confusion matrix.

3.4.1 Data pre-processing

Data pre-processing is the process of transforming the dataset to eliminate contradictions and noise from the data to prepare it for the processing task. As such, it is an important and widely-used part of the data analysis process. A data-cleaning technique was therefore used in this part of the work to make the structure of the two datasets suitable ahead of the model development activity.

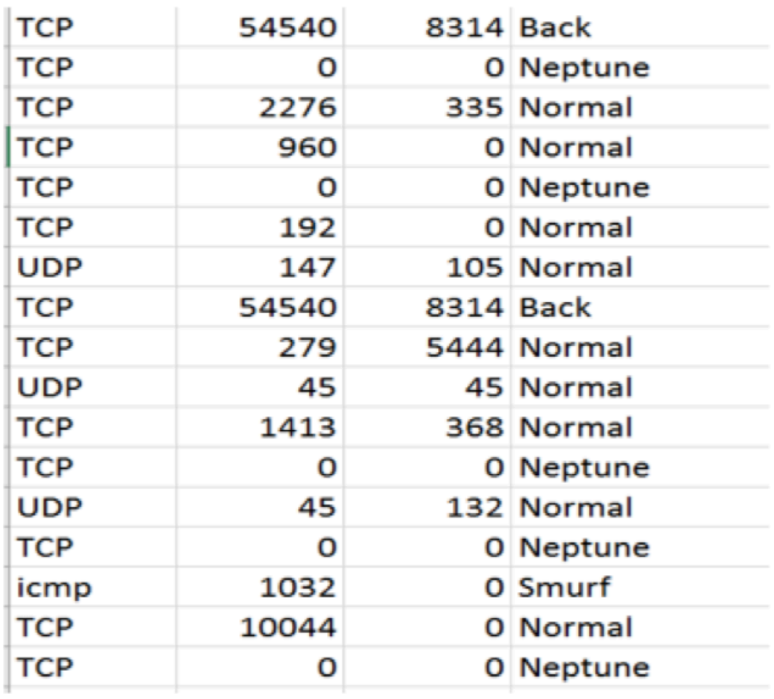
A discretization technique was then applied to the dataset to transform the different values contained in the initial structure of the dataset, to ensure that any continuous values were translated in suitable discrete values. This is important as it allows for optimisation of the processing time and provides a higher accuracy rate to be achieved in the prediction model. The Weka tool (Aljawarneh, Aldwairi and Yassein, 2018) was used to undertake this discretization activity, as follows.

The discretization filter was applied to the 'src_bytes' and 'dst_bytes' features of the two datasets by 'binning'; that is, the range of numerical values for each feature were divided into three bins (or intervals) of equal size. After applying the discretization filter to both datasets, the bins for each feature were named A, B and C. Table 3.5 presents the discretized range of the two originally-continuous features in the datasets.

Discretized bin of src_bytes	Name of each bin	Discretized bin of dst_bytes	Name of each bin
0.5-250.5	M	0.5-668.5	M
250.5-inf	H	668.5-inf	H
-inf-0.5	L	-inf-0.5	L

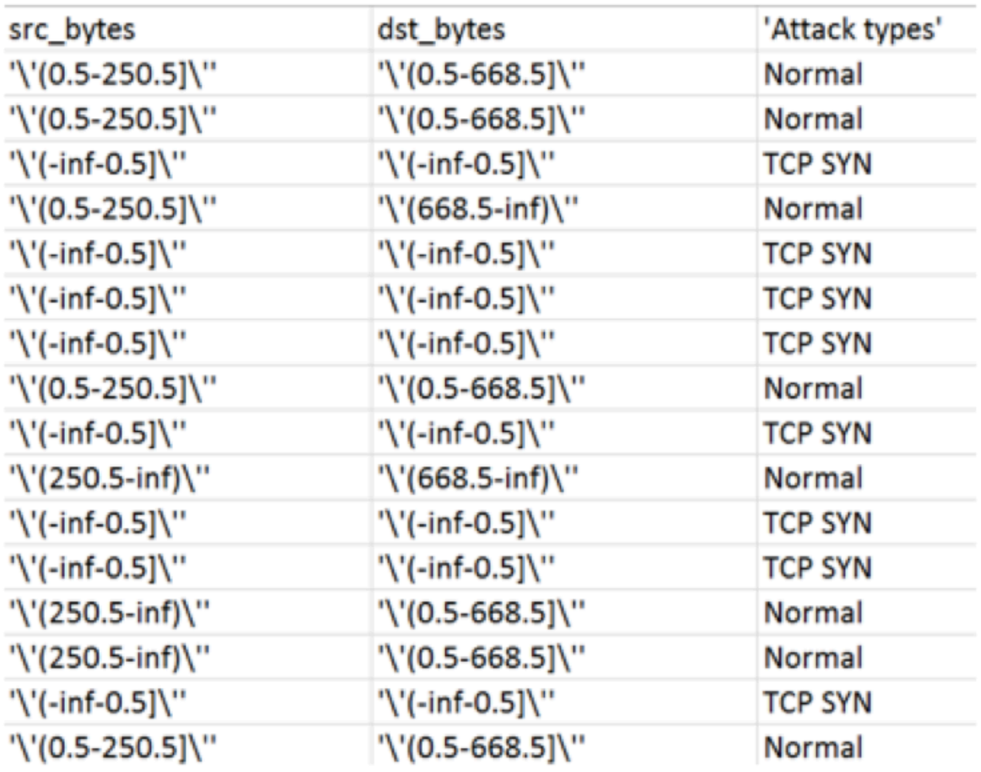
Table 3.2 Name of each bin in discretized dataset

Figures 3.4 and 3.5 show the initial and final structure, respectively, of an extract of the NSL-KDD dataset to illustrate the impact of the discretization activity on it.



TCP	54540	8314	Back
TCP	0	0	Neptune
TCP	2276	335	Normal
TCP	960	0	Normal
TCP	0	0	Neptune
TCP	192	0	Normal
UDP	147	105	Normal
TCP	54540	8314	Back
TCP	279	5444	Normal
UDP	45	45	Normal
TCP	1413	368	Normal
TCP	0	0	Neptune
UDP	45	132	Normal
TCP	0	0	Neptune
icmp	1032	0	Smurf
TCP	10044	0	Normal
TCP	0	0	Neptune

Figure 3.4 The initial structure of an extract of the NSL-KDD dataset

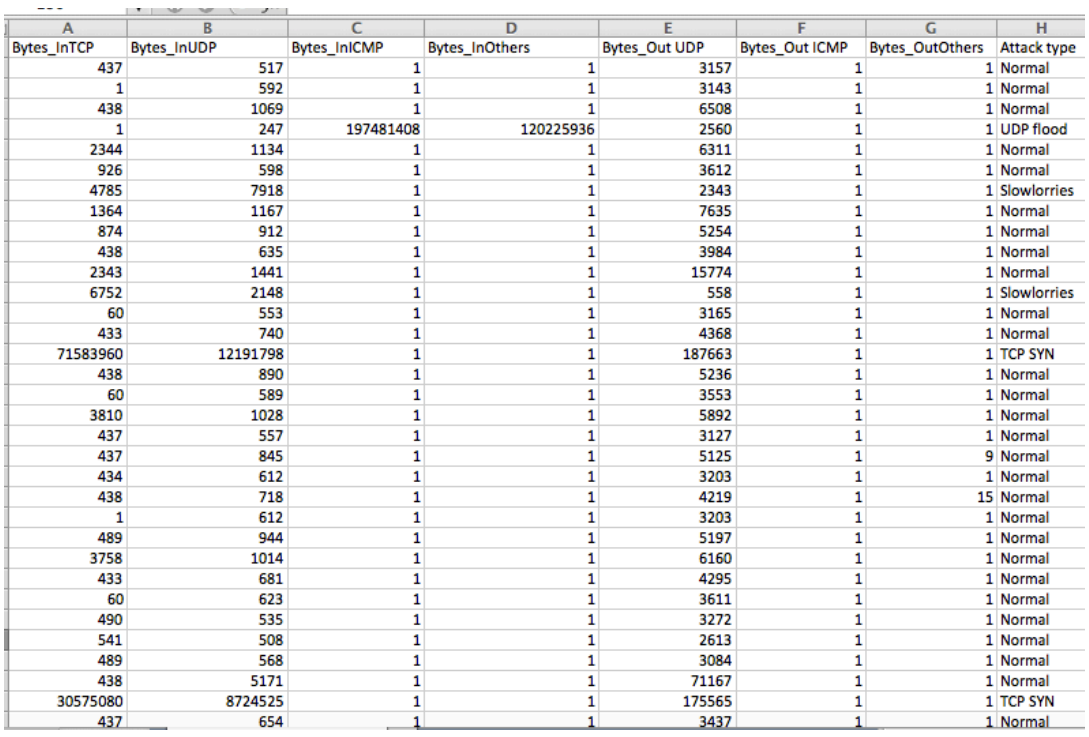


src_bytes	dst_bytes	'Attack types'
'\'(0.5-250.5]\''	'\'(0.5-668.5]\''	Normal
'\'(0.5-250.5]\''	'\'(0.5-668.5]\''	Normal
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(0.5-250.5]\''	'\'(668.5-inf)\''	Normal
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(0.5-250.5]\''	'\'(0.5-668.5]\''	Normal
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(250.5-inf)\''	'\'(668.5-inf)\''	Normal
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(250.5-inf)\''	'\'(0.5-668.5]\''	Normal
'\'(250.5-inf)\''	'\'(0.5-668.5]\''	Normal
'\'(-inf-0.5]\''	'\'(-inf-0.5]\''	TCP SYN
'\'(0.5-250.5]\''	'\'(0.5-668.5]\''	Normal

Figure 3.5 The post-discretization structure of an extract of the NSL-KDD dataset

Next, resampling was undertaken in Weka to address imbalances in the distribution of the data across the eight classes in the NSL-KDD (training) dataset, which could have caused lower accuracy in the subsequently-developed model's prediction of classes. This resulted in over-sampling the minority classes and under-sampling the majority classes to create a more balanced distribution in the dataset.

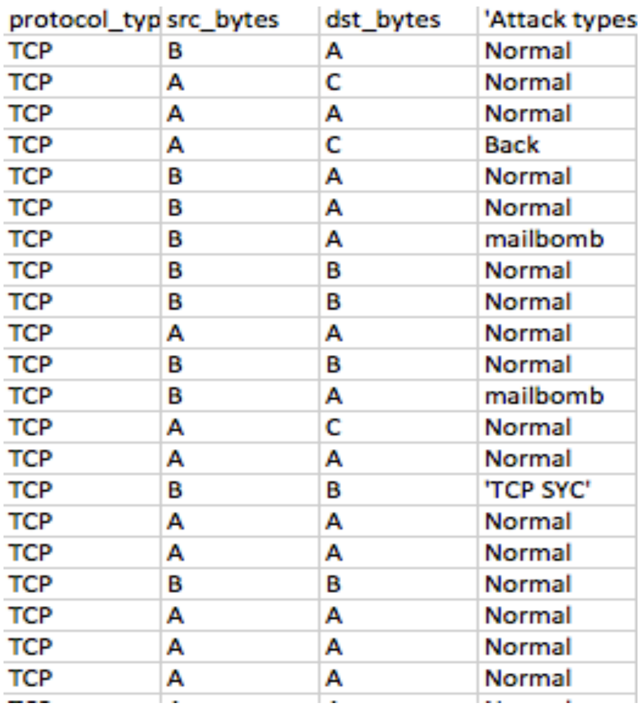
Finally, a process was applied to 'unpack' the time from CIDD dataset. Figure 3.6 presents an extract of the CIDD dataset to show its initial structure. As well as the attack type or normal traffic identifier, each row/instance of the CIDD dataset includes five seconds of aggregated packet data organised under four types of protocol: TCP; ICMP; UDP; and others. Data related to the 'other' protocol category were not used because the types of protocol were not specified.



A	B	C	D	E	F	G	H
Bytes_InTCP	Bytes_InUDP	Bytes_InICMP	Bytes_InOthers	Bytes_Out UDP	Bytes_Out ICMP	Bytes_OutOthers	Attack type
437	517	1	1	3157	1	1	Normal
1	592	1	1	3143	1	1	Normal
438	1069	1	1	6508	1	1	Normal
1	247	197481408	120225936	2560	1	1	UDP flood
2344	1134	1	1	6311	1	1	Normal
926	598	1	1	3612	1	1	Normal
4785	7918	1	1	2343	1	1	Slowlorries
1364	1167	1	1	7635	1	1	Normal
874	912	1	1	5254	1	1	Normal
438	635	1	1	3984	1	1	Normal
2343	1441	1	1	15774	1	1	Normal
6752	2148	1	1	558	1	1	Slowlorries
60	553	1	1	3165	1	1	Normal
433	740	1	1	4368	1	1	Normal
71583960	12191798	1	1	187663	1	1	TCP SYN
438	890	1	1	5236	1	1	Normal
60	589	1	1	3553	1	1	Normal
3810	1028	1	1	5892	1	1	Normal
437	557	1	1	3127	1	1	Normal
437	845	1	1	5125	1	9	Normal
434	612	1	1	3203	1	1	Normal
438	718	1	1	4219	1	15	Normal
1	612	1	1	3203	1	1	Normal
489	944	1	1	5197	1	1	Normal
3758	1014	1	1	6160	1	1	Normal
433	681	1	1	4295	1	1	Normal
60	623	1	1	3611	1	1	Normal
490	535	1	1	3272	1	1	Normal
541	508	1	1	2613	1	1	Normal
489	568	1	1	3084	1	1	Normal
438	5171	1	1	71167	1	1	Normal
30575080	8724525	1	1	175565	1	1	TCP SYN
437	654	1	1	3437	1	1	Normal

Figure 3.6 The initial structure of an extract of the CIDD dataset

To make the structure of the CIDD dataset closer to that of the NSL-KDD dataset, each record was divided to create three records, one for each protocol type considered (TCP; ICMP; and UDP). Figure 3.7 presents an extract that shows the 'unpacked' form of the CIDD dataset for a set of TCP records.



protocol_typ	src_bytes	dst_bytes	'Attack types
TCP	B	A	Normal
TCP	A	C	Normal
TCP	A	A	Normal
TCP	A	C	Back
TCP	B	A	Normal
TCP	B	A	Normal
TCP	B	A	mailbomb
TCP	B	B	Normal
TCP	B	B	Normal
TCP	A	A	Normal
TCP	B	B	Normal
TCP	B	A	mailbomb
TCP	A	C	Normal
TCP	A	A	Normal
TCP	B	B	'TCP SYC'
TCP	A	A	Normal
TCP	A	A	Normal
TCP	B	B	Normal
TCP	A	A	Normal
TCP	A	A	Normal
TCP	A	A	Normal

Figure 3.7 The unpacking the time structure of an extract of the CIDD dataset

As can be seen in Figure 3.7, the resulting dataset has four columns, which are similar to those extracted from the NSL-KDD dataset. At this stage, the CIDD pre-processed dataset was ready for use in testing the model once developed. The next sub-section reports the processes that were involved in developing the model.

3.4.2 Machine learning approach for developing the model

A Machine Learning (ML) approach was adopted to develop the detection model as they have been widely used in IDS work (Keegan *et al.*, 2016). Moreover, this method offers the ability to learn from experience (using training data) to predict from the unseen data (the test data). ML approaches mainly use two different types of learning: supervised or unsupervised. Supervised learning solves the classification problem by using one of many well-defined Artificial Intelligence algorithms. In a very basic example, assuming there is input X (training data) and output Y, an algorithm learns from input X to find the relation between X and Y. Unsupervised learning is used when there is input data X, but no output variables. The goal in unsupervised learning is to model the structure in the data to understand more about the data, seeking to discover interesting structures/relationships.

Since the datasets in this work (introduced in section 3.3) are classified and labelled, a supervised learning approach was followed.

3.4.3 Machine learning classification algorithm

The process of classification aims to use the existing data to predict the associated classes of the data. For example, each instance of the NSL-KDD dataset includes a few features that classified a record/line in the dataset as either one of a number of DDoS attack types (see Figure 3.6 for examples) or as normal traffic. The classification algorithm should be able to distinguish between normal network traffic and the different types of attack traffic. If able to do so, it would give a predictive model for non-cloud data which could then be used with the cloud dataset to provide information about the relationship between the two datasets, allowing us to see how/whether the non-cloud trained model can predict cloud-based attacks.

With the aim of developing the most accurate model possible given the constraints of the datasets, we applied different algorithms to the NSL-KDD dataset to assess how accurate each algorithm was in terms of prediction. In addition to the accuracy factor, speed was another important issue used to select the classification algorithm owing to the high dimensionality of both datasets. Based on analysis of relevant literature, the following classification algorithms were identified as being the most widely-used to solve the classification problem in IDS: decision tree; neural network; and Naïve Bayes (Ahmed, Naser Mahmood and Hu, 2016).

Table 3.7 reports the accuracy of the results of applying each of these three algorithms on the NSL-KDD dataset.

Classification Algorithm	Accuracy
Neural network Multi layers Perceptron	42%
Naïve Bayes	85%
Decision Tree J48	57%

Table 3.3 accuracy of the applied algorithms on the NSL-KDD dataset

As can be seen in Table 3.3, of these three algorithms, the Naïve Bayes classification algorithm was the most accurate. (It also performed marginally faster, but this was not a key issue in the model development phase). Therefore, it was decided to use this algorithm as the classifier. The algorithm will be described in the remainder of this section.

The Naïve Bayes classification is a supervised machine-learning algorithm and probabilistic classifier that has proved to be effective, fast and accurate for a range of real-world scenarios (Om, 2012). The Naïve Bayes model also has the advantage of being easy and simple to build. The Naïve Bayes classifier predicts the product classes (in this case the different types of DDoS attack) based on the independence hypothesis, which means that the presence of a specific feature in a class (in our case the protocol type, src_bytes and dst_bytes) is not dependent on the presence of the other features. Naïve Bayes uses Bayes theorem, which describes conditional probabilities – the likelihood of an event based on the relevant prior knowledge of that event. For example, in the context of our model, it is concerned with the likelihood of a specific attack occurring based on the values of the existing three features in the dataset. The following formula calculates the conditional probability of the event:

$P(H|E) = P(E|H) * P(H)/P(E)$

In this formula, the aim is to calculate the likelihood of given hypothesis (H). In our context, the hypothesis can be assumed to be the probability of predicting the different types of DDoS attack. The sign or evidence (E) can be presumed to be the available features in the dataset, such as protocol type, src_bytes and dst_bytes. For example, to calculate the probability of predicting the Smurf attack (P|E) among other attack types (such as PoD or UDP Flood) and normal network traffic, we compute the probability of each type of attack P(H) and also the probability of each feature P(E). The class with the highest score is the predicted class.

The Naïve Bayes approach has been introduced briefly to demonstrate the performance of this algorithm in classifying the non-cloud dataset. In this study, all calculations were automatically undertaken by a specific tool, which will be introduced in the next sub section.

3.4.4 Machine learning implementation

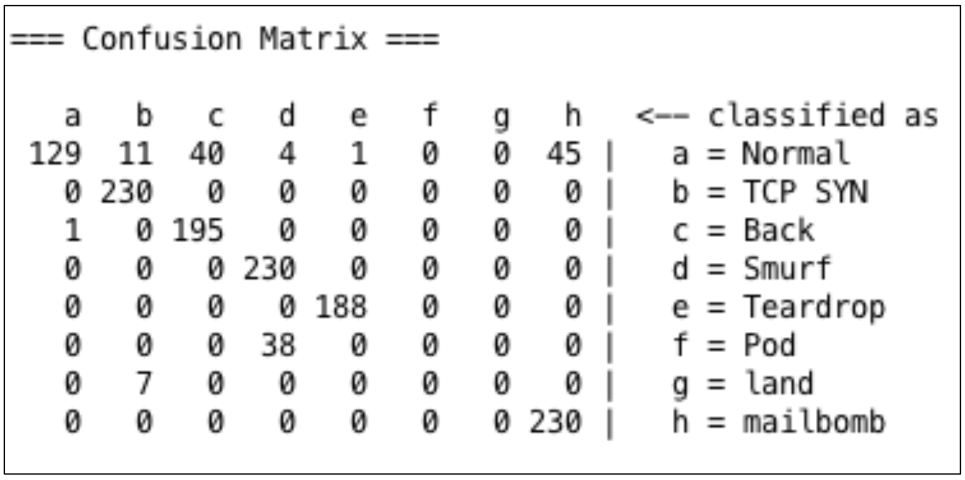
There are various machine learning tools/languages that are commonly used in data analysis, such as Python, R, MATLAB and the Waikato Environment for Knowledge Analysis (Weka). This study used Weka version 3.8 to implement the chosen classier and filter. Most of the available tools offer very similar functionality; Weka is, though, free and widely used, making it an appropriate choice for this work. Weka offers data mining and machine learning capabilities, including filtering, classification, clustering and ranking. Moreover, it has been broadly used for research and commercial applications (Frank, Hall and Witten, 2016) and it easy to use. For all of these reasons, it was an appropriate choice as the analysis environment for this study.

Having selected the Weka environment, the NSL-KDD dataset was used to train the model and then test it with the testing dataset (CIDD). First, we wanted to know how well the classifier predicted the different types of DDoS attack identified in the non-cloud dataset in order to understand the level of accuracy of the model. The evaluation of the performance of the classifier will be reported in the next sub-section.

3.4.5 Evaluation of the classifier

Cross validation and sensitivity analysis were used to evaluate the results of the classifier/resulting model. First, a 10-fold cross validation package was applied to the data. This software package divided the NSL-KDD dataset into 10 groups, of which nine groups were selected for training automatically by the Weka package and the remaining group was used to test the model.

A confusion matrix was then used to present a summary of the prediction outcomes of the classification model. The generated matrix shows the number of correctly and incorrectly predicted instances of each class, providing insights into the errors being made by the classification model. Figure 3.8 presents the confusion matrix resulting from the classification model when applied to the NSL-KDD dataset.



```
=== Confusion Matrix ===

   a   b   c   d   e   f   g   h   <-- classified as
 129  11  40   4   1   0   0  45 |   a = Normal
   0 230   0   0   0   0   0   0 |   b = TCP SYN
   1   0 195   0   0   0   0   0 |   c = Back
   0   0   0 230   0   0   0   0 |   d = Smurf
   0   0   0   0 188   0   0   0 |   e = Teardrop
   0   0   0  38   0   0   0   0 |   f = Pod
   0   7   0   0   0   0   0   0 |   g = land
   0   0   0   0   0   0   0 230 |   h = mailbomb
```

Figure 3.8 Confusion matrix showing the result of the classification applied to the NSL-KDD dataset

The instances on the main diagonal (top left to bottom right) indicate the correct prediction (i.e., where each attack type from the dataset is correctly predicted by the model). If these numbers are high for all rows/attack types, it demonstrates a good classification outcome.

In Figure 3.8, most of the attack types are classified correctly. Of these attack types, Land, PoD and some of the normal cases have been classified incorrectly. However, the general accuracy of the classifier is 89.103%, suggesting high overall accuracy with the noted exceptions.

The next section will report the results of the applying the model to the cloud dataset and explain the way in which the confusion matrix approach was used in this part of the study in relation to each cloud attack type.

3.5 Analysis of the Results

This section will report the results of applying the classification model that was trained on the non-cloud data (NSL-KDD) to the cloud data (CIDD). The predicted outcomes of the classification model will be then analyzed using a confusion matrix, but, in this instance, it will be used in a different/non-standard way. The standard confusion matrix describes the results of applying a developed classification model on a testing dataset that contains the same classifiers as the training set. For example, having trained our model with the non-cloud data, it would normally be tested with another non-cloud dataset containing the same features, with the aim of the classification model predicting the same type of attacks in this new dataset.

However, in this work, the types of the DDoS attack in the two datasets are not the same. The Naïve Bayes classification model was trained with, and tested on, the non-cloud data, which contained identifiers for normal traffic and the following seven attack types: TCP SYN; Back; Smurf; Teardrop; Ping of Death (PoD); Land; and Mailbomb (see description of the NSL-KDD dataset in section 3.3.1). The cloud data contained identifiers for normal traffic and the following seven attack types: TCP SYN; UDP Flood; ICMP Flood; DNS Flood; Ping of Death (PoD); Land; and Slowloris.

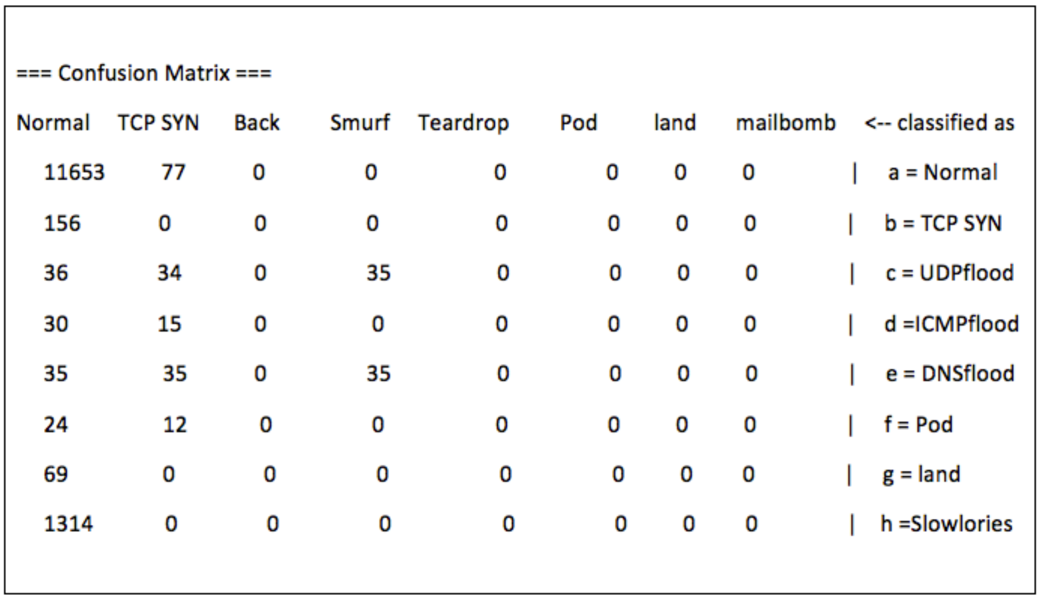
Testing the model on the cloud data leads to the creation of a confusion matrix that shows how the model has classified the normal and attack type instances in the cloud data in terms of normal traffic and the non-cloud attack types. In this way, the confusion matrix can be used to identify where there is a set of cloud data instances of one attack type (for example, DNS Flood) classified as a particular non-cloud attack type (for example, TCP SYN). The values of the features/classifiers in the model for these (in our example, DNS Flood) cloud data instances can then be compared with the feature data for all of the instances of the identified/classified non-cloud attack (in our example, TCP SYN) in the original test data (NSL-KDD). This may help us to understand the correspondence between cloud and non-cloud attack types and subsequently identify areas in which the model would have to be developed to more effectively identify and classify cloud attacks.

A threshold was set to focus the analysis arising from the confusion matrix, with only the cases where there were 30 or more instances of a cloud attack having been identified as a particular non-cloud attack being explored. This threshold was determined by simple observation of the values in the confusion matrix (see Figure 3.9). These cloud data instances were then considered by extracting and analyzing the distributions of the (discretized) values of the three core features (see section 3.3.3) and comparing them to the comparable features from the relevant non-cloud attack instances.

3.5.1 Transfer learning: applying the classification model to the CIDD dataset

Transfer learning is the application of a model that was trained for solving one particular task to another, different task (Dai et al., 2007). In contrast to the classical machine learning approach, this technique is more helpful where there is a shortage of data (or a limited dataset). Applying transfer learning could be useful in developing and training the model using the non-cloud dataset to facilitate the prediction of attacks in the cloud dataset. Making use of the technique in this research was, as far as we are aware, the first time that has been applied to the area of cloud-based intrusion detection.

Here, we applied the model trained on the non-cloud dataset to classify intrusions (of different types) in the cloud dataset. Figure 3.9 presents the confusion matrix arising from applying the model to the CIDD dataset.



```
=== Confusion Matrix ===

Normal  TCP SYN  Back   Smurf  Teardrop   Pod   land  mailbomb  <-- classified as
  11653    77     0       0        0       0     0     0       |  a = Normal
  156      0      0       0        0       0     0     0       |  b = TCP SYN
  36       34     0       35       0       0     0     0       |  c = UDPflood
  30       15     0       0        0       0     0     0       |  d =ICMPflood
  35       35     0       35       0       0     0     0       |  e = DNSflood
  24       12     0       0        0       0     0     0       |  f = Pod
  69       0      0       0        0       0     0     0       |  g = land
  1314     0      0       0        0       0     0     0       |  h =Slowlories
```

Figure 3.9 Result of classification based on testing with CIDD dataset based

As Figure 3.9 shows, in the first row, normal instances were classified by the model correctly 11653 times, but 77 instances were classified as TCP SYN attacks.

In the second row, all of the cloud TCP SYN instances were classified as normal traffic instances.

In the third row, the UDP flood attack instances were classified as normal instance 36 times, as TCP SYN attacks 34 times, and as Smurf attacks 35 times.

In the fourth row, the ICMP Flood attack instances were classified as normal instances 30 times.

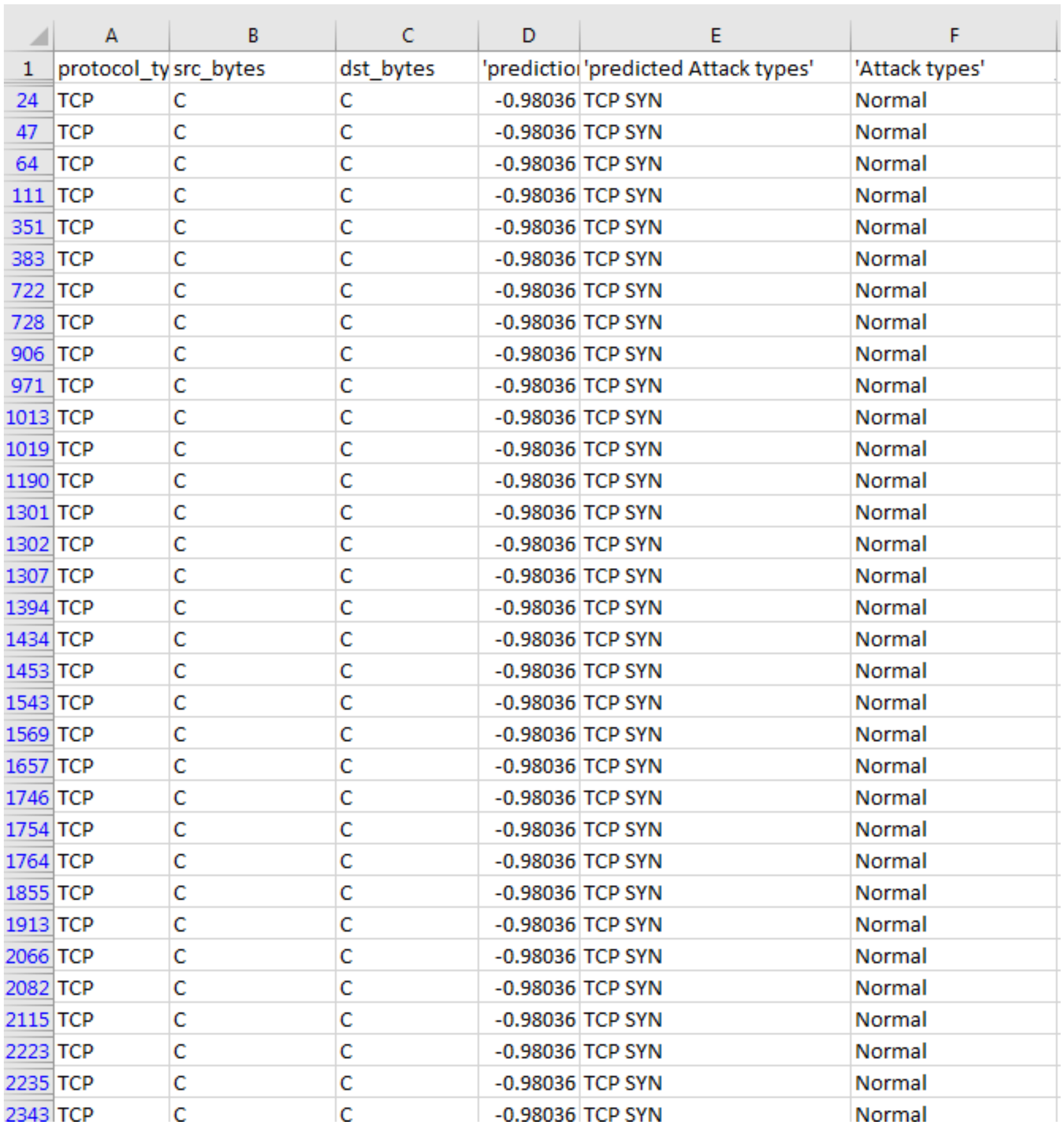
In the fifth row, 35 DNS Flood attack instances were classified as being normal traffic instances, 35 were classified as being TCP SYN attacks and 35 were classified as being Smurf attacks.

In the last two rows of the confusion matrix, all of the instances of Land attack and Slowloris attack were classified as normal traffic (69 times and 1314 times, respectively).

The next section will analyse the cases that have been presented above.

3.5.2 Analysis of the results of applying the classification model to the cloud dataset

To understand the classifications made by the model, the correspondences and the differences between cloud attack types and non-cloud attack types were explored by analyzing the underlying data of the cases identified in the confusion matrix that met the set threshold (see section 3.5.1). These 'above threshold' instances of normal traffic and then each attack type in the cloud dataset (the rows in Figures 3.9) that were classified by the model as specific non-cloud attack types, or as normal traffic, (the columns in Figure 3.9) were isolated and examined. For example, Figure 3.10 shows a screenshot of the 77 normal instances in the cloud data set that were classified by the model as a TCP SYN attacks. This extraction process was undertaken for each item in the confusion matrix that met the threshold. Each case, row by row, will now be discussed.



	A	B	C	D	E	F
1	protocol_ty	src_bytes	dst_bytes	'predictio	'predicted Attack types'	'Attack types'
24	TCP	C	C	-0.98036	TCP SYN	Normal
47	TCP	C	C	-0.98036	TCP SYN	Normal
64	TCP	C	C	-0.98036	TCP SYN	Normal
111	TCP	C	C	-0.98036	TCP SYN	Normal
351	TCP	C	C	-0.98036	TCP SYN	Normal
383	TCP	C	C	-0.98036	TCP SYN	Normal
722	TCP	C	C	-0.98036	TCP SYN	Normal
728	TCP	C	C	-0.98036	TCP SYN	Normal
906	TCP	C	C	-0.98036	TCP SYN	Normal
971	TCP	C	C	-0.98036	TCP SYN	Normal
1013	TCP	C	C	-0.98036	TCP SYN	Normal
1019	TCP	C	C	-0.98036	TCP SYN	Normal
1190	TCP	C	C	-0.98036	TCP SYN	Normal
1301	TCP	C	C	-0.98036	TCP SYN	Normal
1302	TCP	C	C	-0.98036	TCP SYN	Normal
1307	TCP	C	C	-0.98036	TCP SYN	Normal
1394	TCP	C	C	-0.98036	TCP SYN	Normal
1434	TCP	C	C	-0.98036	TCP SYN	Normal
1453	TCP	C	C	-0.98036	TCP SYN	Normal
1543	TCP	C	C	-0.98036	TCP SYN	Normal
1569	TCP	C	C	-0.98036	TCP SYN	Normal
1657	TCP	C	C	-0.98036	TCP SYN	Normal
1746	TCP	C	C	-0.98036	TCP SYN	Normal
1754	TCP	C	C	-0.98036	TCP SYN	Normal
1764	TCP	C	C	-0.98036	TCP SYN	Normal
1855	TCP	C	C	-0.98036	TCP SYN	Normal
1913	TCP	C	C	-0.98036	TCP SYN	Normal
2066	TCP	C	C	-0.98036	TCP SYN	Normal
2082	TCP	C	C	-0.98036	TCP SYN	Normal
2115	TCP	C	C	-0.98036	TCP SYN	Normal
2223	TCP	C	C	-0.98036	TCP SYN	Normal
2235	TCP	C	C	-0.98036	TCP SYN	Normal
2343	TCP	C	C	-0.98036	TCP SYN	Normal

Figure 3.10 Screenshot of the result of classification on the CIDD

To understand why normal cloud traffic instances may have been classified as (non-cloud) TCP SYN attacks, we looked at the distribution of the data for the three features/classifiers for the TCP SYN attack instances in the non-cloud (benchmark) data and for the 77 'misclassified' cloud data cases. The results are presented in Figure 3.11.

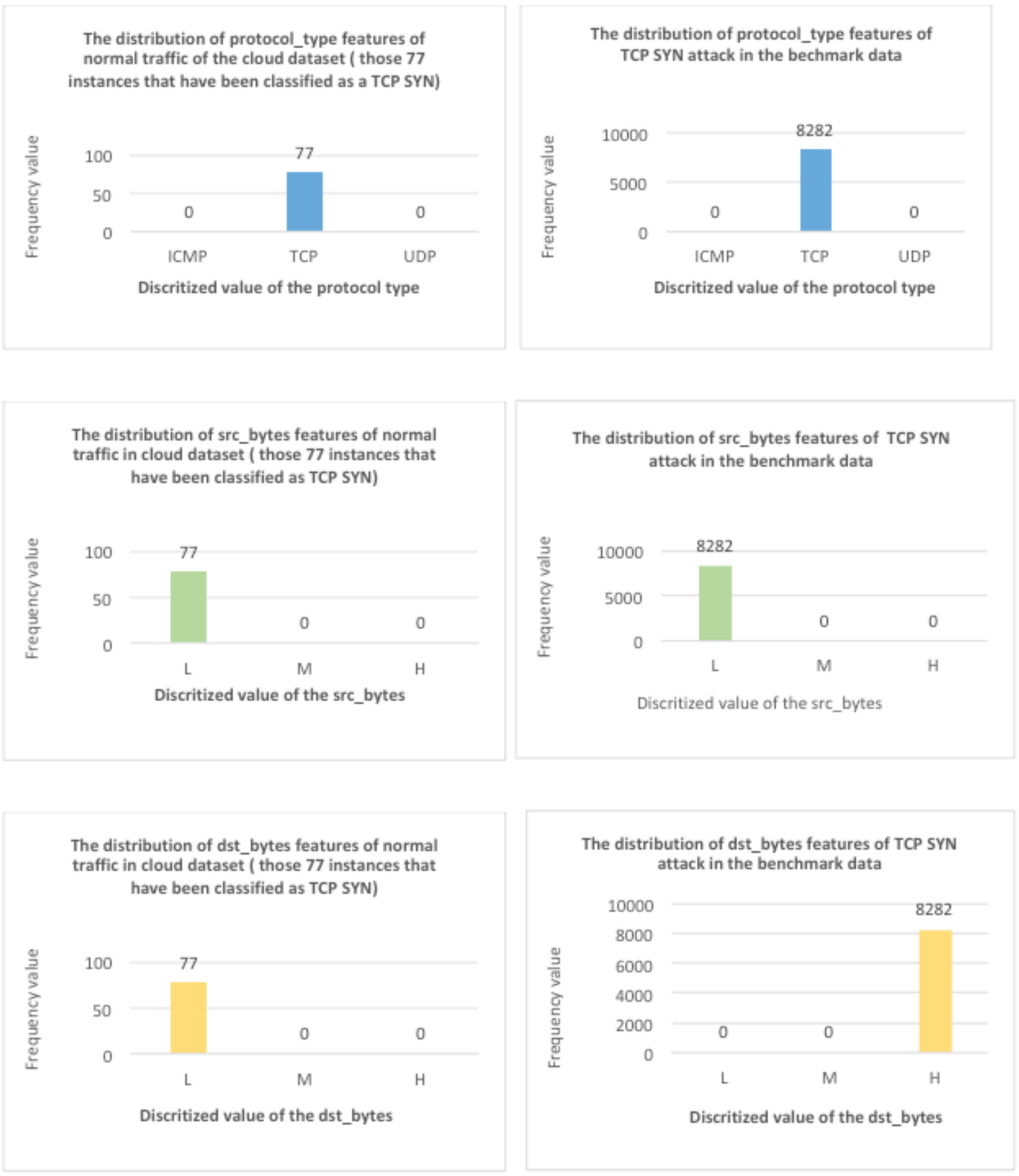


Figure 3.11 The distribution of Normal instances of CIDD that have been classified as a TCP SYN attack in the benchmark dataset

Returning to the description of the TCP SYN attack type in section 2.5, the attacker generates frequent TCP connections with the target by not sending the expected ACK message; this results in the target machine generating a high number of SYN-ACK messages. This results in the number of bytes that have been sent from the target machine to the attacker machine (dst_bytes) being higher than the number of bytes transmitted from the attacker machine to the target machine (src_bytes).

In Figure 3.11, perhaps unsurprisingly, all 77 of the normal cases in the cloud dataset that are classified as being TCP SYN and all of the 8282 TCP SYN cases in the benchmark data have the TCP protocol type. Moreover, the src_bytes distribution categories of all of the cases are the same (in the low category). However, the distribution of the dst_bytes in the cloud cases are all in the 'low' discretized category whereas the benchmark cases are all in the 'high' discretized category. This suggests that the model is classifying these normal cloud cases as being TCP SYN on the basis of the protocol and src_bytes features only.

As can be seen in Figure 3.12, the 156 TCP SYN cases in the cloud dataset that are classified as being normal are split across the three protocol types. The normal instances in the benchmark data are also split across the ICMP, TCP and UDP protocol types, though there are proportionally fewer ICMP instances. The src_bytes values in the cloud cases are in the 'low' and 'high' discretized categories whereas the benchmark cases are in spread across all three discretized categories (low, medium and high), though many fewer in the 'medium' category. Finally, there are instances of dst_bytes values in the cloud cases are in the 'low' and 'high' discretized categories, whereas in the benchmark cases there are instances of in all three categories (low, medium and high), though, again, many fewer in the medium category. This suggests that the model is classifying these TCP SYN cloud cases as being normal on the basis of the protocol type and combination of the src_bytes and dst_bytes in the 'low' and 'high' discretized categories.

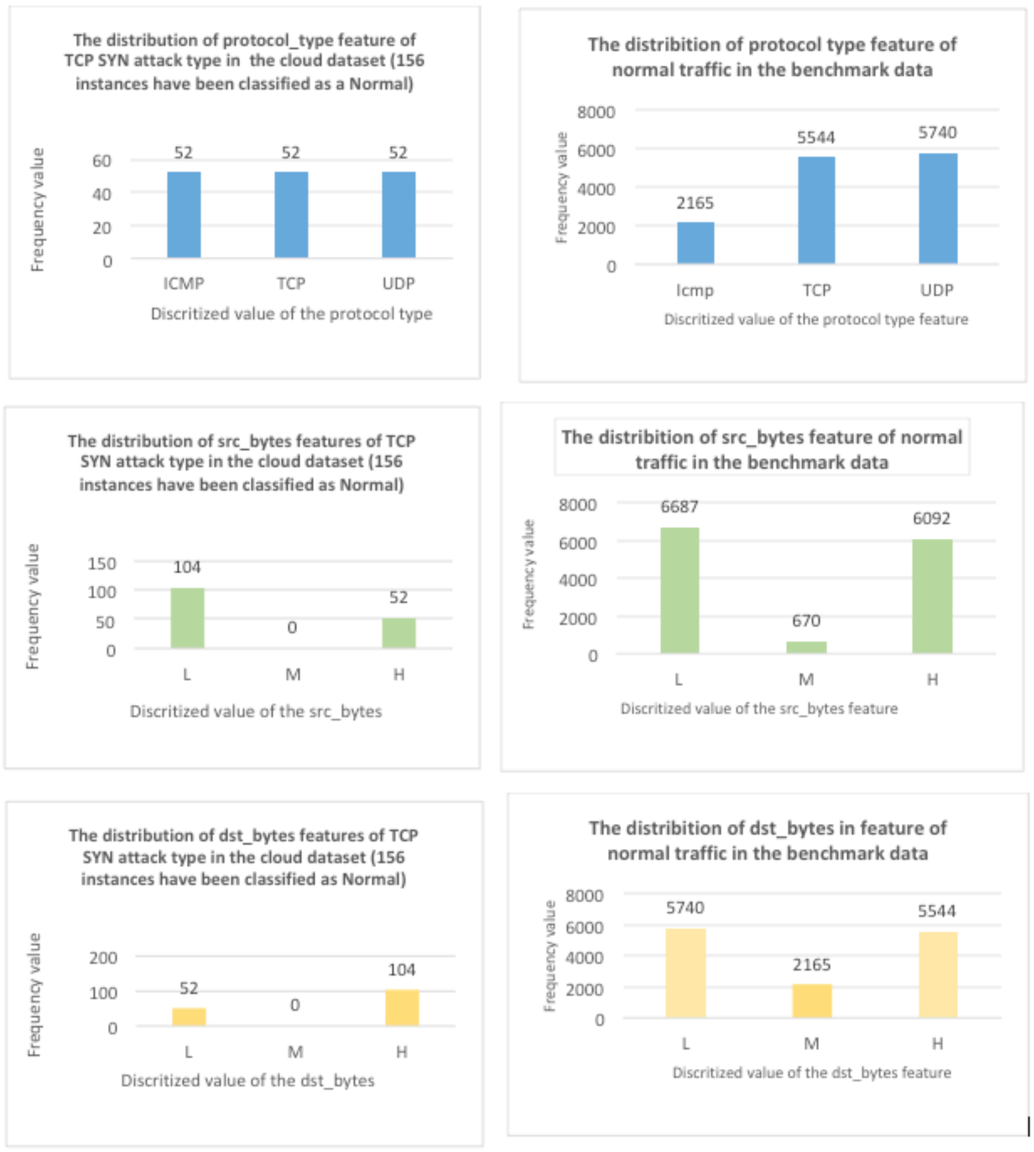


Figure 3.12 Distribution of TCP SYN attack instances that have been classified as normal instances in the benchmark data

Moving on to the UDP attacks type, in the third row of Figure 3.9, it is important to remember that in UDP attacks the attacker generates UDP packets which are sent to the targeted cloud server; the target is not expecting these UDP packets in relation to the associated application on the port, resulting in the reply queue being filled and the target responding with continuous ICMP packet messages to the sender (which are not received as the attacker machine is using a fake/spoofed IP address) (see section 2.5). This results in the number of bytes associated with the ICMP packets that have been sent from the target machine to the attacker machine (dst_bytes) being higher than the number of bytes transmitted from the attacker machine to the target machine (src_bytes)

In Figure 3.13, the 36 cases of UDP attack in the cloud dataset that were classified as being normal have TCP (one instance) and UDP (35 instances) protocol types, whereas the normal cases in the benchmark have instances from all three protocol types (ICMP, TCP and UDP). The src_bytes values in the cloud cases are in the 'medium' (35 instances) and 'high' (1 instance) discretized categories, whereas there are benchmark cases in the all three categories (6687 low, 670 medium and 6092 high). Finally, there are instances of dst_bytes values in the cloud cases in each of the three categories (1 low, 29 medium and 6 high) and the same is true for the benchmark cases (5740 low, 2165 medium and 5544 high). This suggests that the model is classifying these UDP cloud cases as being normal on the basis of a combination of TCP and UDP protocols, 'low' and 'high' discretized values in src_bytes and the dst_bytes features.

As has been noted earlier in relation to the impact of unpacking the time on the structure of cloud dataset, UDP cases are also bound to have multiple protocols, which is contrary to the features of this type of attack (see section 2.5).

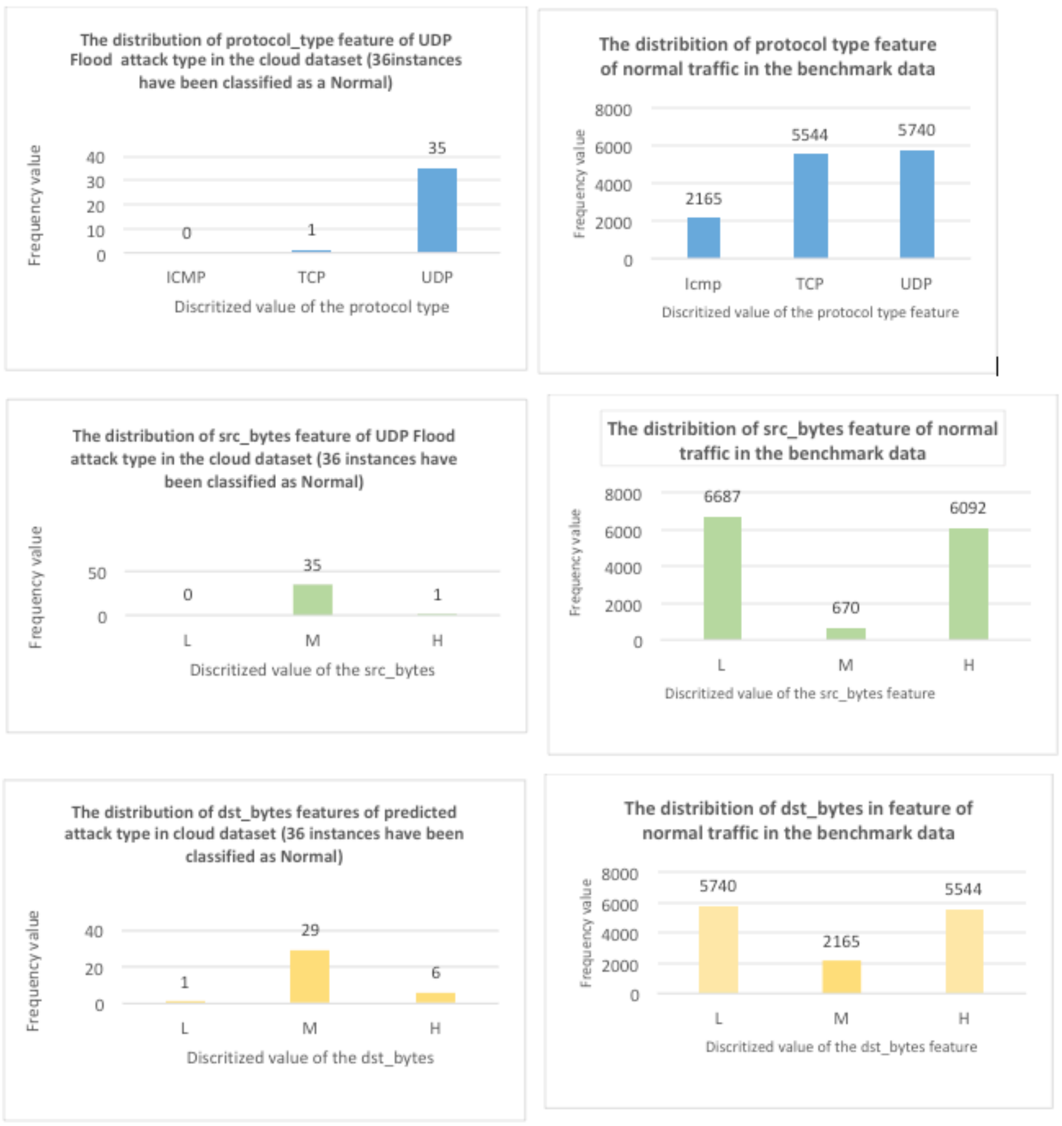


Figure 3.13 The distribution of UDP Flood attack instances that have been classified as normal instances in benchmark data

In Figure 3.14, the other 34 cases of the UDP attack in the cloud dataset were classified as being TCP SYN attack and all the 8282 cases of TCP SYN attack in the benchmark data have TCP protocol type. The src_bytes value in the cloud dataset (34 instances) and TCP SYN (8282 instances) in the benchmark data are in the 'low' discretized category. Finally, there are instances of the dst_bytes value in the cloud dataset that are in the 'low' discretized category (34 low) whereas the dst_bytes value of all 8282 cases of TCP SYN in the benchmark data are in the 'high 'discretized category. This suggests that the model is classifying these UDP cases as being TCP SYN on the basis of the combination of TCP protocol and the 'low' discretized value of the src_bytes feature.

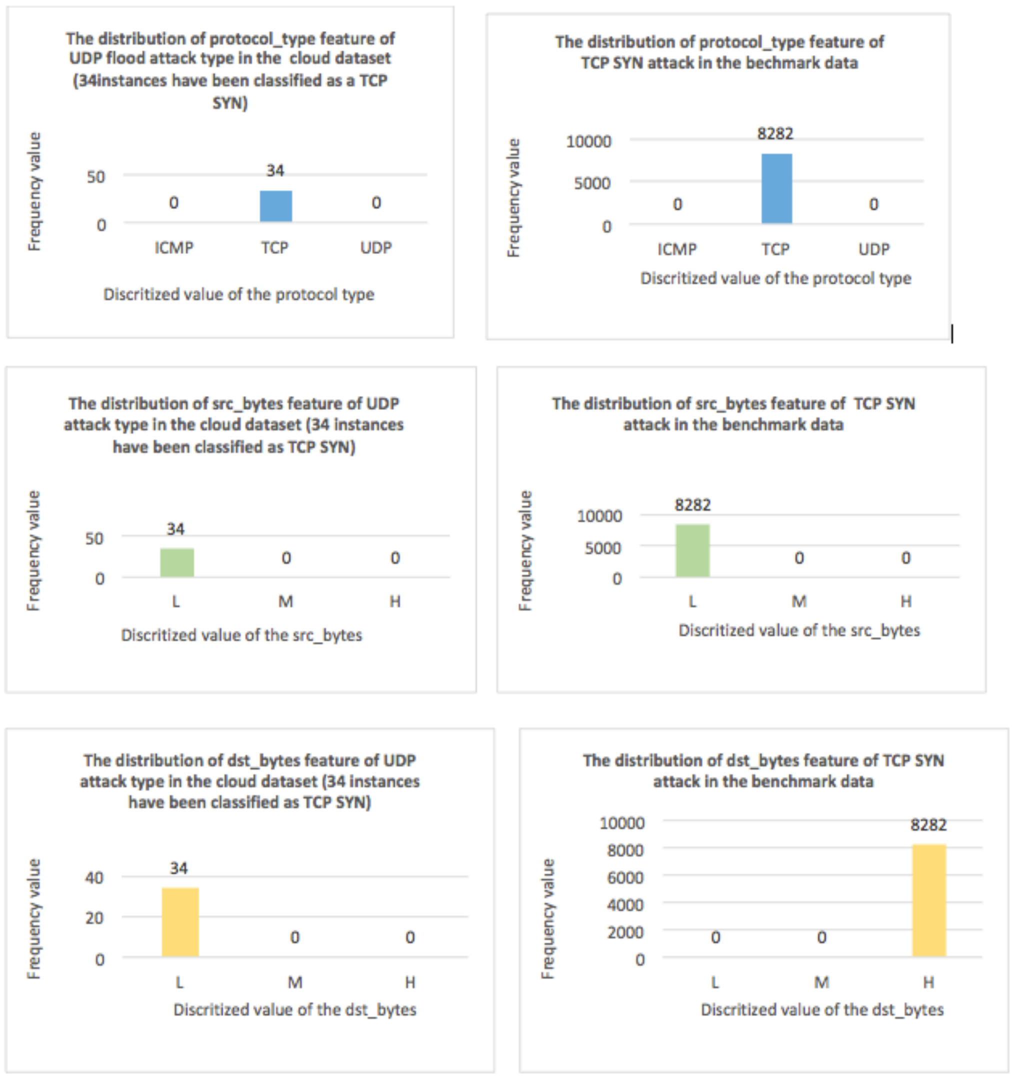


Figure 3.14 The distribution of UDP Flood attack instances that have been classified as TCP SYN instances in benchmark data

In Figure 3.15, the last 35 cases of UDP attack were classified as being Smurf attacks and all the 529 cases of Smurf attack in the benchmark data have the ICMP protocol type. The src_bytes values of cloud cases are in the 'high' discretized category whereas there are instances (529) in the benchmark data that are in the 'low' discretized category. Finally, the value of the dst_bytes in the cloud cases are in the 'low' discretized category where there are cases in the benchmark data that are in the 'medium' discretized category. This suggests that the model is classifying these UDP cases on the basis of the ICMP protocol type only.

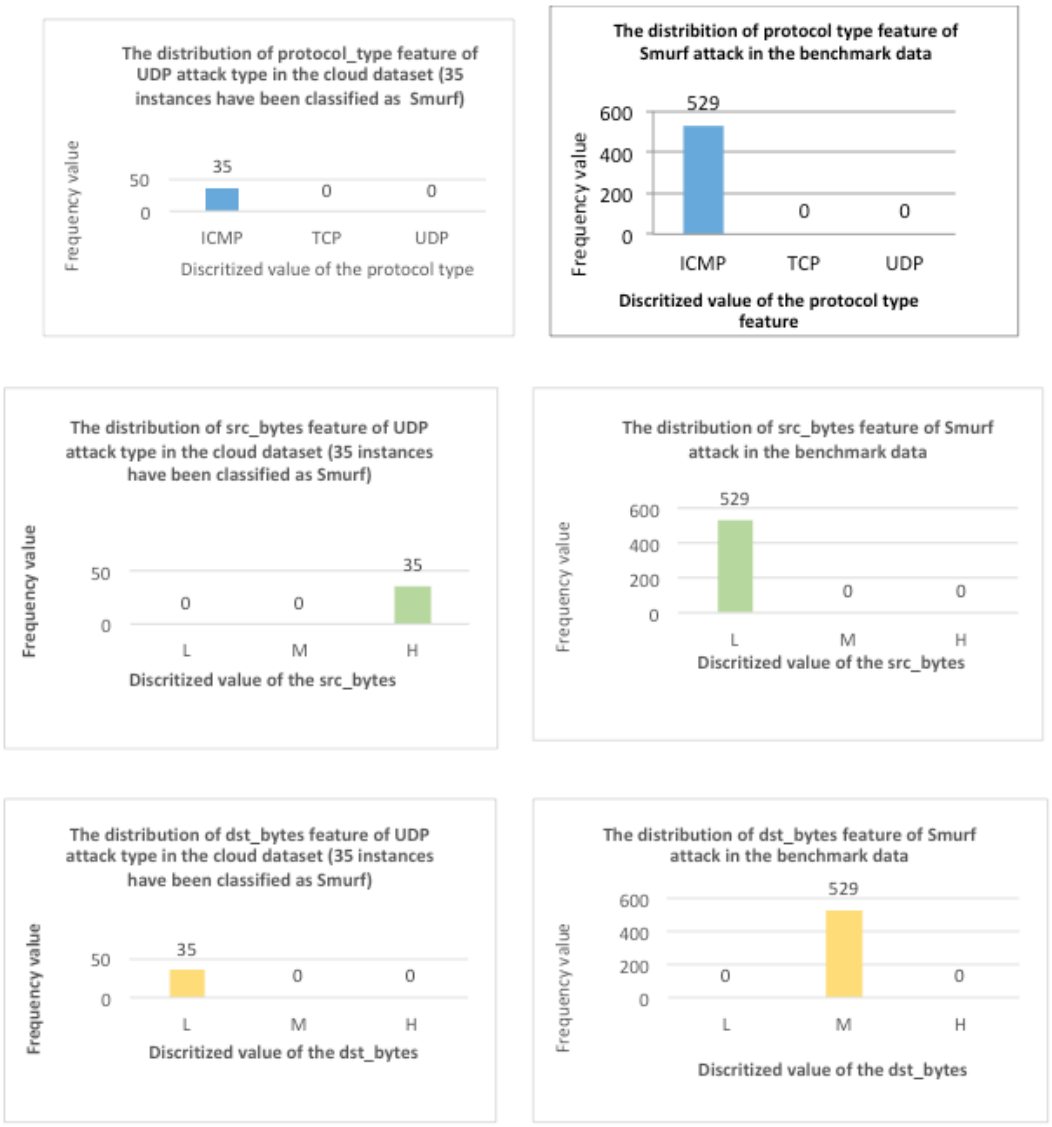


Figure 3.15 The distribution of UDP Flood attack instances that have been classified as Smurf attack instances in benchmark data

Moving to the DNS flood attack type, in the fifth row of Figure 3.9, this type of attack is based on the high number of DNS requests to the DNS server that lead to the generation of UDP traffic to overwhelm the DNS server. This results in the number of bytes of UDP packets that have been sent from the targeted DNS server to the attacker machine (dst_bytes) being higher than the number of bytes transmitted from the attacker machine to the targeted DNS server (src_bytes).

In Figure 3.16, 35 cases of DNS Flood were classified as being normal having the UDP protocol type (35 instances) and there are instances in the benchmark data that have all three types of protocol (TCP, UDP and ICMP). The src_bytes value in the cloud cases are in the 'medium' discretized category whereas the normal cases in the benchmark data are spilt across all three discretized categories (low, medium and high), though many fewer in the 'medium' category. Finally, there are instances of the dst_bytes value in the cloud cases that are in the medium discretized category, whereas the dst_bytes value in the normal cases are in all three discretized categories (low, medium and high), though, again, many fewer are in the 'medium' category. This suggests that the model is classifying these DNS Flood attack cases on the basis of the UDP protocol type and the combination of the 'medium' discretized value of the src_bytes and dst_bytes features.

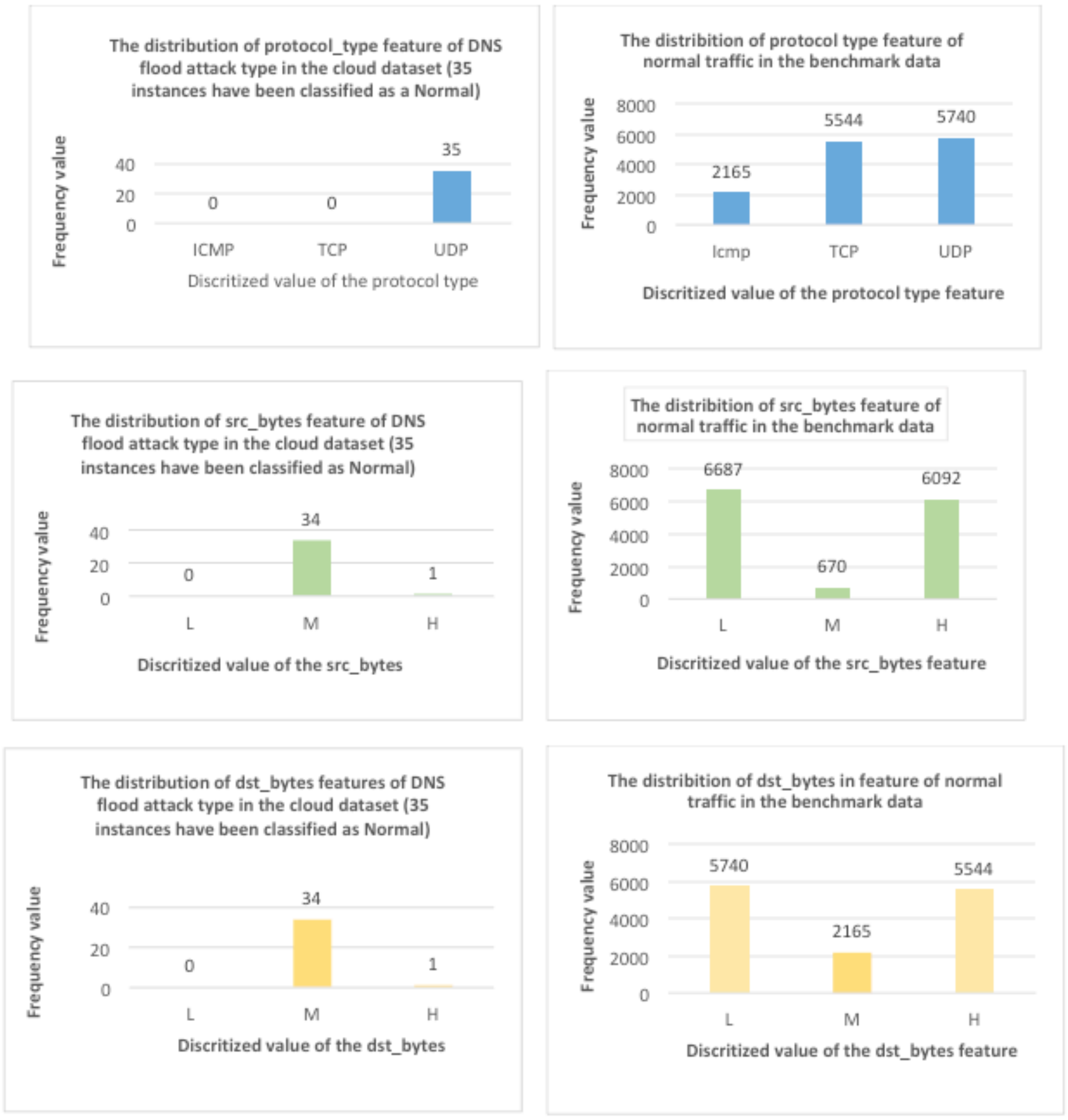


Figure 3.16 The distribution of DNS Flood attack instances that have been classified as a normal traffic instances in the benchmark data

The other 35 cases of DNS attack in Figure 3.17 are classified as being TCP SYN attacks; all the TCP SYN cases in the benchmark data have TCP protocol type. The src_bytes value of the 35 cloud cases and all of the 8282 instances of the benchmark data are in the 'low' discretized category. Finally, there are instances in the cloud data that have dst_bytes in the 'medium' discretized category whereas the all the 8282 cases in the benchmark data are in the 'high' discretized category. This suggests that the model is classifying the DNS

Flood attack on the basis of the TCP protocol and the ‘low’ discretized category of the src_bytes feature.

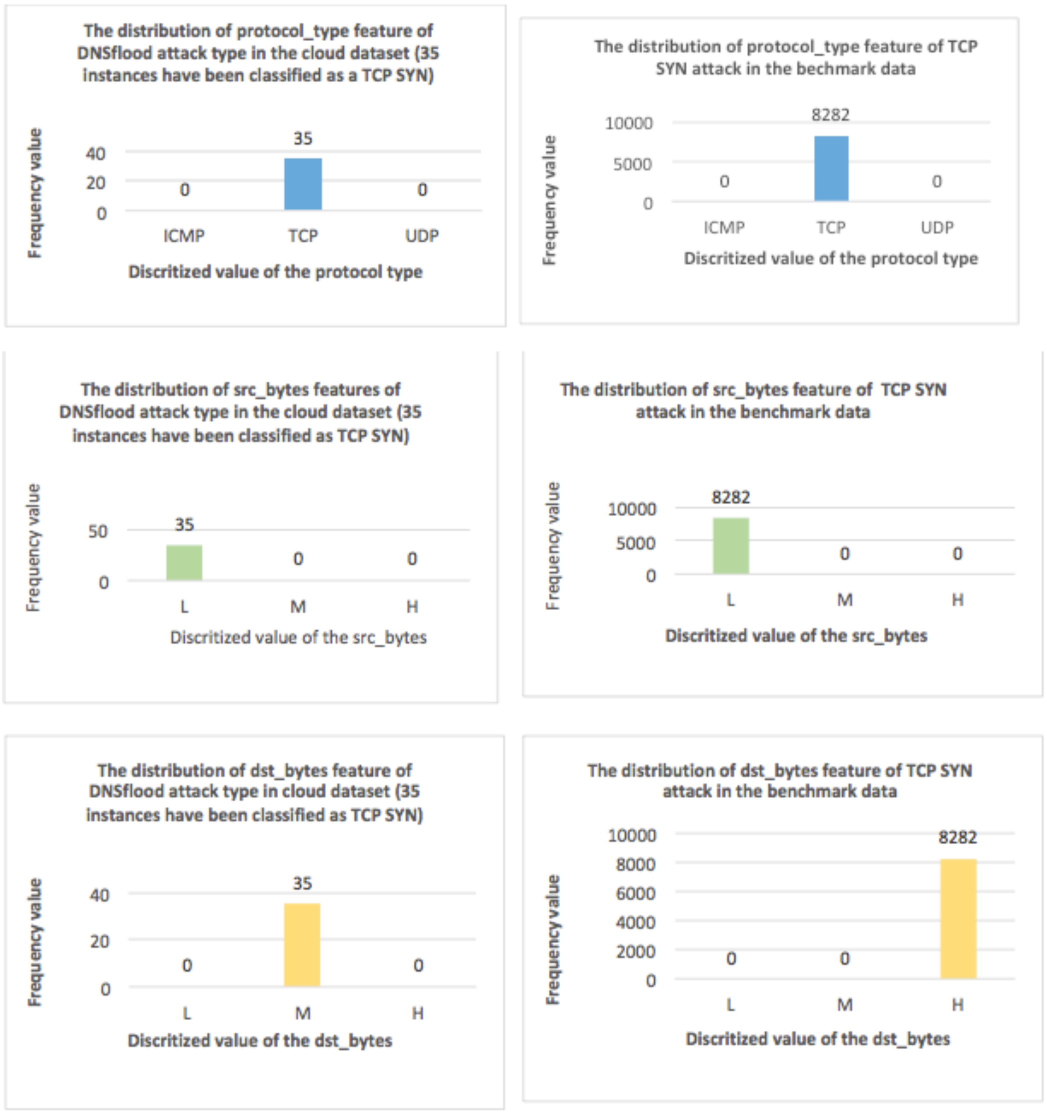


Figure 3.17 The distribution of DNS Flood attack instances that have been classified as a TCP SYN attack instances in the benchmark data

The last 35 cases of DNS Flood in Figure 3.18 are classified as being Smurf attacks; all the Smurf attacks in the benchmark data have the ICMP protocol type. The src_bytes value of the cloud cases is in the ‘high’ discretized category, whereas there are instances in the benchmark data that all are the ‘low’ discretized value. Finally, the dst_bytes value of the cloud cases are in the ‘low’ discretized category whereas all the 529 cases of the benchmark data are in the 'medium' discretized category. This suggests that the model is classifying the DNS flood cases on the basis of the ICMP protocol only.

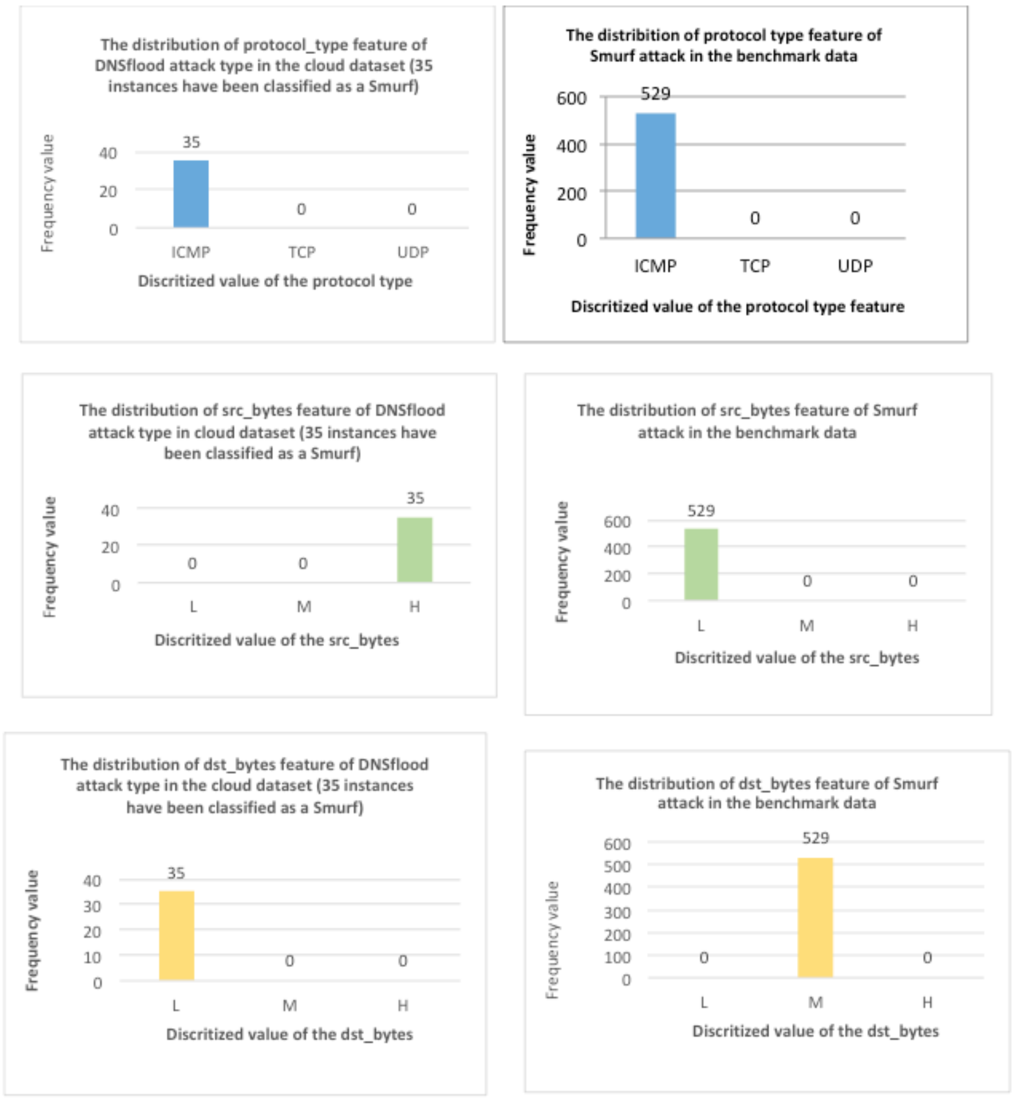


Figure 3.18 Distribution of DNS Flood attack instances that have been classified as a Smurf attack traffic instances in the benchmark data

Moving to the Land attack, in the seventh row of Figure 3.9, this type of attack is generated based on the mal-formed TCP packets that have the same source port and destination port. This makes an empty connection reply to itself until all of the resources have been consumed by the attack. This results in the number of bytes associated with the TCP packets that have been sent from the target machine to the attacker machine (dst_bytes) being higher than the number of bytes transmitted from the attacker machine to the target machine (src_bytes).

In Figure 3.20, the 69 cases of Land attack in the cloud dataset that are classified as being normal are split across the three protocol types. The normal cases in the benchmark data are also split across the TCP, ICMP and UDP protocol types, though there are proportionally fewer ICMP instances. The src_bytes values in the cloud cases are in the 'low' and 'high' discretized categories, whereas the benchmark cases are spread across all three discretized categories (low, medium and high), though there are many fewer in the 'medium' category. Finally, there are instances of dst_bytes values in the cloud cases are in the 'low' and 'high' discretized categories, whereas in the benchmark cases there are instances in all three categories (low, medium and high), though, again, many fewer in the 'medium' category. This suggests that the model is classifying these TCP SYN cloud cases as being normal on the basis of the protocol type and a combination of the src_bytes and dst_bytes in the 'low' and 'high' discretized categories.

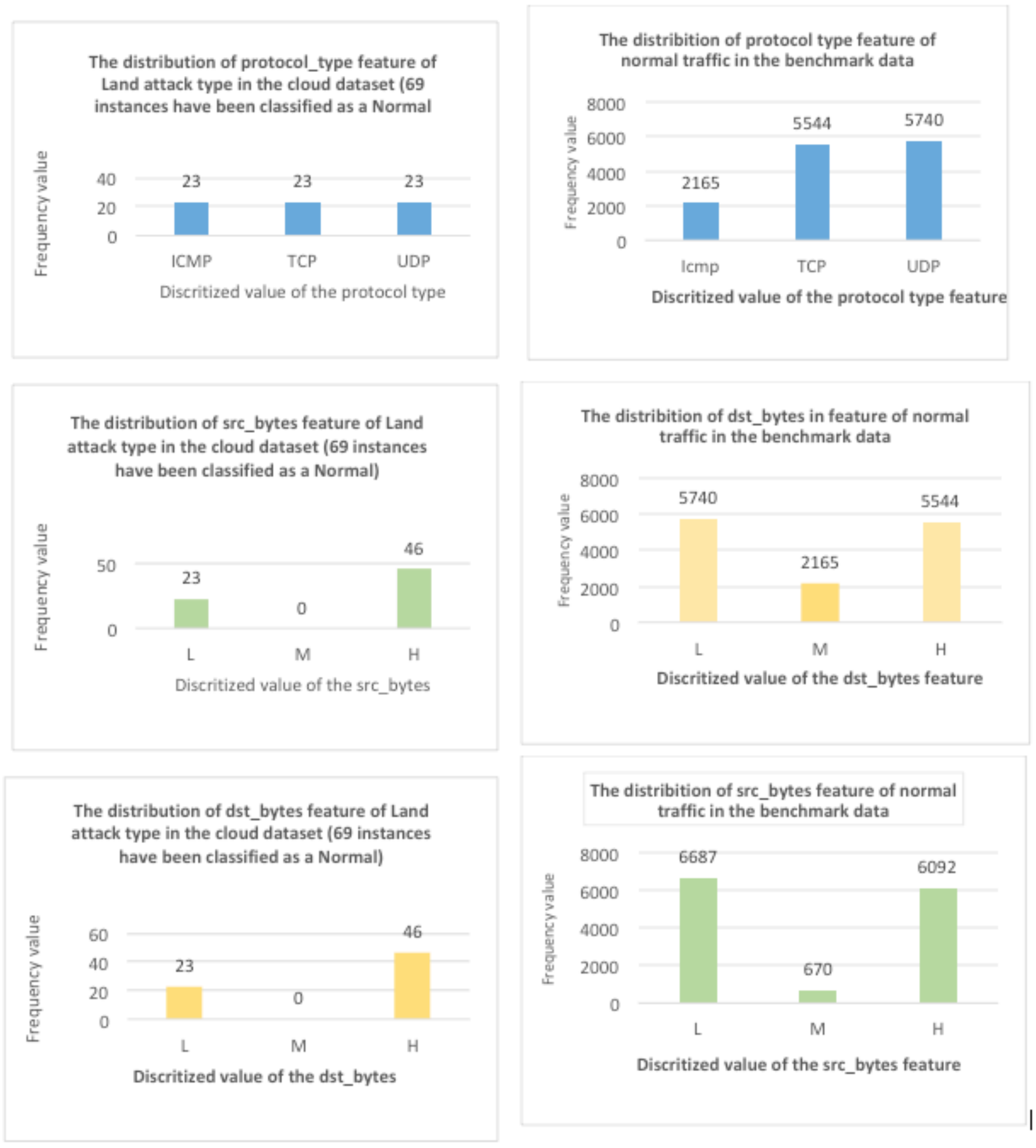


Figure 3.19 Distribution of Land attack instances that have been classified as a Normal traffic instances in the benchmark data

Moving to the Slowloris attack type, in the last row of the Figure 3.19, this type of attack is generated based on incomplete HTTP packets. The attacker holds the connection open and does not release it until the system reaches the maximum number of allowable open connections. This results in the number of bytes associated with the HTTP packets that have been sent from the target machine to the attacker machine (dst_bytes) being higher than the number of bytes transmitted from the attacker machine to the target machine (src_bytes).

As can be seen in Figure 3.20, the 1314 Slowloris cases in the cloud dataset that are classified as being normal are split across the three protocol types. The normal instances in the benchmark data are also split across the ICMP, TCP and UDP protocol types, though there are proportionally fewer ICMP instances. The src_bytes values in the cloud cases and the benchmark cases are in the 'low', 'medium' and 'high' discretized categories, though there are many fewer in the 'medium' category. Finally, there are instances of dst_bytes values in the cloud cases and benchmark data in each of the 'low', 'medium' and 'high' discretized categories, though, again, there are many fewer in the 'medium' category. This suggests that the model is classifying these Slowloris cases as being normal on the basis of the protocol type and combination of the src_bytes and dst_bytes feature.

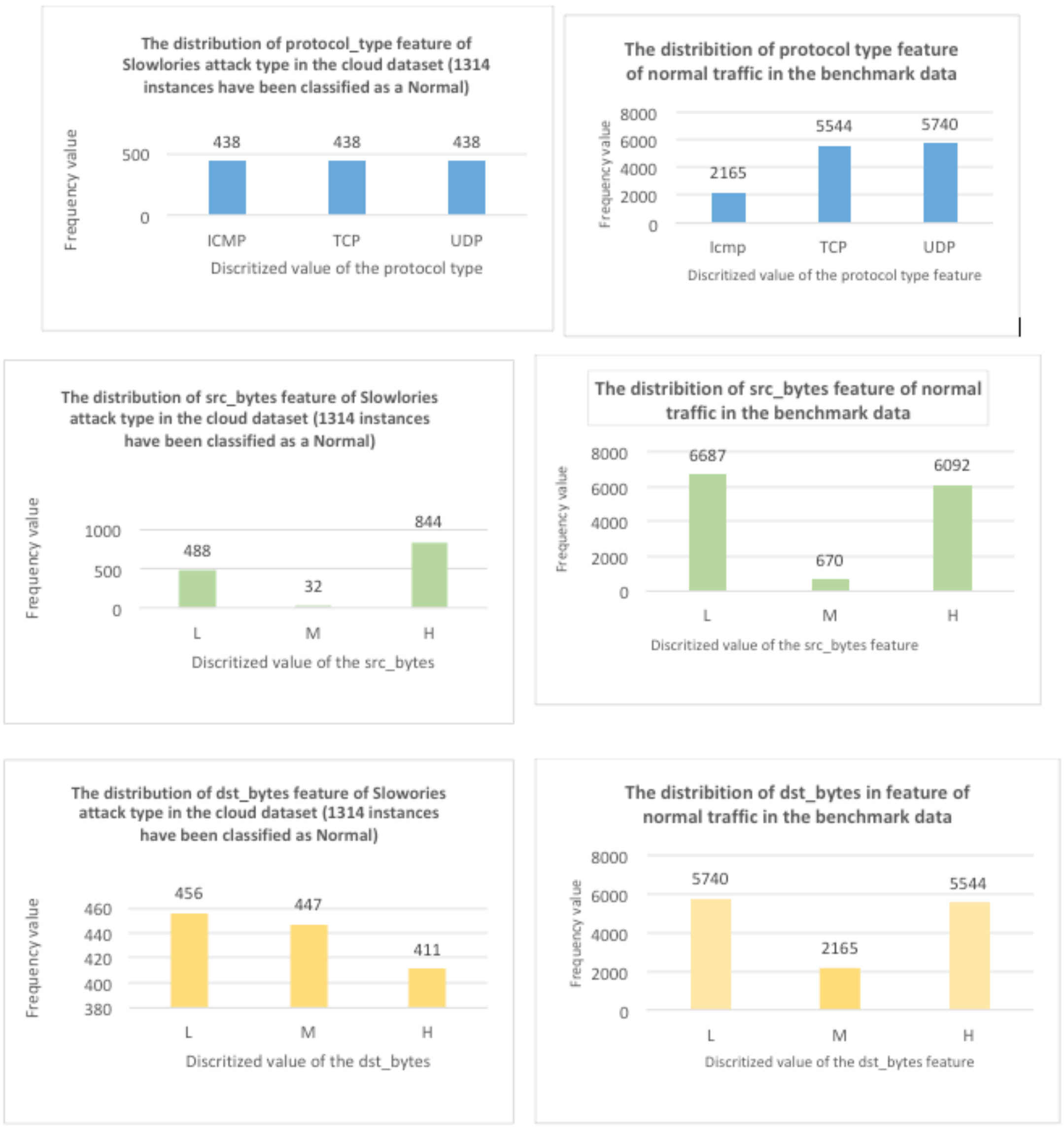


Figure 3.20 The distribution of Slowloris attack instances that have been classified as a normal traffic instances in the benchmark data

3.5.3 Reflection and discussion

This sub section will represent high-level analysis of the confusion matrix results and it will then analyse the impact of the 'unpacking the time' issue on the classification results. The analysis identifies a number of issues related to the developed model which will frame the focus of the next chapter.

Table 3.4 represents those cloud cases in Figure 3.9 that are above the threshold (as such, the PoD and ICMP Flood attack types are excluded).

Actual cloud attack and normal	Classified attack type and normal			The factor that the model suggests								
	Normal	TCP SYN	Smurf	Protocol_types			Src_bytes			Dst_bytes		
				TCP	UDP	ICMP	L	M	H	L	M	H
Normal		✓		✓			✓					
TCP SYN	✓			✓	✓	✓	✓		✓	✓		✓
UDP Flood	✓			✓	✓			✓		✓	✓	✓
UDP Flood		✓		✓			✓					
UDP Flood			✓			✓						
DNS Flood	✓				✓			✓	✓		✓	✓
DNS Flood		✓		✓			✓					
DNS Flood			✓			✓						
Land	✓			✓	✓	✓	✓		✓	✓		✓
Slowloris	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 3.4 Analysis of the results of the 'above threshold' cases

Each row in Table 3.4 reports cloud cases that are classified as other types of attack. The classification model classifies these cases on the basis of the protocol type (TCP, UDP and ICMP), src_bytes and dst_bytes (L, M and H indicate the discretized values). The first row of Table 3.4 represents Normal cases that are being classified as TCP SYN attacks. The rest of the table represents TCP SYN, UDP Flood, DNS Flood, Land and Slowloris cases that are being classified as Normal, TCY SYN or Smurf attacks. For the UDP Flood and DNS Flood types, three rows for each are presented as they are being classified as more than one non-cloud type (Normal, TCP SYN and Smurf). The table then shows the classifier values on which analysis of the underlying data suggests the model may be classifying in each case.

As can be seen in Figure 3.9, all of the cloud cases are classified as Normal traffic, TCP SYN and Smurf attacks. From the cloud data, most of the instances of TCP SYN, UDP Flood, DNS Flood, Land and Slowloris have been classified as Normal traffic (see rows 3, 6, 9 and 10 of Table 3.4). This is because there is the whole range of protocol types and values for src_bytes and dst_bytes in these instances, and there are no other features through which the model can discern them as the other types of attack. The classification as Normal traffic in these cases is likely to be because in the non-cloud dataset on which the model was trained, there are many 'Normal' packet-based cases with different combinations of values for the features, making it the 'default choice' for the model.

There are also instances of CIDD Normal traffic, and UDP Flood and DNS Flood attacks that are being classified as TCP SYN attacks on the basis of the TCP protocol type and a 'low' discretized value for the src_bytes classifier. Finally, it seems that some instances of UDP Flood and DNS Flood in the cloud data are being classified as Smurf attacks on the basis of the ICMP protocol only.

As Figure 3.8 shows, the outcome of the classification model when applied to the NSL-KDD dataset is quite accurate, even with the small feature set. Although there are cases such as Land, PoD, and some of the Normal cases that have not been classified correctly, the overall performance of the model is good on the non-cloud dataset.

When the model is applied to the cloud dataset, its performance is markedly different, as can be seen in Figure 3.9. For example, the TCP SYN, PoD and ICMP flood (another name for Smurf) attack types are common to both datasets which should lead to them being classified as the same type of attack by the model. However, the model mis-classifies every instance of these three attack types in the CIDD dataset.

This problem seems to arise from the structure of the cloud dataset where the 'unpacking time' issue, explained in section 3.4.1, has influenced the structure of the cloud dataset. The structure of the cloud dataset means that each instance/row in the cloud dataset is very likely to have multiple protocol types (since packets are being aggregated into a single data instance/row) whereas each instance in the NSL-KDD dataset includes a single protocol type as it represents a single packet. As a result, TCP SYN cases in the cloud data are very likely to have multiple protocol types, and that is why they have been classified as Normal cases. This is also true for UDP Flood, DNS Flood, Land and Slowloris. Moreover, where the UDP flood cases include TCP protocol packets they have been classified as TCP SYN attacks, and where they include ICMP protocol packets they have been classified as Smurf attacks. This also happens for DNS Flood cases: where DNS cases include TCP protocol packets they have been classified as TCP SYN attacks, and where they include ICMP protocol packets they have been classified as Smurf attacks.

In addition to the structural issue of the datasets, the limited number of overlapping features causes the model to classify some instances of UDP flood as Normal, TCP SYN or Smurf (see Figure 3.9) because there are no other features than can be discerned by the model to classify the instances as the UDP attack type. This is also true for DNS flood cases that have been classified as Normal, TCP SYN and Smurf.

3.6 Summary

This chapter aimed to leverage one of the well-established non-cloud datasets and analyse it in relation to one of the few available cloud datasets to develop a detection model. This was explored through a combination of a machine learning classifier and transfer learning to remap the intrusion types. However, the results of the analysis show that, while the model performed well on the non-cloud dataset, it is of limited value in classifying attacks in the CIDD dataset owing to the different structures of the two datasets, the small overlapping feature set and the different attack types. Given that there are very few publicly-available cloud intrusion detection datasets, there is a need to generate new datasets of this type that have a common structure, address a standard set of attack types (which can be expanded as new attack types emerge) and include a wider range of features to be able to use them to compare results with those of existing work in the area and allow researchers to validate their work/findings on other comparable datasets. This will be the focus of the next chapter.

Chapter 4 : Generating the Cloud Intrusion Detection Dataset

4.1 Introduction

As argued in chapter 3, the analysis related to the developed model showed that while the model performed well when applied to the non-cloud dataset, its performance was limited in classifying attacks in the CIDD dataset because of the different structures of the two datasets, the small overlapping feature set, and the different attack types. Given that there are only a few publicly-available cloud intrusion detection datasets, to be able to improve the model and use it to undertake comparative analysis with existing work in this area, there is a need to generate a new dataset that has a similar structure to the available non-cloud dataset(s), contains the same types of DDoS attack, and comprises a broader range of features. Therefore, this chapter reports the work undertaken to generate a cloud intrusion detection dataset to address the identified issues.

The remainder of the chapter is structured as follows. Section 4.2 will review existing approaches for generating intrusion detection datasets in order to determine which approach best meets the requirements of this research. Section 4.3 will then present the method that the research followed in this phase, detailing the test-bed-architecture that was developed for this piece of the experimental work, including the software and hardware specification. Section 4.4 will explain the process of running the experiment to generate the cloud intrusion dataset. Finally, section 4.5 will present the resulting dataset on which the next stage of the research depended.

4.2 Existing Approaches to Generating Intrusion Detection Datasets

As explained in chapter3, there are only two publicly-available cloud intrusion detection datasets, and there is no standard benchmark cloud intrusion detection dataset for researchers to use. Moreover, there is a lack of systematic approaches in the literature that may be applied to generate intrusion detection datasets. In order to move towards developing a standard benchmark dataset, it is important to review how the existing standard intrusion detection datasets were generated. (Chapter 3 has already introduced the existing non-cloud and cloud intrusion detection datasets, in sections 3.3.1 and 3.3.2, respectively).

Table 4.1 categorizes these datasets in terms of the approach by which they were created – synthetic, real-time and emulation – and provides a definition of each approach type and sets out the tools that each approach type typically uses. It is worth noting that using a simulation approach might also have been be a viable alternative method to employ in generating the cloud intrusion detection dataset. To the best of our knowledge, however, there is no appropriate literature about this approach in the area of intrusion detection. Therefore, it was decided to consider the three identified, existing methods that have been commonly used to generate intrusion detection datasets in other research in order to identify the most suitable approach to adopt for this work. Each of the three approaches will now be explained in more detail.

Approach	Definition	Related datasets to each approach
Synthetic	Synthetic data is artificial data that is generated by various machine learning algorithms. These algorithms have the capability to learn from sample data to create a model that may subsequently generate a synthetic dataset.	Not specified
Real-time	Real-time data is collected from a real environment. This type of data is related to incoming and outgoing network traffic of real workstations (PCs) of users in organisations. All the generated traffic of this environment could be captured from the edge of the router by using network traffic tools such as Tcpdump and so on.	KYOTO
Emulation	Emulated data is collected from a recreated environment. This test-bed environment provides all the software and hardware requirements for cybersecurity experiment, such as a router, firewall, PCs, etc. This enables researchers to configure the network topology to generate the desired dataset.	CIDD, NSL-KDD, CAIDA, DEFCON, LBNL, ADFAUNSW-NB15

Table 4.1 A summary of the existing approaches for generating intrusion detection datasets and examples of resulting datasets

The first category to be considered is the 'Synthetic' approach. This approach creates datasets of artificial data by utilizing machine learning algorithms. For example, Pham, Nguyen and Nguyen (2014) generated a synthetic intrusion dataset from the KDD dataset by using machine learning algorithms with Support Vector Machines (SVM). These algorithms include a set of mathematical functions that are capable of learning from input data (in this case the KDD dataset). The input data (also called the training data) includes a number of features and sample data points/items. The algorithm maps the input data to the features of the dataset and learns from the input data in order to generate the model. This model is then used to generate the required synthetic data.

There are useful tools, such as Weka (Frank, Hall and Witten, 2016) and ID2T, that support the generation of synthetic datasets (see, for example, Vasilomanolakis *et al.*, 2016). The main requirement for generating a dataset using this approach is to have at least a small set of sample data to use as input/training data. In our case, where there is no sample cloud dataset with the required features, it was, though, not an appropriate approach to use.

The second category is the 'Real-Time' approach, which generates datasets from real environments, such as by capturing the data from the edge router of an organisation. The KYOTO dataset was generated using this category of approach; it is made up of a collection of real-time network traffic from the 'honeypots project' at Kyoto University (Song, Takakura and Okabe, 2012). For commercial organisations, however, there are often confidentiality issues they mean that, even if they gather datasets using this approach, they are understandably reluctant to make public the data. As such, using the 'Real-Time' approach was not feasible in this research. This is to be expected; because of the unavailability of real-time datasets from commercial organisations, most academic research applies the synthetic or emulation approaches to generate the datasets (Rao and Naidu, 2017).

The final category is the 'Emulation' approach which uses a computer laboratory environment that emulates a real-work network configuration to generate datasets. The environment will include several physical machines (which may run on various operating systems), switches, firewalls, network connections, and so on, to form the topology of the test network. Depending on the defined goal of the specific research work, as well as generating normal network traffic, the physical machine/PCs will be equipped with tools to generate particular types of attack. The network traffic is captured by the network traffic recorder and analyzer tools, with the recorded file constituting the dataset.

The research reported in the remainder of this thesis used the emulation approach. It is the approach used to create most of the commonly-available intrusion detection datasets, including CIDD, NSL-KDD, CAIDA, DEFCON, LBNL, ADFA and UNSW-NB15. As such, the majority of research publications in the field are based on (or make use of) datasets generated using this approach. Moreover, relevant hardware was available in the university, making it more straightforward and affordable to conduct the dataset generation and associated experimental work using this approach.

Having explained and justified the overall approach taken to generate the dataset, the next section will present the detail of the approach and explain the hardware and software configuration, justifying choices that were made, for example in terms of the specific tools employed.

4.3 Detailed Approach to Generating the Cloud Intrusion Detection Dataset

This section will explain the detailed application of the emulation approach for this research and will set out the software and hardware requirements for generating the dataset in the cloud testbed environment, justifying the choices that were made.

As mentioned in section 4.2, this research aimed to generate a dataset in an emulated cloud environment. In order to inform the test-based design for the research, the starting point was to review the approach taken to generating the CIDD dataset, in terms of the cloud configuration used; however, not enough information about its test-bed architecture was publicly available. Therefore, to determine suitable software- and hardware-level requirements for the test--bed configuration, the architectures of other relevant emulation systems, and the tools that they used, were analyzed. The findings will be presented in subsequent sections as the different parts of the test-bed are presented.

Before moving into the detail, Table 4.2 presents a high-level view of the requirements of the emulation test-bed for generating the cloud intrusion dataset. This shows three main layers in the test-bed configuration: the platform-level specification; the elements of the cloud experimental environment; and the software tools needed to generate and capture network traffic in the cloud environment. The platform-level specification includes all of the software and hardware requirements of the OpenStack cloud platform (see section 4.3.1). The cloud experimental environment layer contains all of the entities required to form the experimental environment (see section 4.3.2). The final piece in Table 4.2 details all of the tools that were required to be installed on the relevant elements in the created cloud environment in order to generate the required dataset (see section 4.3.3).

Cloud test-bed environment			
Platform-level specification			
Software requirements	Choices	Hardware requirements	Choices
Cloud operating system	OpenStack	Memory Hard disk Network CPU (2 processors)	23 GB 466.8 GB 2 processors
Linux operating system	Ubuntu		
Elements of the cloud experimental environment			
Requirements		Choices	
Image (bootable operating system)		Linux server image- Python server image	
Network		External network Internal network Subnets Network interface IP addresses Router	
Virtual machine Instance		8 virtual machines	
Required software to generate and capture the network traffic in the cloud environment			
Requirements		Choices	
Normal traffic script		Python	
DDoS attacks script		Python	
Web server		Python	
Web browser		Lynx	
Network traffic capture		Tcpdump	

Table 4.2 High-level view of the requirements of the emulation test-bed

4.3.1 Platform-level specification

As shown in table 4.2, to develop the cloud-based test-bed environment, it was essential to install a cloud computing platform. This sub-section will introduce the platform's software and hardware requirements.

A range of elements were required to form the platform (the infrastructure of the experimental environment), including virtual machines, a virtual network and a virtual router. To inform the choice of exactly how to implement these elements, various cloud computing platforms that offer IaaS were considered. Based on an analysis of relevant literature, Eucalyptus and OpenStack were found to be the two most common cloud computing platforms that have been used for the evaluation of intrusion detection systems in cloud-computing environments.

The Elastic Utility Computing Architecture for Linking Your Program to Useful System (Eucalyptus) is an open source cloud platform, released in 2008, that provides IaaS for organisations. The Eucalyptus cloud platform was used as the infrastructure for generating the CIDD dataset and has also been used in other research in the field. As such, Eucalyptus would have been a reasonable choice for the cloud platform.

However, the researcher had no experience in using Eucalyptus. In contrast, prior experience had been gained with OpenStack, so it was decided to install an OpenStack instance as the cloud computing platform for this phase of the research. OpenStack offers very similar functionality to Eucalyptus and has also been widely used in the research area (Lindgren, 2013). The remainder of this sub-section will introduce the OpenStack cloud platform.

OpenStack is an open source cloud platform developed by NASA and Rackspace. It provides Infrastructure as a Service (IaaS) by offering great pools of computing, storage and networking resources through a dashboard that gives control to the user on the provision of these resources. OpenStack includes three main components: Nova; Neutron; and Swift. Nova supports the creation of virtual machines and also provides provisioning of virtual compute instances. Neutron offers an Application-Programming Interface (API) that allows network connectivity to be set up and the creation and management of virtual networking devices such as switches, routers and subnets. Finally, Swift is cloud storage software which supports the creation of virtual machine images and allows the storage and retrieval of large amounts of data using an API.

OpenStack has certain software and hardware dependencies. First, there is a need to install Ubuntu Linux as a host operating system. Second, though there are various configurations for the hardware specification of Open Stack (for example, different components of OpenStack can run on different PCs), each physical machine that is used must have at least one processor and 4GB of memory (RAM).

As part of the set-up of the test-bed, OpenStack was installed on PCs with the minimum hardware requirement to test its performance in terms of speed. It was observed that the performance was very slow, so the hardware specification was increased. Two processors were added to the PC and its memory was increased to 23GB. This resulted in an improved performance of Open Stack once installed.

Having set up the cloud platform, the next step was to develop the software environment through which to generate the dataset. The following sub section explain all of the elements that made up this environment.

4.3.2 Elements of the cloud experimental environment

As shown in Table 4.2, to make the cloud experimental environment, it was necessary to create a number of elements on the cloud platform including images, external network, internal network, subnets, network interface, IP address, router, and virtual machine instances. Figure 4.1 represents all elements of the OpenStack environment, comprising eight virtual machine instances, internal network, router, and external network. This subsection will introduce all of the required elements that were created to build the cloud test-bed environment.

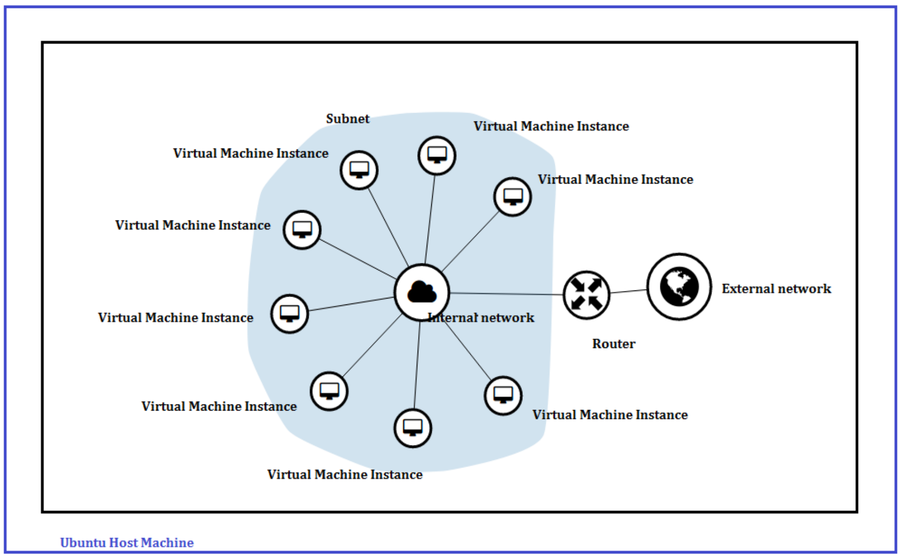


Figure 4.1 All the required elements of the cloud test-bed environment

The cloud test-bed environment was initiated by building cloud images, as it was the first, required step. This is a file including a virtual disk that has a bootable operating system installed on it. Cloud images were utilized to create virtual machine instances within the OpenStack. To launch virtual machine instances, it was necessary to first build the virtual network infrastructure. To do this, it was required to create the external network, which was connected to the physical network infrastructure. The external network provided IP addresses to virtual machine instances in the cloud and supplied a means of access to the outside world through internal network

As such, it was necessary to create the internal network since it was a requirement of connecting to the external network. To connect the external network to the internal network, a router was created to provide the required connectivity between the two networks.

Once all of the fundamental components of the network infrastructure had been created, eight virtual machine instances were launched – this was deemed to be a reasonable number for generating the desired traffic. The images, internal network and external network were allocated to each virtual machine. Associated IP addresses were then assigned to each of the virtual instances to allow them to communicate with each other and the outside world.

Section 4.3.3 will report the next stage of the experiment, explaining the tools that were installed on each virtual machine in order to generate normal traffic as well as the different types of DDoS attack traffic.

4.3.3 Required software tools to generate and capture the network traffic in the cloud environment

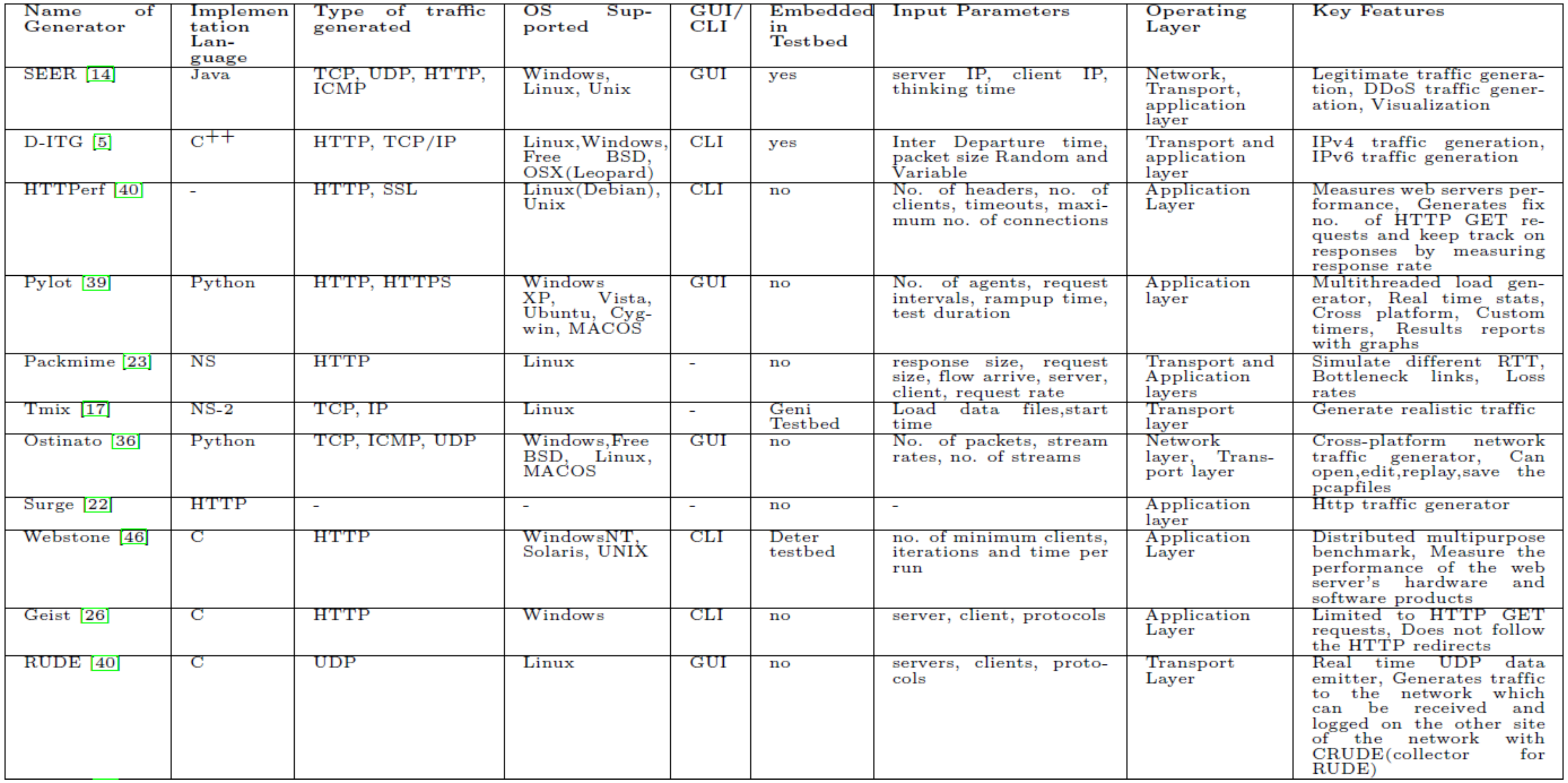
Once the required cloud infrastructure had been developed, it was essential to set up relevant software tools to generate DDoS attacks against the test-bed cloud architecture as well as to generate normal traffic, and to capture the generated traffic for further study. Based on the analysis of relevant literature, Table 4.2 shows all of the software requirements, and the list of chosen tools employed in this research, including Python normal traffic script, Python DDoS attack scripts, Tcpdump network traffic capture, Python web server, Lynx web browser. Moreover, as it was necessary to analyse all of the existing software tools for generating the dataset, this stage of this research reports in more detail than the first two stages. All of the details related to the justification of the choices made in terms of the software tools employed will now be explained.

As mentioned earlier, all of the required software tools were determined by reviewing the list of software specifications used to generate the CIDD dataset and other emulation-based datasets. However, not enough information in terms of software specification tools was publicly available in relation to the emulated datasets. Therefore Table 4.3 presents the result of the analysis of software requirement tools used in the generation of the UNSW-NB15, ADF, NSL-KDD and CIDD datasets. As Table 4.3 shows, the required software includes a network normal traffic generator, DDoS attack generator tool, network traffic capture and analyzer tool, web server and web browser, each of which will now be explained.

Name of dataset	Traffic generator tool	DDoS attack simulation tool	Network traffic capture and analyzer tool	Web Resources	References
UNSW-NB15	LXIA PerfectStorm	LXIA PerfectStorm	Tcpdump/BroIDS	SQL	Moustafa (2015)
ADFA	Metasploit	Metasploit	Not specified	SSH, FTP, SQL,PHP	Creech and Hu (2013)
CIDD	Not specified	Hping3	Custom-bases Sniffer	Siege	(Kumar, Lal and Sharma, 2016))

Table 4.3 Summary of existing software tools for emulated intrusion detection datasets

A network traffic generator tool was used to generate normal network traffic in the controlled lab environment (in our case OpenStack). There are various network traffic generator tools defined for a specific purpose in terms of testing network devices, the supported operating system, the supported network layer, the type of traffic, being embedded in the test-bed, and so on (Botta, Dainotti and Pescapé, 2012). For example, Table 4.3 presents the ixia PerfectStorm tool used for generating network traffic as well as DDoS attacks in UNSW-NB15, and show that the Metasploit tool was used in the ADFA dataset for both traffic generation and DDoS attack traffic. However, no information on the technical details of these tools was available. Therefore, to determine the most suitable tool to generate normal network traffic, relevant literature on existing network generator tools was analyzed. Table 4.4 presents findings on the most commonly-used network traffic generator tools and DDoS attack generator tools (Behal and Kumar, 2017).



Name of Generator	Implementation Language	Type of traffic generated	OS Supported	GUI/ CLI	Embedded in Testbed	Input Parameters	Operating Layer	Key Features
SEER [14]	Java	TCP, UDP, HTTP, ICMP	Windows, Linux, Unix	GUI	yes	server IP, client IP, thinking time	Network, Transport, application layer	Legitimate traffic generation, DDoS traffic generation, Visualization
D-ITG [5]	C++	HTTP, TCP/IP	Linux,Windows, Free BSD, OSX(Leopard)	CLI	yes	Inter Departure time, packet size Random and Variable	Transport and application layer	IPv4 traffic generation, IPv6 traffic generation
HTTPerf [40]	-	HTTP, SSL	Linux(Debian), Unix	CLI	no	No. of headers, no. of clients, timeouts, maximum no. of connections	Application Layer	Measures web servers performance, Generates fix no. of HTTP GET requests and keep track on responses by measuring response rate
Pylot [39]	Python	HTTP, HTTPS	Windows XP, Vista, Ubuntu, Cygwin, MACOS	GUI	no	No. of agents, request intervals, rampup time, test duration	Application layer	Multithreaded load generator, Real time stats, Cross platform, Custom timers, Results reports with graphs
Packmime [23]	NS	HTTP	Linux	-	no	response size, request size, flow arrive, server, client, request rate	Transport and Application layers	Simulate different RTT, Bottleneck links, Loss rates
Tmix [17]	NS-2	TCP, IP	Linux	-	Geni Testbed	Load data files,start time	Transport layer	Generate realistic traffic
Ostinato [36]	Python	TCP, ICMP, UDP	Windows,Free BSD, Linux, MACOS	GUI	no	No. of packets, stream rates, no. of streams	Network layer, Transport layer	Cross-platform network traffic generator, Can open,edit,replay,save the pcapfiles
Surge [22]	HTTP	-	-	-	no	-	Application layer	Http traffic generator
Webstone [46]	C	HTTP	WindowsNT, Solaris, UNIX	CLI	Deter testbed	no. of minimum clients, iterations and time per run	Application Layer	Distributed multipurpose benchmark, Measure the performance of the web server's hardware and software products
Geist [26]	C	HTTP	Windows	CLI	no	server, client, protocols	Application Layer	Limited to HTTP GET requests, Does not follow the HTTP redirects
RUDE [40]	C	UDP	Linux	GUI	no	servers, clients, protocols	Transport Layer	Real time UDP data emitter, Generates traffic to the network which can be received and logged on the other site of the network with CRUDE(collector for RUDE)

Table 4.4 Network traffic and DDoS attack generator tools (Behal and Kumar, 2017)

However, most of these tools were not compatible with OpenStack in terms of supported operating system, command line and interface-based features, image size and image type. For example, it was initially decided to choose Metasploit, which is a popular penetration testing platform that includes all of the required tools to test network security. However, it was observed that the large image size and the interface-based features of the Metasploit caused issues within OpenStack and crashed the system. Therefore, literature analysis was undertaken in relation to other tools that had a smaller image size and command line features which were compatible within the cloud. As result, Python was chosen to be employed on OpenStack for this research.

Python is an object-oriented high-level programming language with dynamic semantics used for web and app development. It is also a scripting language that is easy to use, and it supports libraries and packages. Moreover, a large majority of web application such as Google, YouTube and so on are based on Python. The availability of a normal traffic script, its compatibility with OpenStack and its command line-based feature, meant that it was seen as being the best tool to use.

To generate DDoS attacks, it was essential to install a DDoS attack simulation software tool. This tool generates different types of DDoS attack against its target, which is either based on bandwidth or the server. It includes a range of features for each type of DDoS attack, such as fluctuated and continuous attack rates, protocol type and so on. Since the focus of this research was to generate network, transport and application layer attack types (see Figure 2.4), it was required that the software tool generate ICMP, UDP, HTTP and TCP protocols with a continuous attack rate. Based on the relevant literature analysis related to DDoS simulation tools such as Metasploit, Hping 3, and lxia (as presented in Table 4.3) and other relevant, existing tools, Python was seen as being closer to the requirements of this research. This was because lxia is a commercial tool and was not freely accessible; Metasploit was not compatible with the test-bed environment, and Hping 3 was not capable of generating a DDoS attack. As already mentioned, the command line feature of Python, its compatibility with OpenStack and the availability of different types of DDoS attack scripts made it a more suitable tool to use than others.

To capture normal and DDoS attack traffic, it was essential to use a network traffic capture and analyzer tool. This class of tool is capable of monitoring and saving the incoming and outgoing live traffic. As such, it provides useful statistical information about network traffic flow, network packets and so on. This knowledge is beneficial in terms of understanding network behaviours, detecting network attacks and improving the network planning strategy. For example, source and destination IP address information shows who is originating and receiving the network traffic; port information displays the utilization of the network application.; the packet and byte counts indicate the amount of network traffic. There are various tools that address all of these purposes, including Tcpdump, Bro and sniffer tools (see Table 4.3). Wireshark and T-shark are also popular network capture and analyzer tools and include the same functionality, with only minor differences in terms of whether they are command line, offer an easy to use interface, and how they present network flow and packet features. Since this research was seeking a command line network traffic capture tool to be compatible with OpenStack, and that could extract specific features for detection of DDoS attacks, the Tcpdump network traffic capture and analysis tool was seen as being the best choice from the available tools.

Tcpdump is a command-line network analyzer and monitoring tool that allows a high-level view of the whole network and its graphical visualization. It is capable of real-time packet capture, network monitoring and protocol analysis. More importantly, the Tcpdump tool provided the possibility for this research to export and capture a wide range of features of normal and attack network traffic for further analysis.

As the target of the DDoS attack is web servers or network resources, it was necessary to install servers in the defined target zone (see Figure 4.2) of OpenStack. A web server is a software that uses Hyper Transfer Protocol (HTTP) to serve web resources (such as emails, downloading a request for file transfer and so on) to the incoming Internet request to the end user. A user can send the request to access all of the authorized web resources through a web browser.

As shown in table 4.3 SQL, SSH, FTP and PHP web servers were used in creating the UNSW-NB15, ADFA and CIDD datasets. However, not enough information about the type of these services was available. Therefore, the relevant literature on the required resources associated with DDoS attack was analyzed. The findings, summarized in Table 4.5, include vulnerable servers and websites that are defined for the purpose of ethical hacking (such as Metaspoiltable, Google Gruyere and BWAPP).

The name of the server	Description	References
Metaspoiltable	This is a Linux-based virtual machine that is used for penetration testing. It is created by the Rapid7 Metasploit team and includes intentional vulnerabilities to be exploited for research purposes.	(Metasploit, 2019)
Google Gruyere	This website contains vulnerabilities, such as DoS attacks, that are designed for educational hacking experiments. It helps security researchers and web developers to understand how hackers exploit these weaknesses of the website to improve security mechanisms.	(Leban, Bendre and Tabriz, 2017)
Buggy Web Application (BWAPP)	This is an insecure PHP web application that contains a MYSQL database. It is used for educational and web security testing purposes to help researchers to discover and prevent web vulnerabilities.	(Mesellem, 2014)

Table 4.5 Some of the common vulnerable webservers used for penetration testing

However, these servers were not compatible with OpenStack owing to issues with graphical interface features and large image size. Instead, a Python web server and Python web browser were seen as suitable choices for this research to use as they offer a command line interface, are compatible with OpenStack and simplify the configuration of the Python server. As mentioned earlier, the standard library of Python includes integrated modules, including those of web server and web browser. A Python HTTP server and Lynx web browser were used in this research. Lynx is a text-based web browser which uses a command-line interface. These modules can be invoked for client web server communication, as required in this research.

4.4 Data Collection Process

This section will explain the process of data collection. To this end, it presents the actions of a cloud user performing a range of different DDoS attacks and normal activities in the emulated cloud environment to generate the desired network traffic dataset by employing the essential tools that were explained in the previous section.

4.4.1 Designed Scenario of the Experiment

Once the infrastructure of the experimental environment had been developed and equipped with all of the required software tools, the next stage was to generate the normal as well as the DDoS attack network traffic data. To generate the dataset by using the installed tools, it was necessary to design a scenario in which a specific role was allocated to each virtual machine (VM) within the OpenStack set-up. Table 4.7 explains all of the task that were assigned to each VM, identifying the target VM, DDoS attacker VMs and normal VMs.

Type of scenario	Name of the entity	Allocated roles
Target zone	Virtual Machine Instance 1 (VM1)	VM1 hosted the HTTP python server, and the target of all types of DDoS attack. Tcpdump captured all the incoming traffic to this zone.
Attack scenario	Virtual Machine Instance 2 (VM2)	VM2 generated a Slowloris attack.
Attack scenario	Virtual Machine Instance 3 (VM3)	VM3 generated UDP flood and DNS Flood attacks.
Attack scenario	Virtual Machine Instance 4 (VM4)	VM4 generated a TCP SYN attack.
Attack scenario	Virtual Machine Instance 5 (VM5)	VM5 generated a ICMP Flood attack.
Legitimate scenario	Virtual Machine Instance 6 (VM6)	VM6 sent a normal request to the server in VM1 through a Lynx browser.
Legitimate scenario	Virtual Machine Instance 7 (VM7)	VM7 sent a normal request to the server in VM1 through a Lynx browser.
Legitimate scenario	Virtual Machine Instance 8 (VM8)	VM8 sent a normal request to the server in VM1 instance 1 through a lynx browser.

Table 4.6 The designed scenario for each virtual machine instance in the cloud

As Table 4.6 shows, VM1 was used to host the HTTP Python server and was defined to be the target of the DDoS attacks. As such, the Tcpdump tool was installed on VM1 to capture all of the incoming traffic to this zone. VM2, VM3, VM4 and VM5 were determined as attacker machines and included all of the required DDoS Python scripts – Slowloris, UDP

Flood, TCP SYN, DNS Flood, PoD and ICMP Flood. VM6, VM7 and VM8 were defined as machines/PCs of legitimate users, equipped with Python Lynx and used for normal user activities, such as downloading files, accessing web pages, and so on. Section 4.4.2 will provide an explanation of how each of the scenarios detailed in Table 4.6 was run.

4.4.2 Generating and Collecting the Traffic Data

Once all of the scenarios that comprised the experiment had been designed, the next stage was to implement each of the scenarios and collect all of the generated network traffic.

Figure 4.2 presents a pictorial representation of the cloud test-bed environment, including all of the required entities that were explained in Table 4.2. In the data-capture period, as Figure 4.2 shows, the server was running on VM1 and 'listening' to all of the incoming traffic to this zone. Then, a relevant Python script associated with each attack type was separately run in a different time period to attack the server. Each of these attack scripts were designed based on the behaviour of the relevant DDoS attack (see sections 2.5.1 and 2.5.2). The Tcpdump tool that had been installed on VM1 captured and saved all of the different type of attack and normal traffic for subsequent analysis.

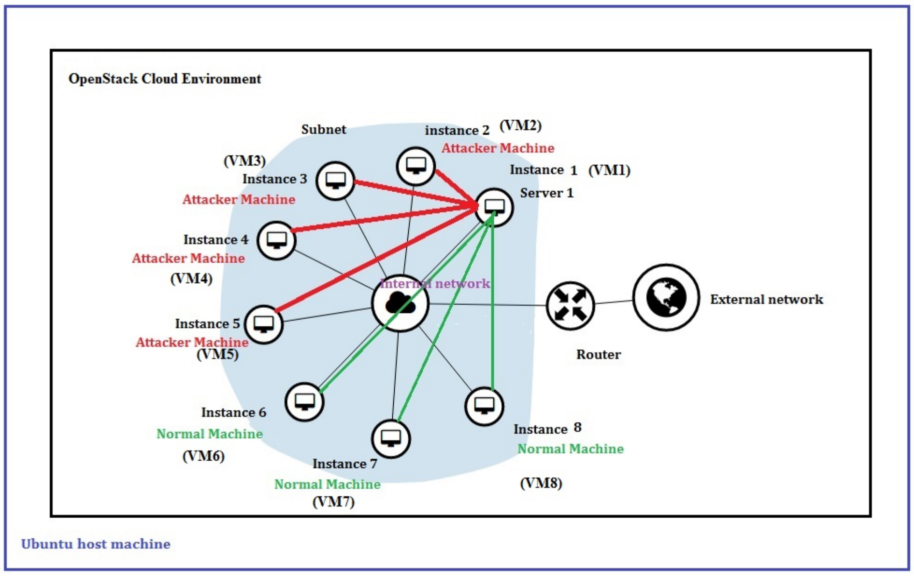


Figure 4.2 The cloud experimental environment

4.5 Summary

This chapter has reported the approach taken to generating the cloud-based intrusion detection dataset for subsequent analysis in the remainder of the thesis. To achieve this aim, this chapter first reviewed and analyzed the different approaches that could be used to create the intrusion detection dataset in order to assess which approach best met the requirement of this work. Based on this analysis, the emulation approach was chosen to generate the dataset. The chapter then created the software and hardware specifications to form the experimental environment and selected, with justifications, the tools to be used to generate the different types of DDoS attack and normal network traffic. Having created the specified environment, the desired dataset was generated using the emulated OpenStack cloud environment using various Python modules. The next chapter will analyse the resulting dataset.

Chapter 5 : Developing Intrusion Detection Models

5.1 Introduction

Chapter 4 presented the method for generating the desired dataset to be used to improve the intrusion detection model and to undertake comparative analysis with existing work in this area. To be able to build a classification model and analyse its performance there was a need for pre-processing to be undertaken on the dataset to ensure that it was in an appropriate form. This chapter therefore reports the pre-processing work that was undertaken to ensure that the structure of the data was appropriate for the classification task and subsequent analysis.

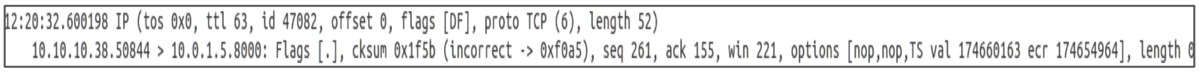
The remainder of this chapter is structured as follows. Section 5.2 will present the features and structure of the dataset to demonstrate the importance of the data pre-processing that was undertaken the dataset that was created/captured as a result of the work reported in chapter 4. Section 5.3 will then present all of the details of the data pre-processing, including finding the commonalities in the different types of malicious and legitimate network traffic datasets, and the parsing and processing of the dataset to transform its structure into a suitable format for the classification task. The choices of the time intervals/slices that were used for comparative analysis will also be explained in this section. Section 5.4 will then present the results of the classification task on the processed dataset. Finally, section 5.5 will present the subsequent analysis of the results of the classification model using different time slices.

5.2 Dataset

This section will introduce the structure of the generated dataset, including its features, in order to define the required preprocessing steps.

The experiment described in chapter4 generated datasets comprising legitimate/normal network traffic and six types of DDoS attacks: DNS flood; ICMP flood; UDP flood; TCP SYN flood; Slowloris; and Ping-of-Death (PoD). The original dataset was collected by tcpdump in the experimental environment (see chapter 4). As tcpdump tool was installed on the server, it captured all different type of the network traffic that passed the TCP/IP network model to the target zone (server) (see section 2.5 for detailed explanation). All the collected instances of network traffic consisting all the protocol header information such as IP packet header information namely: (timestamp, tos, ttl, id, offset, flags, protocol type, length, source IP address, destination IP address); TCP packet header information: (source port number, destination port number, flags, Checksum, seq ,ack, win, options, length); ICMP packet header information: (Type of the message, ICMP length, id, seq, checksum); UDP packet header information: (source port, destination port, UDP length and UDP checksum) (see sections 2.5.1 and 2.5.2 for detailed explanations). When instances of network connection utilised a different type of protocol, such as TCP, UDP or ICMP protocol, the relevant network traffic instance also contained the associated packet header information for the relevant protocol type. Figures 5.1 to 5.7 present and explain the different protocol header information of normal traffic, TCP SYN flood, ICMP flood, UDP flood, Slowloris, Pod and DNS flood respectively, to illustrate the structure of the headers of each attack type.

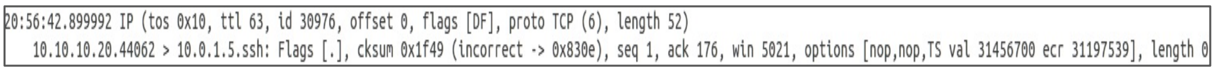
Figure 5.1 presents the screenshot of instance of normal traffic in which, a user sent a request (this request is in the form of the packet) to the server to access the web browser via the establishment of the TCP SYN connection (see section 2.5.5 for detailed description). Therefore, normal instances of the output of the tcpdump data including all the transmitted and received information that derived from the protocol header in packets such as IP packet header and TCP packet header (see sections 2.5.1 and 2.5.2 for the detailed explanation of packet headers information).



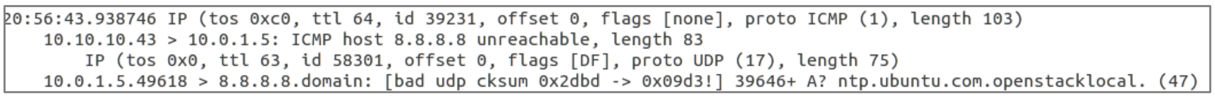
```
12:20:32.600198 IP (tos 0x0, ttl 63, id 47082, offset 0, flags [DF], proto TCP (6), length 52)
   10.10.10.38.50844 > 10.0.1.5.8000: Flags [.], cksum 0x1f5b (incorrect -> 0xf0a5), seq 261, ack 155, win 221, options [nop,nop,TS val 174660163 ecr 174654964], length 0
```

Figure 5.1 Screenshot of the features of IP packet header and TCP packet header in normal traffic

Figure 5.2 presents the screenshot of the features of an IP packet header, a TCP packet header, and an ICMP packet header information. As explained in section 2.5.2, in the TCP SYN attack, the attacker exploits the TCP connection by first sending the frequent SYN packet to the target to establish the connection. Therefore, a tcpdump displays a TCP and IP packet header information in the first section of Figure 5.2. In the second phase of the TCP SYN attack, a server sent the SYN-ACK message to the sender (attacker), and because of the nature of the attack, the server never received the expected ACK message from the attacker. It leaves the connection open and resulting in inaccessibility. Moreover, according to the TCP/IP network model, when the server is unavailable, an ICMP packet sent to the server that the "host is unreachable". Therefore, tcpdump displays the ICMP packet header information and IP packet header information in the second part of tcpdump data in Figure 5.2.



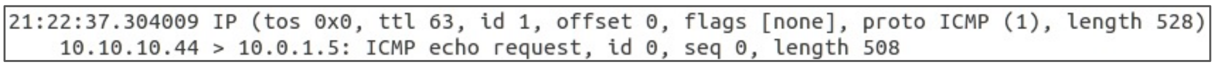
```
20:56:42.899992 IP (tos 0x10, ttl 63, id 30976, offset 0, flags [DF], proto TCP (6), length 52)
   10.10.10.20.44062 > 10.0.1.5.ssh: Flags [.], cksum 0x1f49 (incorrect -> 0x830e), seq 1, ack 176, win 5021, options [nop,nop,TS val 31456700 ecr 31197539], length 0
```



```
20:56:43.938746 IP (tos 0xc0, ttl 64, id 39231, offset 0, flags [none], proto ICMP (1), length 103)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 83
        IP (tos 0x0, ttl 63, id 58301, offset 0, flags [DF], proto UDP (17), length 75)
    10.0.1.5.49618 > 8.8.8.8.domain: [bad udp cksum 0x2dbd -> 0x09d3!] 39646+ A? ntp.ubuntu.com.openstacklocal. (47)
```

Figure 5.2 Screenshot of the features of IP packet header, ICMP packet header and TCP packet header in TCP SYN flood attack

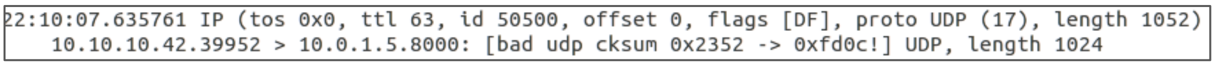
Figure 5.3 presents the screenshot of the features of an IP packet header and an ICMP packet header in the ICMP flood attack. An attacker exploited the ICMP protocol by sending ICMP packets with the "ICMP echo request" message to the server (see section 2.5.1 for detailed explanation). Therefore, the output of the tcpdump in Figure 5.3 presents the information of the IP and ICMP packet header information.



```
21:22:37.304009 IP (tos 0x0, ttl 63, id 1, offset 0, flags [none], proto ICMP (1), length 528)
    10.10.10.44 > 10.0.1.5: ICMP echo request, id 0, seq 0, length 508
```

Figure 5.3 Screenshot of the features of IP packet header and ICMP packet header information in ICMP flood attack

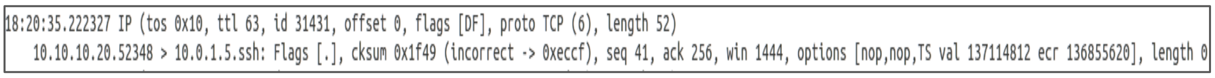
Figure 5.4 presents the screenshot of the features of an IP packet header and an UDP packet header in the UDP flood attack. An attacker exploited the UDP protocol by sending UDP packets to the server (see section 2.5.2 for detailed explanation). Therefore, the output of the tcpdump in Figure 5.4 presents the information of the IP and UDP packet header information.



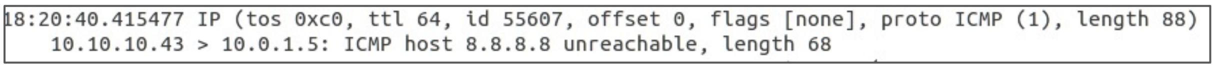
```
22:10:07.635761 IP (tos 0x0, ttl 63, id 50500, offset 0, flags [DF], proto UDP (17), length 1052)
    10.10.10.42.39952 > 10.0.1.5.8000: [bad udp cksum 0x2352 -> 0xfd0c!] UDP, length 1024
```

Figure 5.4 Screenshot of the features of IP packet header and UDP packet header information in UDP flood attack

Figure 5.5 presents the screenshot of the features of an IP packet header, a TCP packet header, and an ICMP packet header information in Slowloris attack. An attacker initiated the attack by sending the legitimate packet to the server to open a connection through a TCP connection (see section 2.5.3 for detailed explanation). Therefore, a tcpdump displays a TCP and IP packet header information in the first section of Figure 5.5. A result of the successful Slowloris attack was unavailability of the server that generated the ICMP packet header “host is unreachable” that can be seen in the second part of Figure 5.5.



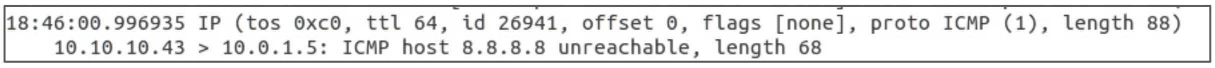
```
18:20:35.222327 IP (tos 0x10, ttl 63, id 31431, offset 0, flags [DF], proto TCP (6), length 52)
   10.10.10.20.52348 > 10.0.1.5.ssh: Flags [.], cksum 0x1f49 (incorrect -> 0xeccf), seq 41, ack 256, win 1444, options [nop,nop,TS val 137114812 ecr 136855620], length 0
```



```
18:20:40.415477 IP (tos 0xc0, ttl 64, id 55607, offset 0, flags [none], proto ICMP (1), length 88)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 68
```

Figure 5.5 Screenshot of the features of IP packet header, ICMP packet header and TCP packet header in Slowloris attack

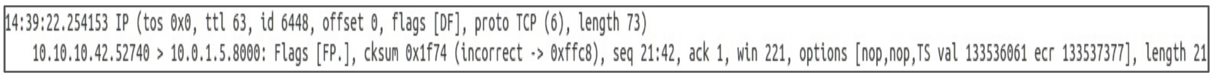
Figure 5.6 presents the screenshot of the features of an IP packet header and an ICMP packet header in the POD attack. An attacker exploited the ICMP protocol by sending ICMP packets with the “ICMP echo request” message to the server (see section 2.5.1 for a detailed explanation). Therefore, the output of the tcpdump in Figure 5.6 presents the information of the IP and ICMP packet header information.



```
18:46:00.996935 IP (tos 0xc0, ttl 64, id 26941, offset 0, flags [none], proto ICMP (1), length 88)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 68
```

Figure 5.6 Screenshot of the features of IP packet header and ICMP packet header in POD attack

Figure 5.7 presents the screenshot of the features of an IP packet header and a UDP packet header in the DNS flood attack. An attacker exploited the UDP protocol by sending UDP packets to the server (see section 2.5.2 for detailed explanation). The output of the tcpdump in Figure 5.4 presents the information of the IP and UDP packet header information.



```
14:39:22.254153 IP (tos 0x0, ttl 63, id 6448, offset 0, flags [DF], proto TCP (6), length 73)
   10.10.10.42.52740 > 10.0.1.5.8000: Flags [FP.], cksum 0x1f74 (incorrect -> 0xffc8), seq 21:42, ack 1, win 221, options [nop,nop,TS val 133536061 ecr 133537377], length 21
```

Figure 5.7 Screenshot of the features of IP packet header and TCP packet header in DNS attack

As Figures 5.1-5.7 illustrate, each type of network traffic offered different protocol header information and so had differing structures, therefore it was necessary to parse all of the datasets to ensure that they were in a suitable/consistent format in order to undertake the classification task. The next section will explain all of the pre-processing steps that were applied to the original dataset.

5.3 Data pre-processing

As mentioned in section 5.1, each type of malicious traffic data and legitimate traffic data had a different structure. As such, it was necessary to define a set of pre-processing tasks to transform the original data to the standard format for the classification task. This section will introduce the different phases of the data pre-processing that were applied to the original dataset to ensure that it was consistent and in a suitable format for the classification task.

Before moving into the detail, Figure 5.8 presents a high-level view of the different pre-processing tasks, showing three main phases.

Phase 1 consisted of parsing the data to extract the required features and designing a Python script to convert the format of the text dataset to a CSV file (see section 5.3.1). Phase 2 consisted of finding the commonalities in the features of all of the datasets (see section 5.3.2). The final phase, Phase 3, consisted of processing the data using specific Python scripts designed for each dataset (normal traffic and each attack type) to calculate specific measures for each IP address in relation to each protocol for each attack type then to concatenate the files into a single dataset before, finally, applying a script to generate datasets based on a range of different time slices (different time intervals) for subsequent analysis and comparison (see section 5.3.3).

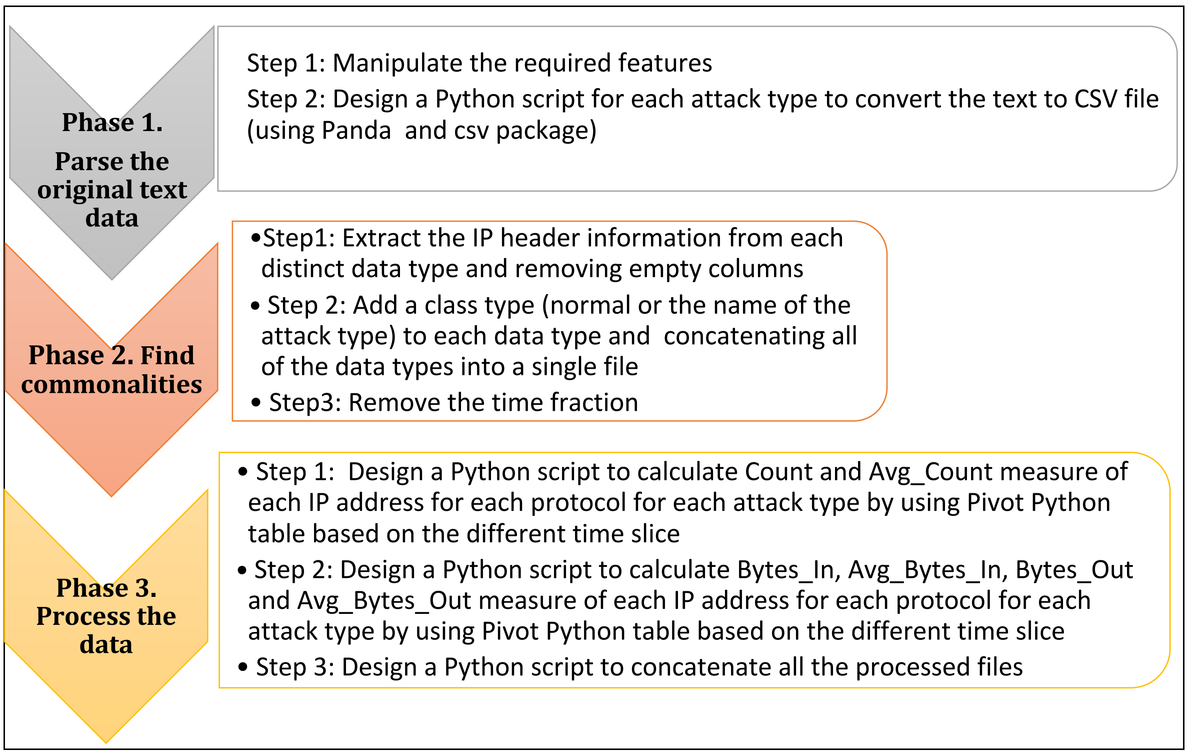
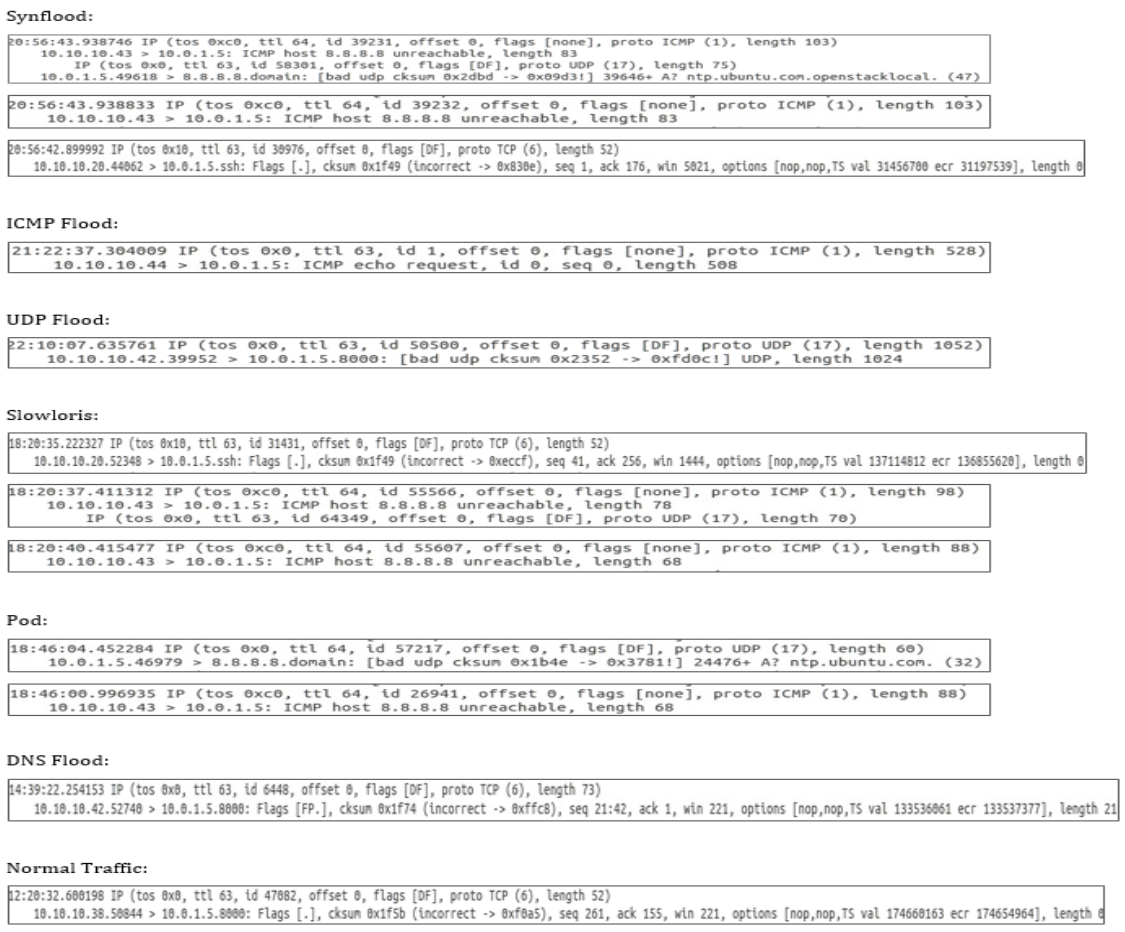

Phase 1.
Parse the original text data
Step 1: Manipulate the required features
Step 2: Design a Python script for each attack type to convert the text to CSV file (using Panda and csv package)
Phase 2. Find commonalities
•Step1: Extract the IP header information from each distinct data type and removing empty columns
• Step 2: Add a class type (normal or the name of the attack type) to each data type and concatenating all of the data types into a single file
• Step3: Remove the time fraction
Phase 3. Process the data
• Step 1: Design a Python script to calculate Count and Avg_Count measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice
• Step 2: Design a Python script to calculate Bytes_In, Avg_Bytes_In, Bytes_Out and Avg_Bytes_Out measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice
• Step 3: Design a Python script to concatenate all the processed files

Figure 5.8 Data pre-processing phases

5.3.1 Phase 1: parsing the original text data

As explained in section 5.2, each malicious and legitimate traffic type had a different structure of features. It was therefore essential to first convert the structure of each dataset into a standard format required to undertake the classification task – this constituted Phase 1 of the pre-processing. This sub-section will explain the pre-processing tasks that were applied to each dataset (one comprising normal network traffic and six comprising data for each of the attack types) to make it suitable for the subsequent classification task.

Figure 5.9 presents an extract from the original datasets illustrating each type of traffic (normal and the different forms of attack traffic). All of the generated datasets were captured in the form of text files. The data records were variable when the attack behaviour changed in terms of the type of protocol that the attack used. This caused each record of the data to have either two, three or four lines of related information (as illustrated in Figures 5.2-5.7). However, this structure of the data was not suitable for use with the classification model. Therefore, it was important to transform each dataset into a suitable form for use by the classification model, such as into the CSV format. A range of tools was available for this data pre-processing task, including (R, 2019). However, it was decided to use Python because of the researcher's existing experience with this programming language.


Synflood:

```
20:56:43.938746 IP (tos 0xc0, ttl 64, id 39231, offset 0, flags [none], proto ICMP (1), length 103)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 83
        IP (tos 0x0, ttl 63, id 58301, offset 0, flags [DF], proto UDP (17), length 75)
    10.0.1.5.49618 > 8.8.8.8.domain: [bad udp cksum 0x2dbd -> 0x09d3!] 39646+ A? ntp.ubuntu.com.openstacklocal. (47)
```

```
20:56:43.938833 IP (tos 0xc0, ttl 64, id 39232, offset 0, flags [none], proto ICMP (1), length 103)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 83
```

```
20:56:42.899992 IP (tos 0x10, ttl 63, id 30976, offset 0, flags [DF], proto TCP (6), length 52)
    10.10.10.20.44062 > 10.0.1.5.ssh: Flags [.], cksum 0x1f49 (incorrect -> 0x830e), seq 1, ack 176, win 5021, options [nop,nop,TS val 31456700 ecr 31197539], length 0
```

ICMP Flood:

```
21:22:37.304009 IP (tos 0x0, ttl 63, id 1, offset 0, flags [none], proto ICMP (1), length 528)
    10.10.10.44 > 10.0.1.5: ICMP echo request, id 0, seq 0, length 508
```

UDP Flood:

```
22:10:07.635761 IP (tos 0x0, ttl 63, id 50500, offset 0, flags [DF], proto UDP (17), length 1052)
    10.10.10.42.39952 > 10.0.1.5.8000: [bad udp cksum 0x2352 -> 0xfd0c!] UDP, length 1024
```

Slowloris:

```
18:20:35.222327 IP (tos 0x10, ttl 63, id 31431, offset 0, flags [DF], proto TCP (6), length 52)
    10.10.10.20.52348 > 10.0.1.5.ssh: Flags [.], cksum 0x1f49 (incorrect -> 0xeccf), seq 41, ack 256, win 1444, options [nop,nop,TS val 137114812 ecr 136855620], length 0
```

```
18:20:37.411312 IP (tos 0xc0, ttl 64, id 55566, offset 0, flags [none], proto ICMP (1), length 98)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 78
        IP (tos 0x0, ttl 63, id 64349, offset 0, flags [DF], proto UDP (17), length 70)
```

```
18:20:40.415477 IP (tos 0xc0, ttl 64, id 55607, offset 0, flags [none], proto ICMP (1), length 88)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 68
```

Pod:

```
18:46:04.452284 IP (tos 0x0, ttl 64, id 57217, offset 0, flags [DF], proto UDP (17), length 60)
    10.0.1.5.46979 > 8.8.8.8.domain: [bad udp cksum 0x1b4e -> 0x3781!] 24476+ A? ntp.ubuntu.com. (32)
```

```
18:46:00.996935 IP (tos 0xc0, ttl 64, id 26941, offset 0, flags [none], proto ICMP (1), length 88)
    10.10.10.43 > 10.0.1.5: ICMP host 8.8.8.8 unreachable, length 68
```

DNS Flood:

```
14:39:22.254153 IP (tos 0x0, ttl 63, id 6448, offset 0, flags [DF], proto TCP (6), length 73)
    10.10.10.42.52740 > 10.0.1.5.8000: Flags [FP.], cksum 0x1f74 (incorrect -> 0xffc8), seq 21:42, ack 1, win 221, options [nop,nop,TS val 133536061 ecr 133537377], length 21
```

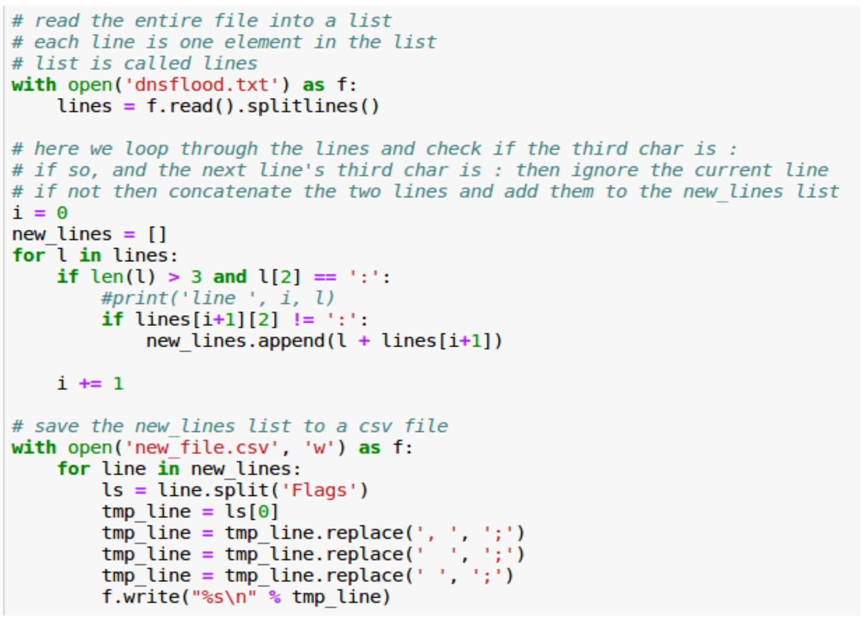
Normal Traffic:

```
12:20:32.600198 IP (tos 0x0, ttl 63, id 47082, offset 0, flags [DF], proto TCP (6), length 52)
    10.10.10.38.50844 > 10.0.1.5.8000: Flags [.], cksum 0x1f5b (incorrect -> 0xf0a5), seq 261, ack 155, win 221, options [nop,nop,TS val 174660163 ecr 174654964], length 0
```

Figure 5.9 An extract of the original dataset

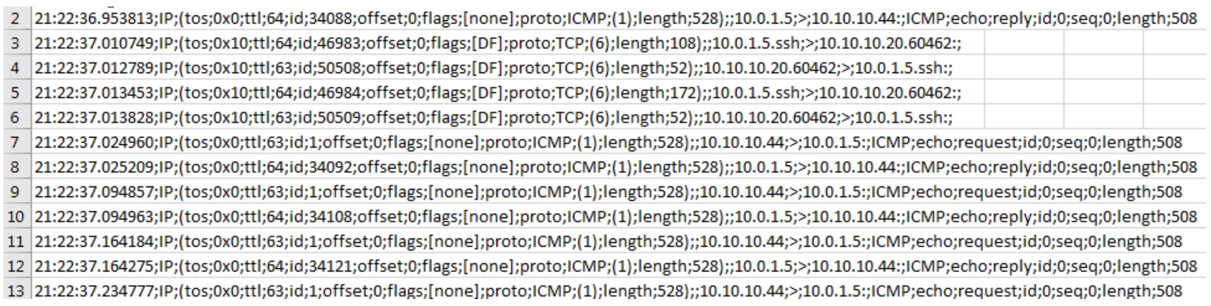
A Python script (see Figure 5.10) was designed to parse each type of text file data. Whenever the script 'observed' more than one line of information for each instance in the dataset, it merged it into a single line of the record. When the parsing of a dataset/text file was complete, the re-formatted data was saved as a CSV format file. Figure 5.11

presents an extract of the result of applying Python code to the original datasets to illustrate the transformation of the original data into a CSV format dataset.



```python
# read the entire file into a list
# each line is one element in the list
# list is called lines
with open('dnsflood.txt') as f:
    lines = f.read().splitlines()

# here we loop through the lines and check if the third char is :
# if so, and the next line's third char is : then ignore the current line
# if not then concatenate the two lines and add them to the new_lines list
i = 0
new_lines = []
for l in lines:
    if len(l) > 3 and l[2] == ':':
        #print('line ', i, l)
        if lines[i+1][2] != ':':
            new_lines.append(l + lines[i+1])

    i += 1

# save the new_lines list to a csv file
with open('new_file.csv', 'w') as f:
    for line in new_lines:
        ls = line.split('Flags')
        tmp_line = ls[0]
        tmp_line = tmp_line.replace(', ', ';')
        tmp_line = tmp_line.replace('  ', ';')
        tmp_line = tmp_line.replace(' ', ';')
        f.write("%s\n" % tmp_line)
```

Figure 5.10 the screenshot of the Python code to parse the original text data to CSV file



```
2  21:22:36.953813;IP;(tos;0x0;ttl;64;id;34088;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.0.1.5;>;10.10.10.44:;ICMP;echo;reply;id;0;seq;0;length;508
3  21:22:37.010749;IP;(tos;0x10;ttl;64;id;46983;offset;0;flags;[DF];proto;TCP;(6);length;108);;10.0.1.5.ssh;>;10.10.10.20.60462:;
4  21:22:37.012789;IP;(tos;0x10;ttl;63;id;50508;offset;0;flags;[DF];proto;TCP;(6);length;52);;10.10.10.20.60462;>;10.0.1.5.ssh:;
5  21:22:37.013453;IP;(tos;0x10;ttl;64;id;46984;offset;0;flags;[DF];proto;TCP;(6);length;172);;10.0.1.5.ssh;>;10.10.10.20.60462:;
6  21:22:37.013828;IP;(tos;0x10;ttl;63;id;50509;offset;0;flags;[DF];proto;TCP;(6);length;52);;10.10.10.20.60462;>;10.0.1.5.ssh:;
7  21:22:37.024960;IP;(tos;0x0;ttl;63;id;1;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.10.10.44;>;10.0.1.5:;ICMP;echo;request;id;0;seq;0;length;508
8  21:22:37.025209;IP;(tos;0x0;ttl;64;id;34092;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.0.1.5;>;10.10.10.44:;ICMP;echo;reply;id;0;seq;0;length;508
9  21:22:37.094857;IP;(tos;0x0;ttl;63;id;1;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.10.10.44;>;10.0.1.5:;ICMP;echo;request;id;0;seq;0;length;508
10 21:22:37.094963;IP;(tos;0x0;ttl;64;id;34108;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.0.1.5;>;10.10.10.44:;ICMP;echo;reply;id;0;seq;0;length;508
11 21:22:37.164184;IP;(tos;0x0;ttl;63;id;1;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.10.10.44;>;10.0.1.5:;ICMP;echo;request;id;0;seq;0;length;508
12 21:22:37.164275;IP;(tos;0x0;ttl;64;id;34121;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.0.1.5;>;10.10.10.44:;ICMP;echo;reply;id;0;seq;0;length;508
13 21:22:37.234777;IP;(tos;0x0;ttl;63;id;1;offset;0;flags;[none];proto;ICMP;(1);length;528);;10.10.10.44;>;10.0.1.5:;ICMP;echo;request;id;0;seq;0;length;508
```

Figure 5.11 An extract of the result of applying Python code to the original datasets, resulting in a CSV formatted dataset

Section 5.3.2 will report Phase 2 of the pre-processing task, which was to find the common features across all of the datasets.

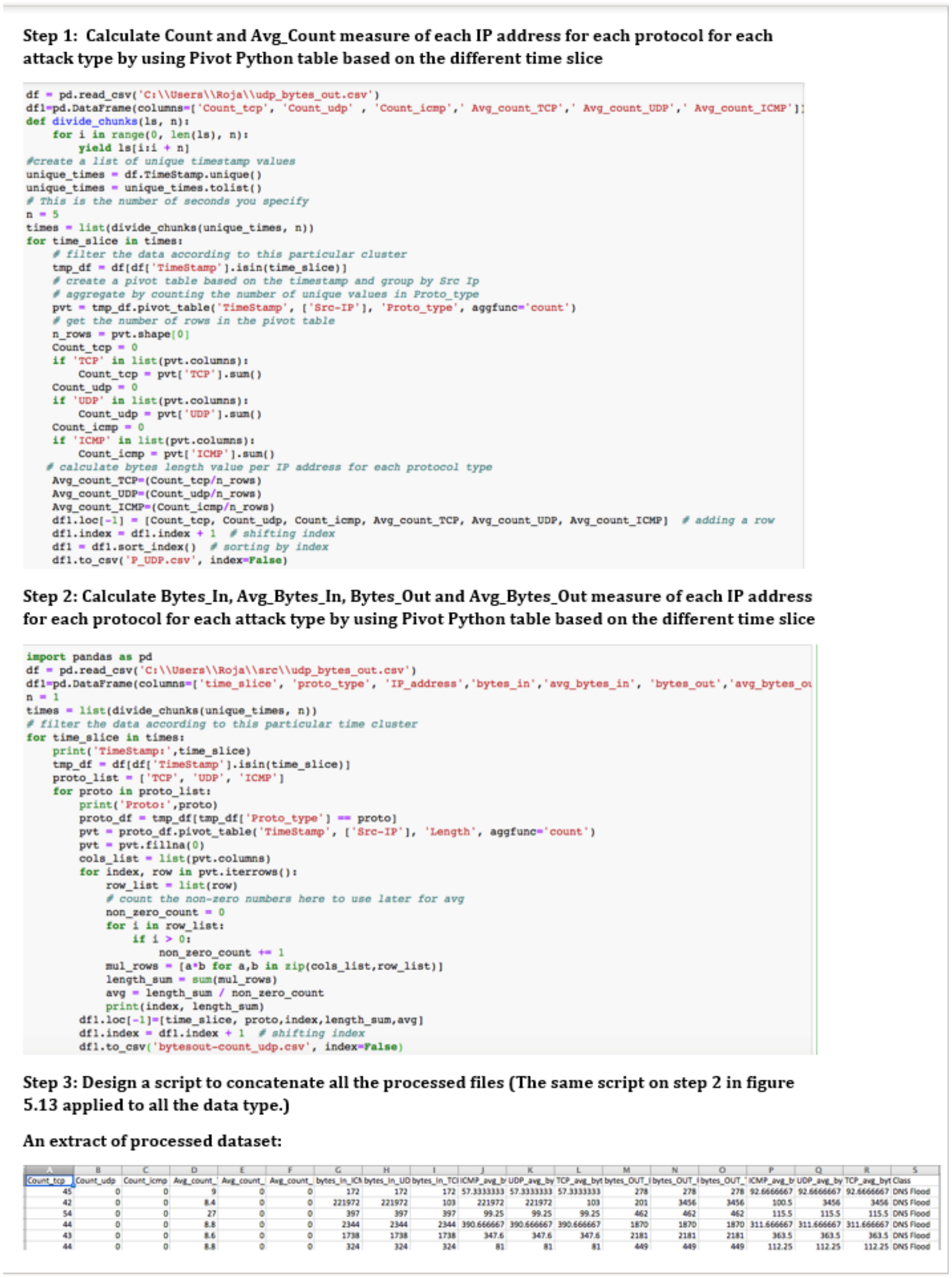
5.3.2 Phase 2: finding commonalities

As described in section 2.5, each malicious and legitimate traffic type had different types of feature. It was therefore essential to identify the commonalities in the features across all of the datasets in order to build the detection model. This section will report the process of finding the common features in the datasets.

In the Phase 2 of the data pre-processing, it was observed that there was a lack of consistent structure in the different attack type data. The data records were variable when the attack behaviour changed in terms of the type of protocol that the attack used. For example, as Figure 5.2 shows, each record in the SYN Flood attack includes IP packet header features, and TCP packet header features, while ICMP flood attacks, as shown in Figure 5.3, obviously exploit the ICMP protocol so generate traffic that contains ICMP packet header features and IP packet header features. (Figures 5.4-5.7 and their accompanying explanations describe the different features of the IP, UDP, ICMP and TCP packet headers in the UDP flood, Slowloris, PoD and DNS flood attacks, respectively).

After careful analysis of the data, a common structure across all of the data types was identified as being contained in the IP packet header features. Therefore, a Python (2019) script was designed and then applied to each of the normal and attack type datasets to extract only the IP packet header features. Figure 5.12 presents each step that was undertaken in Phase 2 of the data pre-processing. In step 1 in Figure 5.12, a Python (2019) script was designed to apply to the data to extract the IP header information from each distinct data type as well as to remove any empty columns across the dataset to which the script was applied. The script also changed the layout of the data by creating, and populating, a CSV spreadsheet comprising columns holding the values for each feature of the IP header information (see step 2 in Figure 5.12). Next, another Python (2019) script was created to add a class type (normal or the name of the attack type) to each data type and then concatenate all of the data types into a single file (see step 2 in Figure 5.12). Finally, it was observed that the time fraction in the 'Time_stamp' column (the first instance of which is highlighted in yellow in the 'output data' of step 1 in Figure 5.12, for ease of identification) was not necessary since only the time itself was required. Therefore, a further Python script was created to remove the time fraction (see step 3 'output data' in Figure 5.12), resulting in the output shown in step 3 in Figure 5.12.

The next subsection will explain the details of the final phase, Phase 3, of data pre-processing.



Step 1: Calculate Count and Avg_Count measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice

```
df = pd.read_csv('C:\\Users\\Roja\\udp_bytes_out.csv')
df1=pd.DataFrame(columns=['Count_tcp', 'Count_udp' , 'Count_icmp',' Avg_count_TCP',' Avg_count_UDP',' Avg_count_ICMP'])
def divide_chunks(ls, n):
    for i in range(0, len(ls), n):
        yield ls[i:i + n]
#create a list of unique timestamp values
unique_times = df.TimeStamp.unique()
unique_times = unique_times.tolist()
# This is the number of seconds you specify
n = 5
times = list(divide_chunks(unique_times, n))
for time_slice in times:
    # filter the data according to this particular cluster
    tmp_df = df[df['TimeStamp'].isin(time_slice)]
    # create a pivot table based on the timestamp and group by Src Ip
    # aggregate by counting the number of unique values in Proto_type
    pvt = tmp_df.pivot_table('TimeStamp', ['Src-IP'], 'Proto_type', aggfunc='count')
    # get the number of rows in the pivot table
    n_rows = pvt.shape[0]
    Count_tcp = 0
    if 'TCP' in list(pvt.columns):
        Count_tcp = pvt['TCP'].sum()
    Count_udp = 0
    if 'UDP' in list(pvt.columns):
        Count_udp = pvt['UDP'].sum()
    Count_icmp = 0
    if 'ICMP' in list(pvt.columns):
        Count_icmp = pvt['ICMP'].sum()
   # calculate bytes length value per IP address for each protocol type
    Avg_count_TCP=(Count_tcp/n_rows)
    Avg_count_UDP=(Count_udp/n_rows)
    Avg_count_ICMP=(Count_icmp/n_rows)
    df1.loc[-1] = [Count_tcp, Count_udp, Count_icmp, Avg_count_TCP, Avg_count_UDP, Avg_count_ICMP]  # adding a row
    df1.index = df1.index + 1  # shifting index
    df1 = df1.sort_index()  # sorting by index
    df1.to_csv('P_UDP.csv', index=False)
```

Step 2: Calculate Bytes_In, Avg_Bytes_In, Bytes_Out and Avg_Bytes_Out measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice

```
import pandas as pd
df = pd.read_csv('C:\\Users\\Roja\\src\\udp_bytes_out.csv')
df1=pd.DataFrame(columns=['time_slice', 'proto_type', 'IP_address','bytes_in','avg_bytes_in', 'bytes_out','avg_bytes_ou
n = 1
times = list(divide_chunks(unique_times, n))
# filter the data according to this particular time cluster
for time_slice in times:
    print('TimeStamp:',time_slice)
    tmp_df = df[df['TimeStamp'].isin(time_slice)]
    proto_list = ['TCP', 'UDP', 'ICMP']
    for proto in proto_list:
        print('Proto:',proto)
        proto_df = tmp_df[tmp_df['Proto_type'] == proto]
        pvt = proto_df.pivot_table('TimeStamp', ['Src-IP'], 'Length', aggfunc='count')
        pvt = pvt.fillna(0)
        cols_list = list(pvt.columns)
        for index, row in pvt.iterrows():
            row_list = list(row)
            # count the non-zero numbers here to use later for avg
            non_zero_count = 0
            for i in row_list:
                if i > 0:
                    non_zero_count += 1
            mul_rows = [a*b for a,b in zip(cols_list,row_list)]
            length_sum = sum(mul_rows)
            avg = length_sum / non_zero_count
            print(index, length_sum)
        df1.loc[-1]=[time_slice, proto,index,length_sum,avg]
        df1.index = df1.index + 1  # shifting index
        df1.to_csv('bytesout-count_udp.csv', index=False)
```

Step 3: Design a script to concatenate all the processed files (The same script on step 2 in figure 5.13 applied to all the data type.)

An extract of processed dataset:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
Count_tcp	Count_udp	Count_icmp	Avg_count_	Avg_count_	Avg_count_	bytes_in_ICN	bytes_in_UD	bytes_in_TCI	ICMP_avg_b	UDP_avg_by	TCP_avg_byt	bytes_OUT_I	bytes_OUT_I	bytes_OUT_T	ICMP_avg_b	UDP_avg_by	TCP_avg_byt	Class
45	0	0	9	0	0	172	172	172	57.3333333	57.3333333	57.3333333	278	278	278	92.6666667	92.6666667	92.6666667	DNS Flood
42	0	0	8.4	0	0	221972	221972	103	221972	221972	103	201	3456	3456	100.5	3456	3456	DNS Flood
54	0	0	27	0	0	397	397	397	99.25	99.25	99.25	462	462	462	115.5	115.5	115.5	DNS Flood
44	0	0	8.8	0	0	2344	2344	2344	390.666667	390.666667	390.666667	1870	1870	1870	311.666667	311.666667	311.666667	DNS Flood
43	0	0	8.6	0	0	1738	1738	1738	347.6	347.6	347.6	2181	2181	2181	363.5	363.5	363.5	DNS Flood
44	0	0	8.8	0	0	324	324	324	81	81	81	449	449	449	112.25	112.25	112.25	DNS Flood

Figure 5.12 The scripts and outputs representing Phase 2 of the data pre-processing

5.3.3 Phase 3: processing the data

As explained in section 5.2, in order to calculate specific measures per IP address for each protocol on the basis of each attack type in the different time frames required, and then concatenate the files into a single dataset, it was necessary to design a suitable script. Running the script would then generate the dataset for a specified time frame. This subsection will present the process of assessing specific measures extracted from the attributes of the IP packet header dataset.

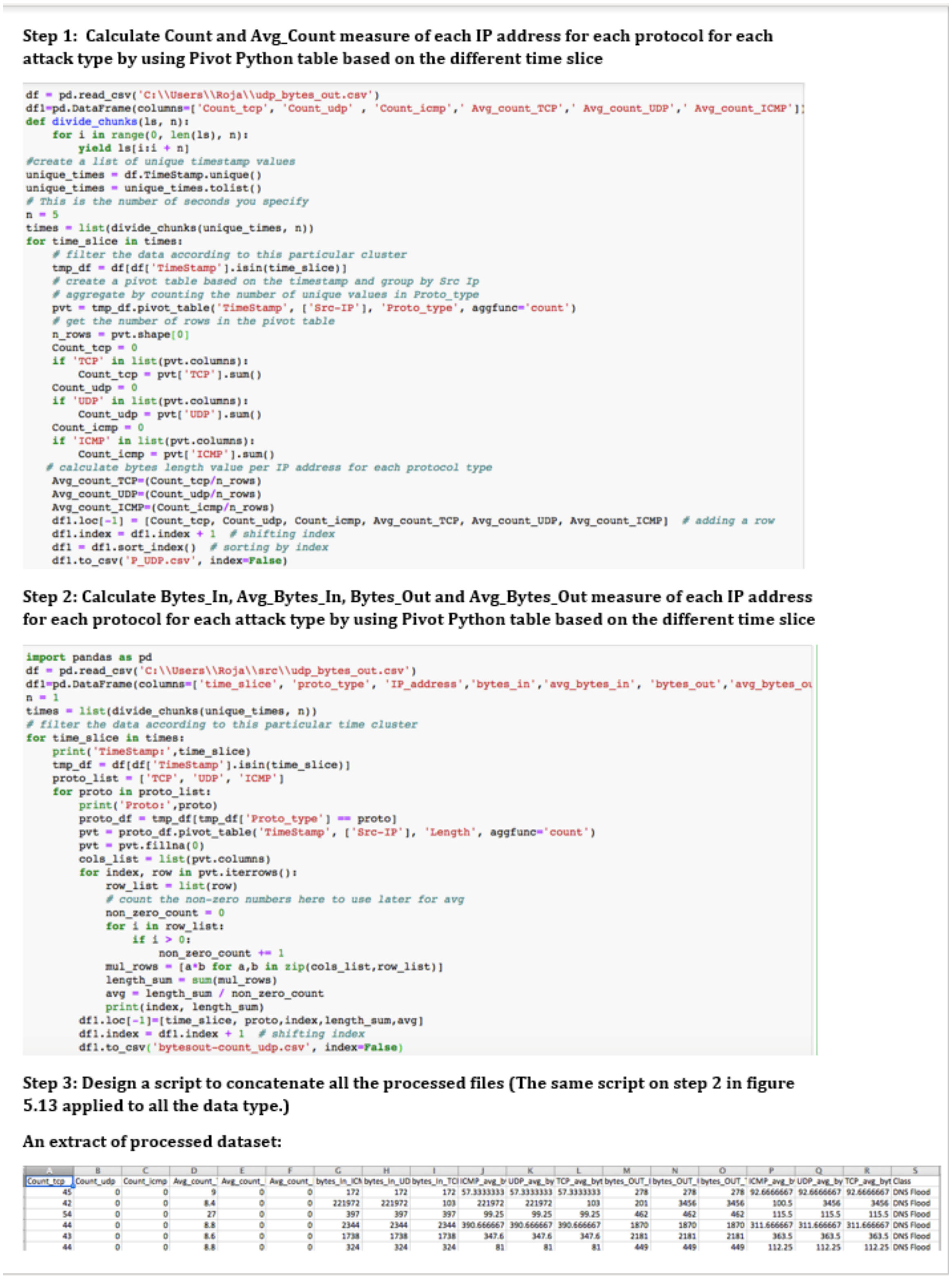
This research used the approach that was applied to the CIDD dataset to calculate a specific measure for each instance in the data files. All of these measures, or features, are presented in Table 5.1.

Features	**Description**
Count	Number of occurrences for an incoming IP address for each of the protocols (TCP, UDP, ICMP)
Avg_Count	Average number of occurrences for an incoming IP address for each of the protocols (TCP, UDP, ICMP)
Bytes_In	Number of bytes received per incoming IP for each of the protocols (TCP, UDP, ICMP)
Avg_Bytes_In	Average number of bytes received per incoming IP address for each of the protocols (TCP, UDP, ICMP)
Bytes_Out	Number of bytes sent per incoming IP for each of the protocols (TCP, UDP, ICMP)
Avg_Bytes_Out	Average number of bytes sent per incoming IP address for each of the protocols (TCP, UDP, ICMP)

Table 5.1 Time-based traffic flow features of CIDD dataset

Table 5.1 shows a description of the time-based traffic flow features of CIDD dataset, which include: Count; Avg_Count; Bytes_In; Avg_Bytes_In; Bytes_Out; and Avg_Bytes_Out. Each of these features was calculated from the relevant IP packet header fields in a specified time interval (this research created datasets for intervals of 1, 3, 5, 7 and 10 seconds – see section 5.3.3 for the justification of these choices). The designed script in the CIDD research work calculated all of the above features in Table 5.1 per incoming IP address for each type of protocol. The Count and Avg_Count measures refer respectively to the number of occurrences for an incoming IP address for each of the protocol types and the average number of occurrences for an incoming IP addresses for each of the protocol types in the specified time interval. Bytes_In and Avg_Bytes_In correspond respectively to the total number of bytes and the average number of bytes received per incoming IP address for each of the protocol types in the specified time interval. Bytes_Out and Avg_Bytes_Out correspond respectively to total the number of bytes and the average number of bytes sent per incoming IP address for each of the protocol types in the specified time interval.

As mentioned earlier, this research used a Python (2019) script using the Python pivot table function to calculate all of the specific measures for each dataset. It then applied the Python (2019) script to the datasets to concatenate all of the files into one single dataset. Figure 5.13 presents each of the three steps that were undertaken in Phase 3 of the data pre-processing.



Step 1: Calculate Count and Avg_Count measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice

```python
df = pd.read_csv('C:\\Users\\Roja\\udp_bytes_out.csv')
df1=pd.DataFrame(columns=['Count_tcp', 'Count_udp' , 'Count_icmp',' Avg_count_TCP',' Avg_count_UDP',' Avg_count_ICMP'])
def divide_chunks(ls, n):
    for i in range(0, len(ls), n):
        yield ls[i:i + n]
#create a list of unique timestamp values
unique_times = df.TimeStamp.unique()
unique_times = unique_times.tolist()
# This is the number of seconds you specify
n = 5
times = list(divide_chunks(unique_times, n))
for time_slice in times:
    # filter the data according to this particular cluster
    tmp_df = df[df['TimeStamp'].isin(time_slice)]
    # create a pivot table based on the timestamp and group by Src Ip
    # aggregate by counting the number of unique values in Proto_type
    pvt = tmp_df.pivot_table('TimeStamp', ['Src-IP'], 'Proto_type', aggfunc='count')
    # get the number of rows in the pivot table
    n_rows = pvt.shape[0]
    Count_tcp = 0
    if 'TCP' in list(pvt.columns):
        Count_tcp = pvt['TCP'].sum()
    Count_udp = 0
    if 'UDP' in list(pvt.columns):
        Count_udp = pvt['UDP'].sum()
    Count_icmp = 0
    if 'ICMP' in list(pvt.columns):
        Count_icmp = pvt['ICMP'].sum()
   # calculate bytes length value per IP address for each protocol type
    Avg_count_TCP=(Count_tcp/n_rows)
    Avg_count_UDP=(Count_udp/n_rows)
    Avg_count_ICMP=(Count_icmp/n_rows)
    df1.loc[-1] = [Count_tcp, Count_udp, Count_icmp, Avg_count_TCP, Avg_count_UDP, Avg_count_ICMP]  # adding a row
    df1.index = df1.index + 1  # shifting index
    df1 = df1.sort_index()  # sorting by index
    df1.to_csv('P_UDP.csv', index=False)
```

Step 2: Calculate Bytes_In, Avg_Bytes_In, Bytes_Out and Avg_Bytes_Out measure of each IP address for each protocol for each attack type by using Pivot Python table based on the different time slice

```python
import pandas as pd
df = pd.read_csv('C:\\Users\\Roja\\src\\udp_bytes_out.csv')
df1=pd.DataFrame(columns=['time_slice', 'proto_type', 'IP_address','bytes_in','avg_bytes_in', 'bytes_out','avg_bytes_ou
n = 1
times = list(divide_chunks(unique_times, n))
# filter the data according to this particular time cluster
for time_slice in times:
    print('TimeStamp:',time_slice)
    tmp_df = df[df['TimeStamp'].isin(time_slice)]
    proto_list = ['TCP', 'UDP', 'ICMP']
    for proto in proto_list:
        print('Proto:',proto)
        proto_df = tmp_df[tmp_df['Proto_type'] == proto]
        pvt = proto_df.pivot_table('TimeStamp', ['Src-IP'], 'Length', aggfunc='count')
        pvt = pvt.fillna(0)
        cols_list = list(pvt.columns)
        for index, row in pvt.iterrows():
            row_list = list(row)
            # count the non-zero numbers here to use later for avg
            non_zero_count = 0
            for i in row_list:
                if i > 0:
                    non_zero_count += 1
            mul_rows = [a*b for a,b in zip(cols_list,row_list)]
            length_sum = sum(mul_rows)
            avg = length_sum / non_zero_count
            print(index, length_sum)
        df1.loc[-1]=[time_slice, proto,index,length_sum,avg]
        df1.index = df1.index + 1  # shifting index
        df1.to_csv('bytesout-count_udp.csv', index=False)
```

Step 3: Design a script to concatenate all the processed files (The same script on step 2 in figure 5.13 applied to all the data type.)

An extract of processed dataset:

A	B	C	D	E	F	G	H	I	J
Count_tcp	Count_udp	Count_icmp	Avg_count_	Avg_count_	Avg_count_	bytes_In_ICN	bytes_In_UD	bytes_In_TCI	ICMP_avg_b
45	0	0	9	0	0	172	172	172	57.3333333
42	0	0	8.4	0	0	221972	221972	103	221972
54	0	0	27	0	0	397	397	397	99.25
44	0	0	8.8	0	0	2344	2344	2344	390.666667
43	0	0	8.6	0	0	1738	1738	1738	347.6
44	0	0	8.8	0	0	324	324	324	81

K	L	M	N	O	P	Q	R	S
UDP_avg_by	TCP_avg_byt	bytes_OUT_I	bytes_OUT_I	bytes_OUT_T	ICMP_avg_b	UDP_avg_by	TCP_avg_byt	Class
57.3333333	57.3333333	278	278	278	92.6666667	92.6666667	92.6666667	DNS Flood
221972	103	201	3456	3456	100.5	3456	3456	DNS Flood
99.25	99.25	462	462	462	115.5	115.5	115.5	DNS Flood
390.666667	390.666667	1870	1870	1870	311.666667	311.666667	311.666667	DNS Flood
347.6	347.6	2181	2181	2181	363.5	363.5	363.5	DNS Flood
81	81	449	449	449	112.25	112.25	112.25	DNS Flood

Figure 5.13 The scripts and outputs representing Phase 3 of the data pre-processing

As this research aimed to analyse the behaviour of the detection model applied to different 'time slices', it was essential to design a script to generate datasets based on specific, defined time intervals. Therefore, in step 1 of Phase 3 (see Figure 5.13), a Python (2019) script using the Python pivot table function was designed to calculate the statistical measures of Count and Avg_count (see Table 5.1) for each protocol for each data type (normal and attack type) based on a specified 'time slice'. Next, another piece of Python (2019) code using the Python pivot table function was created to calculate the Bytes_In, Avg_Bytes_In, Bytes_Out and Avg_Bytes_Out features (see Table 5.1) for each protocol for each data type (see step 2 in Figure 5.14) for the same 'time slice'. Finally, a third piece of Python (2019) code, which was similar to that used in step 2, was applied to the all of the features to concatenate each of the generated files into a single dataset. The output from step 2 was a processed dataset (an extract of which is presented under step 3 in Figure 5.13) that comprises values for each of the features in Table 5.1 calculated for the specified 'time' slice'.

The following time intervals/slices were chosen: 1 second; 3 seconds; 5 seconds; 7 seconds and 10 seconds. The 3 seconds interval mirrored that used in the work by Kumar at al. (2016). The 1 second interval was chosen as the lower bound to explore whether a shorter period than that represented in Kumar et al. (2016) gave the same, better or worse results. Longer periods, up to 10 seconds, were used to explore whether there were changes as the time interval increased and to give a range of datasets through which any changes in detection accuracy could be explored.

The next section will present the results of the classification/detection model on the processed datasets for each of the chosen time intervals.

5.4 The Classification/Detection Models

A similar approach to that reported in sections 3.4.2-3.4.5 was applied to the processed datasets to develop the classification/detection model. However, in addition to using the Naïve Bayes classifier on the processed data, it was decided to assess the accuracy performance of other classification algorithms on the processed dataset to determine whether it would be useful, in terms of improved accuracy and allowing comparison with

Naïve Bayes, to adopt an additional classifier. As a result, the Random Forest algorithm was adopted as it was found to give higher prediction accuracy when compared to other classifiers that were appraised. Consequently, it was seen as being a suitable choice as a second classifier. The Random Forest algorithm will be introduced in the next section before moving on to report the analysis, using confusion matrices, of the predicted outcomes of the two classification models for each of the chosen time intervals.

5.4.1 The Random Forest classifier

The Random Forest approach will now be briefly explained to show the operation of the algorithm in classifying the processed dataset. The Random Forest classifier is a popular supervised machine learning algorithm. This algorithm uses the Bootstrap aggregation ensemble method to solve the classification problem. This method creates the entire 'forest' with uncorrelated decision tree models and then aggregates several decision tree prediction models to enhance the robustness of a single prediction model. The random forest creates a forest by randomly splitting the features (in the case of this research Count, Avg_Count, Bytes_In, Avg_Bytes_In, Bytes_Out, and Avg_Bytes_Out) of the training data (in this case the processed dataset described in section 5.3) into random subsamples. The algorithm then predicts the class label (in our case UDP flood, Slowloris, PoD, DNS flood, TCP SYN, ICMP or Normal) based on the majority 'vote' (or outcome) over all of the decision trees.

As was the case in chapter 3 and also for the application of Naïve Bayes in this part of the study, the classification was undertaken using Weka (Frank et al. 2016) (see section 3.4.4). The section will present the analysis results of the two classifiers (Naïve Bayes and Random Forest) for each of the chosen time intervals.

5.5 Analysis of the Results of Applying the Naïve Bayes and Random Forest Classification Models to the Processed Dataset

This section will report the results of applying the two classification models (Naïve Bayes and Random Forest) trained on the processed data and the evaluation of the models by stratified ten-fold cross-validation (see section 3.4.5).

Weka breaks down the results into a numeric prediction and a confusion matrix. The numeric prediction includes: the Kappa statistic metric that compares the observed accuracy with the expected accuracy of the data: the mean absolute error which measures the average square of the errors; the root mean square, which calculates the square root between the predicated value and actual value; the relative absolute error and the root-relative absolute error, which are, respectively, the mean absolute error and root mean absolute error divided by the corresponding error of the classifier on the dataset that estimated the prior probability of the actual instances.

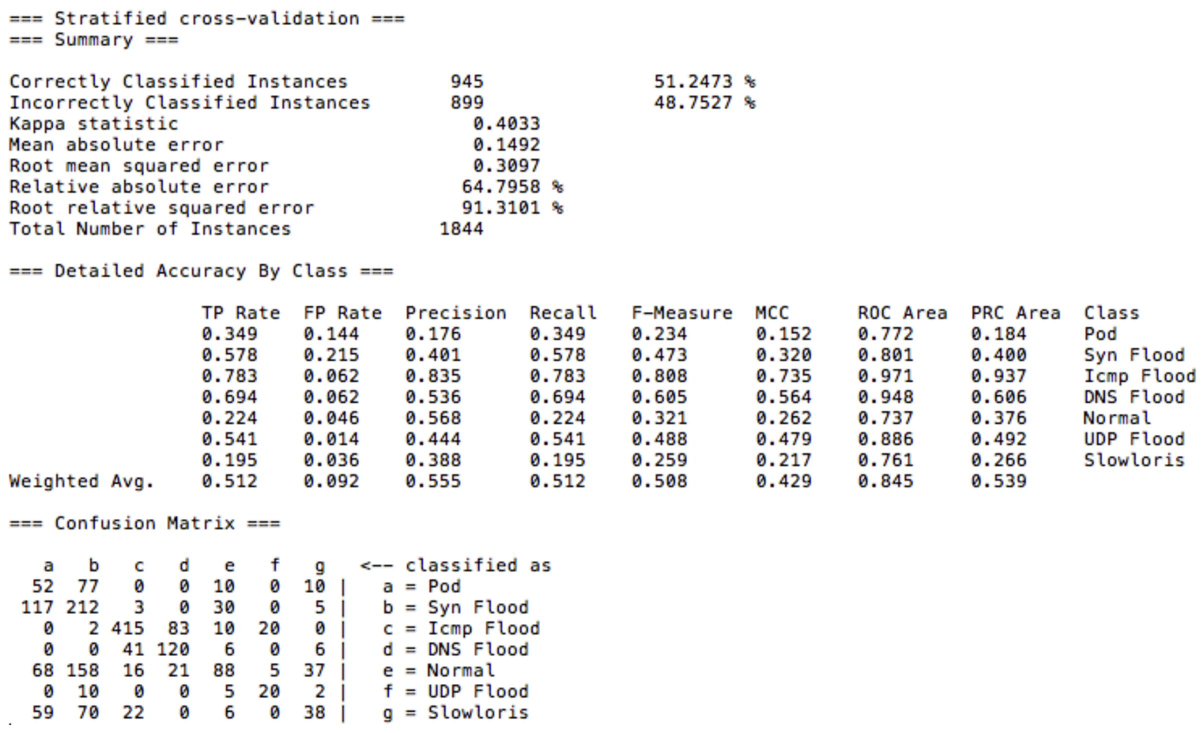
The 'detailed accuracy by class' part of the cross-validation presents several measures: TP Rate, the rate of true positives (TP), defined as instances correctly classified as a given class; FP Rate the rate of false positives (FP), defined as instances falsely classified as a given class; Precision, which is the proportion of instances that are truly of a class (i.e., TP) divided by the total instances classified as that class (i.e., TP+FP); Recall, which is the proportion of instances classified as a given class divided by the real total in that class; F-Measure, which is combined measure for Precision and Recall (calculated as 2 x Precision x Recall / (Precision + Recall)); MCC, which is a balanced measure of the quality of binary (two-class) classifications taking into account true and false positives (TP and FP) and negatives (TN and FN); ROC (Receiver Operating Characteristics) area measurement, which provides a measure of how the classifiers are performing in general; and PRC (Precision Recall) area, which is used to evaluate binary classifiers in imbalanced datasets.

Though these are important measures, the analysis presented in this chapter is based on the confusion matrices themselves. As such, sections 5.5.1-5.5.5 will explain and compare the predicted outcomes of the classification models versus the actual outcomes using confusion matrices (see section 3.4.5) for each of the chosen time intervals of 1, 3, 5, 7 and 10 seconds, respectively. The generated matrices in Figure 5.14-5.24 show the number of correctly and incorrectly predicted instances of each class (attack types and normal traffic), providing insights into the errors being made by two different classification models for each chosen time interval dataset. If these numbers are high for all rows/attack types, it demonstrates a good classification outcome. (The number for instances in a row varies in some cases between the Naïve Bayes and Random Forest classifiers in a specific time interval – i.e., the sums of comparable rows may be different between the classifiers – as a result of resampling of the data to create more balanced datasets).

5.5.1 Predicted versus actual outcomes for the 1 second time interval

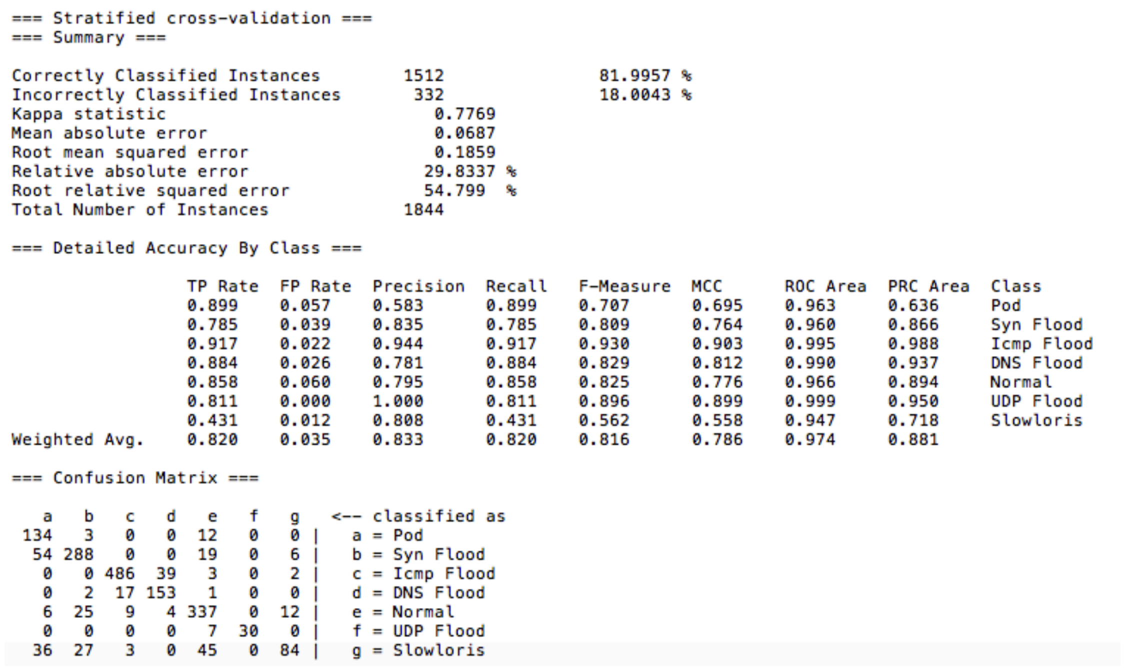
Figures 5.14 and 5.15 present the classification results arising from applying the Naïve Bayes and Random Forest models, respectively, to the processed dataset for the 1 second time interval.

In general, the overall accuracy of the Naïve Bayes classifier is 52.2473 % over the total number of 1844 instances of attack and normal traffic. The general accuracy of the Random Forest classifier is 81.9957% over the total number of 1844 instances, suggesting that the Random Forest classifier has a much higher overall accuracy than the Naïve Bayes classifier. The specific performance of the two models for each attack type, and for normal traffic, for the 1 second time interval dataset will now be discussed.



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         945               51.2473 %
Incorrectly Classified Instances       899               48.7527 %
Kappa statistic                          0.4033
Mean absolute error                      0.1492
Root mean squared error                  0.3097
Relative absolute error                 64.7958 %
Root relative squared error             91.3101 %
Total Number of Instances             1844

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.349    0.144    0.176      0.349    0.234      0.152    0.772     0.184     Pod
                 0.578    0.215    0.401      0.578    0.473      0.320    0.801     0.400     Syn Flood
                 0.783    0.062    0.835      0.783    0.808      0.735    0.971     0.937     Icmp Flood
                 0.694    0.062    0.536      0.694    0.605      0.564    0.948     0.606     DNS Flood
                 0.224    0.046    0.568      0.224    0.321      0.262    0.737     0.376     Normal
                 0.541    0.014    0.444      0.541    0.488      0.479    0.886     0.492     UDP Flood
                 0.195    0.036    0.388      0.195    0.259      0.217    0.761     0.266     Slowloris
Weighted Avg.    0.512    0.092    0.555      0.512    0.508      0.429    0.845     0.539

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
  52  77   0   0  10   0  10 |   a = Pod
 117 212   3   0  30   0   5 |   b = Syn Flood
   0   2 415  83  10  20   0 |   c = Icmp Flood
   0   0  41 120   6   0   6 |   d = DNS Flood
  68 158  16  21  88   5  37 |   e = Normal
   0  10   0   0   5  20   2 |   f = UDP Flood
  59  70  22   0   6   0  38 |   g = Slowloris
```

Figure 5.14 The classification result of the Naïve Bayes model for the 1 second time interval dataset



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1512               81.9957 %
Incorrectly Classified Instances       332               18.0043 %
Kappa statistic                          0.7769
Mean absolute error                      0.0687
Root mean squared error                  0.1859
Relative absolute error                 29.8337 %
Root relative squared error             54.799  %
Total Number of Instances             1844

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.899    0.057    0.583      0.899    0.707      0.695    0.963     0.636     Pod
                 0.785    0.039    0.835      0.785    0.809      0.764    0.960     0.866     Syn Flood
                 0.917    0.022    0.944      0.917    0.930      0.903    0.995     0.988     Icmp Flood
                 0.884    0.026    0.781      0.884    0.829      0.812    0.990     0.937     DNS Flood
                 0.858    0.060    0.795      0.858    0.825      0.776    0.966     0.894     Normal
                 0.811    0.000    1.000      0.811    0.896      0.899    0.999     0.950     UDP Flood
                 0.431    0.012    0.808      0.431    0.562      0.558    0.947     0.718     Slowloris
Weighted Avg.    0.820    0.035    0.833      0.820    0.816      0.786    0.974     0.881

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
 134   3   0   0  12   0   0 |   a = Pod
  54 288   0   0  19   0   6 |   b = Syn Flood
   0   0 486  39   3   0   2 |   c = Icmp Flood
   0   2  17 153   1   0   0 |   d = DNS Flood
   6  25   9   4 337   0  12 |   e = Normal
   0   0   0   0   7  30   0 |   f = UDP Flood
  36  27   3   0  45   0  84 |   g = Slowloris
```

Figure 5.15 The classification result of the Random Forest model for the 1 second time interval dataset

The instances on the main diagonal (top left to bottom right) in Figure 5.14 and Figure 5.15 indicate the correct prediction (i.e., where each attack type from the dataset is correctly predicted by the model). If these numbers are high for all rows/attack types, it demonstrates a good classification outcome.

As can be seen in Figure 5.14, for the 1 second time interval, out of 149 PoD instances, there are 52 cases being classified correctly by the Naïve Bayes classifier. 97 cases are classified incorrectly as Slowloris (10) and SYN Flood (77) and Normal traffic (10) respectively. Similarly, there are 134 cases are being classified correctly by the Random Forest classifier. 15 PoD cases are classified incorrectly as SYN Flood (3) and Normal traffic (12).

Moving on to the SYN Flood attack types, out of the 364 instances, 212 cases are classified correctly by the Naïve Bayes classifier. 152 cases are incorrectly classified as being Slowloris (5), PoD (117), ICMP flood (3), and Normal (30) by the Naïve Bayes classifier. In contrast, 288 cases are classified correctly by the Random Forest classifier. There are 76 SYN Flood cases classified incorrectly as Slowloris (6), PoD (54), and Normal (19) traffic.

Of the 530 ICMP Flood instances, 415 cases are classified correctly by the Naïve Bayes classifier. 115 cases are classified incorrectly as SYN Flood (2), DNS Flood (83), Normal (10), and UDP Flood (20) by the Naïve Bayes classifier. Similarly, there are only 486 cases are being classified correctly by the Random forest. 44 ICMP Flood cases classified incorrectly as Slowloris (2), DNS Flood (39) and Normal traffic (3).

Moving on to the DNS Flood instances, out of 173 instances, 120 cases are classified correctly by the Naïve Bayes classifier. 53 cases are classified incorrectly as a Slowloris (6), ICMP flood (41), and Normal traffic (6) respectively by the Naïve Bayes classifier. Similarly, there are 153 cases classified correctly by the Random Forest classifier and only 20 cases classified incorrectly (as SYN Flood (2), ICMP Flood (17) and Normal cases (1)).

Out of 393 Normal instances, there are only 88 cases classified correctly by the Naïve Bayes classifier. 305 cases are classified incorrectly as Slowloris (37), PoD (68), SYN flood (158), ICMP Flood (16), DNS Flood (21) and UDP Flood (5) by the Naïve Bayes classifier. In contrast, 337 cases are classified correctly by the Random Forest classifier, with 56 cases are being incorrectly classified as Slowloris (12), PoD (6), SYN flood (25), ICMP Flood (9), and DNS Flood (4).

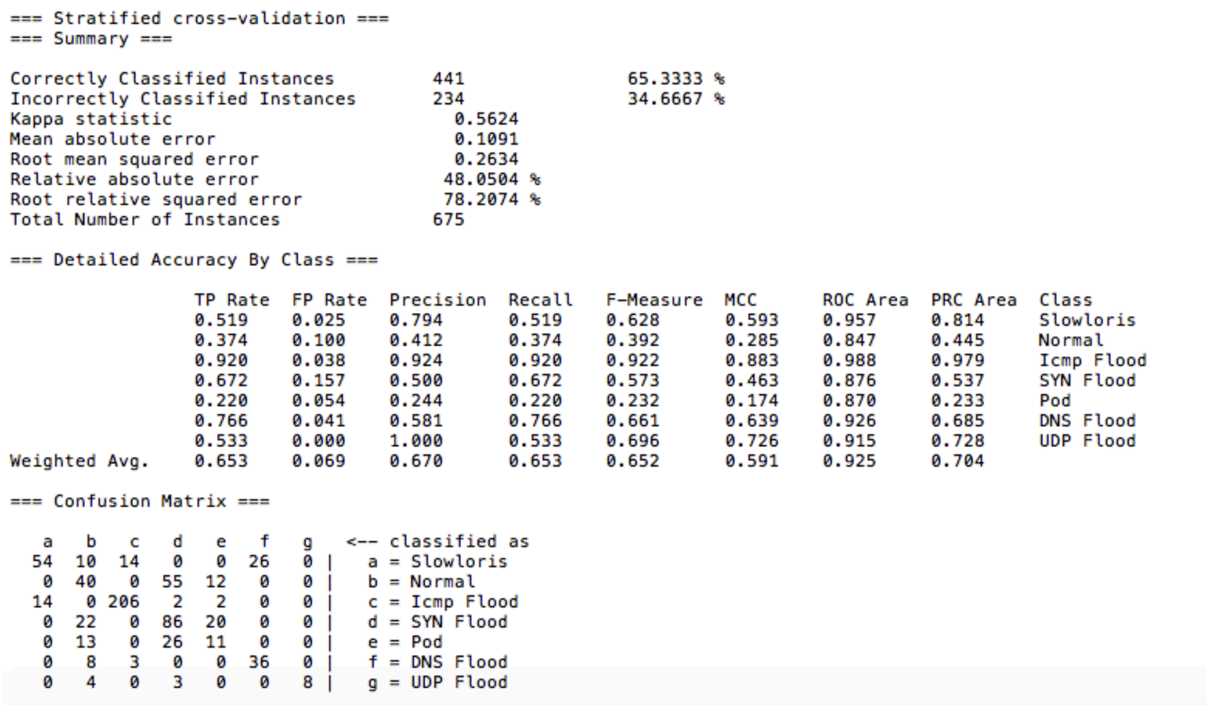
Out of the 37 cases of UDP flood, 20 cases are classified correctly by the Naïve Bayes classifier, with 17 cases classified incorrectly, as SYN Flood (10), Slowloris (2), and Normal traffic (5). Similarly, 30 cases are classified correctly by the Random Forest classifier, with only 7 cases classified incorrectly (all as Normal traffic).

Out of 195 Slowloris instances, 38 cases are classified correctly by the Naïve Bayes classifier. 157 cases are classified incorrectly as PoD (59 instances), SYN Flood (70), ICMP Flood (22), and DNS Flood (6) by the Naïve Bayes classifier. In contrast, in Figure 5.16, there are 84 classified correctly. Only 111 cases are classified incorrectly as a PoD (36), SYN Flood (27), and ICMP Flood (3) by the Random Forest classifier (45 instances were classified as being normal traffic).

5.5.2 Predicted versus actual outcomes for the 3 second time interval

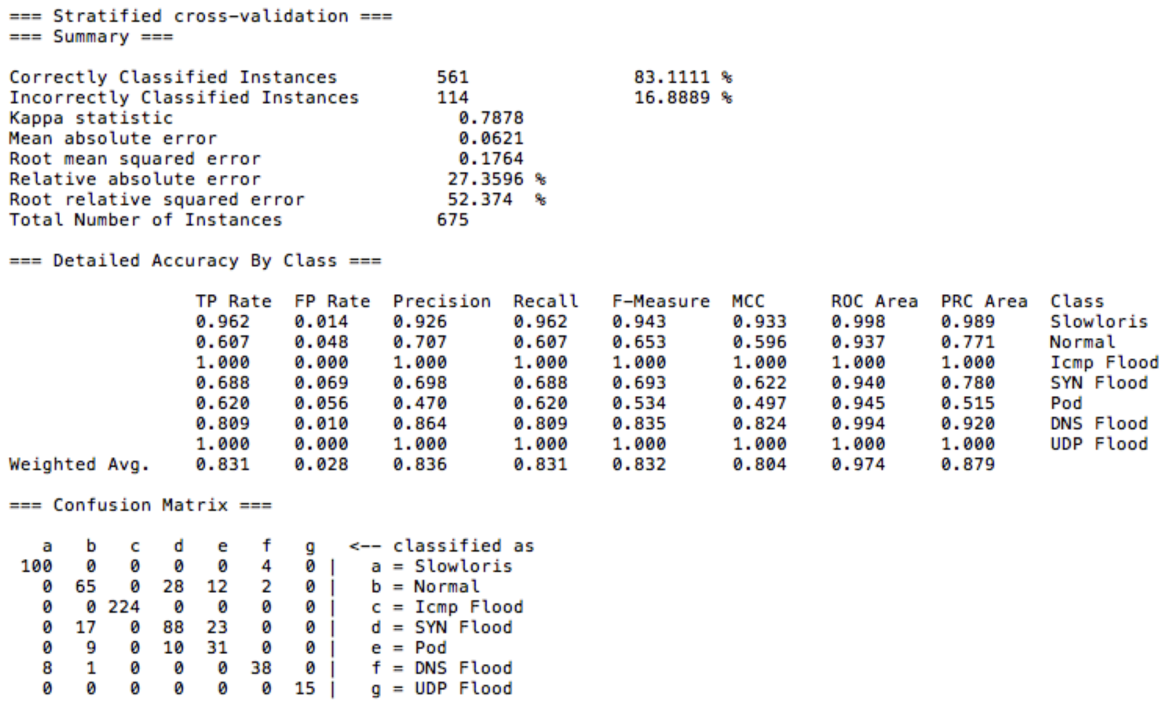
Figures 5.16 and 5.17 present the classification results arising from applying the Naïve Bayes and Random Forest models, respectively, to the processed dataset for the 3 second time interval.

In general, the overall accuracy of the Naïve Bayes classifier is 59.8519% over the total number of 675 instances of attack and normal traffic. The general accuracy of the Random Forest classifier is 71.1111% over the total number of 675 instances, suggesting that the Random Forest classifier has a higher overall accuracy than the Naïve Bayes classifier. The specific performance of the two models for each attack type, and for normal traffic, for the 3 second time interval dataset will now be discussed.



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         441               65.3333 %
Incorrectly Classified Instances       234               34.6667 %
Kappa statistic                          0.5624
Mean absolute error                      0.1091
Root mean squared error                  0.2634
Relative absolute error                 48.0504 %
Root relative squared error             78.2074 %
Total Number of Instances              675

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.519    0.025    0.794      0.519    0.628      0.593    0.957     0.814     Slowloris
                 0.374    0.100    0.412      0.374    0.392      0.285    0.847     0.445     Normal
                 0.920    0.038    0.924      0.920    0.922      0.883    0.988     0.979     Icmp Flood
                 0.672    0.157    0.500      0.672    0.573      0.463    0.876     0.537     SYN Flood
                 0.220    0.054    0.244      0.220    0.232      0.174    0.870     0.233     Pod
                 0.766    0.041    0.581      0.766    0.661      0.639    0.926     0.685     DNS Flood
                 0.533    0.000    1.000      0.533    0.696      0.726    0.915     0.728     UDP Flood
Weighted Avg.    0.653    0.069    0.670      0.653    0.652      0.591    0.925     0.704

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
  54  10  14   0   0  26   0 |   a = Slowloris
   0  40   0  55  12   0   0 |   b = Normal
  14   0 206   2   2   0   0 |   c = Icmp Flood
   0  22   0  86  20   0   0 |   d = SYN Flood
   0  13   0  26  11   0   0 |   e = Pod
   0   8   3   0   0  36   0 |   f = DNS Flood
   0   4   0   3   0   0   8 |   g = UDP Flood
```

Figure 5.16 The classification result of the Naïve Bayes model for the 3 second time interval dataset



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         561               83.1111 %
Incorrectly Classified Instances       114               16.8889 %
Kappa statistic                          0.7878
Mean absolute error                      0.0621
Root mean squared error                  0.1764
Relative absolute error                 27.3596 %
Root relative squared error             52.374  %
Total Number of Instances              675

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.962    0.014    0.926      0.962    0.943      0.933    0.998     0.989     Slowloris
                 0.607    0.048    0.707      0.607    0.653      0.596    0.937     0.771     Normal
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Icmp Flood
                 0.688    0.069    0.698      0.688    0.693      0.622    0.940     0.780     SYN Flood
                 0.620    0.056    0.470      0.620    0.534      0.497    0.945     0.515     Pod
                 0.809    0.010    0.864      0.809    0.835      0.824    0.994     0.920     DNS Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     UDP Flood
Weighted Avg.    0.831    0.028    0.836      0.831    0.832      0.804    0.974     0.879

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
 100   0   0   0   0   4   0 |   a = Slowloris
   0  65   0  28  12   2   0 |   b = Normal
   0   0 224   0   0   0   0 |   c = Icmp Flood
   0  17   0  88  23   0   0 |   d = SYN Flood
   0   9   0  10  31   0   0 |   e = Pod
   8   1   0   0   0  38   0 |   f = DNS Flood
   0   0   0   0   0   0  15 |   g = UDP Flood
```

Figure 5.17 The classification result of the Random Forest model for the 3 second time interval dataset

As can be seen in Figure 5.16, for the 3 second time interval, out of 104 Slowloris instances, 54 cases are classified correctly by the Naïve Bayes classifier. 50 cases are classified incorrectly as Normal (10), ICMP Flood (14), and DNS Flood (26) by the Naïve Bayes classifier. In contrast, in Figure 5.17, 100 Slowloris cases are classified correctly, with only 4 cases classified incorrectly by the Random Forest classifier, all as and DNS Flood.

Out of 107 Normal instances, there are 40 cases classified correctly by the Naïve Bayes classifier. 67 cases are classified incorrectly as an SYN flood (55) and PoD (12) respectively by the Naïve Bayes classifier. In contrast, 65 cases are classified correctly by the Random Forest classifier, with 42 cases incorrectly classified as SYN Flood (28), PoD (12) and DNS Flood (2).

Moving onto the ICMP Flood instances, out of 224 cases, 206 instances are classified correctly by the Naïve Bayes classifier. There are 18 cases classified incorrectly as Slowloris (14), SYN Flood (2) and PoD (2) by the Naïve Bayes classifier. For the Random Forest classifier, all 224 cases are classified correctly.

Out of 128 SYN Flood attack types, 86 cases are classified correctly by the Naïve Bayes classifier. 42 cases are incorrectly classified as being Normal (22), and PoD (20) by the Naïve Bayes classifier. In contrast, 88 cases are classified correctly by the Random Forest classifier, while 40 SYN Flood cases are classified incorrectly as Normal (17) and PoD (23).

Out of 50 PoD instances, there are 11 cases being classified correctly by the Naïve Bayes classifier, while 39 cases are classified incorrectly as Normal (13) and SYN Flood (26). In contrast, there are 31 PoD cases being classified correctly by the Random Forest classifier, with 19 PoD cases being classified incorrectly as Normal (9), and SYN Flood (10).

Moving on to the DNS Flood instances, out of 47 cases, 36 instances are classified correctly by the Naïve Bayes classifier. 11 cases are classified incorrectly as Normal (8), and ICMP Flood (3). Similarly, 38 out of 47 DNS Flood cases are classified correctly by the Random Forest classifier, while 9 cases are classified incorrectly as Slowloris (8) and Normal (1).

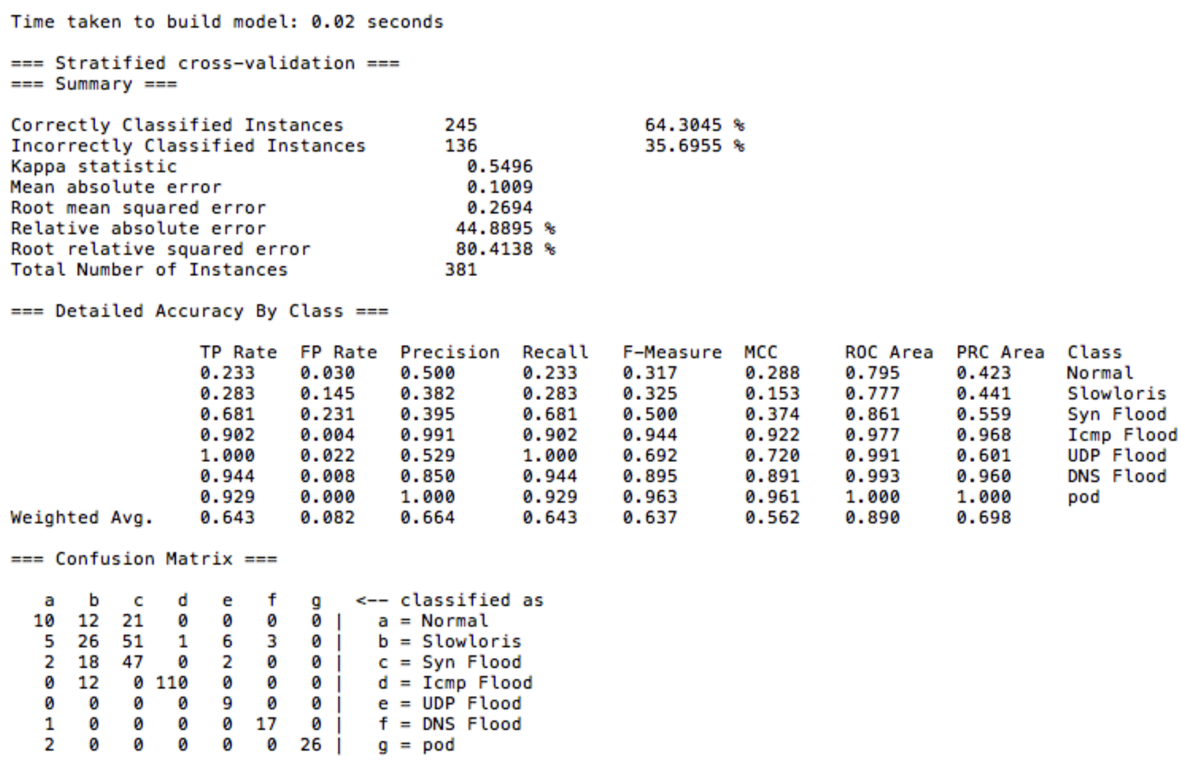
Lastly, out of 15 cases of UDP flood attacks, 8 cases are classified correctly by the Naïve Bayes classifier, with 7 cases being classified incorrectly, as Normal (4) and SYN Flood (3). In contrast, there are 15 UDP cases being classified correctly by the Random Forest classifier, with no cases being classified incorrectly.

5.5.3 Predicted versus actual outcomes for the 5 second time interval

Figures 5.18 and 5.19 present the classification results arising from applying the Naïve Bayes and Random Forest models, respectively, to the processed dataset for the 5 second time interval.

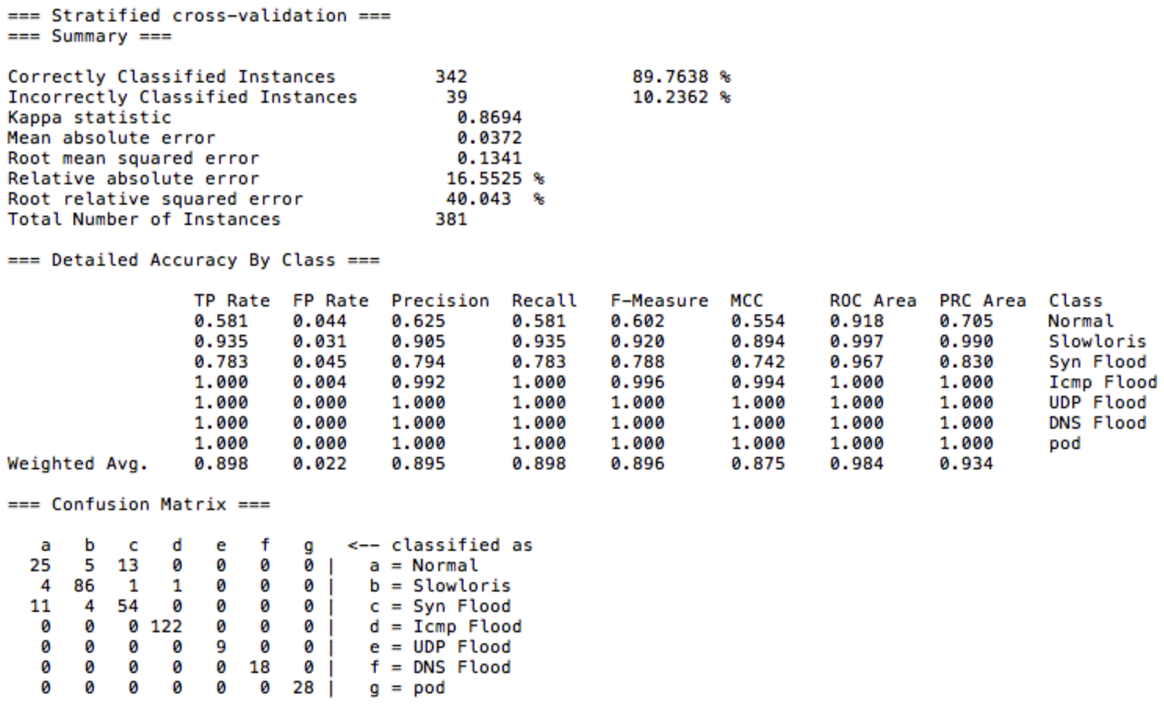
In general, the overall accuracy of the Naïve Bayes classifier is 75.1678% over the total number of 2831 instances of attack and normal traffic. The general accuracy of the Random Forest classifier is 89.7916% over the total number of instances, suggesting that the Random Forest classifier has a higher overall accuracy than the Naïve Bayes classifier.

The specific performance of the two models for each attack type, and for normal traffic, for the 5 second time interval dataset will now be discussed.



```
Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         245               64.3045 %
Incorrectly Classified Instances       136               35.6955 %
Kappa statistic                          0.5496
Mean absolute error                      0.1009
Root mean squared error                  0.2694
Relative absolute error                 44.8895 %
Root relative squared error             80.4138 %
Total Number of Instances              381

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.233    0.030    0.500      0.233    0.317      0.288    0.795     0.423     Normal
                 0.283    0.145    0.382      0.283    0.325      0.153    0.777     0.441     Slowloris
                 0.681    0.231    0.395      0.681    0.500      0.374    0.861     0.559     Syn Flood
                 0.902    0.004    0.991      0.902    0.944      0.922    0.977     0.968     Icmp Flood
                 1.000    0.022    0.529      1.000    0.692      0.720    0.991     0.601     UDP Flood
                 0.944    0.008    0.850      0.944    0.895      0.891    0.993     0.960     DNS Flood
                 0.929    0.000    1.000      0.929    0.963      0.961    1.000     1.000     pod
Weighted Avg.    0.643    0.082    0.664      0.643    0.637      0.562    0.890     0.698

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
  10  12  21   0   0   0   0 |   a = Normal
   5  26  51   1   6   3   0 |   b = Slowloris
   2  18  47   0   2   0   0 |   c = Syn Flood
   0  12   0 110   0   0   0 |   d = Icmp Flood
   0   0   0   0   9   0   0 |   e = UDP Flood
   1   0   0   0   0  17   0 |   f = DNS Flood
   2   0   0   0   0   0  26 |   g = pod
```

Figure 5.18 The classification result of the Naïve Bayes model for the 5 second time interval dataset



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         342               89.7638 %
Incorrectly Classified Instances        39               10.2362 %
Kappa statistic                          0.8694
Mean absolute error                      0.0372
Root mean squared error                  0.1341
Relative absolute error                 16.5525 %
Root relative squared error             40.043  %
Total Number of Instances              381

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
                 0.581    0.044    0.625      0.581   0.602      0.554    0.918     0.705     Normal
                 0.935    0.031    0.905      0.935   0.920      0.894    0.997     0.990     Slowloris
                 0.783    0.045    0.794      0.783   0.788      0.742    0.967     0.830     Syn Flood
                 1.000    0.004    0.992      1.000   0.996      0.994    1.000     1.000     Icmp Flood
                 1.000    0.000    1.000      1.000   1.000      1.000    1.000     1.000     UDP Flood
                 1.000    0.000    1.000      1.000   1.000      1.000    1.000     1.000     DNS Flood
                 1.000    0.000    1.000      1.000   1.000      1.000    1.000     1.000     pod
Weighted Avg.    0.898    0.022    0.895      0.898   0.896      0.875    0.984     0.934

=== Confusion Matrix ===

   a   b   c   d   e   f   g   <-- classified as
  25   5  13   0   0   0   0 |   a = Normal
   4  86   1   1   0   0   0 |   b = Slowloris
  11   4  54   0   0   0   0 |   c = Syn Flood
   0   0   0 122   0   0   0 |   d = Icmp Flood
   0   0   0   0   9   0   0 |   e = UDP Flood
   0   0   0   0   0  18   0 |   f = DNS Flood
   0   0   0   0   0   0  28 |   g = pod
```

Figure 5.19 The classification result of the Random Forest model for the 5 second time interval dataset

As can be seen in Figure 5.18, for the 5 second time interval, out of 43 Normal instances, 10 cases are classified correctly by the Naïve Bayes classifier. 33 cases are classified incorrectly, as Slowloris (12) and SYN flood (21), respectively, by the Naïve Bayes classifier. In contrast, 25 cases are classified correctly by the Random Forest classifier. 18 cases are being incorrectly classified as Slowloris (5) and SYN Flood (13).

Out of 92 Slowloris instances, 26 cases are classified correctly by the Naïve Bayes classifier, while 66 cases are classified incorrectly as Normal traffic (5), SYN Flood (51) , ICMP (1), UDP Flood (6), and DNS Flood (3). In contrast, as Figure 5.19 shows, 86 cases are being classified correctly by the Random Forest classifier, with only 6 cases classified incorrectly, as Normal traffic (4), SYN Flood (1) and ICMP Flood (1).

Out of 69 SYN Flood attack types, 47 cases are classified correctly by the Naïve Bayes classifier. 221 cases are incorrectly classified as being Slowloris (18), Normal (2), and UDP Flood (2) by the Naïve Bayes classifier. 54 cases are classified correctly by the Random Forest classifier, with 15 SYN Flood cases being classified incorrectly, as Normal (11) and Slowloris (4).

Moving to the ICMP Flood instances, out of 122 cases, 110 instances are classified correctly by the Naïve Bayes classifier. There are 12 cases classified incorrectly, all as Slowloris. For the Random Forest classifier, all 122 cases are being classified correctly.

Out of 9 cases of UDP Flood, all are classified correctly by the Naïve Bayes classifier. Similarly, all 9 cases are classified correctly by the Random Forest classifier.

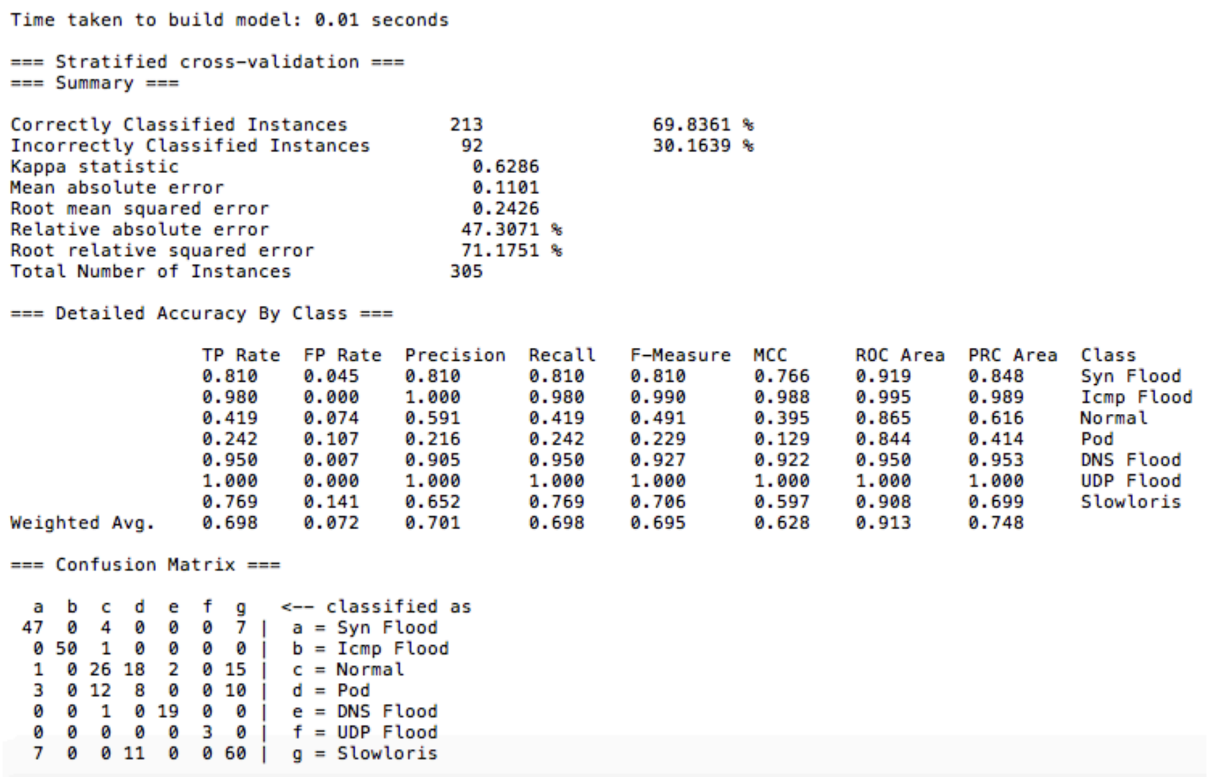
Moving on to the DNS Flood instances, out of 18 instances, 17 cases are classified correctly by the Naïve Bayes classifier, with 1 case being classified incorrectly, as Normal traffic. For the Random Forest classifier, all 18 cases are classified correctly.

Finally, out of 28 PoD instances, 26 cases are classified correctly by the Naïve Bayes classifier, with 2 cases being classified incorrectly, both as Normal traffic. For the Random Forest classifier, all 28 cases are classified correctly.

5.5.4 Predicted versus actual outcomes for the 7 second time interval

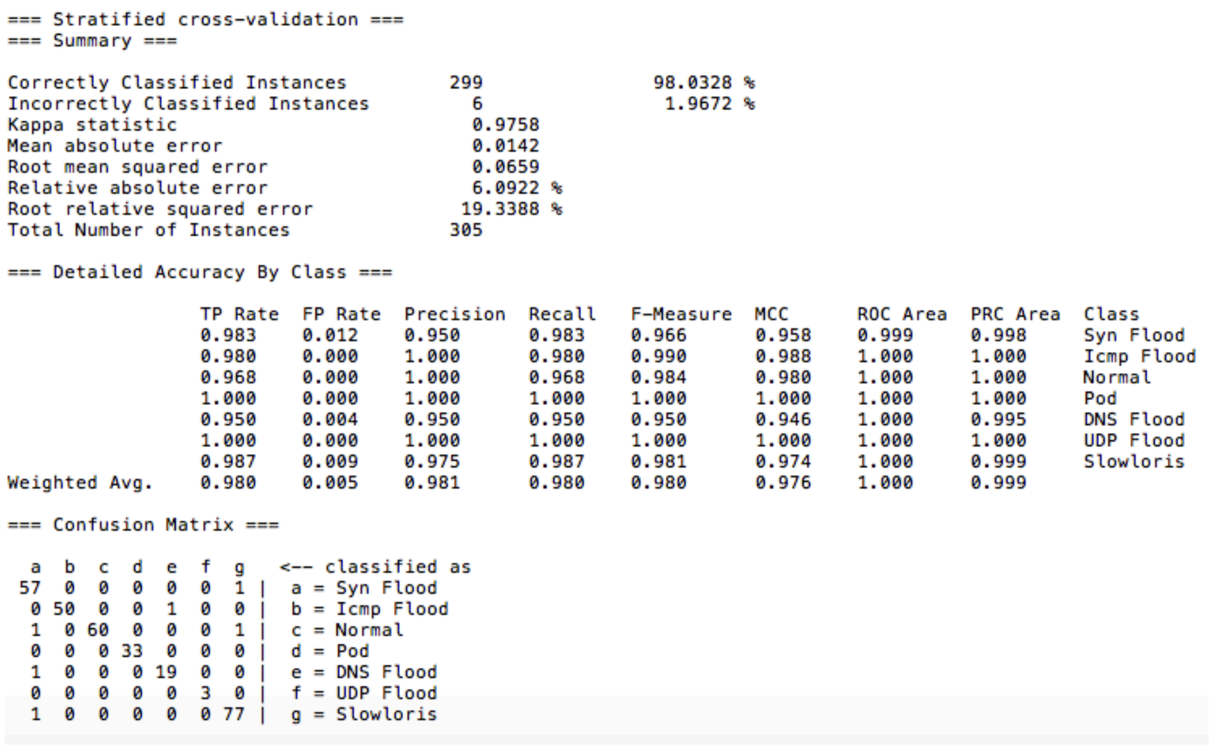
Figures 5.20 and 5.21 present the classification results arising from applying the Naïve Bayes and Random Forest models, respectively, to the processed dataset for the 7 second time interval.

In general, the overall accuracy of the Naïve Bayes classifier is 69.8361% over the total number of instances of attack and normal traffic. The general accuracy of the Random Forest classifier is 98.0328% over the total number of 305 instances, suggesting that the Random Forest classifier has a higher overall accuracy than the Naïve Bayes classifier. The specific performance of the two models for each attack type, and for normal traffic, for the 7 second time interval dataset will now be discussed.



```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         213               69.8361 %
Incorrectly Classified Instances        92               30.1639 %
Kappa statistic                          0.6286
Mean absolute error                      0.1101
Root mean squared error                  0.2426
Relative absolute error                 47.3071 %
Root relative squared error             71.1751 %
Total Number of Instances              305

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.810    0.045    0.810      0.810    0.810      0.766    0.919     0.848     Syn Flood
                 0.980    0.000    1.000      0.980    0.990      0.988    0.995     0.989     Icmp Flood
                 0.419    0.074    0.591      0.419    0.491      0.395    0.865     0.616     Normal
                 0.242    0.107    0.216      0.242    0.229      0.129    0.844     0.414     Pod
                 0.950    0.007    0.905      0.950    0.927      0.922    0.950     0.953     DNS Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     UDP Flood
                 0.769    0.141    0.652      0.769    0.706      0.597    0.908     0.699     Slowloris
Weighted Avg.    0.698    0.072    0.701      0.698    0.695      0.628    0.913     0.748

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 47  0  4  0  0  0  7 |  a = Syn Flood
  0 50  1  0  0  0  0 |  b = Icmp Flood
  1  0 26 18  2  0 15 |  c = Normal
  3  0 12  8  0  0 10 |  d = Pod
  0  0  1  0 19  0  0 |  e = DNS Flood
  0  0  0  0  0  3  0 |  f = UDP Flood
  7  0  0 11  0  0 60 |  g = Slowloris
```

Figure 5.20 The classification result of the Naïve Bayes model for the 7 second time interval dataset



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         299               98.0328 %
Incorrectly Classified Instances         6                1.9672 %
Kappa statistic                          0.9758
Mean absolute error                      0.0142
Root mean squared error                  0.0659
Relative absolute error                  6.0922 %
Root relative squared error             19.3388 %
Total Number of Instances              305

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.983    0.012    0.950      0.983    0.966      0.958    0.999     0.998     Syn Flood
                 0.980    0.000    1.000      0.980    0.990      0.988    1.000     1.000     Icmp Flood
                 0.968    0.000    1.000      0.968    0.984      0.980    1.000     1.000     Normal
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Pod
                 0.950    0.004    0.950      0.950    0.950      0.946    1.000     0.995     DNS Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     UDP Flood
                 0.987    0.009    0.975      0.987    0.981      0.974    1.000     0.999     Slowloris
Weighted Avg.    0.980    0.005    0.981      0.980    0.980      0.976    1.000     0.999

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 57  0  0  0  0  0  1 |  a = Syn Flood
  0 50  0  0  1  0  0 |  b = Icmp Flood
  1  0 60  0  0  0  1 |  c = Normal
  0  0  0 33  0  0  0 |  d = Pod
  1  0  0  0 19  0  0 |  e = DNS Flood
  0  0  0  0  0  3  0 |  f = UDP Flood
  1  0  0  0  0  0 77 |  g = Slowloris
```

Figure 5.21 The classification result of the Random Forest model for the 7 second time interval dataset

As can be seen in Figure 5.20, for the 7 second time interval, out of 58 SYN Flood attack types, 47 cases are classified correctly by the Naïve Bayes classifier, with 7 cases being incorrectly classified, as Slowloris (7) and Normal traffic (4). In contrast, 57 cases are classified correctly by the Random Forest classifier, with 1 SYN Flood cases classified incorrectly, as Slowloris (1).

Moving onto the ICMP Flood instances, out of 51 cases, 50 instances are classified correctly by the Naïve Bayes classifier. There is only 1 case classified incorrectly, as Normal traffic. Similarly, there are 50 cases being classified correctly by the Random Forest classifier, with only 1 case classified incorrectly, as DNS Flood.

Out of 62 Normal instances, only 26 cases are classified correctly by the Naïve Bayes classifier. 36 cases are classified incorrectly, as SYN flood (1), PoD (18), and DNS Flood (2) and Slowloris (15). In contrast, there are 60 cases classified correctly by the Random Forest classifier, with 2cases incorrectly classified, as Slowloris (1) and SYN Flood (1).

Out of 33 PoD instances, there are 8 cases classified correctly by the Naïve Bayes classifier. 25 cases are classified incorrectly, as SYN Flood (3), Normal (12) and Slowloris (10). In contrast, all 33 cases are being classified correctly by the Random Forest classifier.

Moving on to the DNS Flood instances, out of 20 instances, 19 cases are classified correctly by the Naïve Bayes classifier. Only 1 case is classified incorrectly, as Normal traffic. Similarly, there are 19 cases classified correctly by the Random Forest classifier, with 1case classified incorrectly, as SYN Flood.

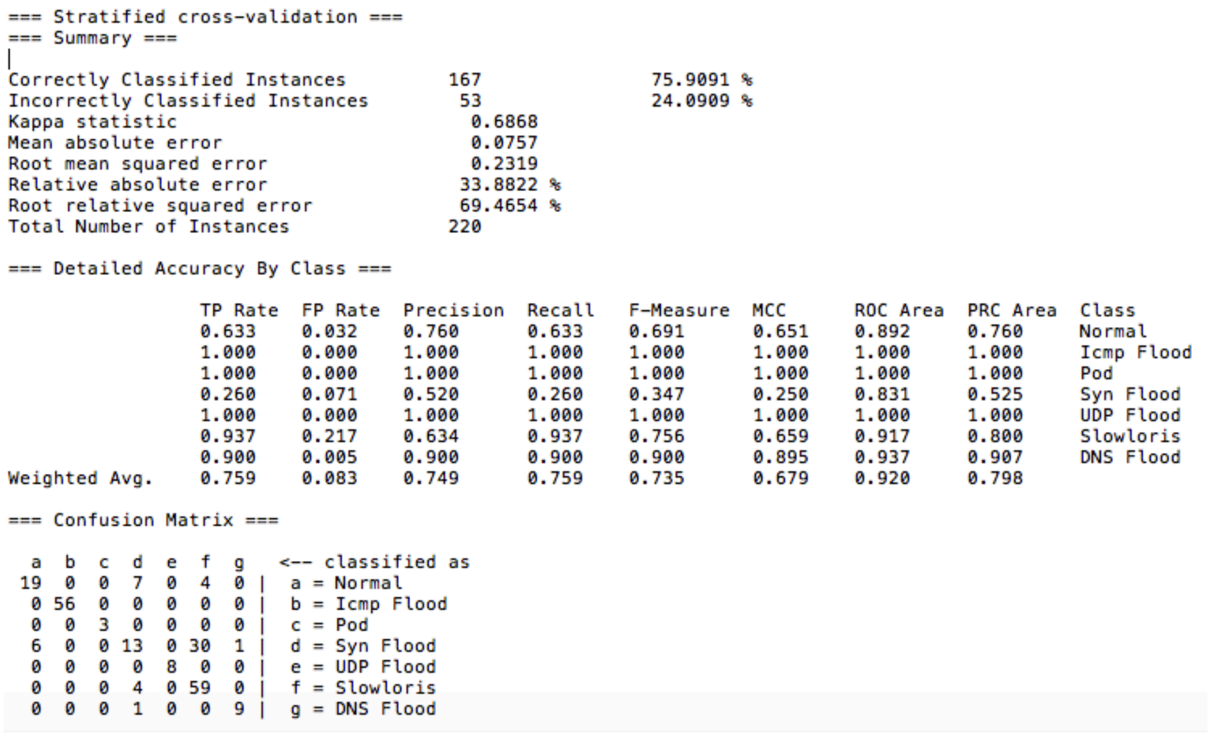
Of the 3 cases of UDP flood, all of the cases are classified correctly by the Naïve Bayes classifier. Similarly, no UDP cases are classified incorrectly by the Random Forest classifier.

Finally, out of 78 Slowloris instances, 60 cases are classified correctly by the Naïve Bayes classifier. 18 cases are classified incorrectly, as SYN Flood (7) and PoD (11). In contrast, as can be seen in Figure 5.21, there are 77 cases being classified correctly by the Random Forest classifier, with only 1 case classified incorrectly, as SYN Flood.

5.5.5 Predicted versus actual outcomes for the 10 second time interval

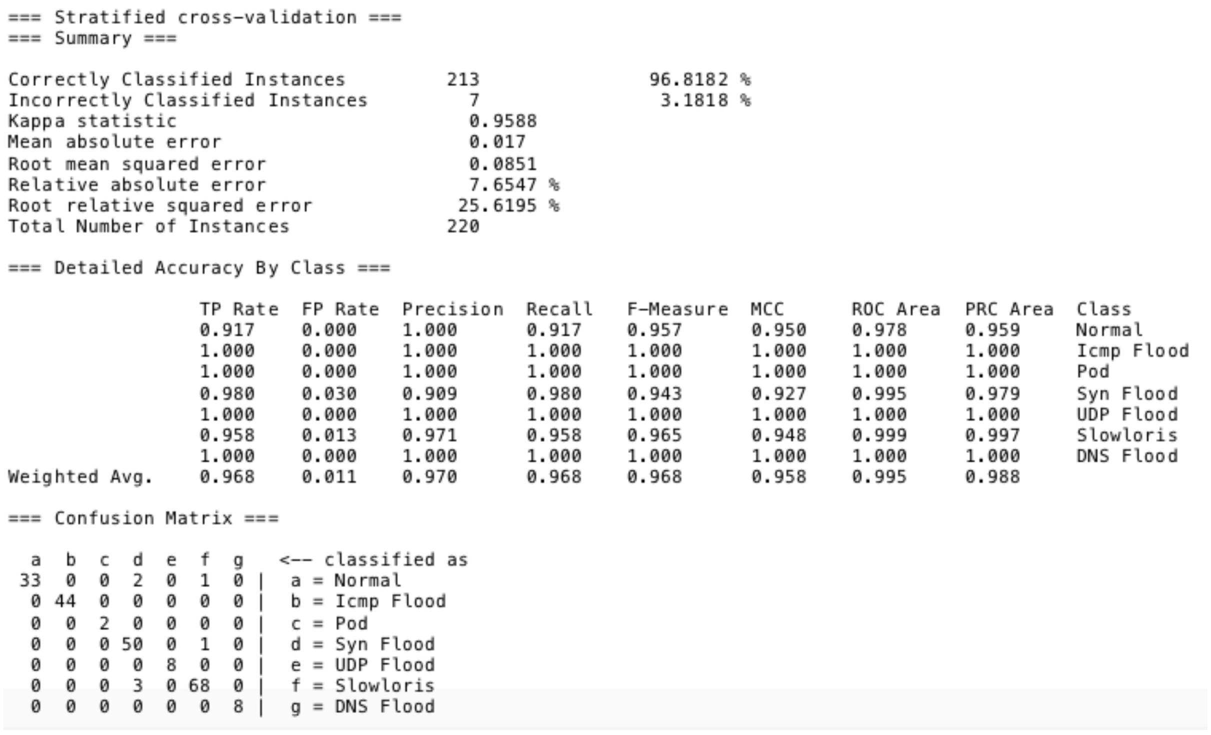
Figures 5.22 and 5.23 present the classification results arising from applying the Naïve Bayes and Random Forest models, respectively, to the processed dataset for the 10 second time interval.

In general, the overall accuracy of the Naïve Bayes classifier is 75.9091 %over the total number of instances of attack and normal traffic. The general accuracy of the Random Forest classifier is 96.8182 % over the total number of 226instances, suggesting that the Random Forest classifier has a higher overall accuracy than the Naïve Bayes classifier. The specific performance of the two models for each attack type, and for normal traffic, for the 10 second time interval dataset will now be discussed.



```
=== Stratified cross-validation ===
=== Summary ===
|
Correctly Classified Instances         167               75.9091 %
Incorrectly Classified Instances        53               24.0909 %
Kappa statistic                          0.6868
Mean absolute error                      0.0757
Root mean squared error                  0.2319
Relative absolute error                 33.8822 %
Root relative squared error             69.4654 %
Total Number of Instances              220

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.633    0.032    0.760      0.633    0.691      0.651    0.892     0.760     Normal
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Icmp Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Pod
                 0.260    0.071    0.520      0.260    0.347      0.250    0.831     0.525     Syn Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     UDP Flood
                 0.937    0.217    0.634      0.937    0.756      0.659    0.917     0.800     Slowloris
                 0.900    0.005    0.900      0.900    0.900      0.895    0.937     0.907     DNS Flood
Weighted Avg.    0.759    0.083    0.749      0.759    0.735      0.679    0.920     0.798

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 19  0  0  7  0  4  0 |  a = Normal
  0 56  0  0  0  0  0 |  b = Icmp Flood
  0  0  3  0  0  0  0 |  c = Pod
  6  0  0 13  0 30  1 |  d = Syn Flood
  0  0  0  0  8  0  0 |  e = UDP Flood
  0  0  0  4  0 59  0 |  f = Slowloris
  0  0  0  1  0  0  9 |  g = DNS Flood
```

Figure 5.22 The classification result of the Naïve Bayes model for the 10 second time interval dataset



```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         213               96.8182 %
Incorrectly Classified Instances         7                3.1818 %
Kappa statistic                          0.9588
Mean absolute error                      0.017
Root mean squared error                  0.0851
Relative absolute error                  7.6547 %
Root relative squared error             25.6195 %
Total Number of Instances              220

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.917    0.000    1.000      0.917    0.957      0.950    0.978     0.959     Normal
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Icmp Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     Pod
                 0.980    0.030    0.909      0.980    0.943      0.927    0.995     0.979     Syn Flood
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     UDP Flood
                 0.958    0.013    0.971      0.958    0.965      0.948    0.999     0.997     Slowloris
                 1.000    0.000    1.000      1.000    1.000      1.000    1.000     1.000     DNS Flood
Weighted Avg.    0.968    0.011    0.970      0.968    0.968      0.958    0.995     0.988

=== Confusion Matrix ===

  a  b  c  d  e  f  g   <-- classified as
 33  0  0  2  0  1  0 |  a = Normal
  0 44  0  0  0  0  0 |  b = Icmp Flood
  0  0  2  0  0  0  0 |  c = Pod
  0  0  0 50  0  1  0 |  d = Syn Flood
  0  0  0  0  8  0  0 |  e = UDP Flood
  0  0  0  3  0 68  0 |  f = Slowloris
  0  0  0  0  0  0  8 |  g = DNS Flood
```

Figure 5.23 The classification result of the Random Forest model for the 10 second time interval dataset

As can be seen in Figure 5.22, for the 10 second time interval out of 30 Normal instances, there are only 19 cases classified correctly by the Naïve Bayes classifier. 11 cases are classified incorrectly, as SYN flood (7), and Slowloris (4). In contrast, 33 of 36 cases are classified correctly by the Random Forest classifier, with only 3 cases being incorrectly classified, as SYN Flood (1) and Slowloris (1).

Moving onto the ICMP Flood instances, all the 56 cases and 44 cases are being classified correctly by the Naïve Bayes and the Random Forest classifier respectively.

Out of the 3 cases of Pod, all are classified correctly by the Naïve Bayes classifier. Similarly, the 2 Pod cases for the Random Forest classifier are both classified correctly.

Out of 50 SYN Flood attack types, 13 cases are classified correctly by the Naïve Bayes classifier. 37cases are incorrectly classified as being Normal (6), Slowloris (30) and DNS Flood (1). In contrast, 50of 51 cases are classified correctly by the Random Forest classifier. There is only one case that is classified incorrectly as Slowloris.

Out of the 8 cases of UDP flood, all are classified correctly by the Naïve Bayes classifier. Similarly, the 8 UDP cases for the Random Forest classifier are both classified correctly.

Out of 63Slowloris instances, 59cases are classified correctly by the Naïve Bayes classifier. 4 cases are classified incorrectly as SYN Flood. In contrast, as shown in Figure 5.23, 68of71 cases are being classified correctly by the Random Forest classifier.

Finally, moving on to the DNS Flood instances, 9 out of 10 cases are classified correctly by the Naïve Bayes classifier. There is only 1 case that classified incorrectly as SYN flood. perhaps surprisingly all 17 DNS Flood cases are classified correctly by the Random Forest classifier.

5.6 Approach to the Analysis of the Results of the Misclassification Errors

This section will present the process of the analysis undertaken to analyse the results of the identified misclassification errors. Because of the high number of features contained in the processed dataset, the analysis approach that was used in chapter 3 (see section 3.5.2 was not a suitable option for use in this phase of the work. Figures 5.24 and 5.25

present an example of one of the misclassified cases (Pod-Slowloris) to illustrate why, and to justify the alternative analysis approach that was used. Having done this, the section will then explain the steps that comprise the alternative analysis approach.

Figure 5.24 shows the Pod attack feature distribution for the 1 second time interval and Figure 5.25 presents the frequency of the Pod attacks that were misclassified as Slowloris attacks. As can be seen in these two figures, the high number of factors involved in the processed dataset makes the visual inspection of the graphs extremely difficult (unlike in the analysis in section 3.5.2 where only three features were being considered). As such, visual inspection of the graphs of the misclassification cases and their comparison to the graphs of the correctly classified attack cases is not a suitable way to develop meaningful interpretations of the misclassification errors for the two classifiers (Naïve Bayes and Random Forest). Therefore, an alternative analysis approach, set out in Figure 5.26, was designed. Each step of this approach will be now explained.

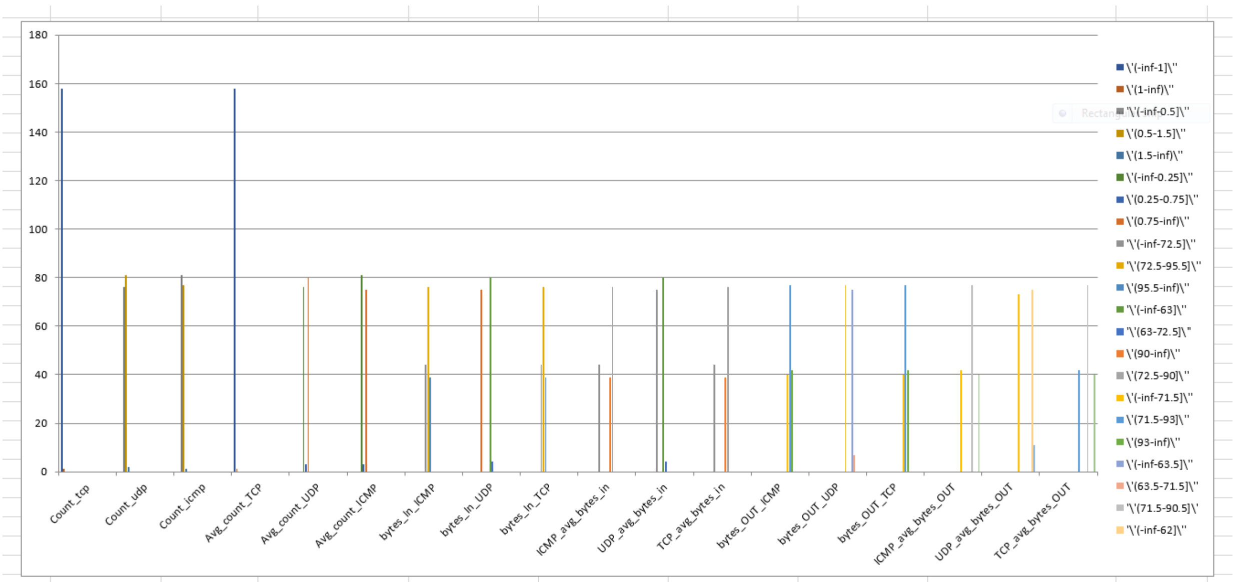


Figure 5.24 Distribution of correctly identified Pod attack instances in terms of the frequency of the discretized feature values for the 1 second time interval

The right hand side of Figures 5.24 and 5.25 shows a colour for each of the features and their discretized values, starting at the top with blue representing 'Count_tcp: Low', then brown representing 'Count_tcp: Medium' and grey representing 'Count_tcp: High', and so on, moving through three values for each feature on the x-axis.

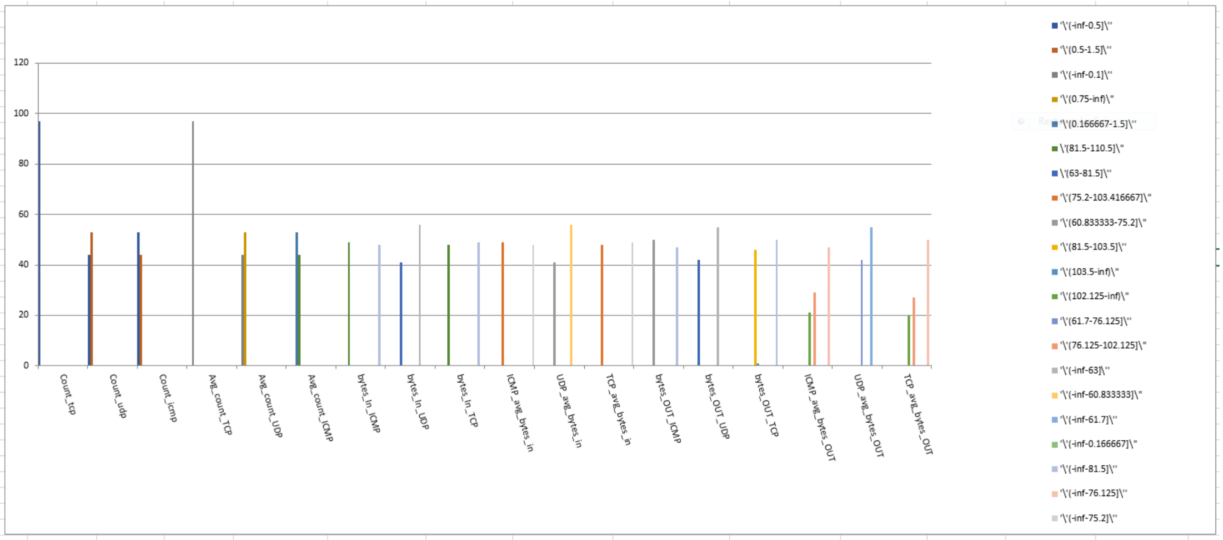


Figure 5.25 Distribution of Pod attacks instances misclassified as Slowloris attacks in terms of the frequency of the discretized feature values for the 1 second time interval

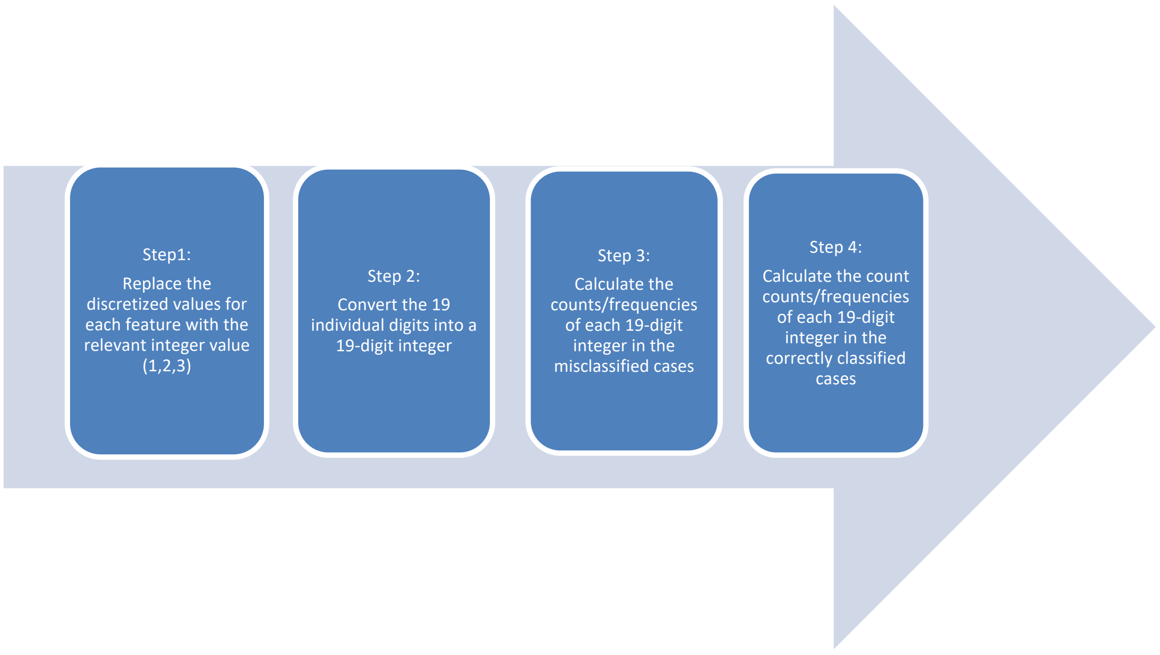
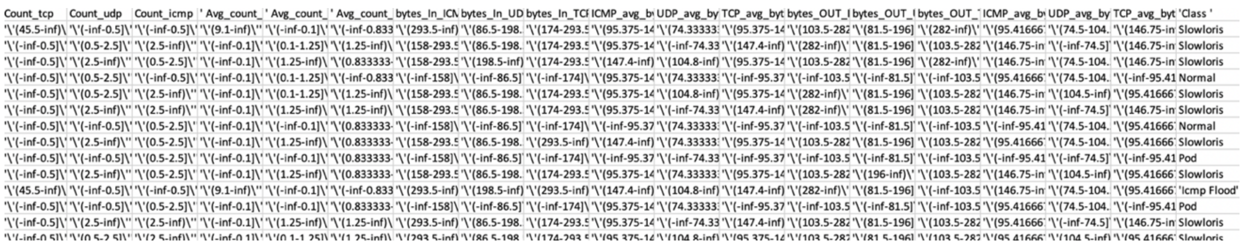

Step1:
Replace the discretized values for each feature with the relevant integer value (1,2,3)
Step 2:
Convert the 19 individual digits into a 19-digit integer
Step 3:
Calculate the counts/frequencies of each 19-digit integer in the misclassified cases
Step 4:
Calculate the count counts/frequencies of each 19-digit integer in the correctly classified cases

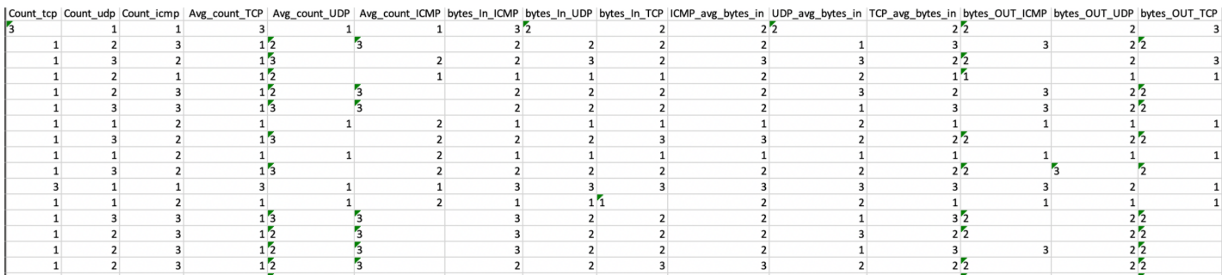
Figure 5.26 Analysis Approach for Misclassification Errors of the classification models

In step 1, the discretized values in the low, medium and high ranges were replaced with 1, 2 and 3 (respectively) for each value of the features in the processed dataset. This resulted in having 19 digits (between 1 and 3) running across the cells for each single record of actual and misclassified data. Figure 5.27 presents the initial structure of an extract of the processed dataset with discretized values, and then Figure 5.28 shows the same extract of the dataset after with all the discretized values replaced by the relevant integers (1, 2 or 3).



Count_tcp	Count_udp	Count_icmp	' Avg_count_	' Avg_count_	' Avg_count_	bytes_In_ICN	bytes_In_UD	bytes_In_TCF	ICMP_avg_b	UDP_avg_by	TCP_avg_byt	bytes_OUT_I	bytes_OUT_I	bytes_OUT_T	ICMP_avg_b	UDP_avg_by	TCP_avg_byt	'Class '
'\'(45.5-inf)\	'\'(-inf-0.5]\'	'\'(-inf-0.5]\'	'\'(9.1-inf)\''	'\'(-inf-0.1]\'	'\'(-inf-0.833	'\'(293.5-inf)	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(74.33333	'\'(95.375-14	'\'(103.5-282	'\'(81.5-196]	'\'(282-inf)\'	'\'(95.41666	'\'(74.5-104.	'\'(146.75-in	Slowloris
'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(2.5-inf)\''	'\'(-inf-0.1]\'	'\'(0.1-1.25]\	'\'(1.25-inf)\	'\'(158-293.5	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(-inf-74.33	'\'(147.4-inf)	'\'(282-inf)\'	'\'(81.5-196]	'\'(103.5-282	'\'(146.75-in	'\'(-inf-74.5]'	'\'(146.75-in	Slowloris
'\'(-inf-0.5]\'	'\'(2.5-inf)\''	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(1.25-inf)\	'\'(0.833333-	'\'(158-293.5	'\'(198.5-inf)	'\'(174-293.5	'\'(147.4-inf)	'\'(104.8-inf)	'\'(95.375-14	'\'(103.5-282	'\'(81.5-196]	'\'(282-inf)\'	'\'(146.75-in	'\'(74.5-104.	'\'(146.75-in	Slowloris
'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(-inf-0.5]\'	'\'(-inf-0.1]\'	'\'(0.1-1.25]\	'\'(-inf-0.833	'\'(-inf-158]\	'\'(-inf-86.5]'	'\'(-inf-174]\	'\'(95.375-14	'\'(74.33333	'\'(-inf-95.37	'\'(-inf-103.5	'\'(-inf-81.5]'	'\'(-inf-103.5	'\'(95.41666	'\'(74.5-104.	'\'(-inf-95.41	Normal
'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(2.5-inf)\''	'\'(-inf-0.1]\'	'\'(0.1-1.25]\	'\'(1.25-inf)\	'\'(158-293.5	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(104.8-inf)	'\'(95.375-14	'\'(282-inf)\'	'\'(81.5-196]	'\'(103.5-282	'\'(146.75-in	'\'(104.5-inf)	'\'(95.41666	Slowloris
'\'(-inf-0.5]\'	'\'(2.5-inf)\''	'\'(2.5-inf)\''	'\'(-inf-0.1]\'	'\'(1.25-inf)\	'\'(1.25-inf)\	'\'(158-293.5	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(-inf-74.33	'\'(147.4-inf)	'\'(282-inf)\'	'\'(81.5-196]	'\'(103.5-282	'\'(146.75-in	'\'(-inf-74.5]'	'\'(146.75-in	Slowloris
'\'(-inf-0.5]\'	'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(-inf-0.1]\'	'\'(0.833333-	'\'(-inf-158]\	'\'(-inf-86.5]'	'\'(-inf-174]\	'\'(-inf-95.37	'\'(74.33333	'\'(-inf-95.37	'\'(-inf-103.5	'\'(-inf-81.5]'	'\'(-inf-103.5	'\'(-inf-95.41	'\'(74.5-104.	'\'(95.41666	Normal
'\'(-inf-0.5]\'	'\'(2.5-inf)\''	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(1.25-inf)\	'\'(0.833333-	'\'(158-293.5	'\'(86.5-198.	'\'(293.5-inf)	'\'(147.4-inf)	'\'(74.33333	'\'(95.375-14	'\'(103.5-282	'\'(81.5-196]	'\'(103.5-282	'\'(95.41666	'\'(74.5-104.	'\'(95.41666	Slowloris
'\'(-inf-0.5]\'	'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(-inf-0.1]\'	'\'(0.833333-	'\'(-inf-158]\	'\'(-inf-86.5]'	'\'(-inf-174]\	'\'(-inf-95.37	'\'(-inf-74.33	'\'(-inf-95.37	'\'(-inf-103.5	'\'(-inf-81.5]'	'\'(-inf-103.5	'\'(-inf-95.41	'\'(-inf-74.5]'	'\'(-inf-95.41	Pod
'\'(-inf-0.5]\'	'\'(2.5-inf)\''	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(1.25-inf)\	'\'(0.833333-	'\'(158-293.5	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(74.33333	'\'(95.375-14	'\'(103.5-282	'\'(196-inf)\'	'\'(103.5-282	'\'(146.75-in	'\'(104.5-inf)	'\'(95.41666	Slowloris
'\'(45.5-inf)\	'\'(-inf-0.5]\'	'\'(-inf-0.5]\'	'\'(9.1-inf)\''	'\'(-inf-0.1]\'	'\'(-inf-0.833	'\'(293.5-inf)	'\'(198.5-inf)	'\'(293.5-inf)	'\'(147.4-inf)	'\'(104.8-inf)	'\'(147.4-inf)	'\'(282-inf)\'	'\'(81.5-196]	'\'(-inf-103.5	'\'(146.75-in	'\'(74.5-104.	'\'(95.41666	'Icmp Flood'
'\'(-inf-0.5]\'	'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(-inf-0.1]\'	'\'(-inf-0.1]\'	'\'(0.833333-	'\'(-inf-158]\	'\'(-inf-86.5]'	'\'(-inf-174]\	'\'(95.375-14	'\'(74.33333	'\'(-inf-95.37	'\'(-inf-103.5	'\'(-inf-81.5]'	'\'(-inf-103.5	'\'(95.41666	'\'(74.5-104.	'\'(-inf-95.41	Pod
'\'(-inf-0.5]\'	'\'(2.5-inf)\''	'\'(2.5-inf)\''	'\'(-inf-0.1]\'	'\'(1.25-inf)\	'\'(1.25-inf)\	'\'(293.5-inf)	'\'(86.5-198.	'\'(174-293.5	'\'(95.375-14	'\'(-inf-74.33	'\'(147.4-inf)	'\'(103.5-282	'\'(81.5-196]	'\'(103.5-282	'\'(95.41666	'\'(-inf-74.5]'	'\'(146.75-in	Slowloris
'\'(-inf-0.5]\'	'\'(0.5-2.5]\'	'\'(2.5-inf)\''	'\'(-inf-0.1]\'	'\'(0.1-1.25]\	'\'(1.25-inf)\	'\'(293.5-inf)	'\'(86.5-198	'\'(174-293.5	'\'(95.375-14	'\'(104.8-inf)	'\'(95.375-14	'\'(103.5-282	'\'(81.5-196]	'\'(103.5-282	'\'(95.41666	'\'(104.5-inf)	'\'(95.41666	Slowloris

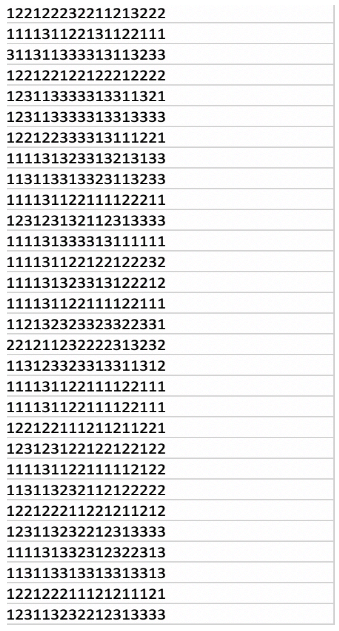
Figure 5.27 Structure of an extract of the processed dataset showing the discretised values



Count_tcp	Count_udp	Count_icmp	Avg_count_TCP	Avg_count_UDP	Avg_count_ICMP	bytes_In_ICMP	bytes_In_UDP	bytes_In_TCP	ICMP_avg_bytes_in	UDP_avg_bytes_in	TCP_avg_bytes_in	bytes_OUT_ICMP	bytes_OUT_UDP	bytes_OUT_TCP
3	1	1	3	1	1	3	2	2	2	2	2	2	2	3
1	2	3	1	2	3	2	2	2	2	1	3	3	2	2
1	3	2	1	3	2	2	3	2	3	3	2	2	2	3
1	2	1	1	2	1	1	1	1	2	2	1	1	1	1
1	2	3	1	2	3	2	2	2	2	3	2	3	2	2
1	3	3	1	3	3	2	2	2	2	1	3	3	2	2
1	1	2	1	1	2	1	1	1	1	2	1	1	1	1
1	3	2	1	3	2	2	2	3	3	2	2	2	2	2
1	1	2	1	1	2	1	1	1	1	1	1	1	1	1
1	3	2	1	3	2	2	2	2	2	2	2	2	3	2
3	1	1	3	1	1	3	3	3	3	3	3	3	2	1
1	1	2	1	1	2	1	1	1	2	2	1	1	1	1
1	3	3	1	3	3	3	2	2	2	1	3	2	2	2
1	2	3	1	2	3	3	2	2	2	3	2	2	2	2
1	2	3	1	2	3	3	2	2	2	1	3	3	2	2
1	2	3	1	2	3	2	2	3	3	2	2	2	2	2

Figure 5.28 Structure of the same extract of the processed dataset after having replaced the discretized values with the relevant integer (1, 2 or 3)

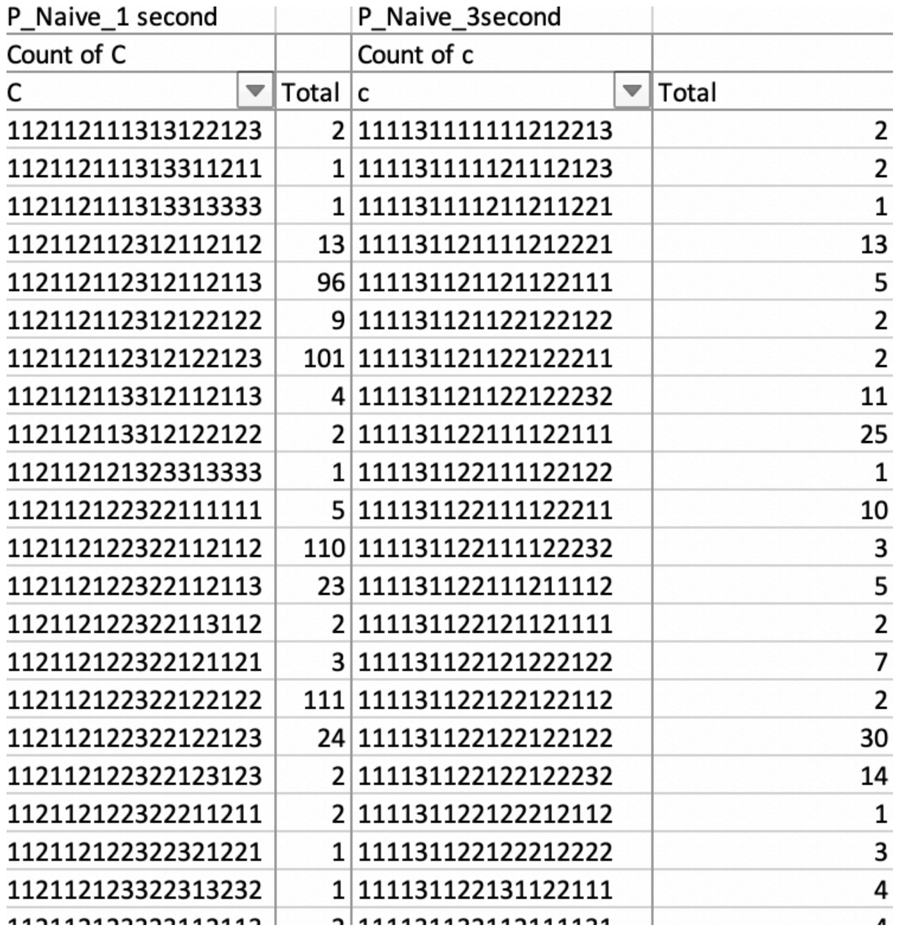
In step 2, the 19 individual digits representing each of the misclassified and actual cases were converted into 19-digits integers (concatenating the individual cells in each row of the dataset). Figure 5.29 present the extract of the resulting 19-digit integers in the processed dataset. These integers represent unique feature patterns that can then be analysed.



1221222322112132222
1111311221311221111
3113113333131132333
1221221221222122222
1231133333133113211
1231133333133133333
1221223333131112211
1111313233132131333
1131133133231132333
1111311221111222111
1231231321123133333
1111313333131111111
1111311221221222322
1111313233131222122
1111311221111221111
1121323233233223311
2212112322223132322
1131233233133113122
1111311221111221111
1111311221111221111
1221221112112112211
1231231221221221222
1111311221111121222
1131132321121222222
1221222112212112122
1231132322123133333
1111313323123223133
1131133133133133133
1221222111212111211
1231132322123133333

Figure 5.29 An extract of the processed dataset showing the generated 19-digit integers

In the steps 3 and 4, a pivot table was used to calculate the counts/frequencies of each 19-digit integer for the misclassified cases and actual cases, respectively. The aim was to obtain the frequency with which each feature pattern occurred over the five different time intervals for correct and misclassified cases, serving as the basis for subsequent analysis. Figure 5.30 presents an extract of the frequency table showing the misclassified cases to illustrate the output of step 3. A corresponding extract of the correctly classified cases could be shown as the output of step 4.



P_Naive_1 second		P_Naive_3second	
Count of C		Count of c	
C	Total	c	Total
112112111313122123	2	111131111111212213	2
112112111313311211	1	111131111121112123	2
112112111313313333	1	111131111211211221	1
112112112312112112	13	111131121111212221	13
112112112312112113	96	111131121121122111	5
112112112312122122	9	111131121122122122	2
112112112312122123	101	111131121122122211	2
112112113312112113	4	111131121122122232	11
112112113312122122	2	111131122111122111	25
112112121323313333	1	111131122111122122	1
112112122322111111	5	111131122111122211	10
112112122322112112	110	111131122111122232	3
112112122322112113	23	111131122111211112	5
112112122322113112	2	111131122121121111	2
112112122322121121	3	111131122121222122	7
112112122322122122	111	111131122122122112	2
112112122322122123	24	111131122122122122	30
112112122322123123	2	111131122122122232	14
112112122322211211	2	111131122122212112	1
112112122322321221	1	111131122122212222	3
112112123322313232	1	111131122131122111	4

Figure 5.30 An extract of the frequency table for misclassified cases

The next section will present an analysis of the misclassification errors in the five different time intervals using these frequencies of the feature patterns in the misclassified and correctly classified cases.

5.7 Analysis of the Results of Misclassification Errors

This section will present an analysis of the misclassification errors identified in the confusion matrix results of the Naïve Bayes and Random Forest models for the different time intervals. The analysis will examine misclassified cases, making use of the work reported in section 5.6 to then compare the pattern of factors on which classifications seem to have been made. (see section 5.8) The aim is to identify a number of accuracy issues related to the developed models in the different time slices which will in turn, frame the focus of the next chapter where the relationships between the features will be more deeply explored.

Table 5.2 summaries all the misclassified instances of the Naïve Bayes and Random Forest classifiers in the 1, 3, 5, 7 and 10-second time intervals. The misclassified cases in each time interval will be analyzed in sections 5.7.2-5.7.6, respectively. Before this, however, section 5.7.1 will explain the thresholds applied when considering misclassification cases in each time interval.

Time slice	Type of the classifier	The actual Instances	The predicted instances
1 second	Naïve Bayes	PoD	SYN Flood, Normal, Slowloris
		SYN Flood	Pod, ICMP Flood, Normal, Slowloris
		ICMP Flood	SYN Flood, DNS Flood, Normal, UDP Flood
		DNS Flood	ICMP Flood, Normal, Slowloris
		Normal	Pod, SYN Flood, ICMP Flood, DNS Flood, UDP Flood, Slowloris
		Slowloris	Pod, SYN Flood, ICMP Flood, Normal
1 second	Random Forest	Pod	SYN Flood, Normal
		SYN Flood	Pod, Normal, Slowloris
		ICMP Flood	DNS Flood, Normal, Slowloris
		DNS Flood	SYN Flood, ICMP Flood, Normal
		Normal	Pod, SYN Flood, ICMP Flood, DNS Flood, Slowloris
		UDP Flood	Normal
		Slowloris	PoD, SYN Flood, ICMP Flood, Normal
3 seconds	Naïve Bayes	Slowloris	Normal, ICMP Flood, Pod
		Normal	SYN Flood, Pod
		ICMP Flood	Slowloris, SYN Flood, Pod
		SYN Flood	Normal, PoD
		Pod	SYN Flood, Normal
		DNS Flood	Normal, ICMP Flood
		UDP Flood	Normal, SYN Flood
3 seconds	Random Forest	Slowloris	DNS Flood
		Normal	SYN Flood, Pod
		SYN Flood	Normal, Pod
		Pod	Normal, SYN Flood
		DNS Flood	Normal
5 seconds	Naïve Bayes	Normal	Slowloris, SYN Flood
		Slowloris	Normal, SYN Flood, ICMP Flood, UDP Flood, DNS Flood
		SYN Flood	Normal, Slowloris, UDP Flood
		ICMP Flood	Slowloris
		DNS Flood	Normal
		PoD	Normal
5 seconds	Random Forest	Normal	Slowloris, SYN Flood

		Slowloris	Normal, SYN Flood, ICMP Flood
		SYN Flood	Normal, Slowloris
7 seconds	Naïve Bayes	SYN Flood	Normal, Slowloris
		ICMP Flood	Normal
		Normal	SYN Flood, Pod, DNS Flood, Slowloris
		Pod	SYN Flood, Normal, Slowloris
		DNS Flood	Normal
		Slowloris	SYN Flood, Normal
7 seconds	Random Forest	SYN Flood	Slowloris
		ICMP Flood	DNS Flood
		Normal	SYN Flood, Slowloris
		DNS Flood	SYN Flood
		Slowloris	SYN Flood
10 Seconds	Naïve Bayes	Normal	SYN Flood, Slowloris
		SYN Flood	Normal, Slowloris, DNS Flood
		Slowloris	SYN Flood
		DNS	SYN Flood
10 Sec	Random Forest	Normal	SYN Flood, Slowloris
		SYN Flood	Slowloris
		Slowloris	SYN Flood

Table 5.2 Summary of the results of the misclassification errors in each time interval

5.7.1 Setting thresholds for the analysis of the misclassification errors in each time interval

Before presenting the misclassification errors associated with the two models/classifiers, it was decided to set thresholds to determine which misclassification cases should be considered. The aim was to focus on the cases (feature patterns) for each model, in each time interval, where higher levels of misclassification had been identified as these cases had the greatest impact on the accuracy of the models.

Table 5.3 summarizes the selected thresholds for the Naïve Bayes and Random Forest classifiers in the five chosen time intervals/slices. The thresholds were determined by simple observation of the values in the confusion matrices, with a threshold value chosen in each case that captured most of the misclassification cases. In the 1 second time intervals, the threshold was set at 20 for the Naïve Bayes and Random Forest classifiers; in the 3 second time interval, the threshold was set at 10 for both classifiers; and in the 5 second time interval the threshold was set at 5 for both classifiers. In the 7, and 10 second intervals, however, the threshold for the both the Naïve Bayes and Random Forest classifiers was set at 1 (i.e., all misclassification cases were examined) as such since the models' accuracies in these time intervals were very high, leaving few cases to be considered.

Time interval	The chosen for threshold for Naïve Bayes	The chosen threshold for Random Forest
1 second	20	20
3 seconds	10	10
5 seconds	5	5
7 seconds	1	1
10 seconds	1	1

Table 5.3 The chosen thresholds for the misclassification analysis

Sections 5.7.2-5.7.6 will, in turn, present the feature patterns of the misclassification cases above the selected threshold for each of the time intervals, then section 5.8 will analyse these feature patterns.

5.7.2 Misclassification cases for the 1 second time interval

To begin to understand the misclassifications made by the model, the misclassification cases were explored by analyzing the underlying data of the cases identified by the models that met the relevant threshold (see Table 5.3). These 'above threshold' instances of normal traffic and each attack type in the processed dataset that were misclassified by the model were isolated and examined. The aim was to identify the particular feature patterns in the set of misclassification instances and then, after each time interval had been considered, to examine changes in the relative frequency of the feature patterns over the different time intervals in an attempt to understand the underlying reasons for the changes in the accuracy of the classifiers as the length of the time interval increased.

To understand the misclassification errors of the Naïve Bayes and the Random Forest models in the 1 second time interval, the distribution of the frequencies of the different feature patterns in the misclassification cases was examined. The results are presented in Figures 5.31 and 5.32. To concentrate on the most common feature patterns represented in the misclassification set, only feature patterns that were found more than 20 times were considered.

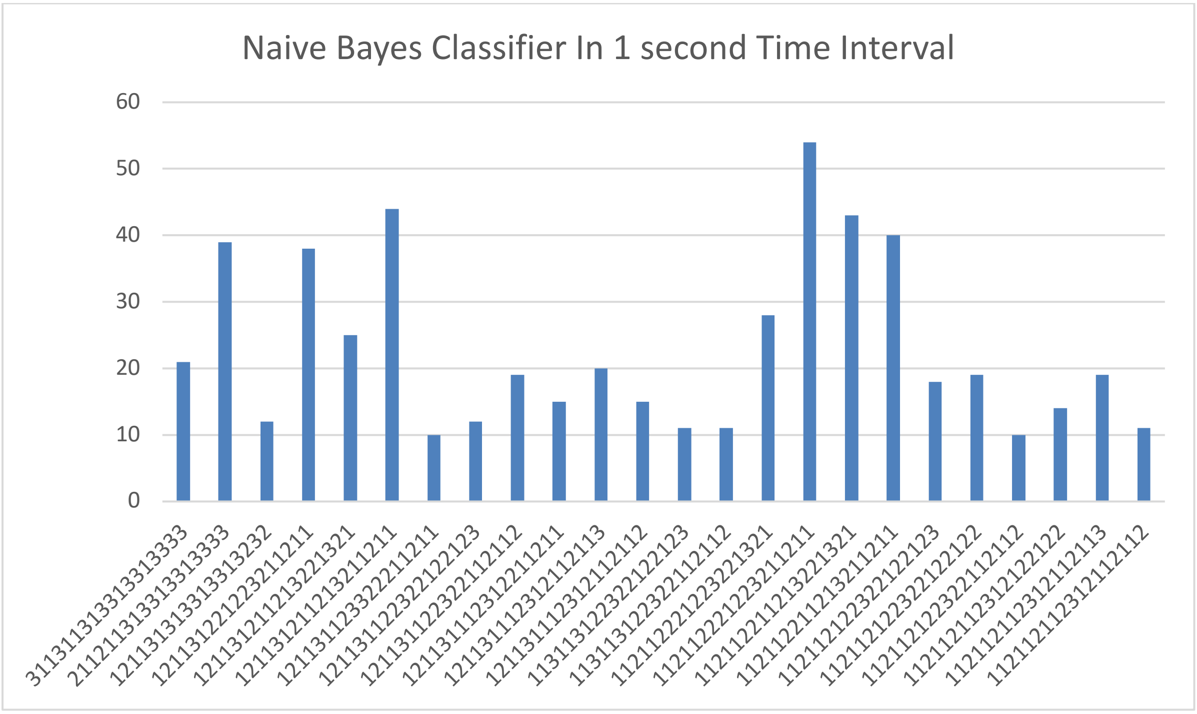


Figure 5.31 The distribution of misclassification feature patterns in the 1 second time interval for the Naïve Bayes classifier

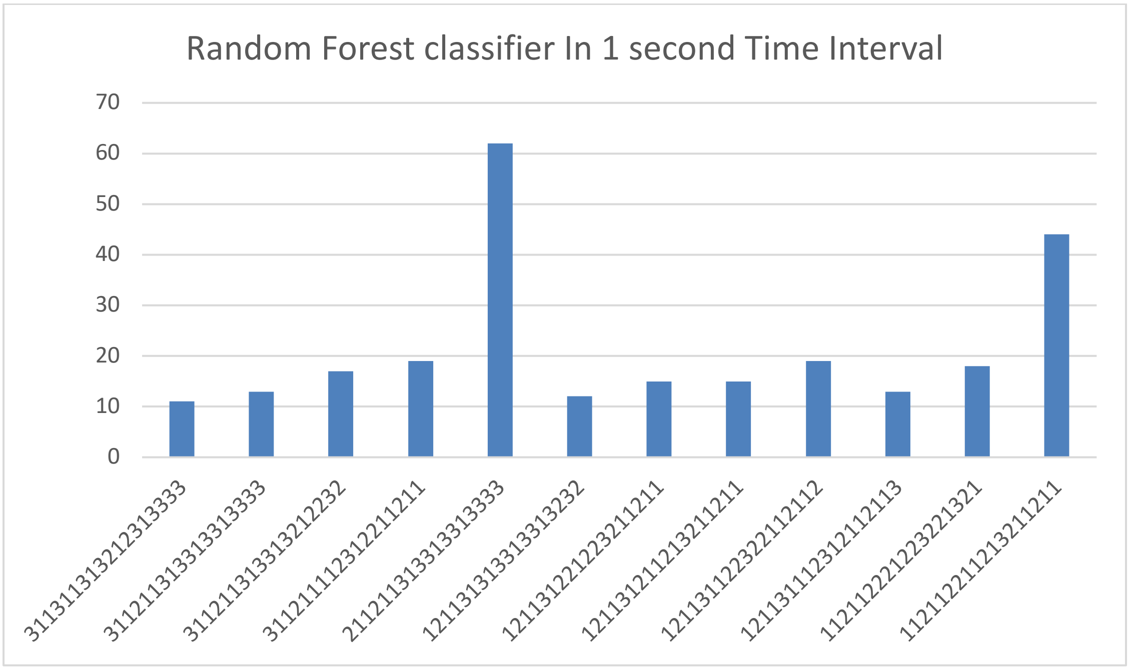


Figure 5.32 The distribution of misclassification feature patterns in the 1 second time interval for the Random Forest classifier

As can be seen in Figures 5.31 and 5.32, there are 9 feature patterns above the frequency threshold of 20 that are misclassified (up to a 50 times) by the Naïve Bayes classifier. In contrast, there are only 2 features patterns that are misclassified by the Random Forest classifier (up to 60 times) in the 1 second time interval.

The next subsection will present the results of the misclassifications in the 3 second time interval.

5.7.3 Misclassification cases for the 3 second time interval

The distribution of the frequencies of the different feature patterns for the misclassification cases for the Naïve Bayes and Random Forest classifiers in the 3 second time interval are presented in Figures 5.33 and 5.34, respectively.

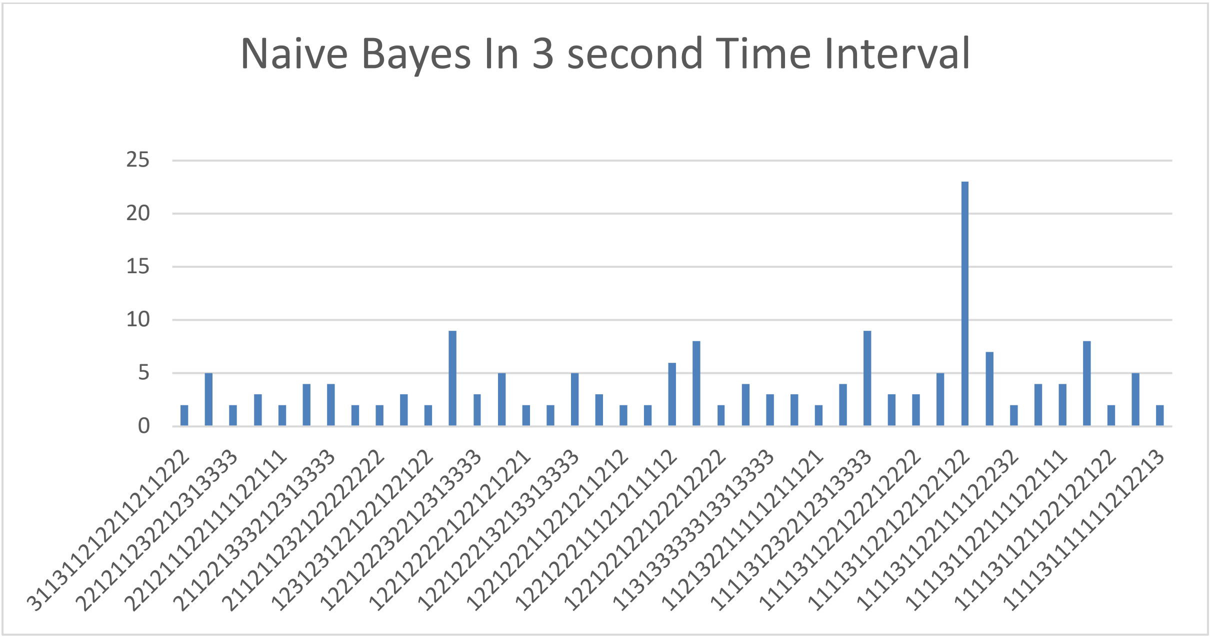


Figure 5.33 The distribution of misclassification feature patterns in the 3 second time interval for the Naïve Bayes classifier

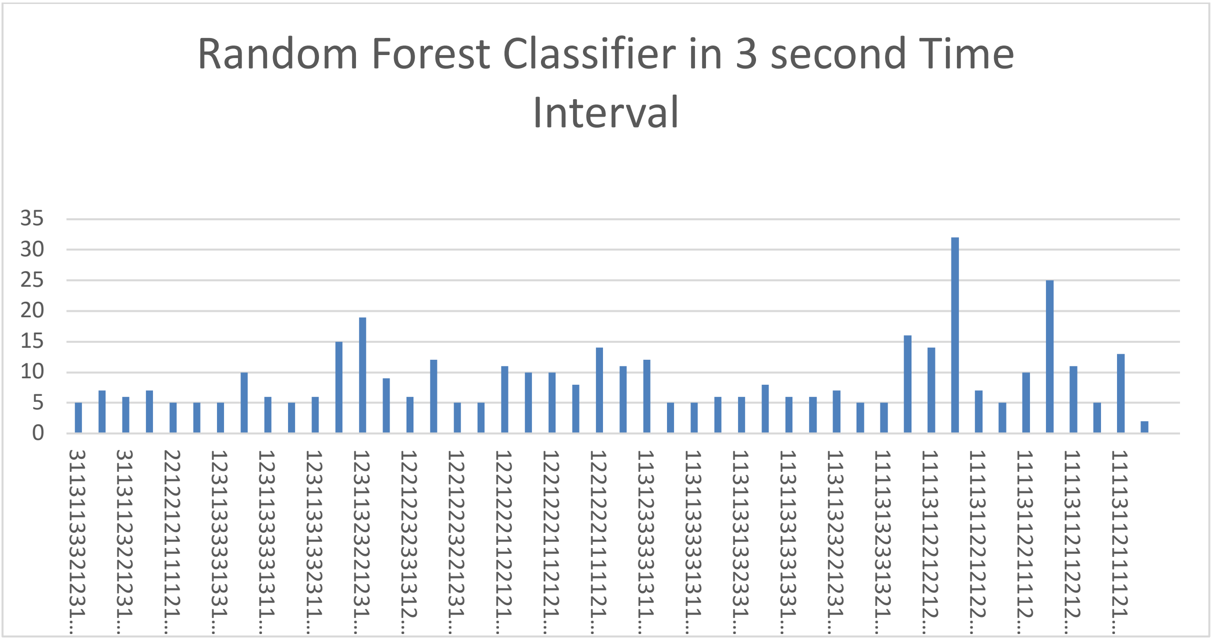


Figure 5.34 The distribution of misclassification feature patterns in the 3 second time interval for the Random Forest classifier

Since the number of records in the 3 second time interval is obviously lower than for the 1 second time interval dataset, a lower threshold was set for the feature patterns that were considered in the 3 second time interval. A lower threshold of 10 was chosen to provide a higher number of feature patterns for consideration.

As Figures 5.33 and 5.34 show, there are 2 feature patterns at or above the threshold of 10 that were misclassified (up to a maximum of 10 times) by the Naïve Bayes classifier in the 3 second time interval, and 13 feature patterns at or above the threshold for the Random Forest classifier.

The next subsection will present the results of the misclassification in the 5 second time interval.

5.7.4 Misclassification cases for the 5 second time interval

The distribution of the frequencies of the different feature patterns for the misclassification cases for the Naïve Bayes and Random Forest classifiers in the 5 second time interval are presented in Figures 5.35 and 5.36, respectively.

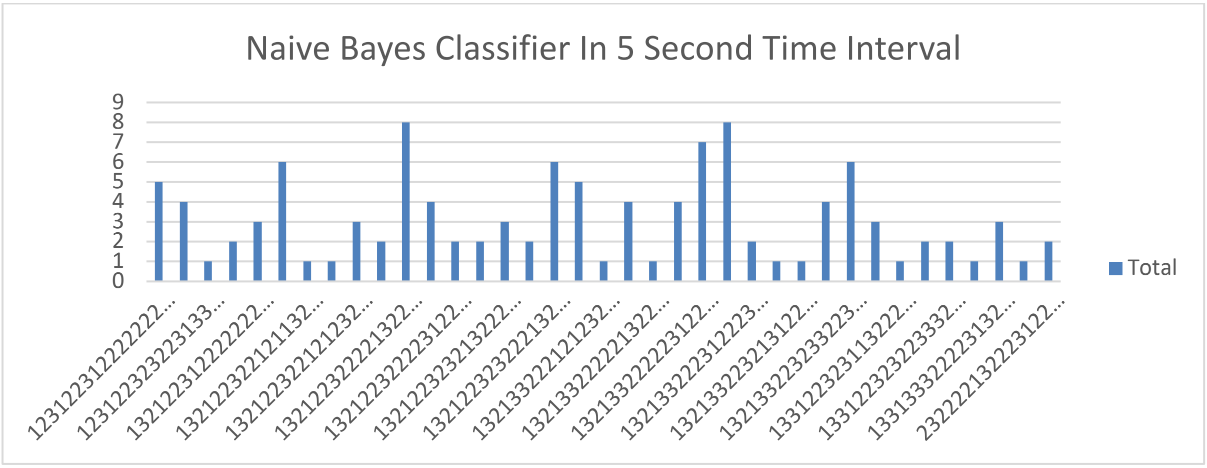


Figure 5.35 The distribution of misclassification feature patterns in the 5 second time interval for the Naïve Bayes classifier

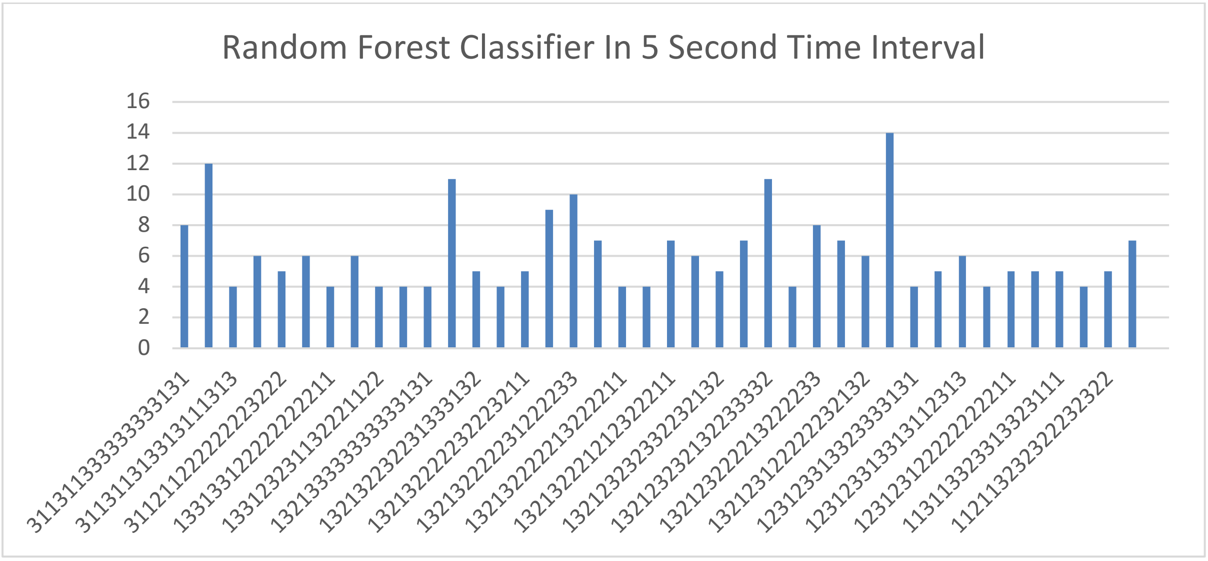


Figure 5.36 The distribution of misclassification feature patterns in the 5 second time interval for the Random Forest classifier

Considering the frequency number of misclassification cases in the 5 second time interval, a threshold of 5 was used.

As Figures 5.35 shows, there are 6 feature patterns at or above the threshold of 5 that were misclassified (up to a maximum of 8 times) by the Naïve Bayes classifier in the 5 second time interval. In contrast, Figure 5.36 shows there are 28 features patterns above the threshold of 5 that were misclassified (up to 14 times) by the Random Forest classifier.

5.7.5 Misclassification cases for the 7 second time interval

The distribution of the frequencies of the different feature patterns for the misclassification cases for the Naïve Bayes and Random Forest classifiers in the 7 second time interval are presented in Figures 5.37 and 5.38, respectively.

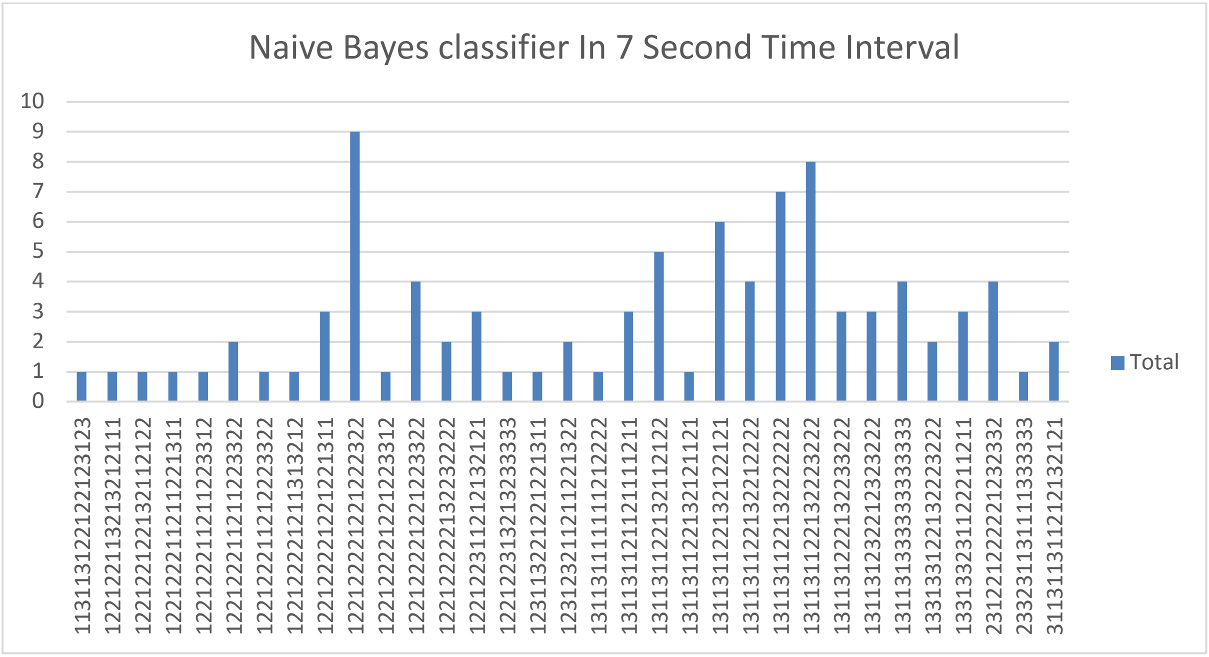


Figure 5.37 The distribution of misclassification feature patterns in the 7second time interval for the Naïve Bayes classifier

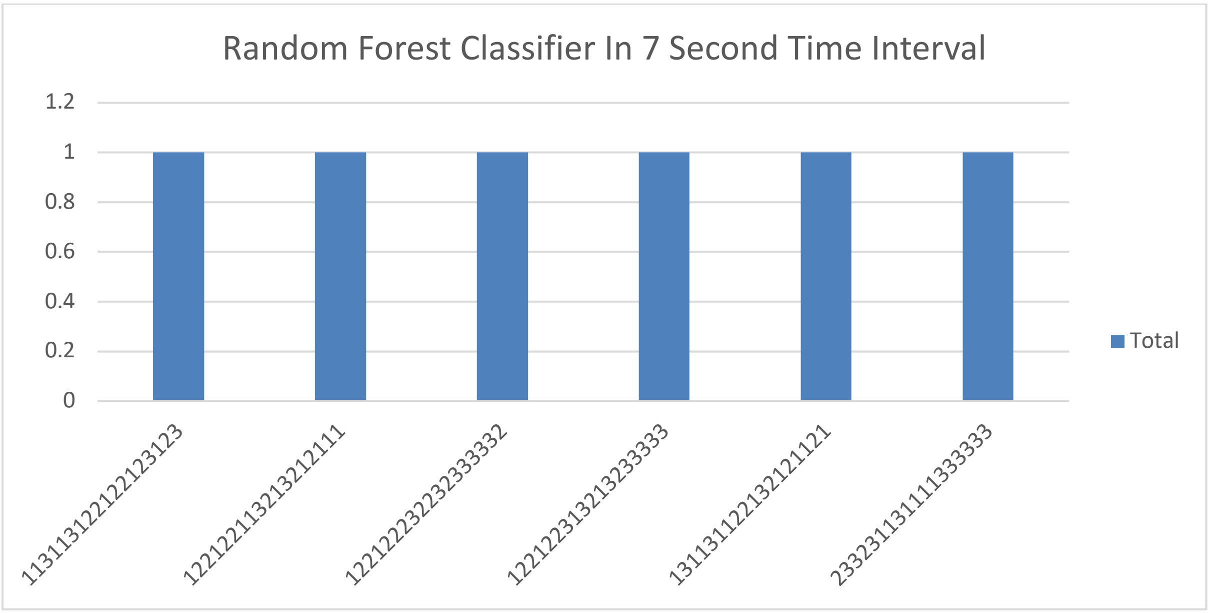


Figure 5.38 The distribution of misclassification feature patterns in the 7 second time interval for the Random Forest classifier

For the 7 second time interval, all misclassification cases were considered (i.e., a threshold of 1 was used).

As Figure 5.37 shows, there are 33 feature patterns at or above the threshold of 1 that were misclassified (up to a maximum of 9 times) by the Naïve Bayes classifier in the 7 second time interval. In contrast, Figure 5.38 shows there are only 6 features patterns at the threshold of 1 that were misclassified (up to 1 time only) by the Random Forest classifier.

5.7.5 Misclassification cases for the 10 second time interval

The distribution of the frequencies of the different feature patterns for the misclassification cases for the Naïve Bayes and Random Forest classifiers in the 10 second time interval are presented in Figures 5.39 and 5.40, respectively.

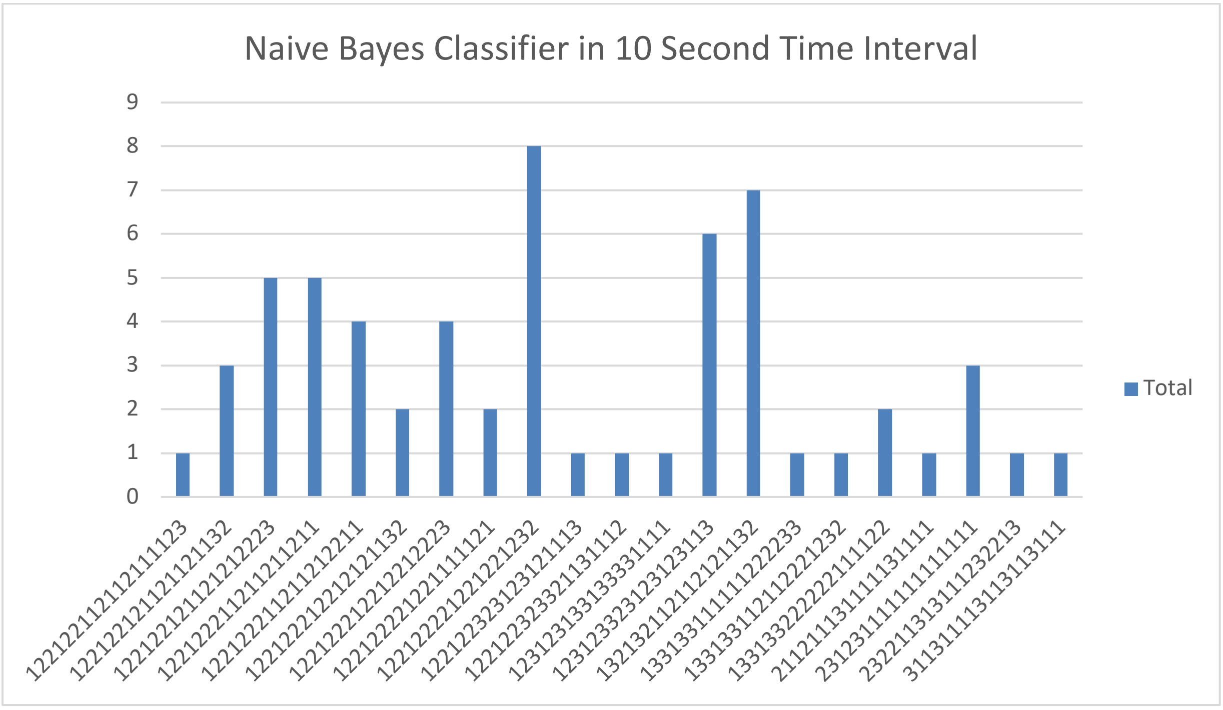


Figure 5.39 The distribution of misclassification feature patterns in the 10second time interval for the Naïve Bayes classifier

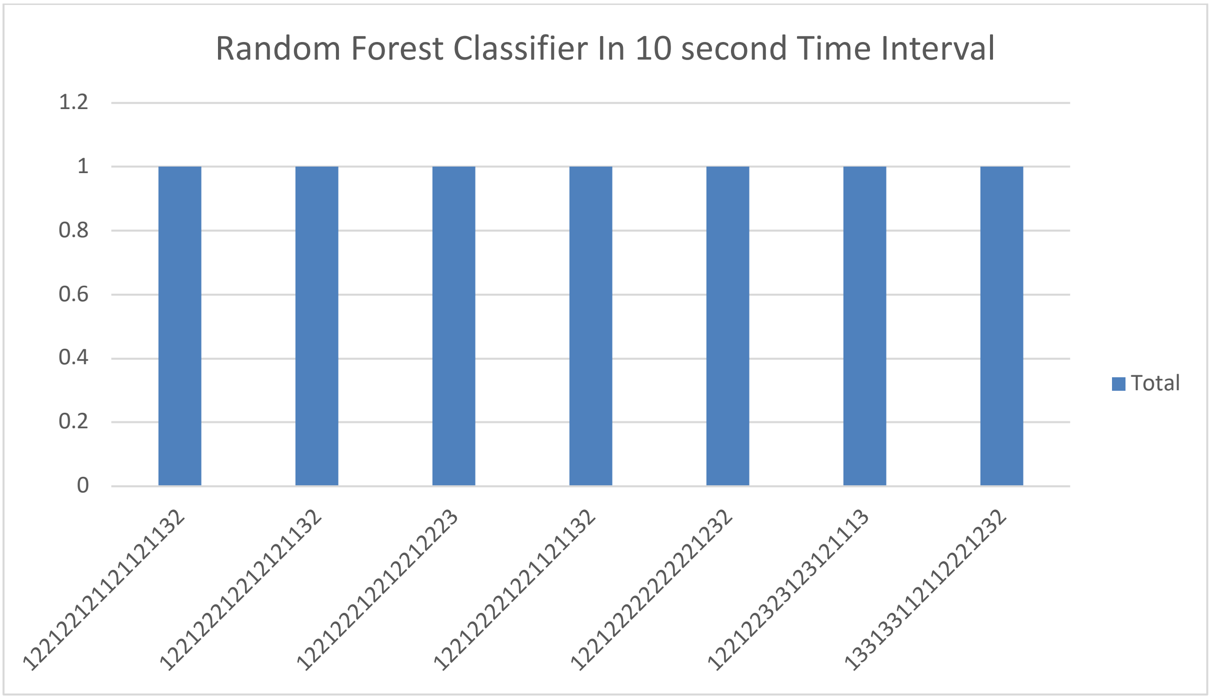


Figure 5.40 The distribution of misclassification feature patterns in the 10 second time interval for the Random Forest classifier

As for the 7 second interval, a threshold of 1 was used for the 10 second interval.

As Figure 5.39 shows, there are 12 feature patterns at or above the threshold of 1 that were misclassified (up to a maximum of 8 times) by the Naïve Bayes classifier in the 10 second time interval. In contrast, Figure 5.40 shows that there are only 7 feature patterns at the threshold of 1 that were misclassified (up to 1 time only) by the Random Forest classifier.

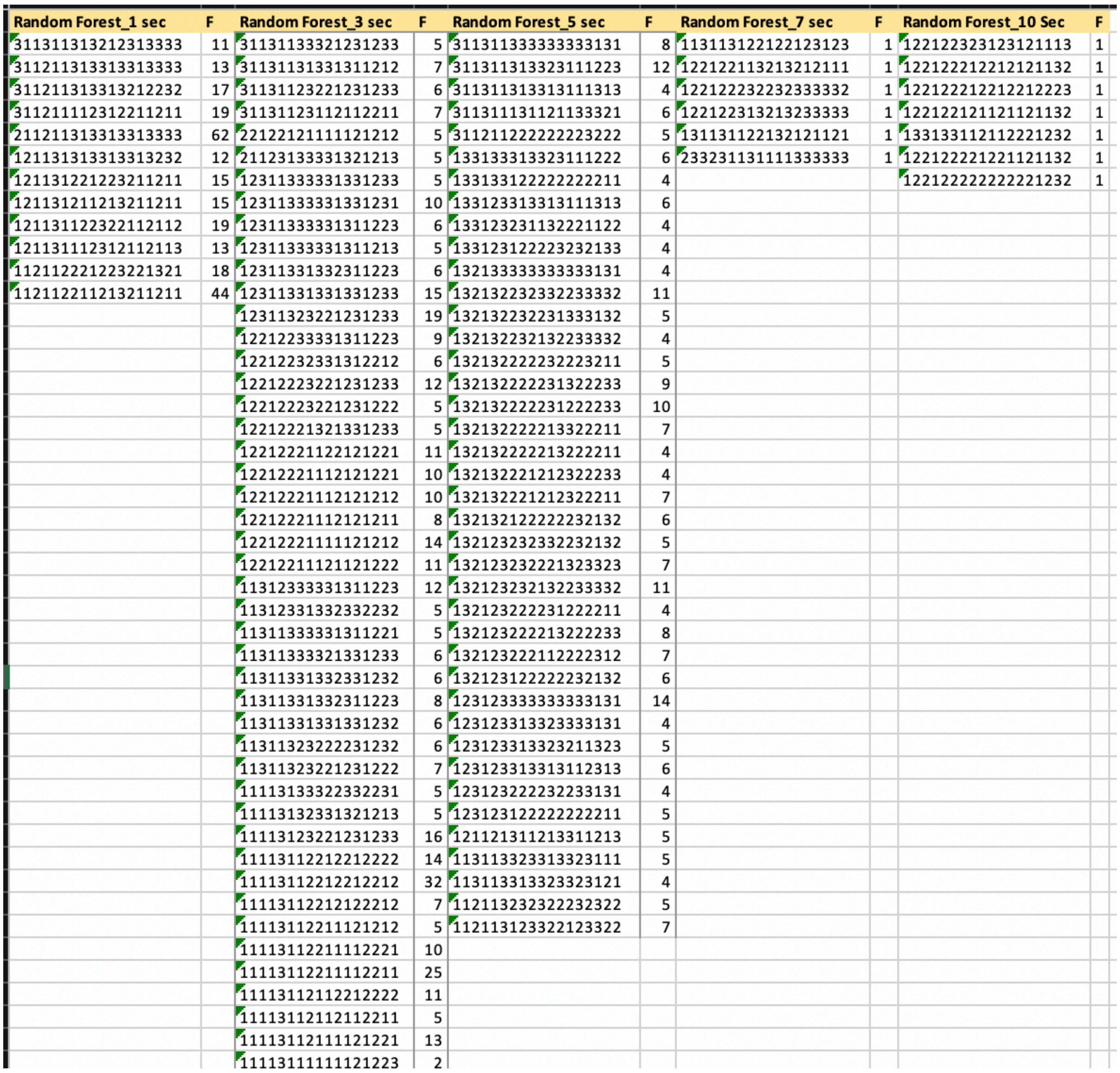
5.8 Reflection and Discussion

This section will present high-level analysis of the feature patterns that represent the misclassification cases associated with the two classifiers (Naïve Bayes and Random Forest) in the 1, 3, 5, 7 and 10 second time intervals. The aim is to understand the changes in accuracy across the time intervals and to examine whether any insights can be drawn by analyzing the feature patterns that represent the misclassification cases in the different time intervals.

Tables 5.4 and 5.5 present, for the Naïve Bayes and Random Forest classifiers respectively, in the 1, 3, 5, 7 and 10 second time intervals, the distribution of the frequencies of the different feature patterns for the misclassification cases above the selected thresholds (see Table 5.3).

Naïve Bayes_1 Sec	F	Naïve Bayes_3 Sec	F	Naïve Bayes_5 Sec	F	Naïve Bayes_7 Sec	F	Naïve Bayes_10 Sec	F
311311313313313333	21	311311212211211222	2	123122312222222211	5	113113122122123123	1	122122112112111123	1
211211313313313333	39	221221211111211112	5	123122322223233131	4	122122113213212111	1	122122121121121132	3
121131313313313232	12	221211232212313333	2	123122323223133132	1	122122122132112122	1	122122121121212223	5
121131221223211211	38	221211222211212122	3	123122323313233233	2	122122211211221311	1	122122211211211211	5
121131211213221321	25	221211122111122111	2	132122312222223122	3	122122211211223312	1	122122211211212211	4
121131211213211211	44	211311232112122222	4	132122312222232132	6	122122211211223322	2	122122212212121132	2
121131123322211211	10	211221333212313333	4	132122322121132123	1	122122211212223322	1	122122212212212223	4
121131122322122123	12	211221212111211112	2	132122322121232211	1	122122221211313212	1	122122221221111121	2
121131122322112112	19	211211232122222222	2	132122322121232233	3	122122221221221311	3	122122221221221232	8
121131112312211211	15	123123323313122212	3	132122322212232132	2	122122221221222322	9	122122323123121113	1
121131112312112113	20	123123122122122122	2	132122322221322233	8	122122221221223312	1	122122323321131112	1
121131112312112112	15	122122333313113233	9	132122322231222211	4	122122221221223322	4	123123133133331111	1
113113122322122123	11	122122232212313333	3	132122322231222322	2	122122222132232222	2	123123323123123113	6
113113122322112112	11	122122232212313232	5	132122322231322211	2	122122311212132121	3	132132112112121132	7
112112221223221321	28	122122222122121221	2	132122323213222322	3	122122313213233333	1	133133111111222233	1
112112221223211211	54	122122222111112211	2	132122323213233332	2	123113221221221311	1	133133112112221232	1
112112211213221321	43	122122213213313333	5	132122323221323323	6	123123211211221322	2	133133222222111122	2
112112211213211211	40	122122211221211221	3	132122323233232132	5	131131111111212222	1	211211131111131111	1
112112122322122123	18	122122211221211212	2	132133222121232211	1	131131121121111211	3	231231111111111111	3
112112122322122122	19	122122211121211121	2	132133222121232233	4	131131122132112122	5	232211313111232213	1
112112122322112112	10	122122211121211112	6	132133222213222233	1	131131122132121121	1	311311113113113111	1
112112112312122122	14	122122211111211112	8	132133222213322211	4	131131122132122121	6		
112112112312112113	19	122122122122212222	2	132133222231222233	7	131131122132212222	4		
112112112312112112	11	122122111211211212	4	132133222231322233	8	131131122132222222	7		
		113133333313313333	3	132133223122233332	2	131131122132223222	8		
		113133121122122122	3	132133223132232133	1	131131222132233222	3		
		112132211111211121	2	132133223213122121	1	131131232212323222	3		
		111131313323321322	4	132133223213223332	4	131131333333333333	4		
		111131232212313333	9	132133223233223332	6	133133122132223222	2		
		111131211121221121	3	133122322121222211	3	133133231122211211	3		
		111131122122212222	3	133122323113221122	1	231221222221232332	4		
		111131122122122232	5	133122323213222322	2	233231131111333333	1		
		111131122122122122	23	133122323223323112	2	311311311212132121	2		
		111131122121222122	7	133122323233222322	1				
		111131122111122232	2	133133222231322233	3				
		111131122111122211	4	133133232132233322	1				
		111131122111122111	4	232222132223122223	2				
		111131121122122232	8						
		111131121122122122	2						
		111131121121122111	5						
		111131111111212213	2						

Table 5.4 The frequencies of the different feature patterns for the misclassification of the Naïve Bayes classifier



Random Forest_1 sec	F	Random Forest_3 sec	F	Random Forest_5 sec	F	Random Forest_7 sec	F	Random Forest_10 Sec	F
311311313212313333	11	31131133321231233	5	311311333333333131	8	113113122122123123	1	122122323123121113	1
311211313313313333	13	31131131331311212	7	311311313323111223	12	122122113213212111	1	122122212212121132	1
311211313313212232	17	31131123221231233	6	311311313313111313	4	122122232232333332	1	122122212212212223	1
311211112312211211	19	31131123112112211	7	311311131121133321	6	122122313213233333	1	122122121121121132	1
211211313313313333	62	22122121111121212	5	311211222222223222	5	131131122132121121	1	133133112112221232	1
121131313313313232	12	21123133331321213	5	133133313323111222	6	233231131111333333	1	122122221221121132	1
121131221223211211	15	12311333331331233	5	133133122222222211	4			122122222222221232	1
121131211213211211	15	12311333331331231	10	133123313313111313	6				
121131122322112112	19	12311333331311223	6	133123231132221122	4				
121131112312112113	13	12311333331311213	5	133123122223232133	4				
112112221223221321	18	12311331332311223	6	132133333333333131	4				
112112211213211211	44	12311331331331233	15	132132232332233332	11				
		12311323221231233	19	132132232231333132	5				
		12212233331311223	9	132132232132233332	4				
		12212232331312212	6	132132222232223211	5				
		12212223221231233	12	132132222231322233	9				
		12212223221231222	5	132132222231222233	10				
		12212221321331233	5	132132222213322211	7				
		12212221122121221	11	132132222213222211	4				
		12212221112121221	10	132132221212322233	4				
		12212221112121212	10	132132221212322211	7				
		12212221112121211	8	132132122222232132	6				
		12212221111121212	14	132123232332232132	5				
		12212211121121222	11	132123232221323323	7				
		11312333331311223	12	132123232132233332	11				
		11312331332332232	5	132123222231222211	4				
		11311333331311221	5	132123222213222233	8				
		11311333321331233	6	132123222112222312	7				
		11311331332331232	6	132123122222232132	6				
		11311331332311223	8	123123333333333131	14				
		11311331331331232	6	123123313323333131	4				
		11311323222231232	6	123123313323211323	5				
		11311323221231222	7	123123313313112313	6				
		11113133322332231	5	123123222232233131	4				
		11113132331321213	5	123123122222222211	5				
		11113123221231233	16	121121311213311213	5				
		11113112212212222	14	113113323313323111	5				
		11113112212212212	32	113113313323323121	4				
		11113112212122212	7	112113232322232322	5				
		11113112211121212	5	112113123322123322	7				
		11113112211112221	10						
		11113112211112211	25						
		11113112112212222	11						
		11113112112112211	5						
		11113112111121221	13						
		11113111111121223	2						

Table 5.5 The frequencies of the different feature patterns for the misclassification of the Random Forest classifier

As can be seen in Tables 5.4 and 5.5, there is unique set of feature patterns for each the misclassification cases above threshold. Surprisingly, none of the misclassification feature patterns are repeated by the classifiers in the different time intervals.

Table 5.6 presents a summary showing the total number of misclassification cases and the accuracy of the models over the different time intervals (1, 3, 5, 7, and 10 seconds).

Time Interval	Total Number of Misclassification In Naïve Bayes	Total Number of Misclassification In Random Forest	Naïve Bayes accuracy	Random Forest Accuracy	Total Number of the cases
1 second	899	332	51.2473 %	81.9957 %	1844
3 seconds	234	114	59.8519%	71.1111%	675
5 seconds	136	39	64.3045 %	89.7638 %	381
7 seconds	92	6	69.8361 %	98.0328 %	305
10 seconds	78	8	75.9091 %	96.8182%	226

Table 5.6 Analysis of the accuracy of the models in different time intervals

The first row in Table 5.6 shows, in the 1 second time interval, that there are 899 misclassification cases for the Naïve Bayes classifier from the total number of 1844 cases, giving an overall accuracy of 51.2473%, while the number of misclassification cases is much lower for the Random Forest classifier (at 332 cases), giving an overall accuracy of 81.9957%.

The second row in Table 5.6 shows that in the 3 second time interval there are 234 misclassification cases for the Naïve Bayes classifier from the total number of 675 cases, giving an overall accuracy of 59.8519. In contrast, there are 114 misclassification cases for the Random Forest classifier, giving an overall accuracy of 71.111%.

Moving to the third row of Table 5.6, for the 5 second time interval there are 136 misclassification cases for the Naïve Bayes classifier from the total number of 381 cases, giving an overall accuracy of 64.3045%, while the number of misclassification cases for the Random Forest classifier is much lower (at 39 cases), leading to an overall accuracy of 89.7638%.

In the 7 second time interval, Table 5.6 shows there are 92 misclassification cases for the Naive Bayes classifier from the total number of 305 cases, giving an overall accuracy of 69.8361%. In contrast, there are only 6 misclassification cases for the Random Forest classifier, leading to an overall accuracy of 98.0328%.

Finally, in the 10 second time interval, the last row in Table 5.6 shows that there are 78 misclassification cases for the Naïve Bayes classifier, giving an overall accuracy of 75.9091 %, while there are only 8 misclassification cases for the Random Forest classifier, leading to an overall accuracy of 96.8182%.

The analysis results of the Naïve Bayes and Random Forest classifiers suggest that the total number of misclassification cases of the Naïve Bayes and the Random Forest classifiers decreases over time, though this is to be expected given the relative reduction in the number of cases overall as the time intervals become bigger. More interestingly, the accuracy of the Random Forest is higher than the Naïve Bayes Classifier across all of the time periods. For the 7 second and 10 second time intervals, the accuracy of the two classifiers is higher than the respective values for the 1, 3 and 5 second intervals. The 7 seconds interval provides the highest accuracy levels for each of the classifiers from the time periods considered.

Most interestingly, perhaps, there do not seem to be consistent sets of features patterns in the misclassification cases that reduce over the time periods (as some feature patterns 'disappear'), so little can be drawn from the analysis in terms of explaining the improved performance of the classifiers as the time intervals increase. This suggests that applying another analysis method might be useful in order to try to discover relationships between the features that might be leading to accuracy improvements as the time intervals increase. This will form the focus of the next chapter.

5.9 Summary

This chapter has reported the building of the classification models (using Naïve Bayes and Random Forest classifiers) and the analysis of their performance in five chosen time intervals. This was explored by observing the accuracy and the misclassification feature patterns of the two models. The results of the analysis show that, while the performance of the classifiers improved over time with the models performing best at the 7 seconds time interval, consistent sets of feature patterns that reduced in frequency across the time intervals were not found. This suggests that there is a need to apply another analysis method to seek to discover the relationships between the features in the models within and across the time intervals that might help to explain the improvements in the accuracy of the models. This will form the focus of the next chapter.

Chapter 6 : Cross-correlation Analysis

6.1 Introduction

As argued in chapter 5, the analysis of the results obtained from the developed classification models showed that the performance of the classifiers in terms of their accuracy and misclassification patterns improved over time. However, consistent sets of feature patterns which reduced in frequency across the time intervals, and which might therefore have explained the improvements in accuracy, were not found. To be able to discover the relationships between the features in the models within and across the time intervals, there was therefore a need to apply another analysis method to seek to improve the accuracy of the models. To this end, Chapter 6 reports the use of cross-correlation analysis that was applied to the dataset in an attempt to discover relationships between pairs of features over time.

The remainder of this chapter is structured as follows. Section 6.1 will introduce the Cross-correlation function (CCF) which was used to undertake the analysis. Section 6.2 will then present all of the techniques that were essential to undertake before applying the CCF to the initial/one second time interval dataset developed in Chapters 4 and 5. Section 6.4 will then report the results of applying the CCF function to the dataset. Section 6.5 will present the subsequent analysis of the results generated by the cross-correction analysis.

6.2 Cross-correlation Analysis

To understand the relationship between features across time in the dataset, which might help to understand the increases in accuracy in the different time intervals considered in Chapter 5, it was essential to apply a time series data analysis technique to the dataset. This section will introduce cross-correlation analysis as the chosen method, and the Cross-Correlation Function (CCF) as the instantiation of the method that was applied.

Cross-correlation is a time-series data analysis approach which aims to find the correlation between two time series (in our case, the time series associated with pairs of features from the dataset developed in Chapter 4 and used in Chapter 5). The data from the time series under consideration can be compared using cross correlation analysis to determine if there is a correlation between the two features. Shifts in time (or lags) can be applied to assess whether the features correlate at different time shifts (i.e., where feature 1 correlates with feature 2 at a lag of one second; that is, feature 1's value at a point in time may correlate with feature 2's value one second later).

For two time series, *x(i)* and *y(i)*, where *i* is the time value (0, 1, 2,…*n*-1), the cross correlation *r* at lag *k* is given by:

$$r = \sum_i \frac{[x(i) - mx * (y(i-k) - my)]}{\sum_i \sqrt{(x(i) - mx)^2} * \sum_i \sqrt{((y(i-k) - my))^2}}$$

Where *mx* and *my* are the respective means of the two time series, *x* and *y*.

When calculated for all lags (*k* = 0, 1, 2,… *n*-1), this results in a cross correlation series:

$$r(d) = \sum_i \frac{[x(i) - mx * (y(i-k) - my)]}{\sum_i \sqrt{(x(i) - mx)^2} * \sum_i \sqrt{((y(i-k) - my))^2}}$$

The denominator in this formula acts to normalize the correlation coefficients (i.e., making all value of *r(d)* between -1 and 1 (-1<=*r(d)*<=1).

The cross-correlation function identifies the level of correlation between the two series *x(i)* and *y(i)* and the following formula can be used to identify the level of correlation that is significant:

$$\frac{2}{\sqrt{n - |k|}}$$

Where n is the number of records and k is the lag that refers to the number of time intervals that divide the two-time series. If the absolute value of the level of correlation (r(d)) is greater than above formula, there is a significant correlation between the two time series.

While there are other techniques for the analysis of time-series data, including support vector machine, Fuzzy Logic, and so on, as the cross-correlation approach has been used for detecting DDoS attacks in a number of pieces of research (Huang and Yi, 2019), it was decided to apply this analysis technique to the time-series data.

Table 6.1 presents a summary of existing work in this area that has made use of cross-correlation analysis. That CCF has been used in the analysis of time series data for DDoS attaches suggests that it is a useful and, at least somewhat, established analysis technique that has been used in a number of ways/contexts by researchers in the area.

No	Title of Paper	A Brief Description of the Paper's Use of the CCF Approach	Reference
1	CCID: Cross-Correlation Identity Distinction Method for Detecting Shrew DDoS	CCF was used to calculate the cross-correlation between the attack flow and the normal flow in three different situations	Huang and Yi (2019)
2	Combining Cross-Correlation and Fuzzy Classification to Detect Distributed Denial-of-Service Attacks	CCF was used to discover changes in the correlation of outgoing and incoming traffic caused by DDoS attack	Wei *et al.* (2006)
3	Cross-correlation Based Synchronization Mechanism of LDDoS Attacks	CCF was used to estimate and compute the delay between two DDoS attack series	(Zhijun *et al.*, 2014)
4	Self-similarity Based DDoS Attack Detection Using Hurst Parameter	CCF was used to exploit self-similarity features of the DDoS attack to distinguish the attack traffic from legitimate traffic	(Deka and Bhattacharyya, 2016)

Table 6.1 Summary of use of cross-correlation approach in detecting DDoS attacks

Before applying the CCF to the dataset, it was necessary to transform the dataset as a result of applying stationarity tests (see section 6.3) to the data. Section 6.3 will report all of the stages undertaken to complete the transformation task on the dataset.

6.3 Transformation Tasks on the Time Series Dataset

In order to transform the dataset's structure to make it suitable for cross-correlation analysis, a series of actions were applied to the time series data. To do this, a suitable programming language had to be selected.

Various programming languages are commonly used in time series data analysis, including Python (2019), R (2019), MATLAB (2019) and Weka (Frank et al. 2016). This study used R to implement all stages of the transformation task and to undertake the cross-correlation analysis, using R's CCF function. Most of the available tools offer very similar functionality; R is, though, free and widely used, making it an appropriate choice for this work. R offers a broad variety of statistical capabilities, including time series analysis, classification, clustering and graphical techniques (R, 2019). Moreover, it has been broadly used for time series analysis and it easy to use (Huang and Yi, 2019). For all of these reasons, it was seen as being an appropriate choice as the analysis environment for use in this part of the study.

Figure 6.1 presents a high-level view of the different steps that formed the data transformation process.

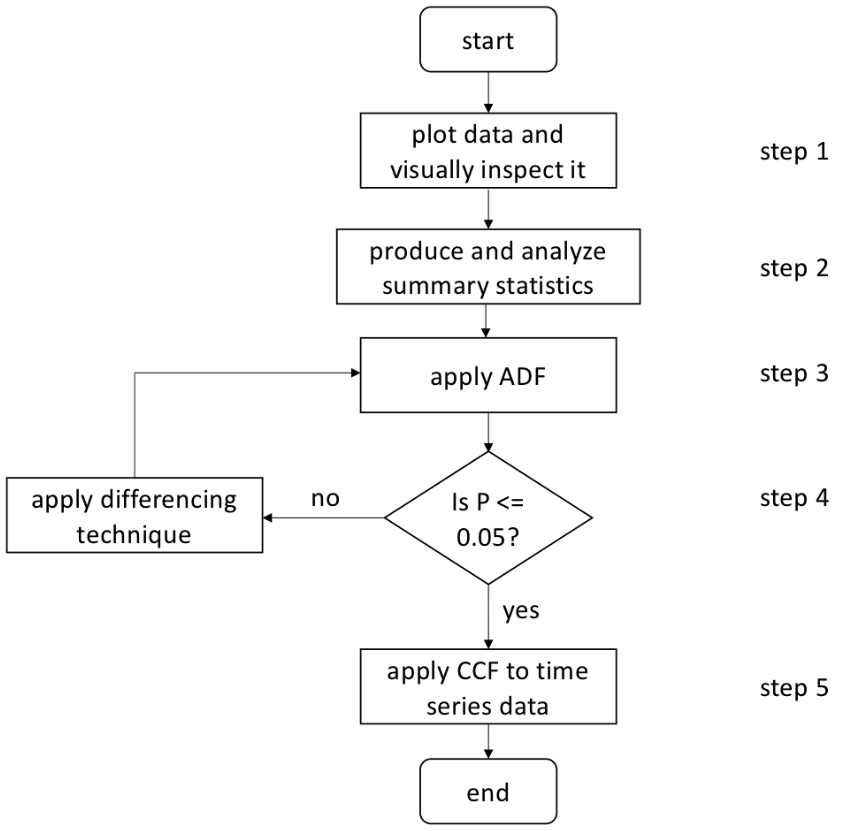
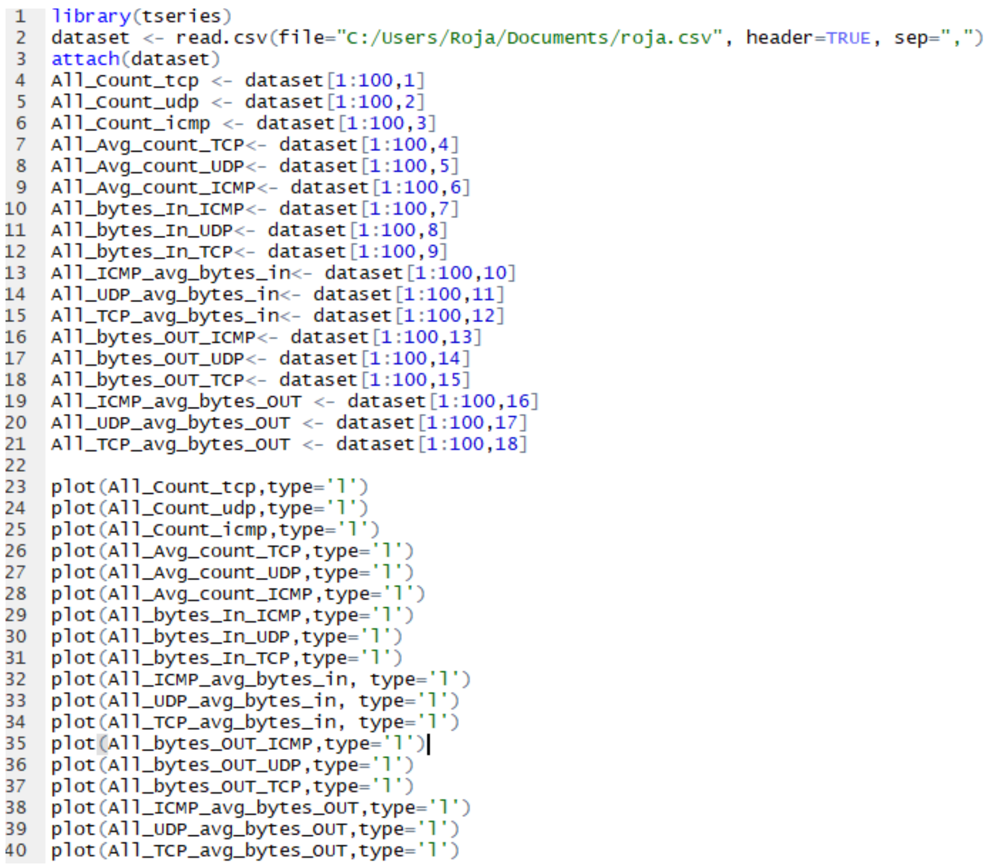

start
plot data and visually inspect it
step 1
produce and analyze summary statistics
step 2
apply ADF
step 3
apply differencing technique
no
Is P <= 0.05?
step 4
yes
apply CCF to time series data
step 5
end

Figure 6.1 Transformation process applied to the time series dataset for each feature

First, a test was applied to assess the stationarity of the data. A stationarity process is used in time series analysis to make the statistical properties of the data, such as mean and variance, constant over time – this is essential when undertaking cross-correlation analysis. It was important to test the stationarity of the time series dataset because sometimes time series data may contain a trend – an upwards or downwards shift in the dataset that results in a varying mean and in a seasonality, meaning regular change that occurs in a set period (such as a calendar year) that consequently results in changes in variance (i.e., non-stationarity). These effects are the most common reasons for a process being non-stationary.

According to the literature on time series data analysis, non-stationary time series have to be transformed to stationary time series prior to the modelling process (in our case, cross-correction analysis) (see, for example, Grogan (2018)). There are three different methods to check the stationarity of a time series, which vary in sophistication (Kang, 2017); from simplest to most sophisticated, they are: plotting the data and visually inspecting the plot (step 1); splitting the data into intervals and comparing the interval-based summary statistics for obvious or significant differences (step 2); and applying a specific statistical test (step 3).

In step 1, it was essential to first understand the total level of the variability of the time series. Therefore, 100 instances of the time series data were selected for plotting purposes to see whether there was any noticeable trend in the generated graphs (indicating non-stationarity). Figure 6.2 presents an R script that was designed to plot all of the features of the time series; this resulted in the generation of 19 graphs (one for each feature). As the number of the generated graphs was high, and all had a similar (lack of) trend, only one example is provided here, for illustrative purposes. Figure 6.3 presents an extract of the result of applying the R code to the original dataset to generate the graph for the 'UDP_avg_bytes_OUT' from the full time series dataset.



```
library(tseries)
dataset <- read.csv(file="C:/Users/Roja/Documents/roja.csv", header=TRUE, sep=",")
attach(dataset)
All_Count_tcp <- dataset[1:100,1]
All_Count_udp <- dataset[1:100,2]
All_Count_icmp <- dataset[1:100,3]
All_Avg_count_TCP<- dataset[1:100,4]
All_Avg_count_UDP<- dataset[1:100,5]
All_Avg_count_ICMP<- dataset[1:100,6]
All_bytes_In_ICMP<- dataset[1:100,7]
All_bytes_In_UDP<- dataset[1:100,8]
All_bytes_In_TCP<- dataset[1:100,9]
All_ICMP_avg_bytes_in<- dataset[1:100,10]
All_UDP_avg_bytes_in<- dataset[1:100,11]
All_TCP_avg_bytes_in<- dataset[1:100,12]
All_bytes_OUT_ICMP<- dataset[1:100,13]
All_bytes_OUT_UDP<- dataset[1:100,14]
All_bytes_OUT_TCP<- dataset[1:100,15]
All_ICMP_avg_bytes_OUT <- dataset[1:100,16]
All_UDP_avg_bytes_OUT <- dataset[1:100,17]
All_TCP_avg_bytes_OUT <- dataset[1:100,18]

plot(All_Count_tcp,type='l')
plot(All_Count_udp,type='l')
plot(All_Count_icmp,type='l')
plot(All_Avg_count_TCP,type='l')
plot(All_Avg_count_UDP,type='l')
plot(All_Avg_count_ICMP,type='l')
plot(All_bytes_In_ICMP,type='l')
plot(All_bytes_In_UDP,type='l')
plot(All_bytes_In_TCP,type='l')
plot(All_ICMP_avg_bytes_in, type='l')
plot(All_UDP_avg_bytes_in, type='l')
plot(All_TCP_avg_bytes_in, type='l')
plot(All_bytes_OUT_ICMP,type='l')
plot(All_bytes_OUT_UDP,type='l')
plot(All_bytes_OUT_TCP,type='l')
plot(All_ICMP_avg_bytes_OUT,type='l')
plot(All_UDP_avg_bytes_OUT,type='l')
plot(All_TCP_avg_bytes_OUT,type='l')
```

Figure 6.2 R script for plotting the data

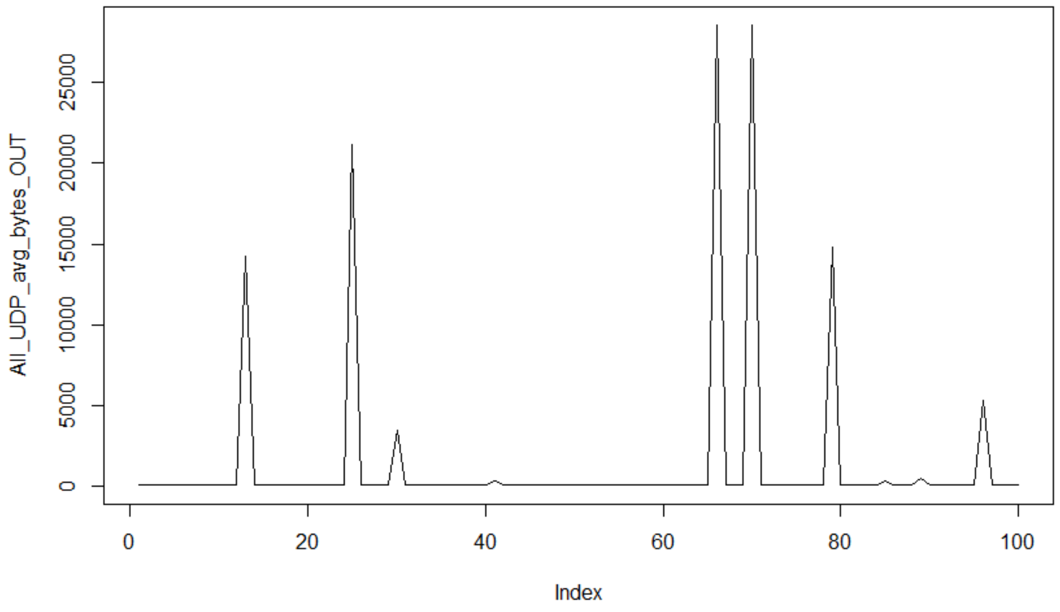


Figure 6.3 Screenshot of an example of stationarity for the ‘UDP_avg_bytes_OUT’ feature

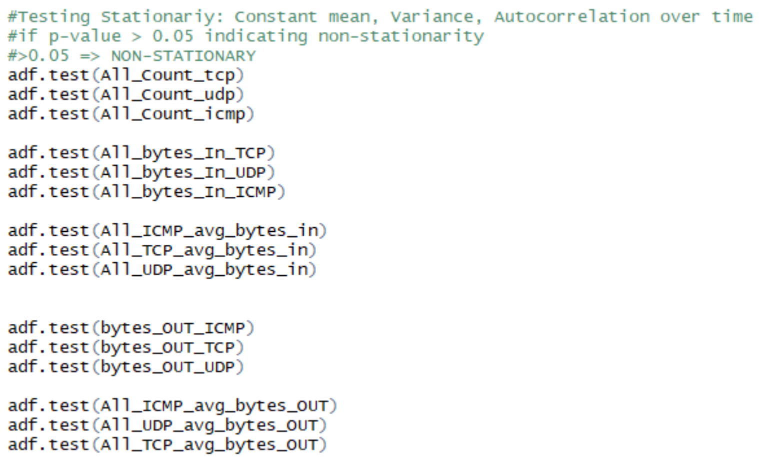
As Figure 6.3 shows, there is no obvious trend in the time series data for the example feature (UDP_avg_bytes_OUT') indicating that the time series for this feature is stationary. If, however, the plotted time series data for any feature had been found to have a trend it would have needed to be removed (i.e., there would have needed to be a transformation applied to the data) before applying the CCF function. This process was undertaken for each feature and the time series data for each feature was found to be stationary.

In step 2, summary statistics were produced to assess whether there were significant differences in them across time intervals in the time series data, as a secondary check of stationarity. This also showed differences that indicated that the data were stationary for each feature. (These data are not reproduced here for reasons of space).

In step 3, a statistical test was used to check that the expectation of the stationarity function was met. As can be seen from Figure 6.1, the test would have been used repeatedly after a differencing technique had been applied until a terminating condition was met (indicating that the time series for each feature had been transformed and met the stationarity test).

There are different stationarity tests that may be applied to time series data, such as the Kwiatkowski–Phillips–Schmidt–Shin (KPPS) and Augmented Dicky-Fuller (ADF) tests (Shumway and Stoffer, 2016). It was decided to apply the ADF statistical test as it is widely used and may avoid the contradictory results that have been identified in relation to KPSS (see Maddala and Kim, 1998).

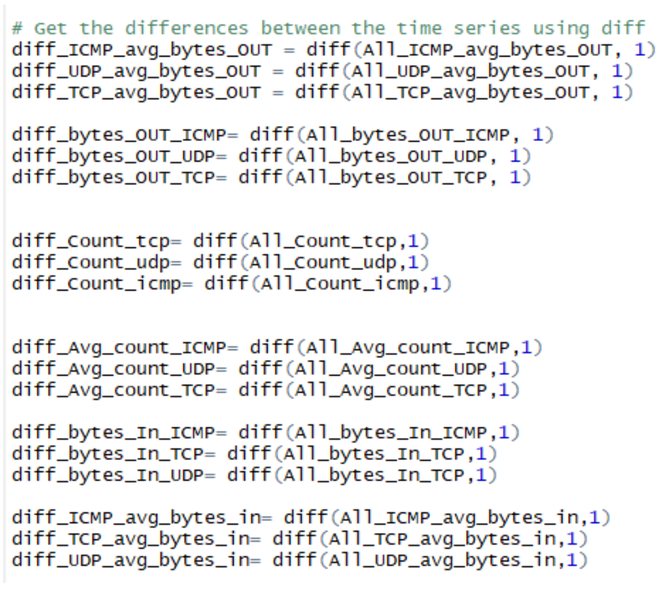
There are two types of hypothesis in the ADF test: a null hypothesis that assumes that the time series possess a unit root (i.e., it is a time-dependent/stochastic process); and the alternative hypothesis that there is no unit root present in the time series data (i.e., that the process producing the data is stationary). If the alternative hypothesis fails to be rejected, the time series is stationary and free of time-dependent structure. If it is rejected (i.e., the null hypothesis is true) it means that the time series is non-stationary, and it does have a time-dependent structure. A p-value equal to or less then 0.05 signifies the acceptance of the alternative hypothesis. The lower (and more negative) the p-value is, the stronger the rejection of the null hypothesis. Figure 6.4 presents the R script that was designed to apply the ADF test to the time series data.



```
#Testing Stationariy: Constant mean, Variance, Autocorrelation over time
#if p-value > 0.05 indicating non-stationarity
#>0.05 => NON-STATIONARY
adf.test(All_Count_tcp)
adf.test(All_Count_udp)
adf.test(All_Count_icmp)

adf.test(All_bytes_In_TCP)
adf.test(All_bytes_In_UDP)
adf.test(All_bytes_In_ICMP)

adf.test(All_ICMP_avg_bytes_in)
adf.test(All_TCP_avg_bytes_in)
adf.test(All_UDP_avg_bytes_in)


adf.test(bytes_OUT_ICMP)
adf.test(bytes_OUT_TCP)
adf.test(bytes_OUT_UDP)

adf.test(All_ICMP_avg_bytes_OUT)
adf.test(All_UDP_avg_bytes_OUT)
adf.test(All_TCP_avg_bytes_OUT)
```

Figure 6.4 R script for ADF test

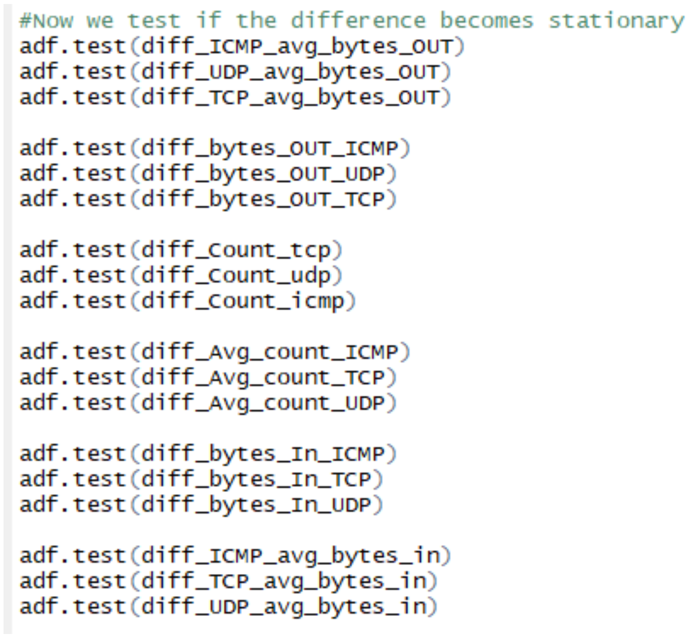
In step 4, a differencing technique was (or would have been) applied to the time series to transform any non-stationary data to stationary time series data. Differencing is one of the most popular methods for transforming the stationarity of time series data. These techniques remove trends and variance from time series by calculating consecutive terms that result in stabilizing the mean. Figure 6.5 presents an example of the R code that was applied to the time series to calculate the differenced values at the specified lag (time interval) – in this case one second.



```
# Get the differences between the time series using diff
diff_ICMP_avg_bytes_OUT = diff(All_ICMP_avg_bytes_OUT, 1)
diff_UDP_avg_bytes_OUT = diff(All_UDP_avg_bytes_OUT, 1)
diff_TCP_avg_bytes_OUT = diff(All_TCP_avg_bytes_OUT, 1)

diff_bytes_OUT_ICMP= diff(All_bytes_OUT_ICMP, 1)
diff_bytes_OUT_UDP= diff(All_bytes_OUT_UDP, 1)
diff_bytes_OUT_TCP= diff(All_bytes_OUT_TCP, 1)


diff_Count_tcp= diff(All_Count_tcp,1)
diff_Count_udp= diff(All_Count_udp,1)
diff_Count_icmp= diff(All_Count_icmp,1)


diff_Avg_count_ICMP= diff(All_Avg_count_ICMP,1)
diff_Avg_count_UDP= diff(All_Avg_count_UDP,1)
diff_Avg_count_TCP= diff(All_Avg_count_TCP,1)

diff_bytes_In_ICMP= diff(All_bytes_In_ICMP,1)
diff_bytes_In_TCP= diff(All_bytes_In_TCP,1)
diff_bytes_In_UDP= diff(All_bytes_In_TCP,1)

diff_ICMP_avg_bytes_in= diff(All_ICMP_avg_bytes_in,1)
diff_TCP_avg_bytes_in= diff(All_TCP_avg_bytes_in,1)
diff_UDP_avg_bytes_in= diff(All_UDP_avg_bytes_in,1)
```

Figure 6.5 R script for applying the differencing function

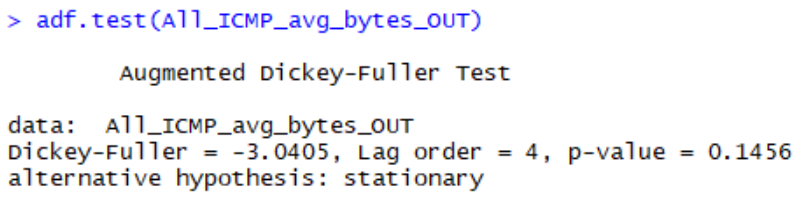
Once the differenced values had been calculated, the ADF test was again applied to the time series to see of the data was now stationary. Figure 6.6 presents the R script that was designed for the ADF test on the time series after calculating the differenced values.



```
#Now we test if the difference becomes stationary
adf.test(diff_ICMP_avg_bytes_OUT)
adf.test(diff_UDP_avg_bytes_OUT)
adf.test(diff_TCP_avg_bytes_OUT)

adf.test(diff_bytes_OUT_ICMP)
adf.test(diff_bytes_OUT_UDP)
adf.test(diff_bytes_OUT_TCP)

adf.test(diff_Count_tcp)
adf.test(diff_Count_udp)
adf.test(diff_Count_icmp)

adf.test(diff_Avg_count_ICMP)
adf.test(diff_Avg_count_TCP)
adf.test(diff_Avg_count_UDP)

adf.test(diff_bytes_In_ICMP)
adf.test(diff_bytes_In_TCP)
adf.test(diff_bytes_In_UDP)

adf.test(diff_ICMP_avg_bytes_in)
adf.test(diff_TCP_avg_bytes_in)
adf.test(diff_UDP_avg_bytes_in)
```

Figure 6.6 R script for testing the differenced value being stationary

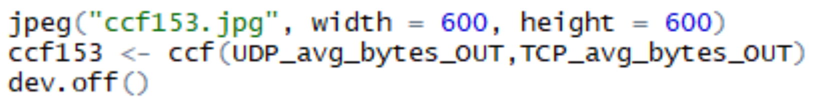
Figure 6.7 shows an example of the output of applying the R code of the ADF test. As can be seen, in this example the *p*-value is less than the 0.05 threshold, which indicates that the time series for this feature ('ICMP_avg_bytes_out') is stationary.



```
> adf.test(All_ICMP_avg_bytes_OUT)

        Augmented Dickey-Fuller Test

data:  All_ICMP_avg_bytes_OUT
Dickey-Fuller = -3.0405, Lag order = 4, p-value = 0.1456
alternative hypothesis: stationary
```

Figure 6.7 Screenshot of the output of the ADF test for 'ICMP_avg_bytes_out'

Finally, in step 5, once all of the time series data for each feature had (if necessary) been transformed into a stationary form, the CCF function was applied to the data to discover the relationship between feature pairs. Figure 6.8 presents an example of the R code that was designed to implement the CCF function to generate the required graphs for a selected feature pair ('UDP_avg_bytes_OUT' and 'TCP_avg_bytes_OUT'). This process was repeated for each feature pair to provide a full set of graphs for analysis.



```
jpeg("ccf153.jpg", width = 600, height = 600)
ccf153 <- ccf(UDP_avg_bytes_OUT,TCP_avg_bytes_OUT)
dev.off()
```

Figure 6.8 R script for CCF applied to 'UDP_avg_bytes_OUT' and 'TCP_avg_bytes_OUT' feature pair

6.4 Organising the Results of Cross-correlation Analysis

Applying the CCF function to the different feature pairs in the time-series, (step 5, described in section 6.3) resulted in 152 graphs being generated. To analyse the results, it was necessary to sort the CCF features-graphs into suitable categories. This section will report the process of sorting the CCF graphs.

Before going into a detailed explanation of the sorting (or categorizing) of the CCF graphs, an example of a CCF graph – relating to two the features 'Count_TCP' and 'Count_ICMP' – is presented in Figure 6.9 in order to explain how the CCF graphs should be interpreted. The level of correlation between the two features (Count_TCP and Count_ICMP) is calculated using the correlation formula (see section 6.2) applied to the values of the two features at lag values between -10 and +10 seconds. Negative lags represent the time series for the first feature being shifted backward in time (for values from -1 to -10 in one second intervals) in relation to the second feature's time series data; whereas positive lags represent the time series for the first feature being shifted forward in time (for values from +1 to +10 in one second intervals) in relation to the second feature's time series data. The time series values at each time lag are then used to calculate the correlation between the time series for that lag. This results in correlation values between the feature pairs at lags from -10 to +10. The significance level for the correlations (positive and negative) is identified by a dotted line on the resulting graph that is produced by the CCF function. As an example, Figure 6.9 shows that there are significant positive correlations (i.e., values that go above the top dotted line) at time lags -3 and at time lag +9. (Significant negative correlations would be signified by values that go below the bottom dotted line).

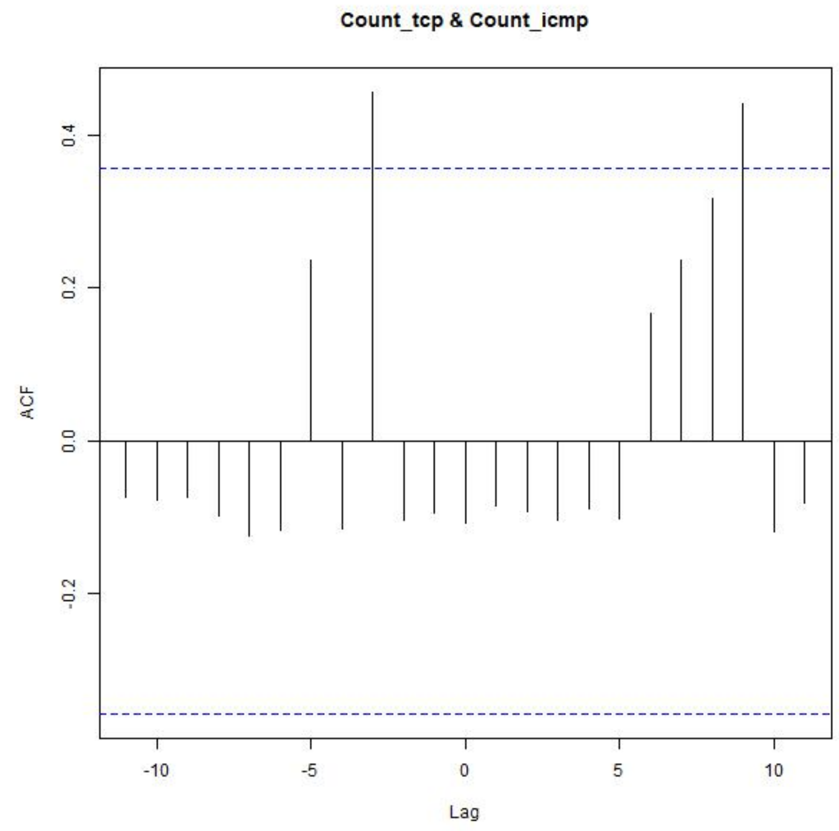


Figure 6.9 An example of CCF features-graph

Figure 6.10 presents the steps through which the CCF feature pair graphs were organized to allow subsequent analysis.

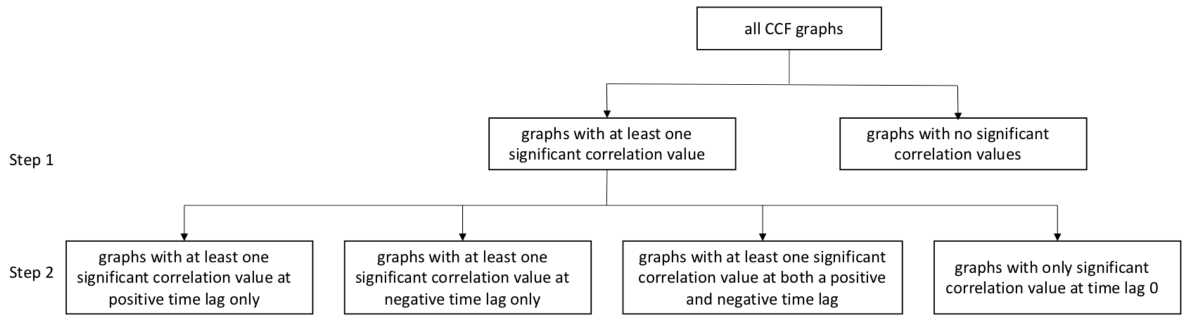

all CCF graphs
Step 1
graphs with at least one significant correlation value
graphs with no significant correlation values
Step 2
graphs with at least one significant correlation value at positive time lag only
graphs with at least one significant correlation value at negative time lag only
graphs with at least one significant correlation value at both a positive and negative time lag
graphs with only significant correlation value at time lag 0

Figure 6.10 The process of sorting the CCF feature-pair graphs

In step 1, the graphs were divided into two categories. Those graphs where there was at least one significant cross-correlation were assigned to the category of 'significant cross-correlation value'; the rest of the graphs were allocated to the category of 'no significant cross-correlation value'. Only the graphs in the 'significant cross-correlation value' group were considered for further analysis.

In step 2, the graphs in the 'significant cross-correlation value' were sorted into four sub-groups, as can be seen in Figure 6.10. The first group comprised those graphs where there was at least one significant correlation at a positive time lag only. The second group comprised those graphs that had at least one significant correlation at a negative time lag only. The third group comprised those graphs that had at least one significant correlation at both a positive time lag and a negative time lag. The fourth, and final, group comprised those graphs where the only significant correlation was at lag 0.

The next section will present the matrix created from the correlation values of these graphs and will provide an analysis of the results.

6.5 Analysis of the Outcome of the CCF Graphs

This section will present an analysis of the matrix created from the correlation values identified from the CCF graphs derived from the approach described in section 6.4.

	Count_tcp	Count_udp	Count_icmp	Avg_count_TCP	Avg_count_UDP	Avg_count_ICMP	bytes_In_ICMP	bytes_In_UDP	bytes_In_TCP	ICMP_avg_bytes_in	UDP_avg_bytes_in	TCP_avg_bytes_in	bytes_OUT_ICMP	bytes_OUT_UDP	bytes_OUT_TCP	ICMP_avg_bytes_OU	UDP_avg_bytes_OUT	TCP_avg_bytes_OUT
Count_tcp	-	NS	9	2	NS	9	8	8	7	8	8	7	8	8	NP	8	8	NP
Count_udp	NS	-	NS	NS	0	NS	NP	NP	NP	NP	NP	NP	NS	NS	NS	NS	NS	NS
Count_icmp	-3	NS	-	3	NS	0	NP	0	10	NP	NS	10	NP	NP	3	NP	NP	3
Avg_count_TCP	-2	NS	-9	-	NS	9	8	8	7	8	8	7	8	8	NP	8	8	NP
Avg_count_UDP	NS	0	NS	NN	-	NS	NP	NS	NP	NP	NS	NP	NP	NS	NS	NP	NS	NS
Avg_count_ICMP	NN	NS	0	NN	NS	-	NS	NP	NP	NS	NP	NP	NP	NP	NP	NP	NP	NP
bytes_In_ICMP	-2	-7	-10	-2	-5	NS	NS	0	0	0	0	0	0	8	0	0	8	0
bytes_In_UDP	NN	-7	0	NN	NS	-9	0	-	7	0	0	7	0	8	NP	0	8	NP
bytes_In_TCP	NN	-4	-1	NN	-4	-2	0	NN	-	0	NP	0	0	1	NP	0	1	NP
ICMP_avg_bytes_in	-2	-5	-10	-2	-4	NS	0	NN	0	-	0	0	0	8	0	0	8	NP
UDP_avg_bytes_in	NN	-7	NS	NN	NS	-9	0	0	-7	0	-	7	0	8	NP	0	8	NP
TCP_avg_bytes_in	NN	-4	-1	NN	-4	-2	0	NN	0	0	NN	-	0	1	NP	1	1	NP
bytes_OUT_ICMP	NN	NS	-10	NN	-1	-1	0	0	0	0	0	0	-	0	NP	0	0	NP
bytes_OUT_UDP	NN	NS	-1	NN	NS	-1	NN	NN	NN	NN	NN	NN	0	-	NP	0	0	NP
bytes_OUT_TCP	-2	NS	-9	-2	NS	-9	0	-8	-7	0	-8	-7	-8	-8	-	8	8	0
ICMP_avg_bytes_OUT	NN	NS	-10	NN	-1	-1	0	0	0	0	0	NN	0	0	NN	-	0	NP
UDP_avg_bytes_OUT	NN	NS	-1	NN	NS	-1	NN	NN	-1	NN	NN	NN	0	0	NN	0	-	NP
TCP_avg_bytes_OUT	-2	NS	-9	-2	NS	-9	0	-8	-7	-8	-8	-7	-8	-8	0	-8	-8	-

Time Lag (modulus)	Frequency
1	16
2	12
3	4
4	4
5	3
6	0
7	15
8	37
9	10
10	6

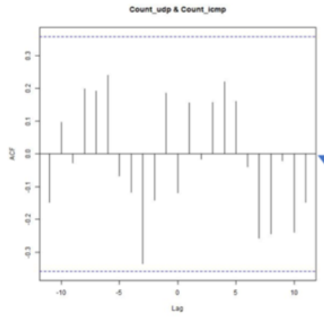
KEY
NN: there is no negative lag in the considered range for which there is a significant correlation between the feature pair
NP: there is no positive lag in the considered range for which there is a significant correlation between the feature pair
NS: there is no significant correction between the feature pair at any negative or positive lag in the considered range
0: there is only a significant correlation between the feature pair at lag 0

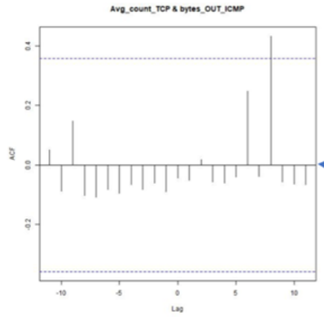
Figure 6.11 The matrix of correlation values extracted from the CCF graphs

Figure 6.11 presents the correlation data, extracted from the CCF graphs, between feature pairs at different time lags (in the range -10 to +10 seconds). The values in the cells in the matrix represent, in the cases where integer values are recorded, the positive or negative lag at which there is the highest significant (positive or negative) correlation value recorded between the relevant feature pair. The other values in the matrix represent instances of the four different cases set out in the key shown as part of Figure 6.11. Where values of 'NN' are shown, it means that there is no negative lag in the considered range for which there is a significant correlation between the relevant feature pair. Similarly, where values of 'NP' are shown, it means that there is no positive lag in the considered range for which there is a significant correlation between the relevant feature pair. Values of 'NS' signify cases where there is no significant correlation between the relevant feature pair at any negative or positive lag in the considered range (-10 to +10 seconds). Values of '0' mean that there is only a significant correlation between the relevant feature pair at lag 0.

Figure 6.11 also shows a frequency table (derived from the values in the cells in the matrix) reflecting the modulus values of the different time lags (i.e., the frequency of 20 for time lag modulus 1 representing the sum of all -1 and +1 values in the matrix). As can be seen from the frequency table, the highest value (37) occurs at time lag (modulus) 8 – that is, when taken together there are more instances of the most significant correlations between feature pairs occurring at time lag -8 and +8 than at any other (modulus) time lag modulus.

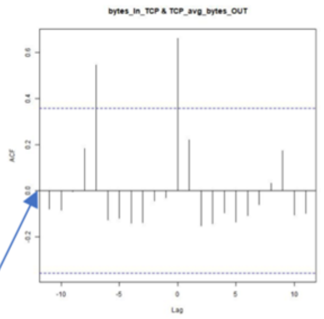
Figure 6.12 then gives an example of a CCF graph relating to each of the cases presented in the Figure 6.11 key (NN, NP, NS,0) as illustrations.





	Count_tcp	Count_udp	Count_icmp	Avg_count_TCP	Avg_count_UDP	Avg_count_ICMP	bytes_In_ICMP	bytes_In_UDP	bytes_In_TCP	ICMP_avg_bytes_in	UDP_avg_bytes_in	TCP_avg_bytes_in	bytes_OUT_ICMP	bytes_OUT_UDP	bytes_OUT_TCP	ICMP_avg_bytes_OUT	UDP_avg_bytes_OUT	TCP_avg_bytes_OUT
Count_tcp	-	NS	9	2	NS	9	8	8	7	8	8	7	8	8	NP	8	8	NP
Count_udp	NS	-	NS	NS	0	NS	NP	NP	NP	NP	NP	NP	NS	NS	NS	NS	NS	NS
Count_icmp	-3	NS	-	3	NS	0	NP	0	10	NP	NS	10	NP	NP	3	NP	NP	3
Avg_count_TCP	-2	NS	-9	-	NS	9	8	8	7	8	8	7	8	8	NP	8	8	NP
Avg_count_UDP	NS	0	NS	NN	-	NS	NP	NS	NP	NP	NS	NP	NP	NS	NS	NP	NS	NS
Avg_count_ICMP	NN	NS	0	NN	NS	-	NS	NP	NP	NS	NP	NP	NP	NP	NP	NP	NP	NP
bytes_In_ICMP	-2	-7	-10	-2	-5	NS	NS	0	0	0	0	0	0	8	0	0	8	0
bytes_In_UDP	NN	-7	0	NN	NS	-9	0	-	7	0	0	7	0	8	NP	0	8	NP
bytes_In_TCP	NN	-4	-1	NN	-4	-2	0	NN	-	0	NP	0	0	1	NP	0	1	NP
ICMP_avg_bytes_in	-2	-5	-10	-2	-4	NS	0	NN	0	-	0	0	0	8	0	0	8	NP
UDP_avg_bytes_in	NN	-7	NS	NN	NS	-9	0	0	-7	0	-	7	0	8	NP	0	8	NP
TCP_avg_bytes_in	NN	-4	-1	NN	-4	-2	0	NN	0	0	NN	-	0	1	NP	1	1	NP
bytes_OUT_ICMP	NN	NS	-10	NN	-1	-1	0	0	0	0	0	0	-	0	NP	0	0	NP
bytes_OUT_UDP	NN	NS	-1	NN	NS	-1	NN	NN	NN	NN	NN	NN	0	-	NP	0	0	NP
bytes_OUT_TCP	-2	NS	-9	-2	NS	-9	0	-8	-7	0	-8	-7	-8	-8	-	8	8	0
ICMP_avg_bytes_OUT	NN	NS	-10	NN	-1	-1	0	0	0	0	0	NN	0	0	NN	-	0	NP
UDP_avg_bytes_OUT	NN	NS	-1	NN	NS	-1	NN	NN	-1	NN	NN	NN	0	0	NN	0	-	NP
TCP_avg_bytes_OUT	-2	NS	-9	-2	NS	-9	0	-8	-7	-8	-8	-7	-8	-8	0	-8	-8	-

KEY
NN: there is no negative lag in the considered range for which there is a significant correlation between the feature pair
NP: there is no positive lag in the considered range for which there is a significant correlation between the feature pair
NS: there is no significant correction between the feature pair at any negative or positive lag in the considered range
0: there is only a significant correlation between the feature pair at lag 0



Time Lag (modulus)	Frequency
1	16
2	12
3	4
4	4
5	3
6	0
7	15
8	37
9	10
10	6

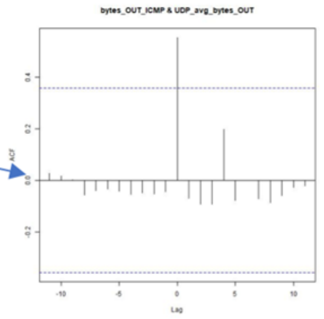


Figure 6.12 An example of CCF graphs relating to each key (NN, NP, NS,0)

As can be seen in Figure 6.12, the example graph for the 'NS' case shows there is no significant correlation between the feature pair (count_udp, count_icmp) at any time lag. The example graph for the 'NN' case shows there is no significant correlation between the feature pair (bytes_OUT_icmp, Avg_count_TCP) at a negative time lag. Similarly, the example graph for the 'NP' case shows that there is no significant correlation between the feature pair (bytes_In_TCP, TCP_avg_bytes_OUT) at a positive time lag. Finally, the example graph for the '0' case shows that the only significant correlation for the feature pair (UDP_avg_bytes_OUT, bytes_Out_ICMP) occurs at time lag 0.

The next section will interpret the data presented in Figure 6.11, exploring whether it may give insights into, and potential explanations for, the performance/accuracy of the models that were presented in chapter 5.

6.6 Discussion and Reflection

This section will consider whether the analysis arising from the CCF data presented in section 6.5 provides a potential explanation for the performance/accuracy levels of the classification models (Naïve Bayes and Random Forest classifies) that were discussed in Chapter 5.

Table 6.2 presents the frequencies for which (the modulus of) each time lag represents the highest significant (positive or negative) correlation value recorded between the relevant feature pair; this represents the data from the table on the right hand side of Figure 6.11 except that the values are summed into the relevant time intervals from Chapter 5 (1, 3, 5, 7 and 10 seconds). In addition, Table 6.2 presents the accuracy levels at these time intervals for the two models (Naïve Bayes and Random Forest) developed and discussed in Chapter 5.

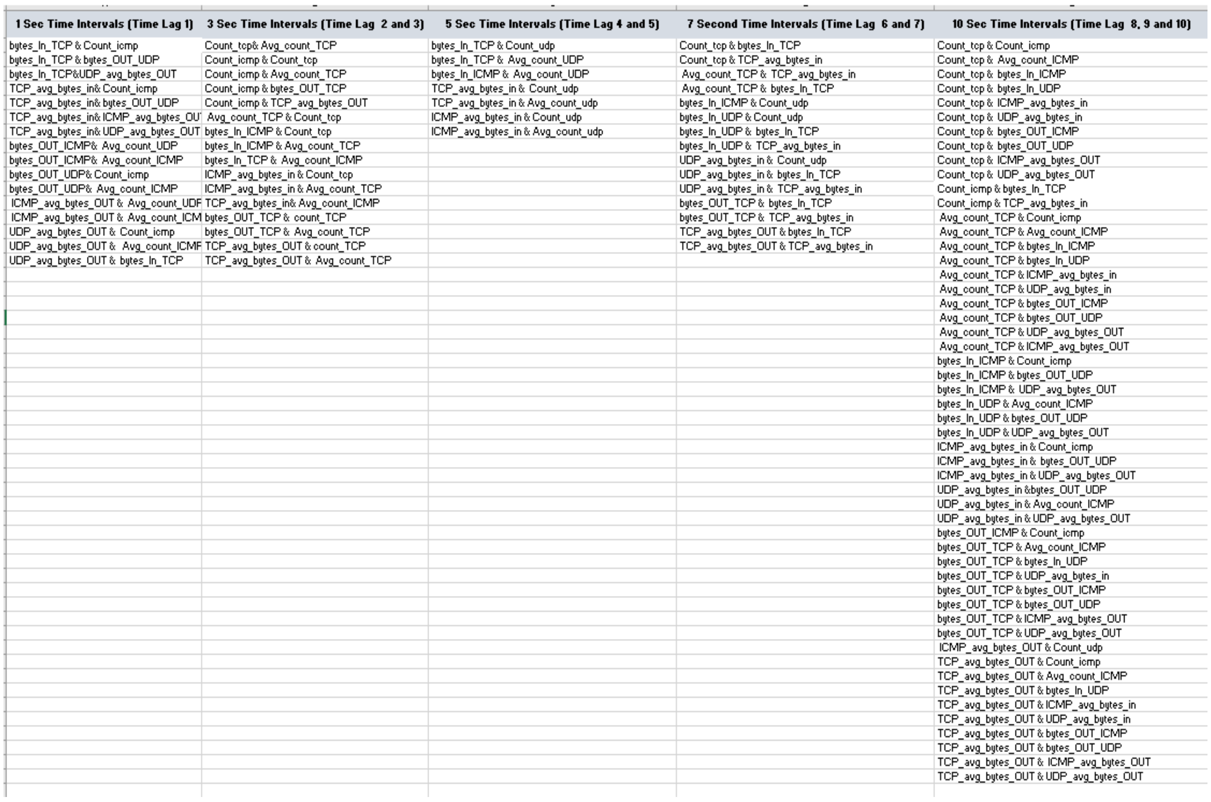
In the 1 second time interval, the associated time lag (modulus) is 1 and there is a 'highest significant correlation' frequency of 20. In this time period, the accuracies of the Naïve Bayes and Random Forest models are 51.2473% and 81.9957%, respectively. The 3 second time interval includes the aggregation of the 'highest significant correlation' frequencies for time lags with a modulus of 2 and 3 seconds, leading to an accumulated frequency of 16. The accuracies of the Naïve Bayes and Random Forest models for the 3 second time interval are 59.8519% and 71.1111%, respectively. Similarly, the 5 second time interval involves the aggregation of the 'highest significant correlation' frequencies for time lags with a modulus of 4 and 5 seconds, leading to an accumulated frequency of 7. The accuracies of the Naïve Bayes and Random Forest models for the 5 second time interval are 64.3045% and 89.7638%, respectively. The 7 second time interval is an aggregation of the 'highest significant correlation' frequencies for time lags with a modulus of 6 and 7 seconds, leading to an accumulated frequency of 15. The accuracies of the Naïve Bayes and Random Forest models for the 7 second time interval are 69.8361% and 98.0328%, respectively. Finally, the 10 second time interval involves the aggregation of the 'highest significant correlation' frequencies for times lags with a modulus of 8, 9 and 10 seconds, leading to an accumulated frequency of 53. The accuracies of the Naïve Bayes and Random Forest models for the 10 second time interval are 75.9091% and 96.8182%, respectively.

Time in Seconds	Time Interval in seconds (from Chapter 5)	Accumulation Frequency Count Within the Time Intervals	Naïve Bayes Accuracy Percentage	Random Forest Accuracy Percentage
1	1	16	51.2473	81.9957
2	3	16	59.851	71.1111
3				
4	5	7	64.3045	89.7638
5				
6	7	15	69.8361	98.0328
7				
8	10	53	75.9091	96.8182
9				
10				

Table 6.2 The frequency data from Figure 6.11 the accuracy of two classifiers in the different time intervals considered

The analysis of the CCF data summarized in Table 6.2 suggests that there is a general increase in the frequency of the feature pairs with a 'highest significant correlation' value as the time intervals increase. Moreover, as noted in Chapter 5 (and reflected in Table 6.2), as the time intervals increase in duration there is improvement in the accuracy of the Naïve Bayes model (from 51.2473% in the 1 second time interval to 75.9091% in the 10 second time interval) and overall improvement in the accuracy of the Random Forest model (from 81.9957% in the 1 second time interval to 96.8182% in the 1 second time interval), though there are exceptions –the decrease between the 1 and 3, and 7 and 10, second intervals. There does not seem to be a clear explanation from the CCF analysis as to why the Random Forest model's accuracy decreases between the 1 and 3, and 7 and 10, second intervals, though it could be that the chosen aggregation is being carried out at intervals that mask the effects in which we are interested. Aggregation of the data into time intervals can have positive aspects, in that it allows the collection of a range of historical information into each datapoint, but also some negatives, since it averages over important features of the data. If the intervals that are chosen are wrong, then it may be that the aggregation has only the negative effects. Further experimental work where different time intervals are explored, and where larger datasets are used, may be useful in exploring these issues further, but this is beyond the scope of the current study.

As a final piece of analysis in this study, the frequency pairs where the 'highest significant correlation' values were found, organized into the time intervals under consideration, were examined to see whether any patterns existed that might provide further insights (see Table 6.3). Perhaps surprisingly, there were no consistent sets of feature pairs present across the different time lags. Further, where a feature was flagged as having a significant correlation with other features, the times at which the 'highest significant correlation' values were identified tended to be relatively consistent (in terms of the time interval into which they fell). That is, particular features seem to have 'highest significant correlation' values with other features across narrow time durations. For example, Count_tcp has 'highest significant correlation values' with 13 other features, and with one exception (with Avg_count_TCP) all of its 'highest significant correlation values' occur between 7 and 9 second (modulus) lags. This narrowness in the time period where an individual feature has its 'highest significant correlation' values with other features seems to be a pattern across the analysis of the CCF data (see Figure 6.11). Since most of the 'highest significant correlation' values were at lags in the range 0-1 seconds, or 7-9 seconds, and this seems to fit with the aggregation levels that performed best, this seems to validate the results from Chapter 5 to some degree. The narrowness issue could, though, be further explored in future work, perhaps with larger datasets and using larger ranges of lags.



1 Sec Time Intervals (Time Lag 1)	3 Sec Time Intervals (Time Lag 2 and 3)	5 Sec Time Intervals (Time Lag 4 and 5)	7 Second Time Intervals (Time Lag 6 and 7)	10 Sec Time Intervals (Time Lag 8, 9 and 10)
bytes_In_TCP & Count_icmp	Count_tcp& Avg_count_TCP	bytes_In_TCP & Count_udp	Count_tcp & bytes_In_TCP	Count_tcp & Count_icmp
bytes_In_TCP & bytes_OUT_UDP	Count_icmp & Count_tcp	bytes_In_TCP & Avg_count_UDP	Count_tcp & TCP_avg_bytes_in	Count_tcp & Avg_count_ICMP
bytes_In_TCP&UDP_avg_bytes_OUT	Count_icmp & Avg_count_TCP	bytes_In_ICMP & Avg_count_UDP	Avg_count_TCP & TCP_avg_bytes_in	Count_tcp & bytes_In_ICMP
TCP_avg_bytes_in& Count_icmp	Count_icmp & bytes_OUT_TCP	TCP_avg_bytes_in & Count_udp	Avg_count_TCP & bytes_In_TCP	Count_tcp & bytes_In_UDP
TCP_avg_bytes_in& bytes_OUT_UDP	Count_icmp & TCP_avg_bytes_OUT	TCP_avg_bytes_in & Avg_count_udp	bytes_In_ICMP & Count_udp	Count_tcp & ICMP_avg_bytes_in
TCP_avg_bytes_in& ICMP_avg_bytes_OU	Avg_count_TCP & Count_tcp	ICMP_avg_bytes_in & Count_udp	bytes_In_UDP & Count_udp	Count_tcp & UDP_avg_bytes_in
TCP_avg_bytes_in& UDP_avg_bytes_OUT	bytes_In_ICMP & Count_tcp	ICMP_avg_bytes_in & Avg_count_udp	bytes_In_UDP & bytes_In_TCP	Count_tcp & bytes_OUT_ICMP
bytes_OUT_ICMP& Avg_count_UDP	bytes_In_ICMP & Avg_count_TCP		bytes_In_UDP & TCP_avg_bytes_in	Count_tcp & bytes_OUT_UDP
bytes_OUT_ICMP& Avg_count_ICMP	bytes_In_TCP & Avg_count_ICMP		UDP_avg_bytes_in & Count_udp	Count_tcp & ICMP_avg_bytes_OUT
bytes_OUT_UDP& Count_icmp	ICMP_avg_bytes_in & Count_tcp		UDP_avg_bytes_in & bytes_In_TCP	Count_tcp & UDP_avg_bytes_OUT
bytes_OUT_UDP& Avg_count_ICMP	ICMP_avg_bytes_in & Avg_count_TCP		UDP_avg_bytes_in & TCP_avg_bytes_in	Count_icmp & bytes_In_TCP
ICMP_avg_bytes_OUT & Avg_count_UD	TCP_avg_bytes_in& Avg_count_ICMP		bytes_OUT_TCP & bytes_In_TCP	Count_icmp & TCP_avg_bytes_in
ICMP_avg_bytes_OUT & Avg_count_ICM	bytes_OUT_TCP & count_TCP		bytes_OUT_TCP & TCP_avg_bytes_in	Avg_count_TCP & Count_icmp
UDP_avg_bytes_OUT & Count_icmp	bytes_OUT_TCP & Avg_count_TCP		TCP_avg_bytes_OUT & bytes_In_TCP	Avg_count_TCP & Avg_count_ICMP
UDP_avg_bytes_OUT & Avg_count_ICM	TCP_avg_bytes_OUT & count_TCP		TCP_avg_bytes_OUT & TCP_avg_bytes_in	Avg_count_TCP & bytes_In_ICMP
UDP_avg_bytes_OUT & bytes_In_TCP	TCP_avg_bytes_OUT & Avg_count_TCP			Avg_count_TCP & bytes_In_UDP
				Avg_count_TCP & ICMP_avg_bytes_in
				Avg_count_TCP & UDP_avg_bytes_in
				Avg_count_TCP & bytes_OUT_ICMP
				Avg_count_TCP & bytes_OUT_UDP
				Avg_count_TCP & UDP_avg_bytes_OUT
				Avg_count_TCP & ICMP_avg_bytes_OUT
				bytes_In_ICMP & Count_icmp
				bytes_In_ICMP & bytes_OUT_UDP
				bytes_In_ICMP & UDP_avg_bytes_OUT
				bytes_In_UDP & Avg_count_ICMP
				bytes_In_UDP & bytes_OUT_UDP
				bytes_In_UDP & UDP_avg_bytes_OUT
				ICMP_avg_bytes_in & Count_icmp
				ICMP_avg_bytes_in & bytes_OUT_UDP
				ICMP_avg_bytes_in & UDP_avg_bytes_OUT
				UDP_avg_bytes_in &bytes_OUT_UDP
				UDP_avg_bytes_in & Avg_count_ICMP
				UDP_avg_bytes_in & UDP_avg_bytes_OUT
				bytes_OUT_ICMP & Count_icmp
				bytes_OUT_TCP & Avg_count_ICMP
				bytes_OUT_TCP & bytes_In_UDP
				bytes_OUT_TCP & UDP_avg_bytes_in
				bytes_OUT_TCP & bytes_OUT_ICMP
				bytes_OUT_TCP & bytes_OUT_UDP
				bytes_OUT_TCP & ICMP_avg_bytes_OUT
				bytes_OUT_TCP & UDP_avg_bytes_OUT
				ICMP_avg_bytes_OUT & Count_udp
				TCP_avg_bytes_OUT & Count_icmp
				TCP_avg_bytes_OUT & Avg_count_ICMP
				TCP_avg_bytes_OUT & bytes_In_UDP
				TCP_avg_bytes_OUT & ICMP_avg_bytes_in
				TCP_avg_bytes_OUT & UDP_avg_bytes_in
				TCP_avg_bytes_OUT & bytes_OUT_ICMP
				TCP_avg_bytes_OUT & bytes_OUT_UDP
				TCP_avg_bytes_OUT & ICMP_avg_bytes_OUT
				TCP_avg_bytes_OUT & UDP_avg_bytes_OUT

Table 6.3 The feature pairs with the most significant correlation values from the CCF analysis in each time interval

Though there is not a direct relationship between the frequencies in Table 6.3 and the accuracy for each time interval for either model, the CCF analysis provides, with the noted anomalies, a starting point from which to further examine why the accuracies of both models increase between the 1 second and 10 second intervals, though more exploration of these issues would be required, through future work, to look at any possible relationships further.

6.7 Summary

This chapter has reported the results obtained by applying cross-correlation analysis to the dataset with lags of -10 to +10 seconds. These results were explored by, where there was one, identifying the 'highest significant correlation' value between the feature pairs in the CCF graphs and the time lag at which it occurred. The results of the analysis show that while there is a general increase in the frequency of the feature pairs with a 'highest significant correlation' value as the time intervals increase, there is not a direct relationship between the frequencies of the feature pairs and the accuracy for each time interval for either model. This suggests that there is a need for future work to seek to discover whether the relationship between feature pairs play a clearer part in explaining the accuracy of the models reported in Chapter 5.

Chapter 7 : Conclusions and Future Work

7.1 Introduction

This chapter will draw together the research that was presented in this thesis, presenting answers to the research questions that were framed in Chapter 1, highlighting the contributions of the work, reflecting on the limitations of the research, and making suggestions for areas of future work. The remainder of this chapter is structured as follows. Section 7.2 will present a brief review of each chapter in this thesis. Sections 7.3 and 7.4 will revisit the research objectives and research questions (framed in Sections 1.4 and 1.5) and seek to provide answers to them. Section 7.5 will present the overall contributions made by the research presented in the thesis. Section 7.6 will reflect on issues that may be seen as limitations of the research that was undertaken. Finally, Section 7.7 will identify areas for future work.

7.2 Review of Preceding Chapters of the Thesis

This section will present a brief summary of each of the preceding chapter in this thesis.

Chapter 1 presented an overview of, and the motivation for, the research reported in the thesis, highlighting the importance of DDoS attacks and their impact in cloud computing environments. The chapter gave a brief introduction to existing research in the area of an Intrusion Detection Systems which seek to identify DDoS attacks in cloud computing environments, and highlighted broad issues associated with this area of work. These issues were used to frame the research questions on which the research reported in this thesis has focused. Finally, Chapter 1 provided a structural guide to the remainder of the thesis.

Chapter 2 presented the cloud computing platform in more detail and explained its core characteristics, the types of service model that it supported and the range of deployment models that exist. The chapter explained several issues that arise as a result of migration to this new computing platform, emphasizing DDoS attacks as one of the most critical threats to the availability of cloud computing services. Chapter 2 then reviewed different type of DDoS attack that target different layers of the network model before reviewing, at a high level, the techniques and methods used by Intrusion Detection Systems to identify and combat this type of attack in cloud computing. The chapter ended by highlighting issues associated with Intrusion Detection Systems, and their accurate detection of DDoS attacks in the cloud environment, to frame the research gap addressed in the remainder of the thesis.

Chapter 3 aimed to develop an accurate detection model based on two different datasets – the NSL-KDD non-cloud dataset and the CIDD cloud dataset – by using machine learning classifiers. It presented the application of transfer learning to remap the different types of DDoS attack between the non-cloud dataset and the cloud dataset. The results of the analysis showed that the performance of the classification model was accurate when applied to the non-cloud dataset, but that their value was limited in classifying attacks in the cloud dataset. It was argued that the limited accuracy of the model arose owing to the different structure of the two datasets, the small overlapping feature set and the different attack types that the datasets contained. Chapter 3 concluded by suggesting that there was a need to generate a cloud intrusion detection dataset on which to undertake further analysis to progress the research effort.

Chapter 4 first reviewed the various methods that could be used to create the cloud intrusion dataset, before justifying the choice of using the emulation approach in this research. The chapter then presented all of the details of the software and hardware environments that were required to form the experimental test-bed. Chapter 4 also presented the tools that were used to generate normal traffic and the different types of DDoS attack required to generate the desired dataset which would have the same structure, a broad set of features and the same type of DDoS attacks as contained within the CIDD dataset.

Using the generated dataset, Chapter 5 reported the development of two detection models, using Naïve Bayes and Random Forest classifiers. The chapter then presented an analysis of the performance of the models in five chosen time intervals, looking at their accuracy and the misclassification feature patterns associated with each model. The chapter showed that the accuracy of the two models generally improved as the time intervals grew longer, identifying that their best performance was at the 7 second time interval. There were, though, no distinct feature patterns in the misclassification cases. Therefore, Chapter 5 ended by suggested that applying another data analysis technique might be useful to seek to discover the relationships between features over time.

In aiming to understand the accuracy of the models over time, Chapter 6 applied cross-correlation analysis to the time series dataset to determine the relationships between feature pairs at different time lags. The chapter explained the processes undertaken to identify the 'highest significant correlation' value between feature pairs in the generated CCF graphs and presented the analysis outcomes, identifying some interesting findings but without finding a direct relationship between frequency of feature pairs with a 'highest significant correlation' value as the time intervals increased and the accuracy for each time interval for either model.

7.3 Revisiting the Research Objectives

This section will revisit each of the research objectives that were set out in Section 1.4 and demonstrate how they have been addressed through this research.

The first objective was to develop an intrusion detection model using two different datasets through the application of machine learning and transfer learning.

The findings of the initial experiment undertaken as part of this research demonstrate that applying transfer learning was helpful in developing detection models where there are two different types of dataset – non-cloud and cloud. However, while the developed model was highly accurate in classifying attacks in the non-cloud dataset, its accuracy was significantly lower when detecting attacks in the cloud dataset. The associated analysis of the first study argued that the lower accuracy arose as a result of the different structures of the two datasets; he small overlapping feature set; and the different attack types that the datasets contained.

The second objective of this work was to generate a cloud intrusion detection dataset, that included a broad range of network traffic and all types of DDoS attack. The findings from the second stage of this research suggest that the creation of the cloud intrusion dataset addressed each of the identified challenges from the first stage. The created cloud dataset captured a broad range of network packet features for the traffic generated by a wide range of DDoS attack types and for normal traffic.

The third objective of this study was to develop intrusion detection models using the generated cloud dataset to analyse the performance of the models and the relationships between attack features within five different time frames.

The findings of the third stage of this study demonstrate that the use of a novel approach to the analysis of the generated dataset using different time intervals as the unit of analysis was useful in comparing the accuracy of the performance of the detection models and the relationship between features over the five chosen time intervals.

The fourth objective of this study was to present a transparent approach to the creation of the cloud dataset, the pre-processing steps required, and the development and analysis of the detection models.

The findings of the final stage of this research demonstrate the provision of a clear and transparent process for generating an emulated cloud-based dataset and undertaking a systematic analysis of it. This, it is argued, should provide a strong basis for comparative analysis to other researchers that intend to undertake similar work.

7.4 Revisiting the Research Questions

This section will revisit each of the research questions that were set out in Section 1.5 and seek to present answers to them.

Research Question1: Can a well-established non-cloud dataset (which includes a range of features and attack types) be used, as part of the application of machine learning and transfer learning, to develop an intrusion detection model that accurately identifies DDoS attack types in one of the few existing cloud datasets?

At a high level, the answer to this question seems to be ‘no’, though the reason for the negative result was important in determining the subsequent direction of the research. To address the first research question, this research used one of the most established non-cloud datasets (KDD-NSl) to develop a detection model by using machine learning classifiers. The accuracy of the outcome of the classification models was high when applied to the non-cloud dataset. A transfer learning approach was then applied to assess the accuracy of the classification model when applied to one of the only publicly-available cloud datasets (CIDD), remapping the different types of intrusion/DDoS attack between the datasets. The accuracy of the model decreased significantly when applied to the CIDD dataset. For example, the model was unable to classify the same type of attack to the same level of accuracy when present in the two datasets. The reduction in accuracy was explained as being associated with the different structures of the datasets. It was argued that this problem arose as a result of the 'unpacking issue' in the CIDD (cloud) dataset, where the data , which was aggregated into specific time periods, had to be disaggregated to create a structure that was comparable to the KDD-NSI (non-cloud) dataset in order for the model to be applied to it. The issues identified in this phase of the work led to the framing of the second research question.

Research Question 2: To address differences in the structure of existing intrusion detection datasets, and the variations in the feature and attack types that they contain, can a new cloud dataset be created that has a similar structure to, broadens the feature set of, and includes the same attack types as, the well-established non-cloud dataset used in the phase of the research that addresses research question 1?

The answer to this question seems to be 'yes'. To address the second research question, this research first reviewed the existing methods for generating intrusion detection datasets, using the analysis to define a suitable approach to generate a cloud dataset which met the requirements of this research related to structure, breadth of feature set and DDoS attack types. As a result, an emulation approach was chosen to create the dataset. Further, the test-bed environment was scoped and equipped with the required software and hardware to generate normal traffic and all of the required types of DDoS attack. A range of preprocessing activities were applied to the generated dataset to transform the structure and features of the dataset to ensure that it was similar to the CIDD dataset. The generated dataset met the requirements set under research question 2, containing a broad range of features and a similar structure to the existing cloud dataset (CIDD). The creation of the dataset was central to the subsequent development of the detection model, which framed the third research question.

Research Question 3: If such a cloud dataset can be developed, can it be used to create accurate DDoS attack detection models and what effect is there on accuracy if the times periods of analysis are varied (choosing a range of periods from 1 second 'time slices' up to 10 second 'time slices')?

The answer to this question is 'yes' – accurate DDoS detection models were created through this work. Specifically, to address the third research question, this research developed two detection models (using Naïve Bayes and Random Forest classifiers) and assessed the accuracy of each model, looking more closely at accuracy through the analysis of the identified misclassification patterns of feature pairs across different time intervals. The analysis showed that the accuracy of the two classification models (Naïve Bayes and Random Forest) generally improved as the time interval increased, with the two models having the highest levels of accuracy at the 7 second time interval. However, while effects on accuracy were found, no pattern was found in the analysis of the feature pairs for the misclassification cases across the different time periods, meaning than no clear explanation was found for the accuracy effects through this phase of the research. The issues identified in this phase of the work led to the framing of the fourth research question.

Research Question 4: Does an analysis of the relationship between pairs of features in the developed cloud dataset help to explain any variations in accuracy across the time periods, found in answering research question 3?

The answer to this question seems to be 'partly' – while there was a general increase in the frequency of feature pairs with a 'highest significant correlation' value as the time intervals increased, there was no direct relationship between these frequencies of the feature pairs and the accuracy for each time interval for either model. To address the fourth research question, the research applied cross-correlation analysis to the time series dataset to discover relationships between pairs of features at different lags. This was explored by looking at the 'highest significant correlation' value between feature pairs in the generated CCF graphs. The analysis identified that the 'highest significant correlation' values between feature pairs were at the 0-1 and 7-9 second time periods, which fitted the aggregation levels that performed best, and therefore seemed to validate to some degree the results referred to in answering research question 3.

Research Question 5: Can the approach to the creation of the cloud dataset, the processing of the dataset prior to the creation of the detection models, the model development, and the analysis of the dataset be presented in a transparent and clear way such that other researchers would be likely to be able to meaningfully compare their results to those reported in this thesis?

The answer to this question is 'yes'. Through the work reported in Chapter 4 – where the environment for the creation of the cloud dataset is clearly explained, Chapter 5 – where the process is set out through which the two intrusion detection models are developed, tested and initially analyzed, and Chapter 6 – where the process of additional analysis using CCF is described, this thesis has presented what is argued to be a transparent and clear process that other researchers could use to undertake similar research, allowing meaningful comparison of results.

7.5 Contributions

The research reported in this thesis has led to four contributions to the field.

The first contribution (reported in Ahmadi, Macredie and Tucker (2018)) lies in the novel application of transfer learning to build a detection model in a context where there has been argued to be a shortage of publicly-available cloud-based intrusion detection datasets. This approach was helpful in developing detection models using two different types of dataset – non-cloud and cloud. While the developed model was highly accurate in classifying attacks in the non-cloud dataset, its accuracy was lower in detecting attacks in the cloud dataset. The value of the approach, though, comes mostly from the associated analysis which argued that the lower accuracy arose as a result of the different structure of the two datasets, the small overlapping feature set, and the different attack types that the datasets contained. These identified challenges motivated the rest of the research reported in the thesis.

The second contribution made by this research is a practical contribution to the creation of a new cloud-based dataset, generated from an emulated cloud computing environment. The dataset generated as part of this research addressed the deficiencies of two currently-available datasets by capturing a broad range of network packet features for the traffic generated by a wide range of DDoS attack types and for normal traffic. The dataset may be used for the purpose of developing detection models, and for comparative analysis work by other researchers in the area.

The third contribution of this research centres on the use of a novel approach to the analysis of the generated dataset using different time intervals as the unit of analysis and comparing the accuracy of the model when applied to them. This approach is, to the best of our knowledge, the first time that there has been a systematic study of the effect on the accuracy of a developed intrusion detection model of aggregating network data into different time intervals.

The fourth contribution made by this research is in the provision of a clear and transparent process for generating an emulated cloud-based dataset and undertaking systematic analysis of it. The lack of transparency in the approach taken in published studies was noted in this thesis and it is therefore suggested that the process detailed in this research effort might be useful to the other researchers that intend to undertake similar work. This would provide a strong basis for comparative analysis and, importantly, would save researchers time in terms of creating the experimental environment and defining a structured approach to analysis.

7.6 Research Limitations

Like any other research project, the research reported in this thesis has a set of limitations which will be acknowledged and discussed in this section.

The first limitation relates to the choice of the approach that was taken to generate and collect the cloud intrusion detection dataset (see Chapter 4). This research used an emulation approach to create the desired cloud intrusion detection dataset because other more ecologically valid methods, such as using a real company's data, were not feasible – such data are commercially sensitive and are not made publicly available for researchers to use. Generating and collecting real network data in the university's network was also not seen as being a feasible approach as one of the requirements of the experiment was to generate attack traffic, and this would obviously have caused serious issues within the university's environment. Therefore, to generate the dataset within the university's computing environment, the dataset was created in a restricted, emulated cloud environment, isolated from the university network and including a limited number of physical machines. Though this is a common approach in research of this type, it could be argued to be less valid that using real attack data. It also led to a very time-consuming process in which it took almost six months to set up the experimental platform on which to generate the data. This caused some limitations in the extent of the dataset that was ultimately generated since only a limited amount of time could be committed to generating the dataset, as part of the overall research undertaken, and many issues had to be overcome in order for a suitable dataset to be generated.

A second issue that may be seen as a limitation is that analysis of the generated dataset did not extend beyond the 10 second time interval. This decision was made for two reasons. First, it was felt that the time series data would not support longer frames of analysis given its length (i.e., when 10 second time slices were taken, the number of records available for analysis was relatively small). Second, the number of limited physical PCs in the experimental environment and the capabilities of the machines in terms of memory and CPU power were under significant pressure to handle the processing of the dataset as collected, giving concerns about whether an even larger dataset could be processed with the available resources.

A third limitation is that only 1, 3, 5, 7 and 10 second time intervals were used. Though this seemed a sensible choice when setting up the analysis (reported in Chapters 5 and 6), it may have been useful to undertake analysis on each one second interval (i.e., 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 seconds) to provide even more information to underpin the analysis of the changes in accuracy of the models over time and the relationships between feature pairs.

The final issue that may be regarded as a limitation of this research stems from the lack of comparison of the developed models against models produced in other research work. As has already been noted, though, the lack of transparency in the approach taken to produce and/or analyse models, and the differences in the datasets that are used, makes it extremely difficult, if not impossible, to make meaningful comparisons, so it was felt that this would not be a fruitful avenue to follow.

7.7 Future Work

The limitations identified in section 7.6 may be used to frame directions for future work.

As noted earlier in this chapter, this research used an emulation approach to generate the cloud intrusion detection dataset to (at least to some extent) address the noted lack of publicly-available cloud intrusion detection datasets. One avenue for future work in this area is to extend the emulated experimental environment to a more significant setting, including more powerful physical machines that host cloud operating systems, and that are equipped with all of the necessary tools to generate datasets in daily then weekly periods of time. Though beyond the scope of this research, this would create a massive amount of data that, through its analysis, should be even more beneficial for building accurate intrusion detection models.

As already explained, this research analyzed the cloud dataset up to 10-second time intervals and explored the accuracy of the detection model in five chosen time periods (1, 3, 5, 7, and 10 seconds). Another possible area of future work is to analyse the cloud dataset above 10-second time intervals. This would allow comparison of the analysis of larger time intervals against the analysis reported in this research, which may help to create more accurate detection models and/or to understand the point at which the model 'collapses' and becomes less accurate. This could help inform the design of IDS, providing useful information on the most suitable time period or periods for analysis, and to explore whether this varies by attack type. Larger datasets, in line with those suggested in the first area for future research, would, of course, be required.

Staying on the issue of time intervals, the existing dataset (and any lager datasets generated in the future) could use smaller time intervals (such as one second granularity), as suggested in section 7.5, to see if any additional insights could be gained into the variations in model accuracy and changes in the relationships between feature pairs.

Another aspect of analysis could be used to frame a further area of future research. This research applied cross-correlation analysis on the generated cloud dataset to understand the relationship between feature pairs at different time lags. The analysis identified that the 'highest significant correlation' values between feature pairs occurred at the 0-1 and

7-9 second lags, but that there did not seem to be a direct relationship between the frequencies of the 'highest significant correlation' for feature pairs at different lags and the accuracy of the detection model for the different time intervals. In future research, the correlation between feature pairs could be explored at lags greater than 10 seconds to see if this might provide further insights.

A final analysis-related area of future work would be to use different time series analysis techniques, such as hidden Markov models, to explore the misclassification patterns identified in the detection model.

References

Ahmadi, R., Macredie, R. D. and Tucker, A. (2018) *Intrusion Detection Using Transfer Learning in Machine Learning Classifiers Between Non-cloud and Cloud Datasets*. Madrid, Spain: Intelligent Data Engineering and Automated Learning – IDEAL 2018. Available at: https://www.springerprofessional.de/en/intrusion-detection-using-transfer-learning-in-machine-learning-/16260378.

Ahmed, M., Naser Mahmood, A. and Hu, J. (2016) 'A survey of network anomaly detection techniques', *Journal of Network and Computer Applications*. Elsevier, 60, pp. 19–31. doi: 10.1016/j.jnca.2015.11.016.

Aljawarneh, S., Aldwairi, M. and Yassein, M. B. (2018) 'Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model', *Journal of Computational Science*. Elsevier B.V., 25, pp. 152–160. doi: 10.1016/j.jocs.2017.03.006.

Beal, V. (2017) *DNS - Domain Name SystemNo Title*. Available at: https://www.webopedia.com/TERM/D/DNS.html (Accessed: 20 October 2017).

Bhushan, K. and Gupta, B. B. (2018) 'Detecting DDoS Attack using Software Defined Network (SDN) in Cloud Computing Environment', in *2018 5th International Conference on Signal Processing and Integrated Networks, SPIN 2018*. IEEE, pp. 872–877. doi: 10.1109/SPIN.2018.8474062.

Bhuyan, M. H., Bhattacharyya, D. K. and Kalita, J. K. (2015) 'Towards generating real-life datasets for network intrusion detection', *International Journal of Network Security*, 17(6), pp. 683–701.

Carlin, A., Hammoudeh, M. and Aldabbas, O. (2015) 'Intrusion Detection and Countermeasure of Virtual Cloud Systems - State of the Art and Current Challenges', 6(6), pp. 1–15.

Comer, D. E. (2006) *Internetworking With TCP/IP*. Pearson Prentice Hall.

Constantin, L. (2012) *DDoS attacks against U.S. banks peaked at 60 Gbps*. Available at: https://www.computerworld.com/article/2493861/ddos-attacks-against-u-s--banks-peaked-at-60-gbps.html (Accessed: 5 November 2019).

Creech, G. and Hu, J. (2013) 'Generation of a new IDS test dataset: Time to retire the KDD collection', in *IEEE Wireless Communications and Networking Conference, WCNC*, pp. 4487–4492. doi: 10.1109/WCNC.2013.6555301.

Dai, W. *et al.* (2007) 'Boosting for transfer learning', in *Proceedings of the 24th international conference on Machine learning - ICML '07*, pp. 193–200. doi: 10.1145/1273496.1273521.

Datasets, A., Borisaniya, B. and Patel, D. (2015) 'Evaluation of Modified Vector Space Representation Using ADFA-LD and', (2015), pp. 250–264.

David, J. and Thomas, C. (2015) 'DDoS Attack Detection using Fast Entropy Approach on Flow- Based Network Traffic', in *Procedia - Procedia Computer Science*. Elsevier Masson SAS, pp. 30–36. doi: 10.1016/j.procs.2015.04.007.

Deka, R. K. and Bhattacharyya, D. K. (2016) 'Self-similarity based DDoS attack detection using Hurst parameter', 9(17), pp. 4468–4481. doi: 10.1002/sec.

Elshoush, H. T. and Osman, I. M. (2011) 'Alert correlation in collaborative intelligent intrusion detection systems - A survey', *Applied Soft Computing Journal*, 11(7), pp. 4349–4365. doi: 10.1016/j.asoc.2010.12.004.

Frank, E., Hall, M. A. and Witten, I. H. (2016) *The WEKA Workbench*, *Morgan Kaufmann, Fourth Edition*. doi: 10.1016/B978-0-12-804291-5.00024-6.

Huang, C. and Yi, P. (2019) 'CCID : Cross-Correlation Identity Distinction Method for Detecting Shrew DDoS', 2019.

Kawamoto, D. (2009) *DDoS attack affects half of GoGrid's customers*. Available at: https://www.cnet.com/news/ddos-attack-affects-half-of-gogrids-customers/ (Accessed: 5 November 2019).

Kayacik, H., Zincir-Heywood, a N. and Heywood, M. I. (2005) 'Selecting Features for Intrusion Detection : A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets', in *Proceedings of the Third Annual Conference on Privacy Security and Trust PST2005*. Canada, pp. 3–8. doi: 10.1.1.66.7574.

Keegan, N. *et al.* (2016) 'A survey of cloud-based network intrusion detection analysis', *Human-centric Computing and Information Sciences*. Springer Berlin Heidelberg, 6(1), p. 19. doi: 10.1186/s13673-016-0076-z.

Kumar, R., Lal, S. P. and Sharma, A. (2016) 'Detecting Denial of Service Attacks in the Cloud', in *Proceedings - 2016 IEEE 14th International Conference on Dependable, Autonomic and Secure Computing, DASC 2016, 2016 IEEE 14th International Conference on Pervasive Intelligence and Computing, PICom 2016, 2016 IEEE 2nd International Conference on Big Data*. Auckland, New Zealand: IEEE, pp. 309–316. doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.70.

Leban, B., Bendre, M. and Tabriz, P. (2017) *Web Application Exploits and Defenses*. Available at: https://google-gruyere.appspot.com/ (Accessed: 5 December 2019).

Lindgren, H. (2013) *Performance Management for Cloud Services: Implementation and Evaluation of Schedulers for OpenStack*. Available at: http://www.diva-portal.org/smash/record.jsf?pid=diva2:636233.

Long (2012) *Mafiaboy's Moment*. Available at: https://www.wired.com/2012/02/feb-7-2000-mafiaboys-moment/.

Mesellem, M. (2014) *bWAPP*. Available at: http://users.telenet.be/mmeit/bwapp/index.htm.

Metasploit (2019) *No Title*. Available at: https://www.tutorialspoint.com/metasploit/index.htm

(Accessed: 5 December 2019).

MIT Technology Review (2019) *The first DDoS attack was 20 years ago. This is what we've learned since.* Available at: https://www.technologyreview.com/s/613331/the-first-ddos-attack-was-20-years-ago-this-is-what-weve-learned-since/.

Modi, C. *et al.* (2013) 'A survey of intrusion detection techniques in Cloud', *Journal of Network and Computer Applications*. Elsevier, 36(1), pp. 42–57. doi: 10.1016/j.jnca.2012.05.003.

Moustafa, N. S. J. (2015a) 'UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection systems (UNSW-NB15 Network Data Set)', in. Available at: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-NB15-Datasets/.

Moustafa, N. S. J. (2015b) 'UNSW-NB15: A Comprehensive Data set for Network Intrusion Detection systems (UNSW-NB15 Network Data Set)', in.

Munson, L. (2015) *Greatfire.org faces daily $30,000 bill from DDoS attack*. Available at: https://nakedsecurity.sophos.com/2015/03/20/greatfire-org-faces-daily-30000-bill-from-ddos-attack/.

Nadiammai, G. V. and Hemalatha, M. (2014) 'Effective approach toward Intrusion Detection System using data mining techniques', *Egyptian Informatics Journal*. doi: 10.1016/j.eij.2013.10.003.

NETSCOUT (2018) *cloud in the crosshairs*. Available at: https://www.netscout.com/report/ (accessed 11 November 2019).

Om, H. (2012) 'A Hybrid System for Reducing the False Alarm Rate of Anomaly Intrusion Detection System'.

Pang, R. *et al.* (2005) 'A First Look at Modern Enterprise Traffic', pp. 15–28.

Pham, T. S., Nguyen, Q. U. and Nguyen, X. H. (2014) 'Generating artificial attack data for intrusion detection using machine learning', *Proceedings of the Fifth Symposium on Information and Communication Technology - SoICT '14*, pp. 286–291. doi: 10.1145/2676585.2676618.

R (2019) *R*.

Rao, C. M. and Naidu, M. M. (2017) 'A Model for Generating Synthetic Network Flows and Accuracy Index for Evaluation of Anomaly Network Intrusion Detection Systems', *Indian Journal of Science and Technology*, 10(14), pp. 1–16. doi: 10.17485/ijst/2017/v10i14/106786.

Rouse, M. (2014) *DEFINITION TCP (Transmission Control Protocol)*.

Sharafaldin, I., Lashkari, A. H. and Ghorbani, A. A. (2018) 'Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization', (Cic), pp. 108–116. doi: 10.5220/0006639801080116.

Shiravi, A. *et al.* (2011) 'Toward developing a systematic approach to generate benchmark datasets for intrusion detection', *Computers & Security*. Elsevier Ltd, 31(3), pp. 357–374. doi: 10.1016/j.cose.2011.12.012.

Shumway, R. H. and Stoffer, D. S. (2016) *Time Series Analysis and Its Applications*. Springer. Available at: https://www.stat.pitt.edu/stoffer/tsa4/tsa4.pdf.

Song, J., Takakura, H. and Okabe, Y. (2012) 'Description of Kyoto University Benchmark Data', pp. 10–12.

Tavallaee, M., Stakhanova, N. and Ghorbani, A. A. (2010) 'Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods', in. IEEE, pp. 516–524. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5464348.

Tavallaee Mahbod, Bagheri Ebrahim, Lu Wei, G. A. (2009) 'IEEE Xplore Document - A detailed analysis of the KDD CUP 99 data set', *IEEE symposium*. Available at: http://ieeexplore.ieee.org/document/5356528/.

Vasilomanolakis, E. *et al.* (2016) 'Towards the creation of synthetic, yet realistic, intrusion detection datasets', in *Proceedings of the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Istanbul, Turkey: IEEE, pp. 1209–1214. doi: 10.1109/NOMS.2016.7502989.

Vixie, C. P., Sneeringer, G. and Schleifer, M. (2012) *21 Oct 2002 Root Server Denial of Service Attack*. Available at: https://web.archive.org/web/20110302164416/http://www.isc.org/f-root-denial-of-service-21-oct-2002 (Accessed: 5 November 2019).

Wei, W. *et al.* (2006) 'Combining Cross-Correlation and Fuzzy Classification to Detect Distributed Denial-of-Service Attacks BT - Computational Science – ICCS 2006', in Alexandrov, V. N. et al. (eds). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 57–64.

Zhijun, W. U. *et al.* (2014) 'Cross-correlation Based Synchronization Mechanism of LDDoS Attacks', 9(3), pp. 604–611. doi: 10.4304/jnw.9.3.604-611.