Anomaly-Based Intrusion Detection by Modeling Probability Distributions of Flow Characteristics

Buse Gul Atli

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 8.9.2017

Thesis supervisor:

Prof. N. Asokan

Thesis advisor:

D.Sc. (Tech.) Yoan Miche

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Buse Gul Atli

Title: Anomaly-Based Intrusion Detection by Modeling Probability Distributions of Flow Characteristics

Date: 8.9.2017 Language: English Number of pages: 12+79

Department of Signal Processing and Acoustics

Professorship: Secure Systems

Supervisor: Prof. N. Asokan

Advisor: D.Sc. (Tech.) Yoan Miche

In recent years, with the increased use of network communication, the risk of compromising the information has grown immensely. Intrusions have evolved and become more sophisticated. Hence, classical detection systems show poor performance in detecting novel attacks. Although much research has been devoted to improving the performance of intrusion detection systems, few methods can achieve consistently efficient results with the constant changes in network communications.

This thesis proposes an intrusion detection system based on modeling distributions of network flow statistics in order to achieve a high detection rate for known and stealthy attacks. The proposed model aggregates the traffic at the IP subnetwork level using a hierarchical heavy hitters algorithm. This aggregated traffic is used to build the distribution of network statistics for the most frequent IPv4 addresses encountered as destination. The obtained probability density functions are learned by the Extreme Learning Machine method which is a single-hidden layer feedforward neural network. In this thesis, different sequential and batch learning strategies are proposed in order to analyze the efficiency of this proposed approach.

The performance of the model is evaluated on the ISCX-IDS 2012 dataset consisting of injection attacks, HTTP flooding, DDoS and brute force intrusions. The experimental results of the thesis indicate that the presented method achieves an average detection rate of 91% while having a low misclassification rate of 9%, which is on par with the state-of-the-art approaches using this dataset. In addition, the proposed method can be utilized as a network behavior analysis tool specifically for DDoS mitigation, since it can isolate aggregated IPv4 addresses from the rest of the network traffic, thus supporting filtering out DDoS attacks.

Keywords: Intrusion Detection, Network Behavior Analysis, Probability Distribution, Hierarchical Clustering, ELM

Preface

This thesis work was conducted at the Security Research team of Nokia Bell Labs under the supervision of Dr. Yoan Miche (Nokia Bell Labs) and Prof. N. Asokan (Aalto University). The work was a part of the degree requirement for the master program in Signal, Speech and Language Processing.

I would like to thank my advisor Dr. Yoan Miche for providing constant support and guidance. Also special thanks to Aapo Kalliola (Nokia Bell Labs) for helping me during the implementation of hierarchical heavy hitter algorithm.

Heartfelt thanks to my supervisor Prof. N. Asokan at Aalto University for providing valuable comments and encouragements.

I would also like to thank to my manager Mr. Gabriel Waller at Nokia Bell Labs for giving me the opportunity of being a part of his team. Thanks to Dr. Ian Oliver, Dr. Silke Holtmanns and other team members of the security research team of the Nokia Bell Labs for their motivation and support.

I wish to thank my boyfriend Bulut Tekgul for encouraging and supporting me in my hardest times. Also thanks to Muge Tetik and Mehmet Okatan for their friendship in Finland.

Most importantly, I would like to express my profound gratitude to my parents and my sister for believing in me throughout my life. I always feel their support, love and strength for all these years despite the kilometers between us.

Espoo, 8.9.2017

Buse Gul Atli

Contents

Abstract . . . ii
Preface . . . iii
Contents . . . iv
Symbols and Operators . . . ix

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Structure of the Thesis . . . 2

2 Intrusion Detection Systems **3**
2.1 Definitions and Terminology . . . 3
2.2 IDS Classification Based on Source . . . 3
2.2.1 Host-based Intrusion Detection . . . 4
2.2.2 Network-based Intrusion Detection . . . 4
2.3 IDS Classification Based on Detection Method . . . 4
2.3.1 Misuse Detection . . . 4
2.3.2 Anomaly Detection . . . 5
2.3.3 Hybrid Methods . . . 5
2.4 Network Anomaly Detection Systems . . . 5
2.4.1 Statistics-based Anomaly Detection . . . 6
2.4.2 Knowledge-based Anomaly Detection . . . 6
2.4.3 Machine Learning-based Anomaly Detection . . . 7
2.5 Network Behavior Analysis . . . 7
2.5.1 Packet-Level Analysis . . . 8
2.5.2 Flow-Level Analysis . . . 8
2.6 Problem Description . . . 8
2.7 Evaluation Metrics in IDS . . . 10
2.7.1 Detection Rate and Precision . . . 10
2.7.2 False Alarms and False Positives . . . 10
2.7.3 CPU Consumption . . . 11
2.8 Overview of the Proposed Solution . . . 11
2.9 Related Work . . . 12
2.10 Summary . . . 13

3 Network Traffic Dataset **15**
3.1 Benchmark Datasets . . . 15
3.2 The ISCX-IDS 2012 Dataset . . . 16
3.3 Packet to Flow Conversion with Argus . . . 18
3.4 Evaluation of Dataset with Simple Machine-Learning Based Anomaly Detection Methods . . . 20
3.4.1 Principal Component Analysis . . . 20
3.4.2 K Nearest Neighbor . . . 22
3.4.3 Gaussian Naïve Bayes . . . 22
3.4.4 Quadratic Discriminant Analysis . . . 23
3.4.5 Multivariate Regression . . . 23
3.4.6 Decision Trees . . . 24
3.4.7 Random Forest . . . 25
3.4.8 Perceptron . . . 25
3.4.9 Extreme Learning Machine . . . 26
3.4.10 Results of Prior Machine Learning Techniques . . . 26
3.5 Related work with the Dataset . . . 29
3.6 Summary . . . 30

4 Methodology **31**
4.1 General Architecture . . . 31
4.2 Trie Data Structure . . . 32
4.3 Hierarchical Heavy Hitters . . . 33
4.4 Mapping Features into Probability Space . . . 35
4.5 Extreme Learning Machines (ELM) . . . 37
4.5.1 Batch ELM . . . 38
4.5.2 On-line Sequential ELM (OS-ELM) . . . 39
4.5.3 One-Class Classification with ELM . . . 41
4.5.4 Design Choices for ELM . . . 43
4.6 Implementation of Different Models . . . 44
4.6.1 Case 1: OS-ELM+PT . . . 45
4.6.2 Case 2: ELM+PT 1 . . . 48
4.6.3 Case 3: ELM+PT 2 . . . 50
4.7 Summary . . . 52

5 Results and Evaluation **54**
5.1 Experimental Setup . . . 54
5.2 Evaluation of Different Models . . . 55
5.2.1 Case 1: OS-ELM+PT . . . 55
5.2.2 Case 2 (ELM+PT 1) and Case 3 (ELM+PT 2) . . . 58
5.3 Evaluation and Comparison of the IDS to State-of-the-Art Approaches 62
5.4 Summary . . . 64

6 Summary and Conclusion **65**

References **66**

Appendix A: Generating Profiles **75**

Appendix B: Argus Flow Level Numerical Features **78**

List of Tables

2.1 Confusion Matrix 10

3.1 Daily Traffic in Dataset 18
3.2 Flow Comparison Results for the Week 20
3.3 Confusion Matrices for ELM with June 13 (a) and June 15 (b) 27
3.4 Comparison of Various Traditional Machine Learning Techniques . . . 29

4.1 Mean Squared Error of Training Data with Different Activation Functions 44

5.1 Confusion Matrices for Case 1 with June 13 (a) and June 15 (b) . . . 56
5.2 ISCX-IDS 2012 Results: Basic ELM, ELM+PT 1 and ELM+PT 2 . . 59
5.3 Average Results of ISCX-IDS 2012 Dataset: Basic ELM, ELM+PT 1, ELM+PT 2, Decision Trees, Linear Regression and KNN 60
5.4 Comparison of Proposed Method Performance to Leading Approaches 63

List of Figures

3.1 Flowchart of Matching Complete Traffic PCAP File with Given XML Profiles . . . 19
3.2 Distribution of Flows for June 14 . . . 20
3.3 PCA Scree Graph . . . 21
3.4 Simple Decision Tree with Five Regions . . . 25
3.5 Single Perceptron Structure . . . 26
3.6 Comparison of Various Traditional Machine Learning Techniques . . . 28

4.1 High Level Description of the Methodology . . . 32
4.2 Example of IP Trie Structure . . . 33
4.3 Illustration of the Hierarchical Heavy Hitters Concept ($N = 100$, $\varphi = 0.1$) . . . 34
4.4 Total Flow Duration Histogram for Normal Cluster 192.x.x.x . . . 36
4.5 Distribution of (a) Source Port and (b) Time-to-Live Features for June 13 . . . 37
4.6 Comparison of OC-SVM and One-Class OS-ELM on 2D Toy Data Moving in (x,y) Plane . . . 42
4.7 Mean Squared Error for Training and Validation Data with Different Number of Neurons . . . 44
4.8 Training (Steps 1,2 and 3) and Updating (Steps 5,6 and 7) Mechanisms for OS-ELM+PT . . . 47
4.9 Training Mechanism of ELM+PT 2 . . . 50
4.10 Training Mechanism of ELM+PT 2 . . . 51

5.1 Detection Rate for June 13 with Different Threshold Values . . . 56
5.2 Estimated PDFs of 3 Different Statistics for Heavy Hitter 192.x.x.x. in June 13 (a) and June 15 (b) . . . 57
5.3 Comparison of Decision Trees, ELM, ELM+PT 1 and ELM+PT 2 for Precision, False Positive Rate, False Alarm Rate and Time Complexity 61

Symbols and Operators

Symbols

c	sampled output vector dimension
d	sampled input vector dimension
d_{ELM}	distance metric
$\mathbf{F}$	matrix of probability density function for different statistics
$\mathbf{I}$	unit vector
N	sample size
N^0	sample size in first time window
N^k	sample size in k'th time window
R	radius
$\mathbb{R}$	set of real numbers
t^i	single moment in time
T	time window
w_{ij}	weight of j'th neuron for input i
x_i	arbitrary input sample
$\mathbf{X}$	A matrix containing the learning data
y_i	arbitrary output sample
$\mathbf{Y}$	A matrix containing the output data
z	query point
β	output weight in a neural network
Δ	bin width
μ	mean
ϕ	activation function
Σ	covariance matrix
θ	threshold
φ	proportion of data stream

Operators

$\tilde{.}$	estimate of a quantity
$\|.\|$	norm
$\|.\|$	determinant
$f(.)$	probability density function of a random variable
$O(.)$	big O notation
X^T	transpose of X
X^{-1}	inverse of X
$X^\dagger$	Moore-Penrose generalized inverse of X
$P(X)$	probability of observing an event X
$P(Y\|X)$	probability of observing an event Y given that X has occurred
exp(.)	exponential function
log(.)	logarithm
$sign(.)$	sign function
$\alpha \cdot \beta$	inner product of vectors α and β
$\sum_{i=1}^{N}$	sum of N input variables

Abbreviations

ARP	Address Resolution Protocol
CPS	Colored Petri Nets
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DR	Detection Rate
ELM	Extreme Learning Machine
EMD	Earth Mover's Distance
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
FTP	File Transfer Protocol
ISCX	Information Security Center of Excellence
HHH	Hierarchical Heavy Hitters
HIDE	Hierarchical Intrusion Detection
HIDS	Host-Based Intrusion Detection System
HTTP	Hyper-Text Transfer Protocol
ID	Intrusion Detection
IDS	Intrusion Detection System
IMAP	Internet Message Access Protocol
IPS	Intrusion Prevention System
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IRC	Internet Relay Chat
KNN	K Nearest Neighbor
LAN	Local Area Network
LS	Least Squares
MSE	Mean Squared Error
NAT	Network Address Translation
NBA	Network Behavior Analysis
NIDS	Network-Based Intrusion Detection System
OS-ELM	Online Sequential Extreme Learning Machine
OC-ELM	One-Class Extreme Learning Machine
PCA	Principal Component Analysis
PDF	Probability Density Function
POP3	Post Office Protocol 3
QDA	Quadratic Discriminant Analysis
ROC	Receiver Operating Characteristic
SLFN	Single Hidden Layer Feed-forward Neural Network
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language

SSH	Secure Shell
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
TTL	Time to Live

Chapter 1

Introduction

1.1 Motivation

In recent years, software services and web-based applications have generated much interest because of advances in networking technology. Information networks, such as voice and video over IP, encryption and authentication techniques are deployed by both freelance developers and enterprises in order to produce more intelligent, reliable services for direct interaction with customers [1]. Nowadays, cloud storage services and the Internet of Things (IoT) have created new businesses and connected the world by converting it into a massive information system. However, the use of these technologies also brings more critical vulnerabilities [2]. Since cloud services store personal and private data in hosting devices, hackers can craft attacks by exploiting these vulnerabilities in both public networks and private devices to steal important information [2]. Moreover, IoT and cloud storage pose a wide range of cyber-risks that threaten not only businesses but also health care, education, banking and government systems [3]. According to [4], global cyber-crime cost 3 trillion dollars in the past year, and it is estimated that the loss for 2021 will be as high as 6 trillion dollars. Therefore, security practices have been the focus of intense research due to the requirements for a safe, secure environment.

Traditionally, network and business assets have been protected using Intrusion Prevention Systems (IPS). For example, firewalls [5] have been used to filter traffic packets by checking the information in the packet header. Other prevention systems include anti-virus programs or user identification and authentication mechanisms [5]. Although these security components have provided useful tools for protecting network confidentiality and integrity, they often neglect another key element of the security strategy: availability [6]. Arbor Networks in [6] reported that intrusion prevention systems are susceptible to numerous vulnerabilities, including flooding, fragmentation and impersonation of legitimate users. Since new vulnerabilities are constantly being discovered and attacks continue to evolve, Intrusion Detection Systems (IDS) have emerged as a potential defense mechanism layer to monitor network and detect intrusions unrecognized by the IPS [5].

Intrusion detection is the act of monitoring and analyzing network communication in order to detect any malicious behavior or unauthorized activity which might intend to compromise the confidentiality, integrity and availability of the transmitted information [7]. Accordingly, intrusion detection systems analyze network events and capture security problems, also known as intrusions [8]. Intrusion detection systems are capable of recognizing malicious activities and triggering an alert or logging the results. Intrusion detection systems can be categorized based on alarm triggering actions using either signatures or anomalies [5]. Signature-based IDSs identify and compare network activities with predefined intrusions. On the other hand, anomaly-based IDSs concentrate on finding unusual patterns which do not conform to the normal baseline. IDSs can also be classified in terms of the monitoring location [5]. Host-based IDSs monitor information from individual hosts or devices in the network, whereas network-based IDSs collect data from multiple sources.

Anomaly-based intrusion detection has been the focus of intense research in recent years [1]-[9]. Despite the significant number of existing studies in this area, more research is needed due to the continuously evolving nature of the attacks. In order to solve this problem, a practical intrusion detection system should be able to detect novel and stealthier attacks when IPS fail to do so.

1.2 Structure of the Thesis

The remainder of this thesis is divided into 5 chapters. Chapter 2 provides a brief introduction to intrusion detection systems and network behavior analysis. Chapter 2 also summarizes the related research in this field, highlights current problems in intrusion detection systems, proposes a solution and presents the evaluation criteria for the proposal. Chapter 3 analyses the ISCX-IDS 2012 network dataset and explains the required preprocessing mechanisms for the proposed model. Additionally, Chapter 3 reviews some state-of-the-art research done by using this dataset. Chapter 4 explains the overall architecture of the proposed IDS, the mathematical methods applied in the model and different variants of this IDS. Chapter 5 assesses the performance of the proposed scheme. Chapter 6 concludes the thesis and suggests possible improvements to the proposed model as future work.

Chapter 2

Intrusion Detection Systems

This chapter presents the most common approaches for intrusion detection and classifies these approaches in terms of the source and detection technique used. Section 2.5 provides a detailed summary of the network behavior analysis method which supports IDS by monitoring traffic. Section 2.6 highlights relevant problems with intrusion detection systems. Section 2.7 introduces the metrics used for evaluation and Section 2.8 briefly describes the proposed solution. Work related to relevant intrusion detection systems is discussed in Section 2.9.

2.1 Definitions and Terminology

In network security, *intrusion* is defined as any sequence of actions which attempts to compromise the integrity, confidentiality or availability of the information in a single system or network. An *intruder* initiates several related steps in the intrusion procedure to violate a given security policy [10]. *Intrusion detection* (ID) is a process of monitoring information, detecting such violations and responding with an action in order to protect the system or network. An *intrusion detection system* (IDS) deploys a set of hardware and software resources that performs intrusion detection.

Intrusion detection systems can be categorized based on the source of information and type of the detection approach. Major classifications are discussed in the following sections.

2.2 IDS Classification Based on Source

Intrusion detection systems can be sorted in different categories depending on the source location and the position of an IDS. Host-based and network-based IDSs are two main classifications based on these contexts [5].

2.2.1 Host-based Intrusion Detection

Host-based IDSs are installed on single hosts such as individual computers where data is collected from local systems [11]. Host-based IDSs (HIDSs) monitor activities such as system integrity, memory, calls and logs. HIDSs provide individualized protection against malicious activities which cannot be detected by network-based IDSs [12]. HIDSs maintain a large history of behavioral information that is used for possible misuse identifications; therefore, they are capable of responding to long term attacks such as data-stealing malwares. However, HIDSs have a poor real-time response and protection against one-time massive intrusions [13]. In addition, since HIDSs are designed mostly to detect insider attacks, they cannot avert outsider compromises [13].

2.2.2 Network-based Intrusion Detection

Network-based IDSs (NIDS) monitor the data coming from multiple resources and attempt to protect the entire network [11]. As a NIDS has a number of own sensors monitoring different sources, it can detect malicious activities affecting multiple hosts [13]. Moreover, NIDSs are usually considered as active components, since many of these systems examine the traffic packets or flows in real time. The main disadvantage of NIDSs is that if they monitor a busy section of the network, their packet processing rate might be smaller than the incoming data rate. Therefore, they might be incapable of processing a large amount of data [14].

2.3 IDS Classification Based on Detection Method

The detection method is another significant characteristic that divides IDSs into two subcategories: signature based systems (misuse detection), which depend on specific knowledge, and anomaly based systems monitoring the behavior in the network [14].

2.3.1 Misuse Detection

Misuse detection (also known as signature-based detection) recognizes attack patterns by comparing network data to known attacks. These identified attacks are stored in the database as signatures [14]. Therefore, the accuracy of these systems is very high and they have very low false positives since they only respond when the observed attack patterns are matched with the known signature [1].

In misuse detection systems, signatures in the database must be maintained and updated frequently to operate reliably and in a short response time. Otherwise, they can fail to detect unknown attacks if they are not in the known database [8].

The open source network intrusion detection tool Snort [15] and The Bro Network Security Monitor [16] are two famous network based misuse detection software products.

2.3.2 Anomaly Detection

The anomaly detection method assumes that normal and attack traffic are different in some aspect. This method depends on the idea that identifying an abnormal behavior in the network could be possible by comparing current traffic to a normal state [17]. For this reason, anomaly-detection IDSs can potentially detect novel attacks without any specific knowledge.

Anomaly-based IDSs extract a comprehensive model of the normal profile by collecting data from users, hosts or network connections [14]. Normal profile modeling can be implemented in an off-line or on-line manner. Normal profiles in off-line aggregation are mainly static, which means unchanged until another request is received by the IDS for generating new profiles. On the other hand, on-line methods can include dynamic thresholding related to certain behavior attributes [8]. Other methods such as rule-based measures [18], machine learning [19], neural networks [20] or genetic algorithms [21] can also be used in anomaly detection systems.

Despite their strong detection rate of new exploits and stealthy attacks, anomaly-based IDSs suffer from high false positive rates due to the unexpected behavior of normal traffic [1]. Moreover, anomaly-based IDSs might require a training phase which needs a large amount of time and data in order to construct normal profiles.

2.3.3 Hybrid Methods

Hybrid IDSs combine misuse and anomaly detection models. They attempt to detect both known and novel attacks. For example, a hybrid IDS can check packet headers for signatures and collect network traffic to define normal behaviors [1].

For example, H-Snort [22] can be considered as a hybrid system that enhances the basic functionalities of the original Snort software.

2.4 Network Anomaly Detection Systems

Anomaly detection systems monitor the network data and try to find patterns deviating from the expected behavior. Anomaly-based IDSs assume that attacks differ from the normal behavior in terms of certain aspects and that attackers have a very limited knowledge about the normal traffic patterns of the target network. [1] Anomaly detection systems have the ability to detect both novel and volumetric attacks like Distributed Denial of Service (DDoS) [6]. Hence, anomaly-based intrusion detection has gained a growing interest in the field of intrusion detection [23]-[24].

Although the main assumption for differentiating normal and attack behaviors in network anomaly detection provides a way of identifying attacks, high rates of false alarms can be observed, since they might label unintentional anomalies as intrusions. Therefore, strategies for measuring the deviation from normal behavior is crucial

in these intrusion detection systems. In order to obtain a more robust anomaly detection system, the IDS module is divided into modeling and detection phases [25]. The modeling phase trains the system, constructs normal profiles and defines a proximity measure for finding an acceptable deviation range from these profiles. The detection phase uses this obtained model to classify new events as outliers or anomalies. If the observed network is performing under normal conditions and it is considered that no attack is observed in the modeling phase, the current traffic or data can be used to update the trained model in order to keep it up-to-date. Deciding feasible deviation metrics and the construction of user profiles can be done with different types of supervised and unsupervised methods. In the following subsections, three different anomaly detection techniques will be introduced.

2.4.1 Statistics-based Anomaly Detection

Statistical methods extract normal network behavior for each protocol, IP address or connection by using a variety of features, including packet rate, inter-packet arrival time, and latency [23]. In statistical anomaly detection schemes, two different network traffic datasets are maintained during the detection process. One dataset refers to the previously trained profile and the other dataset associates with the monitored traffic over time. Observed data is compared to the training data by measuring statistical information, such as mean, standard deviation and approximate distribution of the data. These comparisons produce an anomaly score. If this score is beyond a certain threshold for a specific event, then the intrusion detection system will classify that event as an anomaly [1].

The biggest advantage of these methods is that they do not require the designer to explicitly embed prior knowledge. Furthermore, statistical anomaly detection systems can retain a high performance in terms of accuracy over long periods of time [23]. However, their training phase can be exploited by an intruder in order to generate an attack traffic that imitates the normal behavior. Another challenging aspect of statistical methods is setting a threshold value to separate normal and attack events. Moreover, all behaviors in the training data are difficult to model by only using statistical methods.

Implementing Gaussian random variables in univariate models [17], multivariate statistical analysis that calculates correlation among more than two metrics [26], time series models that also examines inter-arrival times of observations [27] are examples of statistical-based intrusion detection systems.

2.4.2 Knowledge-based Anomaly Detection

Knowledge-based approaches use expert systems which are designed in order to classify aggregated network data. Classification is performed with a set of rules and procedures by checking different attributes identified from the training data [23]. Knowledge-based methods are robust, flexible and scalable [23]. However, developing

a set of rules to extract distinguishing attributes from the training data is difficult and computationally complex in knowledge based methods, since the data needs to be analyzed exclusively.

Finite state machines [28], rule based classifications having rule engines [15], and description languages such as n-grams are a few prominent knowledge-based intrusion detection methods.

2.4.3 Machine Learning-based Anomaly Detection

In machine-learning based anomaly detection, modeling the normal behavior is done by learning patterns in the network data [23]. Some of these algorithms can support on-line learning and make predictions on network data arriving in a streaming fashion. Therefore, IDSs using these type of algorithms can adjust learning parameters and possibly perform better compared to other anomaly detection techniques. However, this might be considered as a drawback due to high resource consumption. Another major disadvantage arises from over-fitting [25], since some of these methods are computationally complex and can learn the detail or noise of the received network traffic [23]. Consequently, they might show poor performance in classifying new events in the received network traffic.

Many unsupervised and supervised machine learning methods have been implemented according to the network data properties. Bayesian analysis representing relationships between variables and predicting future events [29], Markov chains systems calling event sequences from past observations [24], outlier detection by clustering observed data according to a proximity measure [19], genetic algorithms inspired by evolutionary biology [21] and a range of neural network methods simulating operation of human brain are typically used machine learning techniques for anomaly detection.

2.5 Network Behavior Analysis

Network Behavior Analysis (NBA) is an activity that enhances the security of a system by monitoring traffic and auditing network data from existing infrastructure devices [30]. Network analyzer tools like Wireshark [31] and Tcpdump [32] are used in a wide range of applications from logging the network traffic to detecting spy-ware and reverse engineering protocols.

Currently, different NBA methods and tools have been used as components in many types of anomaly detection systems. NBA tools aggregate network statistics of all traffic in either packet-level or flow-level.

2.5.1 Packet-Level Analysis

Packet level analysis inspects individual IP packets which are extracted from the network traffic in real time. After capturing packets, the NBA component inspects the highly informative header section and the payload. In addition to the header information, some indirect properties including inter-arrival times, rate and fragmentation levels can also be extracted with packet-level analysis [33].

Packet sniffers are used for packet level analysis, since they log communication traffic, monitor network performance, recognize bottlenecks, retrieve lost data and detect intrusions.

2.5.2 Flow-Level Analysis

According to IPFX terminology [34], a *network flow* (or traffic flow) is a unidirectional sequence of IP packets passing through a point in a certain time interval. Packets belonging to one flow should have invariant header fields, such as the source and destination addresses, the protocol type and the port information. The total number of flows summarizing a complete communication constructs a *flow record*. It reports which hosts communicated with each other, when this connection occurred with which transmission method and other attributes of a specific connection [35]. Similar to packet analysis, flow records can also provide many indirect statistics about network data.

Flow records can produce aggregated input which differs from the entire raw data. Recording only the statistical information rather than the complete IP packets enables to obtain quite small flow records compared to the raw output [35]. Therefore, flow analysis is mostly done on routers or used with NIDS where the detection tool can monitor multiple hosts.

FlowScan [36], SiLK [37] and Argus [38] are well known network flow analyzer tools. A simple usage of Argus (Audit Record Generation and Utilization System) application is explained in section 3.3 with details.

2.6 Problem Description

Network behavior analysis and intrusion detection systems play an important role in cyber-security. For several years, intrusion detection systems have been the focus of increasing work. Although numerous open-source and commercial signature-based intrusion detection systems have been proposed [8, 15, 16], they lack the ability to detect novel attacks. This problem has motivated researchers to focus on anomaly detection [1, 14, 17]. Since anomaly-based intrusion detection approaches assume that there are meaningful differences between normal and anomalous traffic, these approaches can be used to detect new attacks. However, many anomaly-based IDSs have a high false positive rate for detecting volumetric attacks and stealthy floods

generated by botnets [25].

Another significant problem in the intrusion detection approach is the unavailability of a complete, realistic dataset. Many datasets have inadequate characteristics and contain outdated or unlabeled traffic patterns. These deficiencies limit the use of supervised methods for anomaly detection, since these methods require properly labeled datasets. Moreover, many datasets cannot be shared due to privacy issues [39]. Therefore, intrusion detection systems are often evaluated over a few publicly available datasets containing obsolete network routing protocols. Moreover, the constant change in the network traffic can pose new vulnerabilities and unfamiliar types of intrusions as well as it can alter normal network traffic characteristics [40]. Therefore, old benchmark datasets appear less effective for measuring the performance of an intrusion detection system.

Evaluating different IDSs and selecting the best intrusion detection system for providing security in different environments require evaluation metrics. Evaluation metrics should be properly selected in order to measure the effectiveness of an intrusion detection system in terms of its ability to separate attacks from the normal behavior [41]. Standard evaluation criteria developed for assessing IDSs are true positive rate (or detection rate), false alarm and false positive rate. However, alternative evaluation criteria should be used to measure the performance of an intrusion detection system to perform a particular task. For example, the Receiver Operating Characteristic (ROC) curve can be used to analyze the trade-off between false positive and true positive rates or the speed of an IDS for detecting the burst rate of an intrusion can be estimated by measuring response time [40]. Thus, objective, specific metrics should be proposed to measure the capability of an IDS used for different security purposes.

NBA has also generated considerable interest in the field of cyber-security, since NBA tools are capable of aggregating data from many hosts in order to support off-line anomaly detection systems [42]. Although much research has focused on network behavior analysis, few studies have provided efficient models to extract useful profiles from network data. Building comprehensive network behavior profiles from the captured data needs huge memory and time. Therefore, an intrusion detection model that implements cost-effective learning with a complementary network behavior analysis remains a demanding task.

Although extensive studies have been devoted to IDS and anomaly detection, many relevant problems explained above still remain unsolved. Hence, the goal of this thesis is to develop an IDS which combines several individual methods to address these challenges by providing high detection rate with minimal processing requirements and realistic evaluation metrics.

2.7 Evaluation Metrics in IDS

IDS evaluation is an essential task, since intrusion detection systems have to demonstrate how well they operate compared to other IDS tools [14]. Accuracy, sensitivity, specificity, ROC curves, confusion matrix, precision, recall and f-measure are widely calculated evaluation measures in network anomaly detection. Moreover, time and space complexity of the model are useful for assessing the efficiency in real time IDSs.

In the field of pattern recognition and information retrieval, precision and recall are more desired measures than accuracy. They are defined with True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The confusion matrix is another ranking method to compare the actual class label against predicted labels and demonstrates TP, TN, FP, FN values, as shown in Table 2.1. In this table, TP represents the number of normal traffic correctly classified. Similarly, TN gives the number of intrusions correctly detected. FP is the number of attack samples undetected, and FN is the number of normal traffic samples classified as attack. In the following subsections, the chosen metrics based on these measurements are explained.

		Predicted	
		Normal	*Attack*
Actual	*Normal*	True Positive (TP)	False Negative (FN)
	Attack	False Positive (FP)	True Negative (TN)

Table 2.1: Confusion Matrix

2.7.1 Detection Rate and Precision

Precision is a measure of how many truly relevant results are retrieved from all instances [43]. This metric, also known as the detection rate in IDSs, measures how well the system identifies attack or normal traffic. A high detection rate should be a requirement for the trustfulness of an intrusion detection system.

$$\textit{Detection Rate of Normal Traffic} = \frac{TP}{TP + FP} \tag{2.1a}$$

$$\textit{Detection Rate of Attack Traffic} = \frac{TN}{TN + FN} \tag{2.1b}$$

2.7.2 False Alarms and False Positives

The false positive rate attempts to estimate the percentage of malicious traffic detected as normal. The false alarm rate (false negative rate) is the rate of normal behavior detected as attack which triggers an unwanted action. Many anomaly detection algorithms suffer from a large number of false positives and false negatives. A high false alarm rate negatively affects the reliability of IDS model and users stop

responding to every false flag produced by the system. A high false positive rate is also dangerous, since it shows that the IDS fails to detect malicious traffic. In an ideal IDS, false positives must be almost zero and the system should be alerted when only true attacks occur.

$$\textit{False Alarm rate of Normal Traffic} = \frac{FN}{TP + FN} \tag{2.2a}$$

$$\textit{False Positive Rate of Attack Traffic} = \frac{FP}{FP + TN} \tag{2.2b}$$

2.7.3 CPU Consumption

In order to select the best IDS configuration, resource consumption can be measured in terms of CPU and memory usage. For example, the CPU consumption must be considered as a time constraint for real-time intrusion detection systems [44]. If a real-time IDS has a lower processing time than the rate of the arriving traffic, then its performance degrades in terms of packet drops. CPU time can be measured with different tools and presented as epochs or seconds. Since CPU consumption is useful to estimate the response time of an IDS, it can be used as another evaluation metric.

2.8 Overview of the Proposed Solution

In order to improve the detection accuracy of stealthy attacks, this thesis develops a method which combines machine learning techniques and statistical measurements for analyzing network behavior during the intrusion detection process. Different variants of this method are created and evaluated against the dataset provided by The Canadian Institute for Cybersecurity (CIC) [39]. This dataset is used at the flow-level and statistical measurements are collected from flow-level features. The complete set of flow-level features are described and listed in Appendix B.

The scope of the thesis is limited to neural network methods, and we do not consider feature selection techniques for generalizing the model and reducing the time complexity in the presented work. The learning part of the proposed system is trained with the probability density functions of the flow-level features as proposed in [45]. Full details of this approach are provided in Chapter 4.

The model described in this thesis is originated from the theoretical work done by Aapo Kalliola and Yoan Miche [45]. The other parts of the development, modifications, tests and analyses are carried out by the author.

The proposed technique offers a baseline solution for the mitigation problem in volumetric attacks, such as HTTP floods and DDoS attacks. This thesis demonstrates the efficiency of the proposed method in four different attack scenarios and provides a performance analysis using the ISCX IDS 2012 dataset with evaluation metrics focusing on the detection and false positive rates.

2.9 Related Work

Various sequential and off-line methods have been proposed for improving accuracy in intrusion detection systems. Expert systems [46] are extensively used in signature-based intrusion detection, also commonly known as knowledge-based intrusion detection. These types of IDS contain a set of rules which define good quality signatures. Signatures should contain a wide range of information about all possible attempts and means for compromising the host [47]. One of the earliest models for knowledge-based intrusion detection was developed in [48], which used Colored Petri Nets (CPS) to represent each signature. CPSs specify an attack scenario having more than one start state and a unique final state. Another expert system (STAT) was applied in [18], which models attacks as state transition diagrams. Although numerous trademark signature-based IDSs have been proposed, many of these suffer from high false alarm rates [49].

Because of the insufficiencies in signature-based IDSs, much research has been devoted to finding patterns in network data that deviate from expected behavior. Many machine learning techniques [9, 25, 50] have been proposed as a promising approach for anomaly detection. The ADAM model was developed in [9], which implements data mining techniques to discover attacks in an on-line fashion by building a repository from normal behavior observed more frequently in the streaming network data. In the testing phase, ADAM classifies frequent connections based on a trained classification technique by comparing these connections to the normal repository. Although experimental results in [9] show that ADAM is very effective, it only learns normal events in the training data, and this approach still produces a significant number of false alarms. The authors in [50] proposed the Hierarchical Intrusion Detection (HIDE) system, which involves different intrusion detection agents for monitoring system activities coming from servers. Several layers inside each intrusion detection agent provide different functionalities. Despite its efficiency in detecting flooding attacks, HIDE and other statistical techniques have some drawbacks [25]. For instance, these techniques often have difficulty in selecting the best statistical parameters. In addition to these systems and testbeds, classification methods [51, 52], clustering and outlier-based approaches [53, 54] as well as other combination learner systems [55] have been presented for anomaly-based intrusion detection. Classification methods usually give better results than other anomaly-based intrusion detection methods, since they use labeled samples for training. However, they have low detection rate for unknown intrusions unless the relevant information is not used for retraining purposes. The techniques used in clustering and outlier-based approaches highly depend on the parameters of proximity measures and optimization of these parameters are time consuming [25]. Combination learner methods use ensemble methodology to combine different classifiers and obtain a model with high performance. These methods have higher accuracy than the individual classification methods, and can scale for larger, more comprehensive datasets without any performance degradation. However, they are usually slow and can not be used in real-time systems [25].

Recently, artificial neural networks, such as feed-forward neural networks [56] and self organizing maps [57], have received considerable attention for intrusion detection, since they can adapt the interconnection strengths and synaptic weights of the network when the input data changes. Since some of the feedforward neural network using back-propagation method for training suffer from slow learning time and high false positive rates [58], improving the performance of neural networks by changing the weight update rules has been the focus of intense research in intrusion detection. Thus, Extreme Learning Machine (ELM) has emerged as a promising approach for allowing low computation cost and faster training than other feed-forward neural networks. For example, Cheng *et al.* [59] proposed kernel-based extreme learning machines for both binary and multi-class classification in order to detect the type of attack traffic. The authors found that basic ELM has a lower accuracy than kernel-based ELM or support vector models. They also showed that kernel-based ELM methods have better scalability than support vector machines, although these two methods have almost similar computational complexity. In [60], the authors suggested a weighted version of multi-class ELM to manage unbalanced class distribution. The weighted ELM in [60] achieves better performance than support vector machines; however, this approach is less effective in classifying infiltration and probing attacks. The authors in [61] tested a modified version of ELM using multiple kernel boosting and an ensemble approach requiring no feature selection. This approach achieved excellent results in terms of detection performance and false positive rate despite the increased cost of memory usage.

Aggregation techniques for clustering network traffic and blacklisting arrived packets have also generated increased interest in network behavior analysis, which highly supports intrusion detection systems. For example, Kalliola *et al.* [62] focused on end-host clustering in order to mitigate denial of service attacks. In [63], the authors have demonstrated the feasibility of hierarchical clustering technique for DDoS mitigation. However, a more exhaustive mechanism is needed to gather profiles consisting of a wider range of attacks in order to decrease potential false positives.

The solution proposed in this thesis offers a method for learning and predicting the probability density function (PDF) of flow statistics. Since PDF can be used to estimate the likelihood, it can also represent the normal behavior of the network in terms of the probability values of the observed flow statistics. If estimated probability values for many flow statistics at one sample are small, it probably does not belong to the normal behavior and can be an anomaly. Therefore, a probabilistic approach may be efficient to differentiate attack from normal traffic and can be used for network anomaly detection.

2.10 Summary

This chapter has provided necessary definitions and the terminology in the field of intrusion detection. Different IDSs have been reviewed in terms of their advantages

and drawbacks. In addition to the taxonomy of existing IDSs, this chapter has explained the necessity of analyzing network behavior, which can be implemented at either packet or flow level. Section 2.6 has identified some of the existing problems in the area of intrusion detection. Section 2.7 has presented evaluation metrics used for measuring the reliability and the effectiveness of the proposed approach. Section 2.8 has briefly described the proposed method. Finally, recent work has been summarized by presenting the difference between the existing work and the proposed method. The following chapter provides an analysis of a real-life dataset built by the Information Security Center of Excellence (ISCX) [64] and possible preprocessing methods applied on this dataset.

Chapter 3

Network Traffic Dataset

Capturing and generating network traffic is an essential preprocessing step to compare different intrusion detection systems and validate new approaches. In particular, anomaly detection techniques require comprehensive and current network data which resembles a real communication scenario. A proper dataset should be generated in a realistic way with complete and correct labels. Moreover, it should have an unbiased proportion of the normal to attack traffic [64].

This chapter discusses some of the well-known public datasets and their characteristics. After the brief discussion, the ISCX-IDS 2012 dataset used throughout the proposed solution is explained in detail. In addition, preprocessing methods to extract flow records using the Argus tool are presented. Simple ML methods implemented to validate the usability of the dataset are presented and evaluated. Finally, some earlier intrusion detection approaches are reviewed and tested with this dataset.

3.1 Benchmark Datasets

There are numerous benchmark datasets available for evaluating intrusion detection methods and systems. In these datasets, different attack scenarios were generated using simulated environments.

The KDDCup99 dataset [65] is one of the earliest and well-known IDS datasets. It was collected from seven weeks of labeled data containing 41 features. The training dataset consists of approximately 4,900,000 single connection vectors with 24 different attacks, whereas the test dataset contains 300,000 samples with an additional 14 attack scenarios. [25]. There are four types of attacks in the KDDCup99 dataset: denial of service (syn flood), remote to local (guessing password), user-to-root (buffer overflow) and probing. Since this dataset has many problems that result in poor evaluation of anomaly detection methods [25], the NSL-KDD [66] dataset was introduced to solve these issues. The NSL-KDD dataset includes only selected records of the complete KDDCup99 training dataset. In addition, this dataset does not include redundant or duplicate records. Consequently, machine learning algorithms used for

intrusion detection are unbiased towards more frequent records [64]. Furthermore, training and testing set contain various attack samples with a reasonable proportion. This leads to more accurate, comparable performance evaluation, since there is no need to select a portion of the dataset to test the performance of different intrusion detection systems. In [67], the DARPA dataset was presented as another intrusion detection dataset. The DARPA dataset was created by the MIT Lincoln Laboratory in order to detect complex attacks that contain multiple steps. Attack scenarios were simulated with sessions of probing, breaking into system by exploiting vulnerabilities as well as installing and launching DDoS attack against other hosts [25]. The DEFCON dataset in [68], which was captured in a hacker competition, contains only intrusive traffic. This dataset is very different from a real world network, thus it has a limited use. Finally, the CAIDA dataset [69] is a collection of many different types of network data, available for research purposes. However, the CAIDA dataset is very specific to particular events and intrusions, such as DDoS attacks.

Although benchmark datasets are useful for evaluating intrusion detection systems, many of them suffer from some problems such as unrealistic network configuration, unlabeled or incomplete data, limited intrusion scenarios and disproportionate ratio between normal and attack traffic. In order to generate more realistic network scenarios, real-life datasets have been generated in recent years. For example, the UNIBS [70] dataset was collected on the edge router of the campus network in University of Brescia. The dataset was created in three consecutive days by running tcpdump on the faculty router and stored using 20 workstations. In [64], the authors prepared the TUIDS dataset at Tezpur University. This completely labeled dataset was captured in both packet and flow level. The ISCX-UNB dataset [39] was constructed by creating profiles for agents and various multi-stage attack scenarios were implemented to generate intrusions. The proposed solution in this thesis is evaluated with the ISCX-IDS 2012 dataset which is a part of ISCX-UNB datasets, since these datasets are recent, realistic and provide full packet captures with attack or normal labels.

3.2 The ISCX-IDS 2012 Dataset

Although there have been numerous benchmark datasets publicly available, many of them include outdated, unmodifiable, inextensible and irreproducible intrusion scenarios [39]. In order to overcome these shortcomings and create more current traffic patterns, the ISCX-UNB dataset was created by the Canadian Institute for Cybersecurity. It contains many different types of datasets to evaluate anomaly based approaches.

The manually labeled ISCX-IDS 2012 dataset exhibits realistic network behavior and contains diverse intrusion scenarios. Furthermore, it is shared as complete network capture with all internal traces to analyze payloads for deep packet inspection. The ISCX-IDS 2012 dataset includes seven days of both normal and malicious network activity. The dataset was generated by profiles containing abstract representations

of events and behaviors in the network. For example, the communication between a sender and a receiver host over the HTTP protocol can be represented by the packets sent or received, end point properties or other similar characteristics. This representation constructs a single profile. These profiles can generate real traffic for HTTP, SMTP, SSH, IMAP, POP3 and FTP protocols [39]. They were used by agents or human operators to inject different scenarios to the network. Profiles are also shared with actual datasets for research groups to regenerate the network behavior of different applications by modifying these profiles.

The ISCX-IDS 2012 dataset includes two distinct profiles to produce network behaviors and scenarios: α and β profiles. While α profiles introduce anomalous behavior or multi stage attack scenarios in the network, β profiles represent the features and mathematical distributions of the procedures. For example, β profile can include distributions of packet sizes, certain patterns in the payload, request time distribution of a protocol etc. On the other hand, α profiles are based on earlier attacks and include sophisticated intrusions for each consecutive day. According to the α profiles, there are four attack scenarios in the complete dataset.

1. **Infiltrating the network from inside:** In this scenario, an attacker gains access to a host from inside. It starts by gathering information about the target such as IP ranges, mail servers and email accounts by using a buffer overflow vulnerability. Common web application hacking techniques such as SQL injection or Cross Site Scripting were implemented to compromise systems.

2. **HTTP denial of service:** The Slowloris denial of service (DoS) attack tool [71] was used in the second scenario to generate a low bandwidth and greedy attack. The Slowloris tool holds connections open by sending incomplete yet valid HTTP requests to servers at regular intervals to keep the sockets from closing. Since servers allow limited amount of threading, sockets are eventually tied up and no connection can be made [72].

3. **Distributed denial of service using an IRC botnet:** An Internet Relay Chat (IRC) bot was used in this scenario to perform distributed denial of service attack since botnets have the ability of combining previously malicious activities into a single platform.

4. **Brute force SSH:** This final scenario was generated with brutessh tool [73] to acquire an SSH account by running a dictionary brute force attack.

The testbed network architecture consists of 21 interconnected Windows workstations divided into four distinct LANs in order to construct a realistic interconnected network. The fifth LAN comprises different servers providing email, DNS and Network Address Translation (NAT) services. The NAT server operates as an Internet service provider for the entire network. The sixth LAN is built for monitoring and maintenance purposes. A network tap is also responsible for transmitting the traffic to multiple devices and monitoring the network [39]. The publicly available capturing period starts at 20:10:39 on Friday June 11th 2010 and ends at 00:01:06 on Friday June

18th. The statistics for the overall dataset are summarized in Table 3.1. As can be seen in Table 3.1, each attack scenario was implemented for only one specific day and two days include only normal traffic. In addition, the authors in [39] state that the variety of the normal network behavior and the complexity of the attack scenarios increase from June 11th to June 18th.

Day	Date	Description	Size (GB)
Friday	11/6/2010	Normal Activity. No malicious activity	16.1
Saturday	12/6/2010	Infiltrating the network from inside + Normal activity	4.22
Sunday	13/6/2010	Infiltrating the network from inside + Normal Activity	3.95
Monday	14/6/2010	HTTP Denial of Service + Normal Activity	6.85
Tuesday	15/6/2010	Distributed Denial of Service using an IRC Botnet	23.04
Wednesday	16/6/2010	Normal Activity. No malicious activity	17.6
Thursday	17/6/2010	Brute Force SSH + Normal Activity	12.3

Table 3.1: Daily Traffic in Dataset

Despite diverse real-life network intrusion scenarios, this dataset has some flaws. For instance, a considerable portion of the IP packets generated by the maintenance and network monitoring processes were unlabeled and left as unknown. Moreover, when the flow records were extracted from the dataset, it was observed that some flows include NaN (not-a-number) values. This might be explained by the properties of different communications. For example, The Address Resolution Protocol (ARP) requests does not use ports. For this reason, sender and receiver port numbers have NaN values for flows having this communication protocol. In addition to the NaN values, the proportion of attack to normal traffic is relatively small. In order to address these issues, the dataset was preprocessed and sanitized. The following sections are dedicated to these preprocessing steps.

3.3 Packet to Flow Conversion with Argus

Argus is an open source network audit record generation tool, which can produce network flow status for every network transaction. It constructs detailed flow records from live or stored packet contents and can store these reports for network security research. It is currently running on Mac OS X, Linux, Solaris, FreeBSD, OpenBSD, NetBSD, AIX, HP-UX, VxWorks, IRIX, Windows (under Cygwin) and OpenWrt [38].

Argus core client programs can generate protocol-specific flow transaction models from live network or stored pcap files and implement basic functions, including printing, processing, sorting, aggregating, tallying, collecting, distributing and archiving flow records. It is also possible to write a generic configuration file with different options for various purposes.

In the preprocessing phase, a status report was generated for each day with the Argus tool. The flow status interval, which controls the period of report on a flow's activity, was selected as 5 seconds for the conversion of the complete dataset.

The preprocessing phase starts with converting pcap files to unlabeled Argus flow records. After the conversion, all possible or desired features were extracted from the flow records and written to a text file. The current version of the Argus can produce up to 125 features of a one flow; however, only numerical features (59 of them) were extracted from the flow records. The overall list of features with definitions can be examined in Appendix B. From these features, IPv4 values were converted into a decimal format and each octet was considered as a different feature. Consequently, the final input data has 67 numerical features.

After converting the pcap file to an unlabeled Argus output, traffic profiles were read from the provided XML files. Overall information acquired from a profile can be seen in Appendix A. A small script was written in order to match these profiles with flow records. Following the matching process, labels for each flow record were collected as another output. Classes were named as "Normal" (-1), "Unknown" (0) and "Attack" (1). This complete process can be seen as a flowchart in Figure 3.1. The number of flows from each class with respect to the time elapsed in June 14 is shown in Figure 3.2. As illustrated in the figure, the number of flows belonging to the unknown class were relatively low compared to the normal class. In addition, the time interval of the attack scenario can be estimated around 5000 epochs from Figure 3.2.

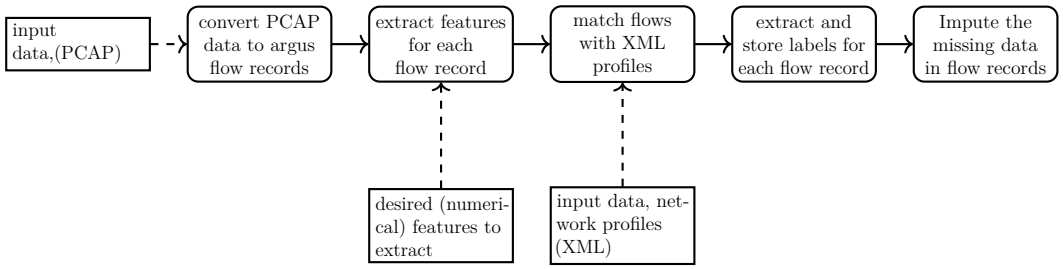


Figure 3.1: Flowchart of Matching Complete Traffic PCAP File with Given XML Profiles

After the flow conversion, it was seen that there were missing statistics (NaN values) in the output files. For example, it was observed that when there are no packet sent in one flow, the mean of packet size becomes unavailable. For these reasons, missing data was imputed with a replacement value -1. This negative value was chosen to emphasize the difference of missing data from the rest of the flow features.

The number of flows and XML profile entries for each day are given in Table 3.2. The first and the second days were given as single, compressed XML profile; therefore, flow records are also combined for these two days.

Day	Argus Flow Records			XML Profiles	
	Normal	Attack	Unknown	matched	total unique
June 11 + June 12	224173	3221	34947	128816	133192
June 13	221148	12853	45179	135080	137160
June 14	220156	12761	46263	169950	171350
June 15	1033972	37402	66332	563583	571220
June 16	663642	0	347914	519146	522251
June 17	612089	3294	19973	180796	397595

Table 3.2: Flow Comparison Results for the Week

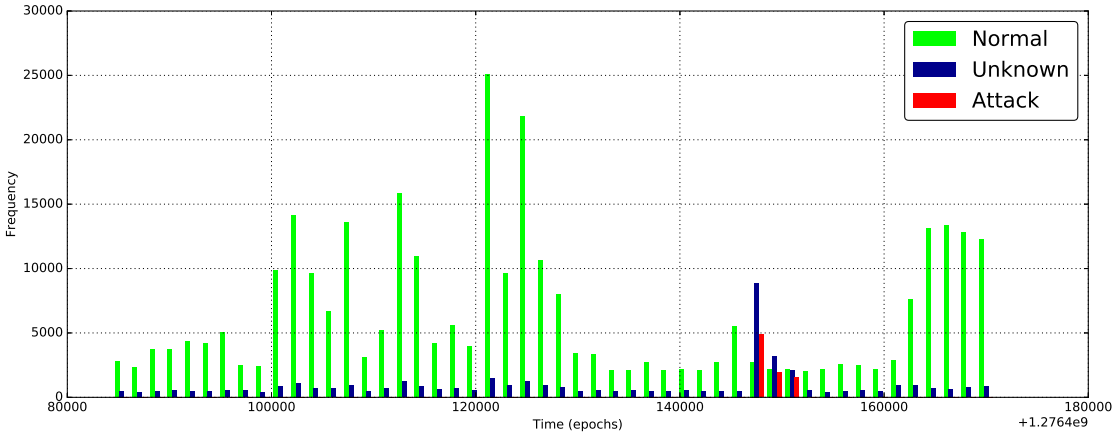


Figure 3.2: Distribution of Flows for June 14

3.4 Evaluation of Dataset with Simple Machine-Learning Based Anomaly Detection Methods

In this section, dimension reduction with principal component analysis and simple machine learning techniques applied to the ISCX-IDS 2012 dataset will be briefly summarized. These methods were implemented and evaluated in order to validate the usability of the dataset.

These machine learning methods were performed using Python 2.7.11, numpy 1.11.0, scipy 0.17.1, Argus 3.0.8.1, in an 64bit Ubuntu 16.04 LTS machine with 8 GB of RAM and CPU of 2.70 GHz. Each method was applied to the largest 20 principal components of the normalized data, which captures 95% of the variance in the original data, and the overall performance was measured with 3-fold cross validation technique.

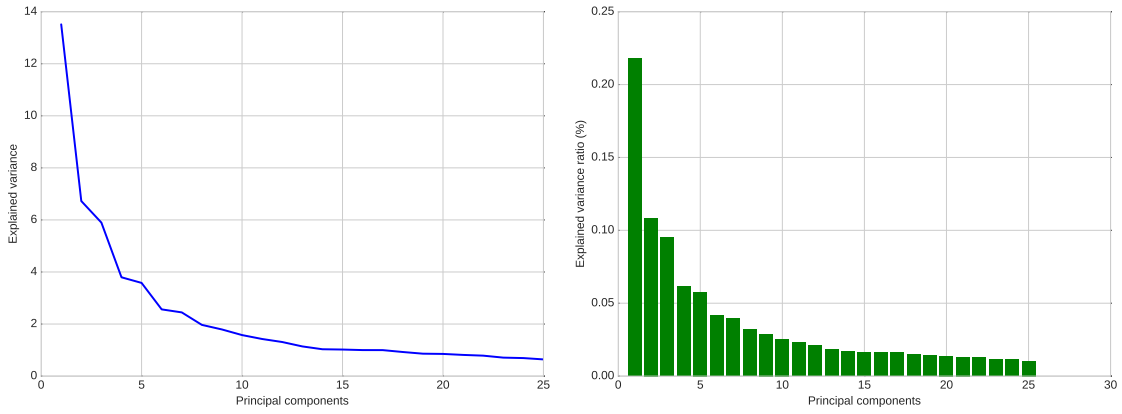
3.4.1 Principal Component Analysis

Principal Component Analysis (PCA) [74, 75] is a simple, non-parametric method, which is used in machine learning, statistics, and in many application areas mostly

for reducing the dimensionality of the data. PCA is mainly implemented for dimensionality reduction and data interpretation for acquiring statistical knowledge.

PCA is an unsupervised projection method finding a mapping from higher dimensional space to a new smaller space with a minimum loss of information. PCA obtains a projection matrix that projects the original data by the basis vectors [75]. If the original data matrix is defined as $\mathbf{X}_{d\times N}$, PCA constructs a projection matrix $\mathbf{W}_{d\times k}$ representing the basis change, where d is the original dimensionality of the data vectors $\mathbf{x}_j$, k is the dimension of the transformed data vectors $\mathbf{y}_j = \mathbf{W}^T\mathbf{x}_j$ after the dimensionality reduction, and N is the number of data vectors $\mathbf{x}_j$. The goal in PCA is to minimize the mean square representation error $||\mathbf{x} - \mathbf{W}\mathbf{W}^T\mathbf{x}||^2$. For a finite dataset, this is equivalent to finding the $d \times k$ matrix $\mathbf{W}$ which minimizes $||\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X}||^2$. Since d is usually much higher than k, this projection maps the input from the original space to a lower dimensional space with minimum loss of information. Finally, the projected data is represented as $\mathbf{Z} = \mathbf{W}^T\mathbf{X}$.

Before applying PCA, it is useful to normalize the data in order to eliminate possible negative effects of having different measurement units for each feature. This preprocessing method produces a mean-centered and unit variance scaled data, and $\mathbf{X}^T\mathbf{X}$ becomes a correlation matrix which will be used by PCA. The column vectors of the optimal PCA transformation matrix $\mathbf{W}_{d\times k}$ consists of k eigenvectors of the data covariance matrix corresponding to its k largest eigenvalues.



(a) (b)

Figure 3.3: PCA Scree Graph

In this work, data normalization is implemented by subtracting the mean μ from the original data and dividing this centered data by the square roots of eigenvalues, as presented in [74]. Therefore, the principal components of $\mathbf{X}_{d\times N}$ will be the eigenvectors calculated from the covariance matrix of the data. These components are extracted and visualized in a scree graph until it reaches a convergence point to retrieve the

variance of the original data. Figure 3.3 illustrates an example of a scree graph, which shows the fraction of the explained variance in a descending order, for June 13. Figure 3.3a plots the amount of variance explained by the each component and Figure 3.3a presents the ratio of explained variance for each component. As shown in the figure, 5 biggest components explains most of the variance, and the total explained variance reaches an almost steady state after some point. As shown in Figure 3.3, 20 principal components can be chosen as a suitable value without losing considerable amount of information, as only less than 5% of the variance are left unexplained.

After the dimension reduction, eight different supervised and unsupervised methods were applied to the dataset. These methods are explained briefly in the following subsections and their results are plotted in Figure 3.6.

3.4.2 K Nearest Neighbor

K nearest neighbor (KNN) is an unsupervised classification method that requires no prior distribution of the sampled data [76]. Each sample x_i is classified by checking its k_n nearest neighbors.

While constructing the nearest neighbor rule, a set of sampled data $x_1, x_2, ..., x_n$ are converted into a metric space. The metric space is usually constructed with a similarity measure such as Euclidean or Manhattan distance functions. This measure is used to obtain the nearest neighbors of a new query point x_i. Finally, the selected k nearest neighbors decide the class through majority voting [76]. Although KNN performs very well, it has a high time complexity since it searches the minimum distance in all the training set for one test sample.

In this preliminary work, 5 equally weighted nearest neighbors were used to predict the label of a query point and the similarity measure was chosen as the Euclidean distance.

3.4.3 Gaussian Naïve Bayes

The Gaussian Naïve Bayes classifier defines a conditional model $P(y_{1...N}|x)$ of the data sample x with N possible outcomes [75]. This model is called posterior probability and represented by Bayes' theorem given in Equation 3.1.

$$P(y_k|x) = \frac{P(y_k)P(x|y_k)}{P(x)}, \tag{3.1}$$

where y_k is the label of class k.

If priors $P(x_k)$ and likelihoods $P(x|y_k)$ can be estimated from the training data, the posterior probability is derived from Bayes' theorem. This model also classifies the test data samples based on highest posterior probability as in Equation 3.2, which

leads to the Maximum A Posteriori estimation. [75]

$$\text{choose } y_i \text{ if } P(y_i|x) = \max_k P(y_k|x). \tag{3.2}$$

In Gaussian Naïve Bayes, the components of the data vectors are assumed to be Gaussian distributed and statistically independent from each other given the class label [75]. Based on these assumptions, the underlying distribution of the data is modeled by multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with location vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The multivariate normal density of observed samples' prior distribution class $k = 1, 2..., N$ can be derived as

$$P(\mathbf{x}|k) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_k|}} \exp(-1/2(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)), \tag{3.3}$$

where the covariance matrix $\boldsymbol{\Sigma}_k$ and mean vector $\boldsymbol{\mu}_k$ are calculated separately for each class k [77].

The Gaussian Naïve Bayes classifier is easy to implement, highly scalable and can be applied with a low computational cost to high dimensional data vectors. Although it assumes that data samples are conditionally independent given the class label, this method can yield reasonably good results even though the components of the data vectors are not Gaussian distributed or the data is not statistically dependent [78].

3.4.4 Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) is a supervised classifier with a quadratic decision boundary, and resembles the Gaussian Naïve Bayes approach. Similar to Naïve Bayes, QDA tries to estimate the posterior probability $P(y_{1...N}|\mathbf{x})$ using Bayes' rule where $\mathbf{x} \in \mathbb{R}^d$. QDA also assumes that observed samples have the characteristic of multivariate normal distribution. Unlike Gaussian Naïve Bayes, QDA does not consider the class conditional independence assumption, and states that data vectors do not have to be independent of other observations given the class label. Therefore, if the covariance matrix $\boldsymbol{\Sigma}$ is diagonal, QDA becomes identical to Gaussian Naïve Bayes [79].

In the case of multiple classes, a QDA predicts the label of a test sample by finding the maximum of posterior distributions which corresponds to Equation 3.2.

QDA performs better with larger training samples as the effect of variance becomes lower and less crucial for this method. Additionally, it can produce better results when the data is moderately non-linear since it assumes a quadratic decision boundary [80].

3.4.5 Multivariate Regression

Logistic regression is a supervised learning model which explains a sample vector $\mathbf{x} \in \mathbb{R}^d$ and desired output $y \in \{0, 1\}$ in a parametric form $P(y|\mathbf{x})$. The desired

output is written as a linear function with weight vector $\mathbf{w} = [w_0, w_1, w_2, ..., w_d]$. The overall distribution can be modeled by logistic regression as in Equation 3.4 and Equation 3.5 [81].

$$Pr(y = 1|\mathbf{x}) = \frac{1}{1 + \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}, \tag{3.4}$$

$$Pr(y = 0|\mathbf{x}) = \frac{\exp\left(w_0 + \sum_{i=1}^{d} w_i x_i)\right)}{1 + \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}. \tag{3.5}$$

In logistic regression, the test data $\mathbf{x}$ is classified by comparing $Pr(y = 0|\mathbf{x})$ and $Pr(y = 1|\mathbf{x})$. The predicted class maximizes the parametric form.

Logistic regression can be extended to the multivariate case when y has N discrete values. In multivariate regression, y is assumed to be expressed as the weighted sum of the input variables $\mathbf{x}$. In this case, the weights w_i of Equation 3.4 and 3.5 are changed into weights w_{ij} associated with the class y_k and the input $\mathbf{x}$. In multivariate logistic regression, the maximum value of $P(Y = y_k|\mathbf{x})$ classifies the testing data as y_k [81].

Logistic regression models are favored in data analysis tools and widely used in medical applications such as diagnosing a patient to find some strong relationships in input features [80]. Unlike QDA, it assumes no Gaussian prior distribution in the training samples. However, these models can be costly and they might deliver unstable results when the training data size is small or the decision boundary is highly non-linear [80].

3.4.6 Decision Trees

Decision trees are non-parametric machine learning methods based on splitting rules and can be used for regression and classification purposes. Decision tree construction is performed with binary splitting as illustrated in Figure 3.4. End nodes in a decision tree lead to a specific region with various class proportions. Therefore, a suitable region is founded for observations and each testing data is predicted as the most frequent class of that region [80].

The purity of each node evaluates the quality of the splits in decision trees. The purity of trees can be calculated with the cross entropy given by Equation 3.6. In the equation, p_{mk} indicates the proportion of training samples from the k-th class in the m-th region. If p_{mk}'s are all near zero or near one, then the cross entropy will produce a smaller value. Therefore, a node with a more negative entropy leads to

more dominated samples from a single class and less classification error for this node [80].

$$H = -\sum_{k=1}^{K} p_{mk} \log(p_{mk}) \tag{3.6}$$

Decision trees are easily interpretable, require minimum data preparation and computational complexity will decrease to log time. Unfortunately, they tend to be unstable since they are affected by small variations in the training samples [82].

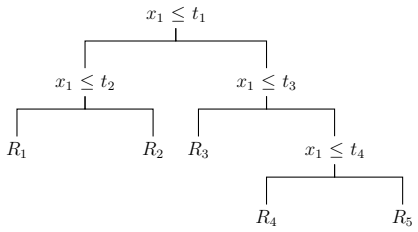


Figure 3.4: Simple Decision Tree with Five Regions

3.4.7 Random Forest

Random forest [83] is an ensemble machine learning method which is based on decision tree bagging. Bagging is a general procedure which aims to reduce the variance of the learning method [82]. As stated in the subsection before, decision trees suffer from high variance; thus, bagging may improve the overall performance

In the training part of the random forest classification, a portion of the data is selected and a decision tree, f_b, is fitted to the sample. This procedure is repeated B times. In the prediction phase, new samples are classified with a majority vote [80].

Random forests are a quite robust machine learning method having high accuracy. Furthermore, they can achieve good performance even for larger datasets [82]. The effect of over-fitting is rare with random forests since they are less sensitive to input variance [80].

3.4.8 Perceptron

The perceptron [84] is a classifier that computes a linear combination of the input features and finds a separating hyperplane. It gives the position of a query sample with respect to the separating hyperplane. Although the perceptron is an old, simple classifier used for trivial classification problems, it is included and tested on the dataset, since it is a former state-of-the-art linear classification tool and can be compared to other linear classifiers in this section. Furthermore, it is considered as the smallest component of neural network systems [84] and ELM, which will be explained in detail in section 4.5.

A basic perceptron model is given in Figure 3.5. As illustrated in the figure, the perceptron takes a weighted sum of the input. It typically applies a non-linear activation function to this weighted sum and predicts the class of the input sample with a threshold value decided beforehand.

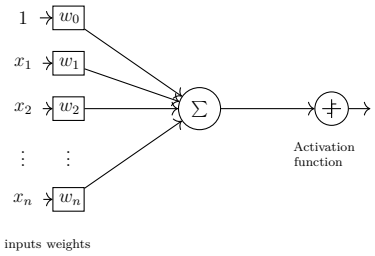


Figure 3.5: Single Perceptron Structure

The perceptron learning algorithm fits a separating hyperplane to a decision boundary using weights $w_0, w_1..., w_n$. The main objective of the algorithm is to minimize the distance of misclassified samples using a stochastic gradient algorithm [82]. In the case of multiple classes, the perceptron algorithm can be combined with a one-vs-all scheme.

Although the perceptron is very fast even with large datasets, the algorithm does not converge well when the data is not linearly separable. Moreover, the solution highly depends on the initial values of weights [85].

3.4.9 Extreme Learning Machine

Extreme Learning Machine (ELM) [86] is a single hidden layer feedforward neural network. It can be used for both regression and classification problems. The ELM method provides a simple learning algorithm which is faster than those of traditional feedforward networks. The ELM algorithm initializes input and hidden node weights randomly. This learning method solves the learning problem by the Least Squares (LS) method.

Since ELM forms the learning part of this thesis, section 4.5 will provide a more detailed discussion of this method.

3.4.10 Results of Prior Machine Learning Techniques

Machine learning techniques indicated above were applied to each day of the dataset and evaluation metrics were extracted for discussion. Figure 3.6 and Table 3.4 show a comparison between these techniques. On June 12, a sample of infiltration attack was used in training. Since this attack was performed on distinct ports, the detection rate of normal and attack data was quite high for many machine learning algorithms, as seen in the figure. The same case can be seen on June 17 with brute force SSH

attack. The measurements of attack data on June 16 were not marked on the figure since this day contains zero attack. In addition, Figure 3.6 reveals that the attack traffic on June 14 has the worst precision and false positive rate. This discrepancy is expected because June 14 contains a HTTP DoS attack created with the Slowloris tool. This type of stealthy intrusions is hard to detect, since they look like legitimate flows, follow protocol rules and handshake procedures [72]. Similarly, false positives and false alarms are high on June 15 due to the DDoS attack. However, the false positive rate of the attack data on June 15 differs from the expected results. This inconsistency can be explained by checking normal-to-attack ratio of the data. June 15 has considerably more attack flows and false predictions than those of other days, as illustrated in Table 3.3. Therefore, this day has a lower false positive rate of attack flows.

As can be seen in Figure 3.6 and in Table 3.1, the time complexity of many methods increases as the data size grows. Normal flows have high precision and low false alarm rate in almost every day in the dataset. However, it was found that the precision of unknown labels varies each day. This might be caused by the nature of these flows. Since unknown flows are transpired by the maintenance and monitoring, they are treated differently than normal or attack data.

From these eight machine learning methods, KNN, logistic regression and decision trees show better performance than those of other techniques. However, KNN and decision trees have quite high time complexity. Moreover, logistic regression has a quite low detection rate for DoS attacks. On the other hand, ELM shows relatively poor performance despite its low time complexity. KNN suffers from the curse of dimensionality [87], which leads to a huge increase in the volume of the training data. Thus, KNN requires more computations if no dimension reduction technique for simpler representation is applied to the data. Therefore, the performance of the proposed method will be mainly compared to the decision trees and ELM.

		Predicted		
		Normal	*Unknown*	*Attack*
Actual	*Normal*	220024	1012	112
	Unknown	14842	26009	4328
	Attack	2423	597	9833

(a)

		Predicted		
		Normal	*Unknown*	*Attack*
Actual	*Normal*	1026753	621	6598
	Unknown	30502	35775	55
	Attack	3469	0	33933

(b)

Table 3.3: Confusion Matrices for ELM with June 13 (a) and June 15 (b)

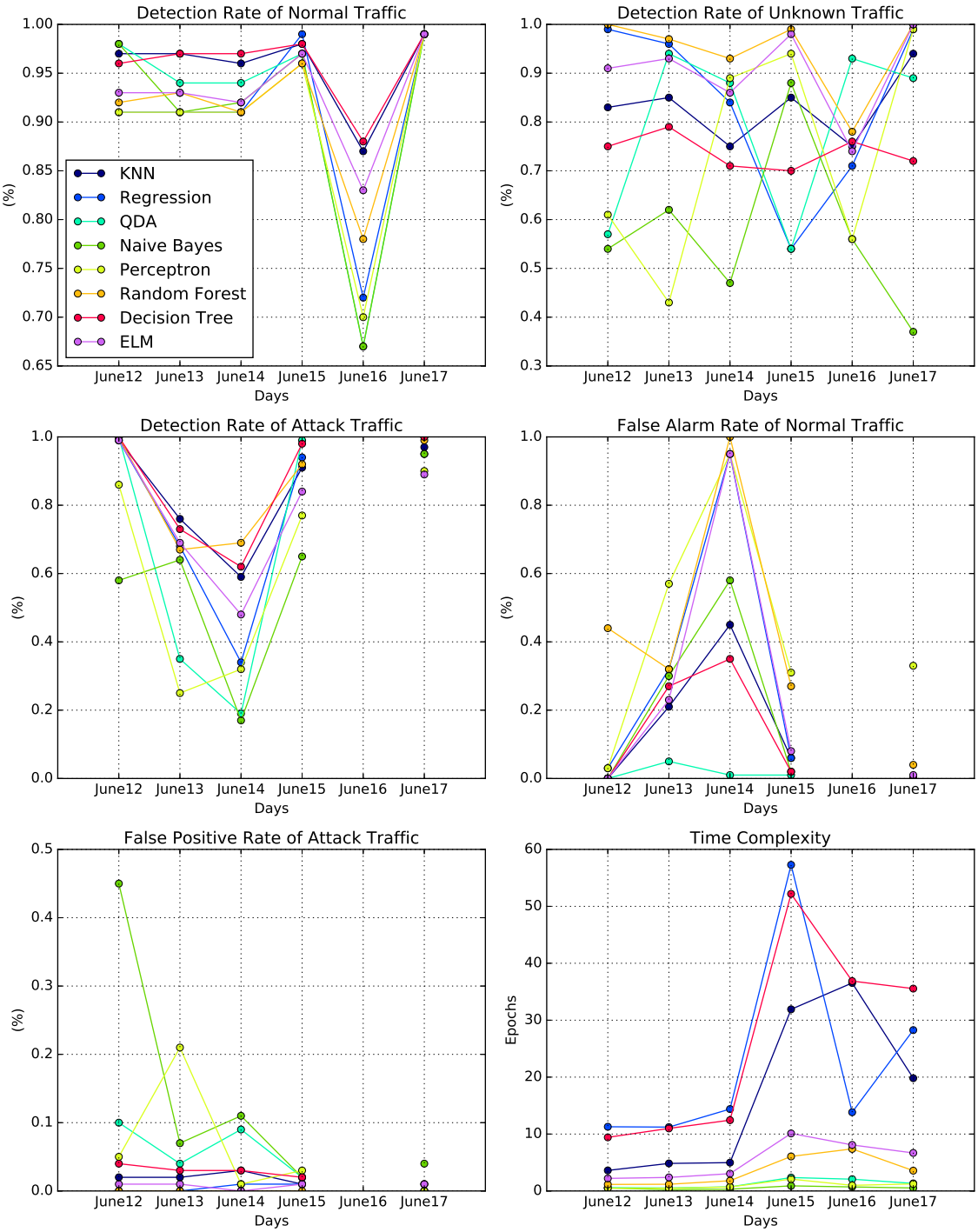


Figure 3.6: Comparison of Various Traditional Machine Learning Techniques

Detection Rate of Normal Traffic / Detection Rate of Unknown Traffic

Classifier	June 12	June 13	June 14	June 15	June16	June 17	June 12	June 13	June 14	June 15	June16	June 17
KNN	0.97	**0.97**	0.96	0.98	0.87	**0.99**	0.83	0.85	0.75	0.85	0.75	0.94
Linear Regression	0.91	0.91	0.91	0.99	0.72	**0.99**	**0.99**	0.96	0.84	0.54	0.71	0.99
QDA	**0.98**	0.94	0.94	0.97	0.67	**0.99**	0.57	0.94	0.88	0.54	**0.93**	0.89
Naïve Bayes	**0.98**	0.91	0.92	0.97	0.67	**0.99**	0.54	0.62	0.47	0.88	0.56	0.37
Perceptron	0.91	0.91	0.91	0.96	0.70	**0.99**	0.61	0.43	0.89	0.94	0.56	0.99
Random Forest	0.92	0.93	0.91	0.96	0.78	**0.99**	**1.00**	**0.97**	**0.93**	**0.99**	0.78	**1.00**
Decision Tree	0.96	**0.97**	**0.97**	0.98	**0.88**	**0.99**	0.75	0.79	0.71	0.70	0.76	0.72
ELM	0.93	0.93	0.92	0.97	0.83	**0.99**	0.91	0.93	0.86	0.98	0.74	**1.00**

Detection Rate of Attack Traffic / False Alarm Rate of Normal Traffic

Classifier	June 12	June 13	June 14	June 15	June16	June 17	June 12	June 13	June 14	June 15	June16	June 17
KNN	0.99	**0.76**	0.59	0.91	-	0.97	**0.00**	0.21	0.45	0.06	-	**0.00**
Linear Regression	0.99	0.68	0.34	0.94	-	0.95	0.03	0.32	0.95	0.06	-	**0.00**
QDA	**1.00**	0.35	0.19	**0.99**	-	**1.00**	**0.00**	**0.05**	**0.01**	**0.01**	-	**0.00**
Naïve Bayes	0.58	0.64	0.17	0.65	-	0.95	0.00	0.30	0.58	0.02	-	**0.00**
Perceptron	0.86	0.25	0.32	0.77	-	0.90	0.03	0.57	0.95	0.31	-	0.33
Random Forest	**1.00**	0.67	**0.69**	0.92	-	0.99	0.44	0.32	1.00	0.27	-	0.04
Decision Tree	**1.00**	**0.73**	0.62	0.98	-	**1.00**	**0.00**	0.27	0.35	0.02	-	**0.00**
ELM	0.99	0.69	0.48	0.84	-	0.89	**0.00**	0.23	0.95	0.08	-	0.01

False Positive Rate of Attack Traffic / Time Complexity (in Epochs)

Classifier	June 12	June 13	June 14	June 15	June16	June 17	June 12	June 13	June 14	June 15	June16	June 17
KNN	0.02	0.02	0.03	0.01	-	**0.00**	3.61	4.84	4.98	31.91	36.54	19.81
Linear Regression	**0.00**	**0.00**	0.01	0.01	-	**0.00**	11.28	11.22	14.39	57.29	13.83	28.26
QDA	0.10	0.04	0.09	0.02	-	**0.00**	0.59	0.53	0.72	2.36	2.07	1.33
Naïve Bayes	0.45	0.07	0.11	0.02	-	0.04	**0.51**	**0.24**	**0.31**	**0.90**	**0.71**	**0.53**
Perceptron	0.05	0.21	0.01	0.03	-	**0.00**	**0.51**	0.49	0.71	2.08	0.99	1.24
Random Forest	**0.00**	**0.00**	**0.00**	**0.00**	-	0.01	1.15	1.18	1.80	6.07	7.39	3.57
Decision Tree	0.04	0.03	0.03	0.02	-	0.01	9.42	10.99	12.44	52.19	36.87	35.55
ELM	0.01	0.01	**0.00**	0.01	-	0.01	2.21	2.38	3.04	10.11	8.09	6.68

Bold face: best value for different classifiers

Table 3.4: Comparison of Various Traditional Machine Learning Techniques

3.5 Related work with the Dataset

Network traffic traces play a key role in evaluating the performance of any intrusion detection method. Evaluations are more convincing when IDSs are tested with a comprehensive and complete network dataset. Despite the importance of dataset quality, few research has been done using the ISCX-IDS 2012 dataset.

One of the earliest methods evaluated with the ISCX-IDS 2012 dataset was proposed in [88], which integrated K-means and Naïve Bayes classifiers. The proposed anomaly detection algorithm was tested by selecting incoming packets for a specific host in one day. Kumar *et al.* [89] applied a multi objective genetic algorithm to the KDD Cup 1999 dataset as well as to a subset of the ISCX-IDS 2012 dataset. In [90], decision trees were constructed based on the alerts issued by Snort IDS. In this study, only 5 features (protocol, source IP, source port, destination IP, destination port) were extracted from the dataset and decision trees were generated by using these attributes. In [91], another ensemble learning method was developed and tested with a flow level version of the ISCX-IDS 2012 dataset. The Flowcalc tool [92] was preferred in the packet-to-flow conversion and relatively basic statistics were extracted from the dataset. Tan *et al.* [93] analyzed traffic records by converting them to respective images. This study focused on mitigating DoS attacks and deployed object shape recognition principle with a special distance metric called Earth Mover's Distance (EMD). In addition, the authors in [93] applied PCA to the destination network traffic which has only basic features. Similarly, Vasan *et al.* [94] analyzed the effect

of PCA in intrusion detection by implementing different machine learning algorithms to ISCX-IDS 2012 dataset.

All previous studies in [88]-[94] either selected a subset of complete dataset or extracted only a few features to represent the network traffic. While these approaches can be simple, other complex features should be investigated to detect more stealthy attacks with a good precision. In this thesis, a model is developed using complete traffic capture in flow level with as many features as possible to achieve better detection mechanism.

3.6 Summary

This chapter has presented different benchmark datasets and briefly explained some of their drawbacks. The ISCX-IDS 2012 dataset has been introduced and performance evaluation with various machine learning algorithms has been illustrated. Finally, a review on the prior research using this dataset has been summarized. The following chapter will present the mathematical background constructing the overall architecture of the proposed system and the application of possible different approaches of it.

Chapter 4

Methodology

This chapter provides the mathematical background as well as the methods required for constructing the proposed system. Firstly, the general architecture of the system is explained. Secondly, the function of each component and mathematical method are presented in detail. Finally, the implementation of different on-line and off-line learning mechanisms with the ISCX-IDS 2012 dataset are introduced.

4.1 General Architecture

Figure 4.1 illustrates the high level architecture of the designed system. The proposed intrusion detection system consists of three parts: Network flows aggregation, conversion from feature to likelihood domain, and machine learning. In order to aggregate network data efficiently, prefix trees and the *hierarchical heavy hitters* (HHH) approach [95] were used. As stated in [95], hierarchical algorithms are usually used for network aware-clustering and detecting DoS attack patterns. In addition, the authors stated that hierarchical algorithms can provide better exploration of high-volume clusters than straightforward methods. In the proposed IDS structure, HHH aggregation was defined on the destination IP addresses due to the hierarchical addressing scheme of IP flows or packets. Section 4.3 explains HHH detection for different classes in more detail. After detecting traffic clusters, every numerical feature of each cluster was treated as a set of events in a sample space and probabilities were assigned to these events. In other words, probability density functions were estimated from various numerical flow statistics of all heavy hitters. Finally, machine learning was implemented for each cluster by using probabilities as input.

The complete mechanism with the HHH algorithm and machine learning in the Figure 4.1 was written from scratch. A black box programming method, where the inner functions in the overall system need not to be examined if inputs change, was used in every step in order to ensure that each block is isolated and can be integrated to other systems.

Different variants of the proposed intrusion detection system were implemented in order to test the performance of both batch and streaming processing. In batch

learning, the proposed IDS is trained and tested with high volumes of data. On the other hand, sequential learning was done by using small chunks of the network traffic over time and updating the machine learning part when new patterns are observed. These different implementations are explained in section 4.6.

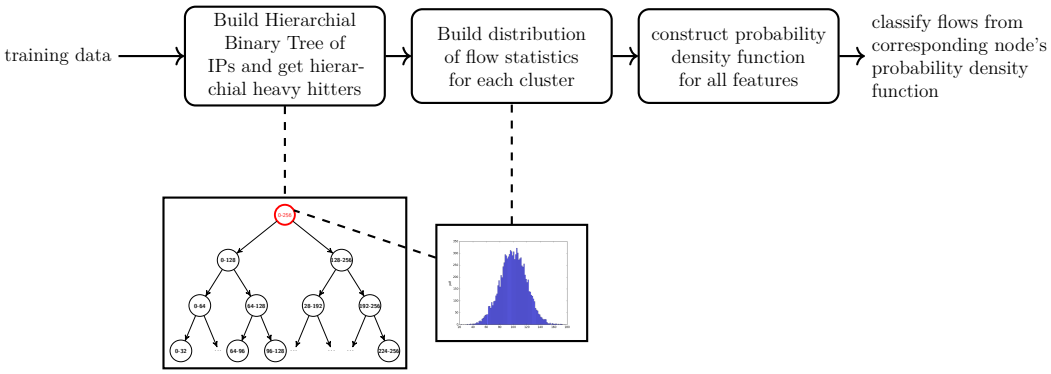


Figure 4.1: High Level Description of the Methodology

4.2 Trie Data Structure

A trie [96], which is also known as a prefix tree, is an ordered data structure based on the prefix of a string. Tries can visualize strings as a tree structure in a hierarchical manner. Tries have some advantages over binary search trees: For example, searching a word with length of M takes $O(M)$ time in the worst case. On the other hand, binary search trees perform $O(\log(n))$ comparisons, where n is the number of nodes. Therefore, binary search trees take $O(M \log(n))$ time for searching a word in the worst case. In [97], it was also stated that tries are more space efficient than binary search trees.

The proposed method has a natural hierarchical structure since it monitors destination IPs and aims to achieve a minimum number of operations, such as searching and inserting an IP value. Therefore, the basic idea of a trie data structure was applied for the IP lookup problem. While maintaining the standard trie structure, the algorithm was designed for 8 bit IPv4 prefix trees to simplify the problem. In this design, the first octet (first three members from the dotted decimal representation) of an 32-bit IPv4 address was used to build prefix trees. The other parts of the IPv4 address representing subnetworks and hosts are not taken into consideration in order to get a tree with smaller depth.

Trie search was implemented for solving the IP address lookup problem [98]. A simple IP lookup operation for the proposed method is illustrated in Figure 4.2. As can be seen in the figure, the most significant 8-bits representing the first octet are extracted from IPv4 destination addresses. In other words, a 255.0.0.0 subnet mask is applied to each IPv4 address in sampled flows. Similar to binary search trees, the

overall structure has a root containing all IPv4 destination addresses. In addition, each node contains an ordered left or right subtree except leaf nodes, as seen in Figure 4.2. In our design, each node n contains the depth, pointers to the children, IP range associated with prefix and the volume of traffic for its sub-trie.

Searching and insertion operations of individual IP addresses are done recursively, which is quite similar to binary search tree operations [99]. It should be noted that trie modification by deletion is not implemented in the thesis.

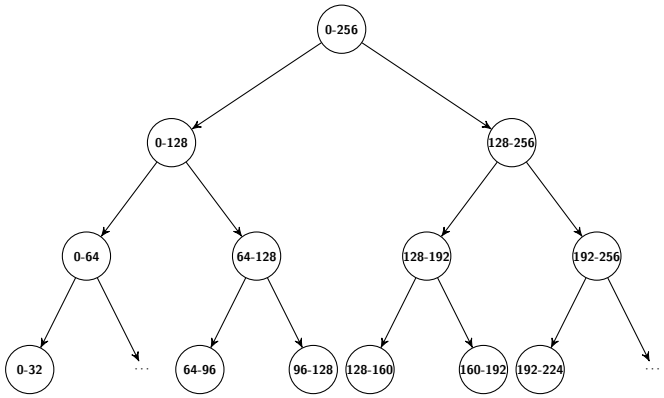


Figure 4.2: Example of IP Trie Structure

4.3 Hierarchical Heavy Hitters

Heavy hitters is a data mining technique used for identifying and clustering frequent patterns in hierarchical data and, since this type of data can be aggregated and visualized at different levels [100]. It was proposed for detecting DoS attacks, since such attacks send huge amount of SYN requests to a victim's machine in order to occupy all the resources of the host system [95]. A *heavy hitter* is a cluster which has more frequent items than some specified threshold value. For example, after aggregating and clustering IP addresses as in Figure 4.2, heavy hitters can be found by comparing the volume of each node to some threshold value. As stated before, an IP address has a hierarchical structure, since it contains 32-bits length information about the network, subnetworks and host. For this reason, *hierarchical heavy hitters* (HHH) can be defined for data streams with IP traffic. Given a proportion φ and a data stream of length N which has hierarchical statistics similar to IPv4 address, p is a HHH node in the hierarchy if it has total number of descendant nodes higher than $N\varphi$ after extracting descendants of p which are already HHH nodes themselves [95].

Figure 4.3 shows an example of HHH results with 100 flow records. In the figure, a threshold value was used as $\varphi = 0.1$ for clustering. It means that a node is designated as an HHH if its *volume* is at least $N \div \varphi = 10$. The volume of a node is defined as the total number of leaf elements under that node minus the cumulative volume of all of its descendant nodes that are already designated as HHHs. Detected

HHHs in Figure 4.3 were drawn and highlighted with a red marker. In the figure, node F is an HHH since it has a volume of 10 from its descendant nodes which are not HHH. On the other hand, node G is not an HHH since its child node is already an HHH, leading to a decrease in its volume from 45 to 2. In this case, G contains less volume than $N\varphi = 10$. The same case can be seen for node B and node C. Node B has 10 descendant nodes after removing its child nodes which are already HHHs; therefore node B is also an HHH. Unlike node B, node C is not an HHH since it has a volume of 2 after subtracting its child nodes which are HHHs.

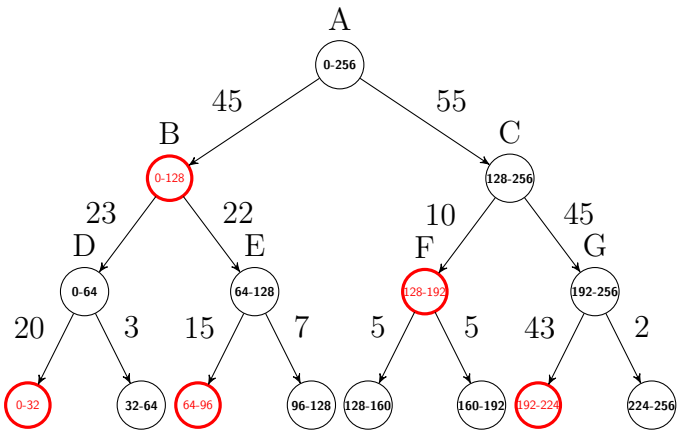


Figure 4.3: Illustration of the Hierarchical Heavy Hitters Concept ($N = 100$, $\varphi = 0.1$)

Algorithm 1: Detecting HHHs from IP Trie

```
1  function ExtractHHH (trie, φ);
   IP trie,freq : An IP trie trie and frequency φ
   HHHs         : list of HHHs
2  threshold = trie.root.volume × φ ;
3  for node n_i in the postorder do
4      if n_i.volume ≥ threshold then
5          HHHs.append(n_i)
6      end
7      for node n_j in the postorder do
8          if n_i in the subtree of n_j then
9              n_j.volume = n_j.volume − n_i.volume;
10         end
11     end
12 end
13 if trie.root.volume ≥ 0 then
14     HHHs.append(trie.root)
15 end
16 return HHHs
```

In the hierarchical heavy hitters algorithm, detected clusters are non-overlapping. This enables each flow to be in only one cluster and not to be found in the parents of that node. In order to ensure that all flows are matched with one cluster, a root node was also included as an HHH if the total flow record has left unclustered items. The procedure for detecting HHHs is summarized in Algorithm 1. In this algorithm, HHHs are detected using postorder traversal method [99], which visits first the left sub-tree, then the right-subtree and finally the root. Detected HHHs are appended into a list of object for further analysis which is explained in the following sections.

4.4 Mapping Features into Probability Space

Assuming that X is a discrete random variable defined on a sample space S, then the probability density function $f(x)$ (also referred as probability mass function for discrete random variables) can be described as

$$f(x) = P(X = x) = P(\{s \in S : X(s) = x\}), \tag{4.1}$$

where $f(x) \geq 0$ and $\sum_x f(x) = 1$. It can also be written by using the Dirac delta function [101] for n possible discrete variables as

$$f(x) = \sum_{i=1}^{n} P_i \times \delta(x - x_i). \tag{4.2}$$

This function can map each real valued random variable X into a probability space. Similarly, the proposed method tries to find a mapping for the probability density function (PDF) by treating the features as random variables.
As explained in subsection 4.3, the most frequent clusters were extracted from the network data with the HHH method. Each of these clusters has d flow features over a time window t. If a cluster has N flows over this period t, then a matrix for the k-th cluster can be formulated with $\mathbf{X}_k$:

$$\begin{aligned} \mathbf{X}_k &= \begin{bmatrix} \mathbf{x}_{k_1} & \cdots & \mathbf{x}_{k_N} \end{bmatrix}^T \\ &= \begin{bmatrix} x^1_{k_1} & \cdots & x^d_{k_1} \\ \vdots & \cdots & \vdots \\ x^1_{k_N} & \cdots & x^d_{k_N} \end{bmatrix}_{N \times d} \end{aligned}, \tag{4.3}$$

where $x^d_{k_n}$ represents the d-th flow feature for the n-th flow in cluster $\mathbf{X}_k$. While classical machine learning methods directly use this matrix in their learning part, the proposed IDS finds the associated probability value for each feature in a sample flow and trains the machine learning part with the representation of $\mathbf{X}_k$ in the probability space.

Equation 4.4 describes the matrix of the constructed probability density functions $\mathbf{F}_{\mathbf{x}_k}$, and Equation 4.5 defines the matrix $\tilde{\mathbf{X}}_k$ as the pdf estimates of features for cluster k in the probability space.

$$\mathbf{F}_{\mathbf{X}_k} = \begin{bmatrix} f^1_{\mathbf{X}^1_k} & \cdots & f^d_{\mathbf{X}^d_k} \end{bmatrix}, \tag{4.4}$$

$$\tilde{\mathbf{X}}_k = \begin{bmatrix} f^1_{\mathbf{X}^1_k}(x^1_{k_1}) & \cdots & f^d_{\mathbf{X}^d_k}(x^d_{k_1}) \\ \vdots & \cdots & \vdots \\ f^1_{\mathbf{X}^1_k}(x^1_{k_N}) & \cdots & f^d_{\mathbf{X}^d_k}(x^d_{k_N}) \end{bmatrix} = \begin{bmatrix} \tilde{x}^1_{k_1} & \cdots & \tilde{x}^d_{k_1} \\ \vdots & \cdots & \vdots \\ \tilde{x}^1_{k_N} & \cdots & \tilde{x}^d_{k_N} \end{bmatrix}_{N \times d}, \tag{4.5}$$

where $f^i_{\mathbf{X}^i_k}$ is the probability density function of the i'th flow statistic for cluster k. It is important to note that $\tilde{x}^i_{k_j}$ refers to the estimated probability of the i-th feature of the j-th flow sample.

Probability density function can be estimated by the use of a histogram approach [87]. The histogram method [87] first partitions a discrete feature x into bins of width Δ_i. Secondly, it counts the number of observations n_i of x which falls into the bin i. Finally, it divides the count with the total number of observations N in order to find the normalized probability value for each bin as formulated in Equation 4.6. If all bins have equal weight $\Delta_i = \Delta$, then it gives an estimated model for the probability value of each bin [87].

$$P_i = \frac{n_i}{N\Delta_i}. \tag{4.6}$$

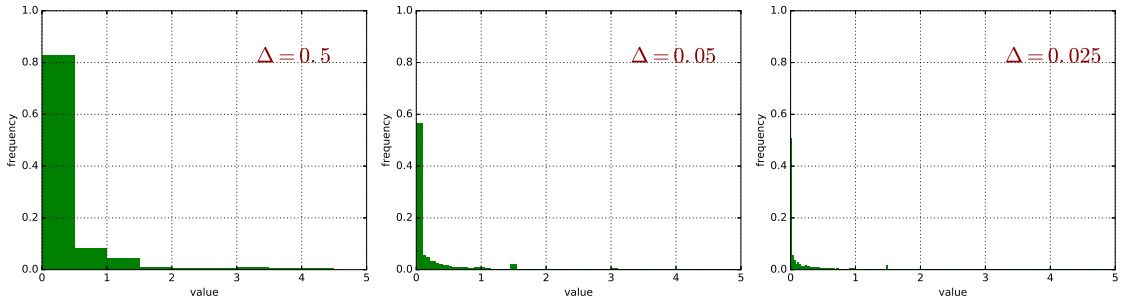


Figure 4.4: Total Flow Duration Histogram for Normal Cluster 192.x.x.x

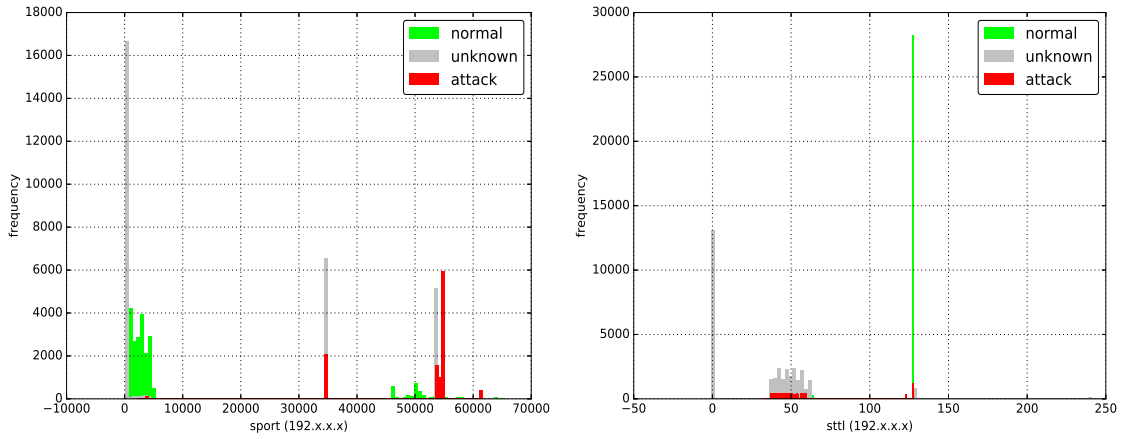
Figure 4.4 illustrates the histogram approach in density estimation based on the different choices of common bin width Δ. The normalized histogram for flow duration within one cluster was plotted in the figure. It can be observed that when Δ is very small, the histogram is unable to estimate the underlying pattern in the network since it shows too much individual data. Conversely, if bins are too large, then the resulting graph was too smooth to capture the real behavior of the histogram [87]. There are numerous methods for finding optimal bin width to determine the goodness of fit in histogram data [102] . The most common method is finding a bin width Δ that minimizes the Mean Squared Error (MSE) [103] between the approximated histogram function and the real distribution. Shimazaki *et al.* [104] also proposed a method to minimize the global measure of estimation accuracy which has less time complexity than calculating the MSE. However, both of these methods might be computationally complex in the proposed approach, since more than fifty histograms have to be calculated for each HHH. Therefore, we chose a simpler method in order

to determine the bin width. The best results for most of the features were obtained when the bin width had been calculated as $\Delta = (\max(\mathbf{X}_k^d) - \min(\mathbf{X}_k^d))/100$ by fixing number of bins to 100 for the d-th feature of the k-th cluster.

After histograms were computed for each feature in one cluster, the dataset itself can be discarded due to the histogram property [87]. Moreover, assuming that data points arrive sequentially, histograms can predict the probability of that point by comparing data with a simple lookup table. In contrast to normalization techniques, it does not suffer from the loss of information since it is insensitive to input vector range.

In order to prove that attack and normal traffic are not identical in all aspects, some selected features were compared for the same destination address. As can be seen in Figure 4.5, comparison results are in very good agreement with this assumption. For a specific destination IPv4 address, the source port and time-to-live statistics were clearly distinguishing features between normal and attack traffic.

After converting input features to probability densities, training and predictions were performed with the Extreme Learning Machine based approaches.



(a) (b)

Figure 4.5: Distribution of (a) Source Port and (b) Time-to-Live Features for June 13

4.5 Extreme Learning Machines (ELM)

The Extreme Learning Machine (ELM) algorithm was proposed by Huang *et al.* in [86]. ELM is one adaptation of single hidden-layer feedforward neural networks (SLFN) [105]. ELM randomly initializes the input weights and updates only the output weights in a single iteration, which differs from other neural network structures in order to reduce computational time [106]. It also provides robust learning capabilities despite the mathematically simple structure [107].

4.5.1 Batch ELM

Considering a set of N arbitrary samples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{1 \leq i \leq N}$ where $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{id}]^T \in \mathbb{R}^d$ and $\mathbf{y}_i = [y_{i1}, y_{i2}, \cdots, y_{ic}]^T \in \mathbb{R}^c$. Then a SFLN with M hidden neurons can be represented as

$$\sum_{i=1}^{M} \beta_i \phi(\mathbf{w}_i^T \mathbf{x}_j + b_i) \quad j \in [1, N], \tag{4.7}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \cdots, w_{id}]^T$ is the weight vector connecting the i-th hidden node to the input nodes, b_i is the bias of the i-th hidden node, $\beta_i = [\beta_{i1}, \beta_{i2}, \cdots, \beta_{ic}]^T$ is the output weight vector connecting the i-th hidden node to the outer nodes and $\phi : \mathbb{R} \to \mathbb{R}$ is the activation function.

If a SLFN can approximate N samples without error, then the difference between the estimated outputs $\tilde{\mathbf{y}}_j$ and actual outputs $\mathbf{y}_j$ should be equal to zero. It also expresses $\mathbf{w}_i$, β_i and ϕ such that

$$\sum_{i=1}^{M} \beta_i \phi(\mathbf{w}_i^T \mathbf{x}_j + b_i) = \mathbf{y}_j \quad j \in [1, N], \tag{4.8}$$

which can eventually be written as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{Y}, \tag{4.9}$$

where

$$\mathbf{H} = \begin{bmatrix} \phi(\mathbf{w}_1^T \mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_M^T \mathbf{x}_1 + b_M) \\ \vdots & \cdots & \vdots \\ \phi(\mathbf{w}_1^T \mathbf{x}_N + b_1) & \cdots & \phi(\mathbf{w}_M^T \mathbf{x}_N + b_M) \end{bmatrix}_{N \times M}, \tag{4.10}$$

and

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_M^T \end{bmatrix}_{M \times c}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix}_{N \times c}, \tag{4.11}$$

assuming that the hidden layer has M nodes.

In [86], Huang *et al.* defines $\mathbf{H}$ as the hidden layer output matrix of the neural network where the i-th column of $\mathbf{H}$ represents the i-th hidden node output.

Unlike back-propagation methods, ELM tries to minimize the squared Euclidean norm of the error matrix $||\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}||^2$ for the randomly initialized input weights $\mathbf{w}_i$ and biases b_i. In most cases, there are more equations than unknowns in the linear system shown in Equation 4.9. Therefore, $\mathbf{H}$ might be defined as a non-square matrix since the number of hidden nodes is less than the number of training samples. In this case, there may not be an exact solution. Therefore, ELM finds the pseudoinverve solution of the system as

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{Y}, \tag{4.12}$$

where $\mathbf{H}^{\dagger}$ is the *Moore-Penrose generalized inverse* [108] of $\mathbf{H}$. This generalized inverse can be calculated by using several methods. If $\mathbf{H}^T\mathbf{H}$ is nonsingular, then the pseudoinverve solution can be converted to the least-squares solution in Equation 4.13.

$$\hat{\beta} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}\mathbf{Y}. \tag{4.13}$$

Singular value decomposition can also be used to calculate the Moore-Penrose generalized inverse [86].

ELM can solve both regression and classification problems. In the case of multi-class classification, the predicted class assignment is based on the closest numerical value when the class labels are converted into numerical representations $\{-1, 0, 1\}$. Since the benchmark dataset consists of three different classes (normal, attack, unknown), ELM can be adapted in the learning phase of the proposed solution.

The detailed theoretical proofs of generalized inverse and the complete ELM procedure can be found in [108] and [86], respectively.

4.5.2 On-line Sequential ELM (OS-ELM)

Most of the SLFN applications are designed as batch-learning training methods. Batch-learning is a type of off-line learning that needs a whole dataset for efficient performance. Although, Huang *et al.* [86] reported that ELM is extremely fast and performs better than other batch training methods, batch learning itself is usually time consuming since it requires many iterations to converge into a solution [109]. Moreover, it needs complete re-computation of the model parameters whenever a new data stream arrives. In addition to this challenge, if the received sequential data arrives faster than the computational time, re-computation may be aborted to process the newly observed data chunk.

In order to avoid these problems, Liang *et al.* [109] developed the On-line Sequential Extreme Learning Machine (OS-ELM) algorithm that can learn the sequential training observations. OS-ELM handles observations that arrive one-by-one and chunk-by-chunk with fixed or varied chunk length. When the new data is received, OS-ELM is able to update the model without requiring the entire past information and without full retraining.

The batch ELM presented in Section 4.5 assumes that all training samples are available for training. OS-ELM divides the training part into two phases: initial learning and sequential learning. In the initial learning phase, it assumes the observed data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N_0}$ as a first chunk and $N_0 \geq M$ where M is the number of hidden neurons. Under those assumptions, the OS-ELM algorithm tries to minimize the l_2 norm of the error in the first chunk (represented as the 0-th chunk) $\|\mathbf{H_0}\boldsymbol{\beta} - \mathbf{Y}_0\|$

where

$$\mathbf{H}_0 = \begin{bmatrix} \phi(\mathbf{w}_1^T\mathbf{x}_1 + b_1) & \cdots & \phi(\mathbf{w}_M^T\mathbf{x}_1 + b_M) \\ \vdots & \cdots & \vdots \\ \phi(\mathbf{w}_1^T\mathbf{x}_{N_0} + b_1) & \cdots & \phi(\mathbf{w}_M^T\mathbf{x}_{N_0} + b_M) \end{bmatrix}_{N_0 \times M}, \tag{4.14}$$

and

$$\mathbf{Y}_0 = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_{N_0}^T \end{bmatrix}_{N_0 \times c}. \tag{4.15}$$

In [109], an orthogonalization method was implemented to find the Moore-Penrose generalized inverse of hidden layer output matrix $\mathbf{H}$. Therefore, the solution to the l_1 minimization problem $\|\mathbf{H}_0\boldsymbol{\beta} - \mathbf{Y}_0\|$ can be solved as

$$\boldsymbol{\beta}^0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}\mathbf{H}_0^T\mathbf{Y}_0. \tag{4.16}$$

After this initial training, a new chunk of data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=N_0+1}^{N_0+N_1}$ with N_1 samples is concatenated to the end of the old dataset to find the new $\boldsymbol{\beta}$ by minimizing

$$\left\| \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \boldsymbol{\beta} - \begin{bmatrix} \mathbf{Y}_0 \\ \mathbf{Y}_1 \end{bmatrix} \right\|. \tag{4.17}$$

Therefore, the output weights $\boldsymbol{\beta}^1$ can be updated with

$$\boldsymbol{\beta}^1 = \boldsymbol{\beta}^0 + (\mathbf{H}_0^T\mathbf{H}_0 + \mathbf{H}_1{}^T\mathbf{H}_1)^{-1}\mathbf{H}_1^T(\mathbf{Y}_1 - \mathbf{H}_1\boldsymbol{\beta}^0). \tag{4.18}$$

The algorithm can be generalized to the $(k+1)$-th chunk of newly arrived dataset $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=(\sum_{j=0}^{k} N_j)+1}^{\sum_{j=0}^{k+1} N_j}$ with N_{k+1} sequential observations. The equations for updating $\boldsymbol{\beta}^{k+1}$ can be written as

$$\begin{aligned} \boldsymbol{\beta}^{k+1} &= \boldsymbol{\beta}^k + \mathbf{P}_{k+1}\mathbf{H}_{k+1}^T(\mathbf{T}_{k+1} - \mathbf{H}_{k+1}\boldsymbol{\beta}^k) \\ \mathbf{P}_{k+1} &= \mathbf{P}_k - \mathbf{P}_k\mathbf{H}_{k+1}^T(\mathbf{I} + \mathbf{H}_{k+1}\mathbf{P}_k\mathbf{H}_{k+1}^T)^{-1}\mathbf{H}_{k+1}\mathbf{P}_k, \end{aligned} \tag{4.19}$$

where

$$\mathbf{P}_{k+1} = (\mathbf{K}_{k+1})^{-1}, \tag{4.20}$$

$$(\mathbf{K}_{k+1})^{-1} = (\mathbf{K}_k + \mathbf{H}_{k+1}^T\mathbf{H}_{k+1})^{-1}, \tag{4.21}$$

and

$$\mathbf{K}_0 = \mathbf{H}_0^T\mathbf{H}_0. \tag{4.22}$$

OS-ELM updates $(\mathbf{K}_{k+1})^{-1}$ using the Sherman-Morrison-Woodbury formula [110].

OS-ELM was found to have better performance than other sequential regression and classification applications in [111]. For this reason, the proposed solution also implemented the on-line learning procedure by the OS-ELM method.

4.5.3 One-Class Classification with ELM

A One-class classifier based on ELM was proposed by Leng *et al.* in [111]. One-class classification is an outlier detection method which differs from normal classification. It samples data from only one target class since training data consists of no or very few samples from the other classes. One-class classification defines a distance function d_{ELM} which is applied to the test samples and target class. With the distance function, this classifier accepts or rejects new observations having a higher distance from the target than some threshold parameter θ. In [111], the One-Class ELM (OC-ELM) algorithm was developed from the assumption that similar objects are close in the feature space. Thus, if similar objects are from a single class, then the estimated $\hat{\mathbf{y}}_i$ and target outputs $\mathbf{y}_i$ should be the same. Then, Equation 4.9 can be combined with the distance function and defined as

$$d_{\text{ELM}}(\mathbf{x}_i) = \|\mathbf{H}(\mathbf{x}_i)\beta - \mathbf{y}_i\|, \tag{4.23}$$

where $\mathbf{H}(\mathbf{x}_i)$ represents the output of the hidden layer for the sample $\mathbf{x}_i$.

In order to achieve the best classification rate, the optimum threshold distance θ to the separating hyperplane must be determined carefully. Extreme cases such as too large or too small θ should be avoided for less generalization error. In [111], a proper method was proposed to select an optimal θ. After calculating the distances of N training samples, they are sorted in terms of magnitude such that $d_{\text{ELM}}(\mathbf{x}_1) \leq \cdots \leq d_{\text{ELM}}(\mathbf{x}_N)$. The function determining this threshold value θ can be chosen as a distance higher than the least deviant sample $d_{\text{ELM}}(\mathbf{x}_1)$ and lower than the most deviant sample $d_{\text{ELM}}(\mathbf{x}_N)$. Then, the decision function for OC-ELM can be defined as

$$\begin{aligned} C_{\text{ELM}} &= \text{sign}(\theta - d_{\text{ELM}}(\mathbf{z})) \\ &= \begin{cases} +1 & \mathbf{z} \text{ belongs to the class} \\ -1 & \mathbf{z} \text{ is an outlier,} \end{cases} \end{aligned} \tag{4.24}$$

for a query point $\mathbf{z}$. A more detailed exploration of the OC-ELM procedure can be found in the original paper [111].

OC-ELM can also be combined with update equations and approaches of on-line sequential ELM in order to achieve a low cost strategy. In intrusion detection systems, OC-ELM can be applied to the network data delivered in real-time in order to construct a normal profile and outliers can be considered as attacks in a single moment or in a time window.

Figure 4.6 shows a comparison between one-class OS-ELM and one class support vector machines (SVM) classifier. One-class SVM [112] is an unsupervised outlier detection method which maps input data to a high-dimensional feature space in order to create a non-linear decision boundary for the target class. One-class SVM has gained considerable attention in machine learning over the last decade due to its

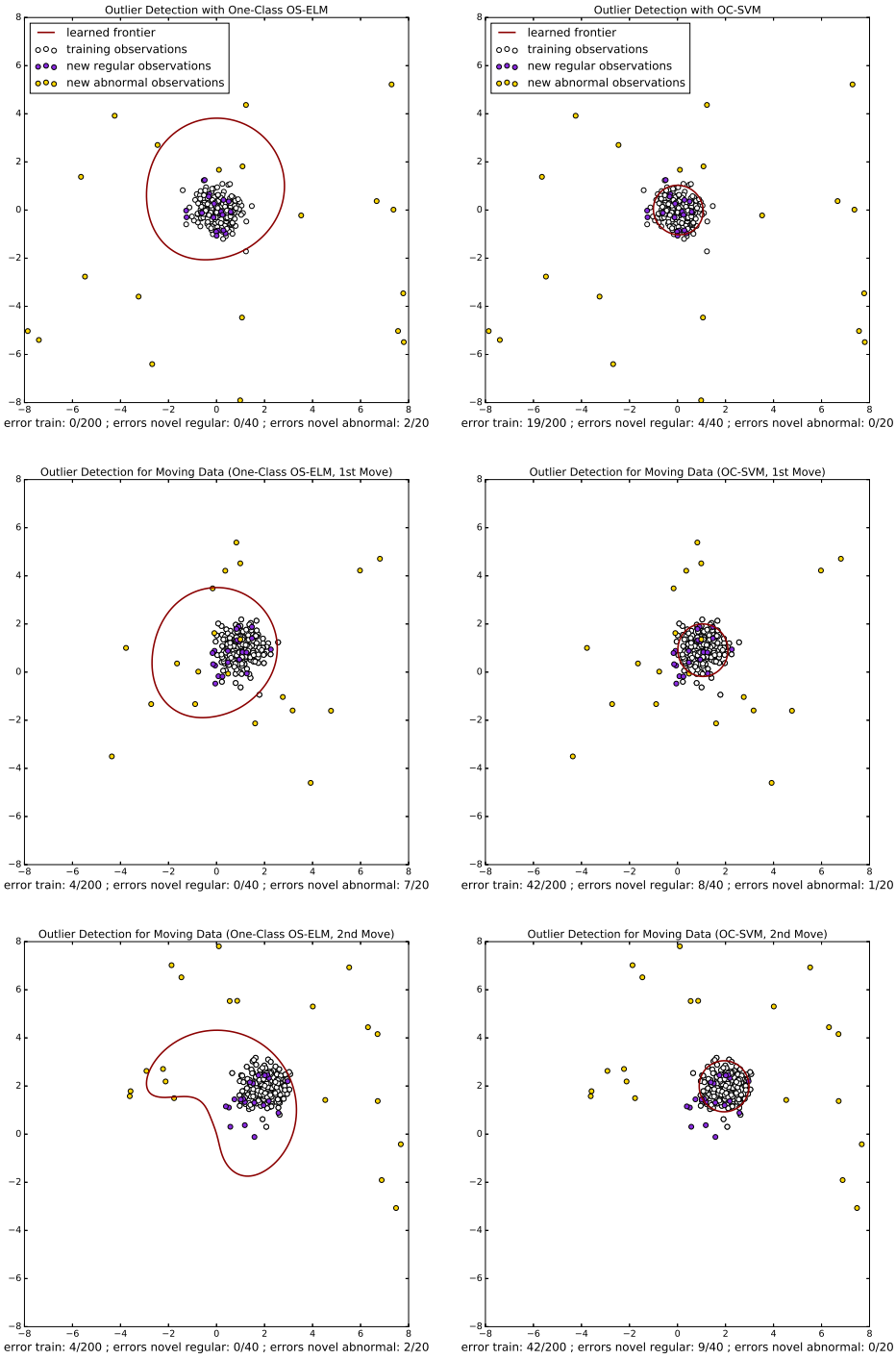


Figure 4.6: Comparison of OC-SVM and One-Class OS-ELM on 2D Toy Data Moving in (x,y) Plane

high accuracy. Thus, the performance of one-class OS-ELM can be interpreted by comparing it to one-class SVM. In the comparison experiments, both one-class SVM and OS-ELM were applied to a synthetically generated toy data. The observations of this data are Gaussian distributed, and a few random outliers are added to the input space. As seen in Figure 4.6, the center of this data slowly moves in the $(+x, +y)$ direction in each step.

In Figure 4.6, both one-class OS-ELM and OC-SVM try to find a description boundary, which is illustrated by the red line, for the training data and label new observations as outliers if they are not inside the red line. First two figures plot the results for the initial training and testing samples as well as draws the original description boundary. Although one-class OS-ELM has a bigger boundary containing the training samples than the OC-SVM has, it successfully detects outliers. It should be noted that we might obtain a better fit for the boundary with different weight initializations. After the initial training, observations move in the $(+x, +y)$ direction. The second and third plots in Figure 4.6 show that the center of the training and test samples moves from $(0, 0)$ to $(1, 1)$ by preserving the underlying distribution. In this case, one-class OS-ELM and OC-SVM update their boundary considering the shift in the data and predict new observations by checking the new boundary conditions. As seen in these two plots in the figure, one-class OS-ELM achieves better performance with less error than OC-SVM does in this step, since one-class OS-ELM updates the definition boundary by checking the data from the previous step and detects new regular observations, which are similar to the old training observations, correctly. The same result can be seen in the last two plots in the same figure. In the last plots, the center of the training and testing observations are shifted to $(2, 2)$, and one-class OS-ELM has less classification error due to its learning and storing capability of sequential training observations. Therefore, one-class OS-ELM was also implemented in one variant of the proposed method.

4.5.4 Design Choices for ELM

The performance of neural networks heavily depends on the dataset as well as the neural network configuration. Therefore, preliminary experiments were done with ELM in order to determine the network performance. For these experiments, mean squared error of training and validation data were calculated for different number of neurons. Figure 4.7 gives the mean squared error of training and validation data for a specific day by using the ELM+PT 2 approach, which will be thoroughly explained in the following section. Although results give a steady error rate after 200 neurons in this figure, 200 neurons were used in hidden layer of all ELM models to save memory and reduce the time complexity. In addition to the time complexity, different activation functions $\phi : \mathbb{R} \to \mathbb{R}$ were tested for this scenario to find the best generalization performance. The most common five activation functions were implemented for 200 neurons and the mean squared error for training data was computed. As can be seen in Table 4.1, the *sigmoid* activation function gives the minimum error. Thus, activation functions of all neurons were selected as *sigmoid*

for a fair comparison.

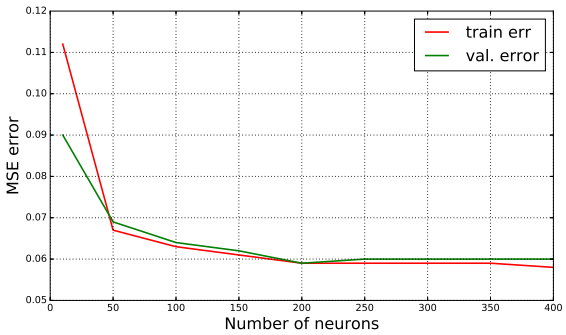


Figure 4.7: Mean Squared Error for Training and Validation Data with Different Number of Neurons

Activation func.	MSE
linear	0.068
sigmoid	0.060
hyperbolic tangent, tanh	0.061
radial basis function with Manhattan distance	0.218
radial basis function with Euclidean distance	0.069

Table 4.1: Mean Squared Error of Training Data with Different Activation Functions

4.6 Implementation of Different Models

This section presents three cases of the proposed IDS applied to the ISCX-IDS 2012 dataset:

1. Case 1 (OS-ELM + PT): It implements on-line learning mechanism by treating the dataset as a data stream as well as learns and retrains a normal profile.

2. Case 2 (ELM+PT 1): It applies batch-learning by accessing the complete dataset.

3. Case 3 (ELM+PT 2): It is another off-line learning mechanism and quite similar to ELM+PT 1. The difference between ELM+PT 1 and ELM+PT 2 is the number of tries constructed from normal, unknown and attack classes.

In all cases, the input data was the flow records extracted from the pcap files with the Argus tool. Each day was trained and tested independently. Additionally, 6 features were eliminated from the learning algorithm, since they have zero variance. This preprocessing produced flow data with 61 different features for evaluating the proposed model.

The proposed algorithm was implemented using the same software tools and on the same computer explained in section 3.4. The performance of these cases will be evaluated in Chapter 5 using the criteria defined in section 2.7.

4.6.1 Case 1: OS-ELM+PT

In this case, the learning model is updated without the need of complete data. Therefore, the basic updating capability of OS-ELM was used to train the model continuously. In addition to the machine learning part, a 8-bit IPv4 prefix tree (IP trie) structure grows dynamically with the new flow arrival in a sliding time window.

Although one-class classification is an unsupervised learning method and built on the target data, this case still checks data labels since there might be unknown flows as stated in Chapter 3. When the dataset was investigated further, it was found that each day starts with a combination of normal and unknown flows. This behavior is useful and supports the proposed initial training part of the one-class classification with OS-ELM.

In the initial training phase, the IP prefix tree was built over a period of time $T_{0:t^0} = [0; t^0]$ from normal flow records. From this structure, hierarchical heavy hitters were extracted, as presented in section 4.3. Initially, each heavy hitter $\mathbf{X}_k$ has N_k^0 flows with d flow features including TTL, average packet size, duration of flow, port numbers, protocols, total number of routers that a flow passes through and other numerical features. If N_k^0 is large enough and heavy hitters have a sufficient number of flow statistics, the distribution of values for $\mathbf{X}_k$ can be constructed over the time period $T_{0:N_k^0}$ as $f_{\mathbf{X}_k}$. As explained in section 4.4, the discrete probability density function (pdf) was produced with the aid of a histogram approach. In these probability density functions, the common bin width Δ was calculated specifically for each feature by a fixed number of bins decided as 100. A lookup table was constructed using these values and stored in each cluster. With this approach, flow statistics can be discarded to decrease the memory usage as stated in section 4.4. After converting flow features of training samples into a matrix for estimated probability values $\tilde{\mathbf{X}}_k$ using a lookup table, the one-class OS-ELM was initialized in each normal heavy hitter with 200 neurons in the hidden layer and sigmoid activation functions in every neuron. Therefore, the output matrix of the hidden layer for the first training part in each heavy hitter can be formulated as

$$\mathbf{H}^0 = \begin{bmatrix} \phi(\mathbf{w}_1^T \tilde{\mathbf{x}}_1 + b_1) & \cdots & \phi(\mathbf{w}_{200}^T \tilde{\mathbf{x}}_1 + b_{200}) \\ \vdots & \cdots & \vdots \\ \phi(\mathbf{w}_1^T \tilde{\mathbf{x}}_{N_k^0} + b_1) & \cdots & \phi(\mathbf{w}_{200}^T \tilde{\mathbf{x}}_{N_k^0} + b_{200}) \end{bmatrix}_{N_k^0 \times 200}, \tag{4.25}$$

where

$$\tilde{\mathbf{X}}_k = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_{N_k^0} \end{bmatrix}^T. \tag{4.26}$$

Initial output weights $\boldsymbol{\beta}^0$ were calculated with Equation 4.16. The threshold value θ was defined as $0.9 \times d_{max}$ where d_{max} represents the maximum distance of a specific sample from the normal data, as explained in section 4.5. Since the input is high dimensional, the idea of the hypersphere in [113] was used to obtain the minimum separating dimension. In this case, OS-ELM tries to find the smallest sphere which contains the normal data samples and separates these from the outliers. OS-ELM decides an optimum ball radius R which is equal to the threshold value θ defined in section 4.5.

In the testing phase, the same statistics for the flow records were collected over a certain time period $T_{t^i,t^j} = [t^i; t^j]$. Each flow was matched to the closest cluster. Matching was performed by comparing the IP value of a test sample with the range of each HHHs. As described in Algorithm 1, HHHs are collected into a structure with a recursive postorder traversal method. This tree traversal method ensures finding the most similar heavy hitter behavior for each cluster. Secondly, the pdf of each feature in matched clusters was estimated using the previously obtained lookup table. Thirdly, these probabilities were given as an input to the one class OS-ELM of matched cluster. Flows with a distance smaller than the threshold θ were labeled as normal flows. Otherwise, they were considered as attacks and discarded from the updating phase.

In the update mechanism, the IP trie grows with the new data and the one class ELM is retrained. In this case, normal predicted flows from the previous time window were added to the IP trie. Secondly, hierarchical heavy hitters were constructed from this growing trie and the learning mechanism was updated with new look up tables. The second procedure was divided into two cases: adding data to existing heavy hitters and initial training of possible new clusters. If the 8-bit IPv4 prefix of a newly arrived flow presents in an HHH node, then the features of this flow are added to this heavy hitter. The initial training of new clusters is done when a node has enough traffic volume to be extracted as an HHH, as the IP trie grows with the received data stream.

Adding Data to Existing Heavy Hitters

In this case, newly obtained heavy hitters are the same as the old HHHs. A new lookup table was extracted from the statistical values which had been gathered over a time period $T_{0,t^i} = [0; t^i]$ where the test phase before this procedure was from the time window $T_{t^{i-1},t^i} = [t^{i-1}; t^i]$. The previous lookup table was replaced with the new lookup table, since it provides a more generalized and updated version of the gathered flow features. By using the new lookup table, the probabilities were estimated for the recently arrived N_k^i flows in the i'th sliding window for a heavy hitter $\mathbf{X}_k$. Consequently, the output matrix of the hidden layer for the i'th time

window in each heavy hitter can be written as

$$\mathbf{H}^i = \begin{bmatrix} \phi(\mathbf{w}_1^T \tilde{\mathbf{x}}_1 + b_1) & \cdots & \phi(\mathbf{w}_{200}^T \tilde{\mathbf{x}}_1 + b_{200}) \\ \vdots & \cdots & \vdots \\ \phi(\mathbf{w}_1^T \tilde{\mathbf{x}}_{N_k^i} + b_1) & \cdots & \phi(\mathbf{w}_{200}^T \tilde{\mathbf{x}}_{N_k^i} + b_{200}) \end{bmatrix}_{N_k^i \times 200}, \tag{4.27}$$

where

$$\tilde{\mathbf{X}}_k = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_{N_k^i} \end{bmatrix}^T. \tag{4.28}$$

Since OS-ELM allows to add new data to the model for retraining, the updated output weights $\boldsymbol{\beta}^i$ can be computed using Equation 4.19. This continuous learning was performed in every time window over data matrix $\mathbf{X}_k$ of each heavy hitter unless the volume of the cluster has a complete network data proportion smaller than φ. Otherwise, $\mathbf{X}_k$ was dropped from the HHH list, as it is no longer a frequent event in the network data.

Adding a New Heavy Hitter to the Model

Adding a new heavy hitter to the model is a straightforward process. If the recently obtained heavy hitter list contains a new cluster which had not been seen before, the probabilistic representation of features and the initial training of the one class OS-ELM was conducted for that heavy hitter by using the exact procedure in the initial training. Hence, the model can continue to grow by adding new subnetworks without any disruptions.

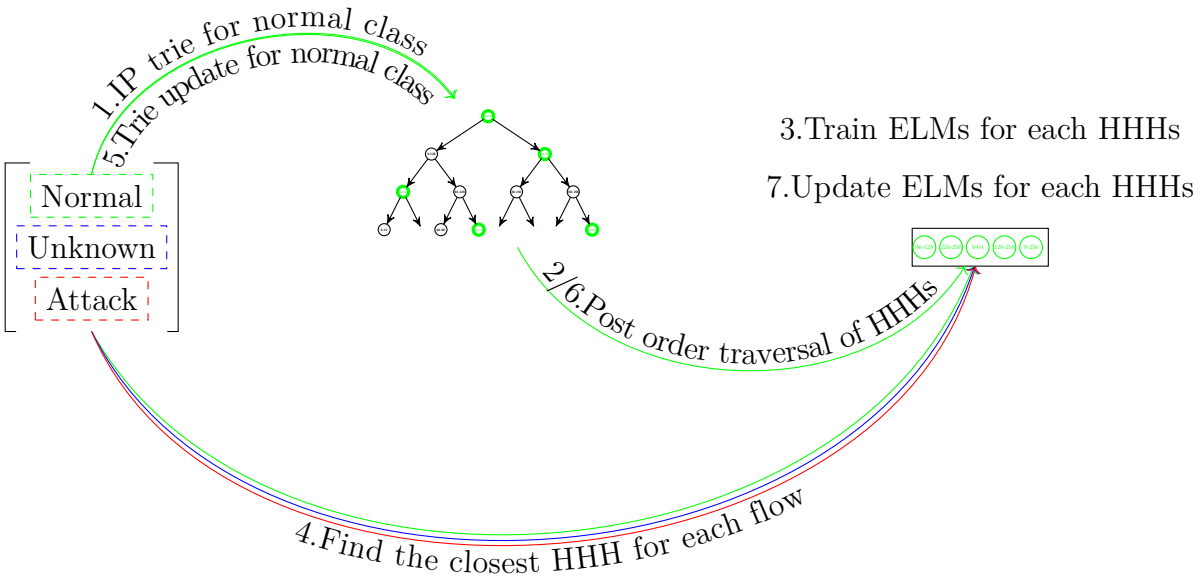


Figure 4.8: Training (Steps 1,2 and 3) and Updating (Steps 5,6 and 7) Mechanisms for OS-ELM+PT

Figure 4.8 illustrates the initial training (steps 1,2,3) and update mechanisms (steps 5,6,7) for the streaming input data shown as matrix. In this case, the main objective is to label outliers as attack. Therefore, unknown flows were either identified as normal or attack. As illustrated in Figure 4.8, training and updating mechanisms follow these steps:

1. In the initial training part, the IP trie for the normal class is obtained.
2. HHHs are identified from the IP trie with a post order traversal method.
3. For each HHH, a lookup table is constructed by calculating histograms and one-class OS-ELMs are trained with the estimated pdf values of each feature in every HHH.
4. In the testing phase, this case tries to find the closest HHH for the recently observed flows and labels each flow with respect to the output of the corresponding ELM.
5. IP trie is updated with the new flows predicted as normal data.
6. Possible new HHHs are detected from the updated IP trie with the same post order traversal method. In the same time, histograms are updated in every old HHH.
7. If a new observation is added to an existing heavy hitter, OS-ELM of this cluster is retrained. If a new HHH is added to the complete model, OS-ELM is initialized for that heavy hitter.

4.6.2 Case 2: ELM+PT 1

As stated earlier, batch learning requires a complete dataset. For this reason, each day in the dataset was randomly split into training and testing samples by using 3-fold cross-validation. This batch learning model was trained with 1/3 of the complete dataset for each day and the remaining part of the data was used to validate the resulting model. In the learning part of this method, ELM was applied with 200 neurons in the hidden layer and the sigmoid activation functions in every neuron.

In the training phase, IP trie and HHHs were obtained by using only normal flows randomly sampled from the training data. A lookup table was constructed with the same method explained in the first training part of Case 1. In contrast to the previous case, ELMs were trained by using complete training set. For each flow, the closest cluster was found by comparing the IPv4 value of the destination address with the IP range of the HHH list. After obtaining the matrix $\tilde{\mathbf{X}}$ for the estimated pdf values of the training data, ELMs of the k-th heavy hitter were trained with this matrix. $\tilde{\mathbf{X}}_k$ training matrix for the k-th HHH contains samples from the normal

$\tilde{\mathbf{X}}_{kn}$, unknown $\tilde{\mathbf{X}}_{ku}$ and attack $\tilde{\mathbf{X}}_{ka}$ classes. In ELM+PT 1, the training data for cluster k is represented as Equation 4.29.

$$\tilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{X}}_{kn} \\ \tilde{\mathbf{X}}_{ku} \\ \tilde{\mathbf{X}}_{ka} \end{bmatrix}. \tag{4.29}$$

The probability values of each feature for all classes were predicted as

$$\begin{aligned} \tilde{\mathbf{X}}_{kn} &= \begin{bmatrix} f^1_{\mathbf{X}^1_{kn}}(\mathbf{X}^1_{kn}) & \cdots & f^d_{\mathbf{X}^d_{kn}}(\mathbf{X}^d_{kn}) \end{bmatrix} \\ \tilde{\mathbf{X}}_{ku} &= \begin{bmatrix} f^1_{\mathbf{X}^1_{kn}}(\mathbf{X}^1_{ku}) & \cdots & f^d_{\mathbf{X}^d_{kn}}(\mathbf{X}^d_{ku}) \end{bmatrix} \\ \tilde{\mathbf{X}}_{ka} &= \begin{bmatrix} f^1_{\mathbf{X}^1_{kn}}(\mathbf{X}^1_{ka}) & \cdots & f^d_{\mathbf{X}^d_{kn}}(\mathbf{X}^d_{ka}) \end{bmatrix} \end{aligned} \tag{4.30}$$

where the k-th normal HHH has a lookup table $f^i_{(\mathbf{X}^i_{kn})}$ for its i-th feature.

The corresponding output of the training samples for the k-th HHH was converted into binarized vectors

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^T_{kn} \\ \mathbf{y}^T_{ku} \\ \mathbf{y}^T_{ka} \end{bmatrix}, \tag{4.31}$$

The correct output was labeled as

$$\begin{aligned} \mathbf{y}^T_{n_i} &= [1, 0, 0] \\ \mathbf{y}^T_{u_i} &= [0, 1, 0]. \\ \mathbf{y}^T_{a_i} &= [0, 0, 1] \end{aligned} \tag{4.32}$$

where $\mathbf{y}^T_{n_i}$ is the normal class sample, $\mathbf{y}^T_{u_i}$ is the unknown class sample and $\mathbf{y}^T_{a_i}$ represents attack class sample in the training dataset.

There might be cases when the training samples of an HHH does not contain any attack or unknown labeled flows. Although this highly indicates that this HHH node probably represents only normal traffic of the complete dataset, the training phase was performed with multi-class classification, as written in Equation 4.31. It should be assumed that the 8-bit IPv4 prefix value of an unknown or attack labeled test sample can match to this HHH and the ELM algorithm should identify this sample correctly. For example, if the learning input of k-th normal cluster consists of N perfectly normal data samples, Equation 4.29 and 4.31 are converted into

$$\begin{aligned} \tilde{\mathbf{X}}_k &= \begin{bmatrix} \tilde{\mathbf{X}}_{kn} \end{bmatrix} \\ &= \begin{bmatrix} \tilde{\mathbf{x}}_{kn_1} & \cdots & \tilde{\mathbf{x}}_{kn_N} \end{bmatrix}^T, \end{aligned} \tag{4.33}$$

and

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^T_{kn_1} \\ \vdots \\ \mathbf{y}^T_{kn_N} \end{bmatrix}^T, \tag{4.34}$$

where

$$\mathbf{y}_{n_i}^T = [1, 0, 0]. \tag{4.35}$$

Figure 4.9 demonstrates the training procedure for batch input data. The training phase is as follows:

1. 8-bit IPv4 prefix tree is obtained from only normal class samples of the training dataset.
2. HHHs are retrieved from this IPv4 trie with a post order traversal method.
3. After extracting the HHHs from the normal data samples, all available training data is matched to its closest HHH and a lookup table for the pdf estimation of every feature is constructed for each HHH.
4. ELMs of each HHH are trained with the estimated pdf values.

Prediction for the remaining dataset is performed in a similar way to the training part. Probabilities are estimated for each feature of every flow in the remaining data by finding the closest HHH. Estimated probabilities are given as an input to the corresponding ELMs in order to predict the labels. These labels are also obtained in a binarized vector shape as in Equation 4.32. Finally, the estimated vectors were categorized as normal, attack or unknown flows for evaluation.

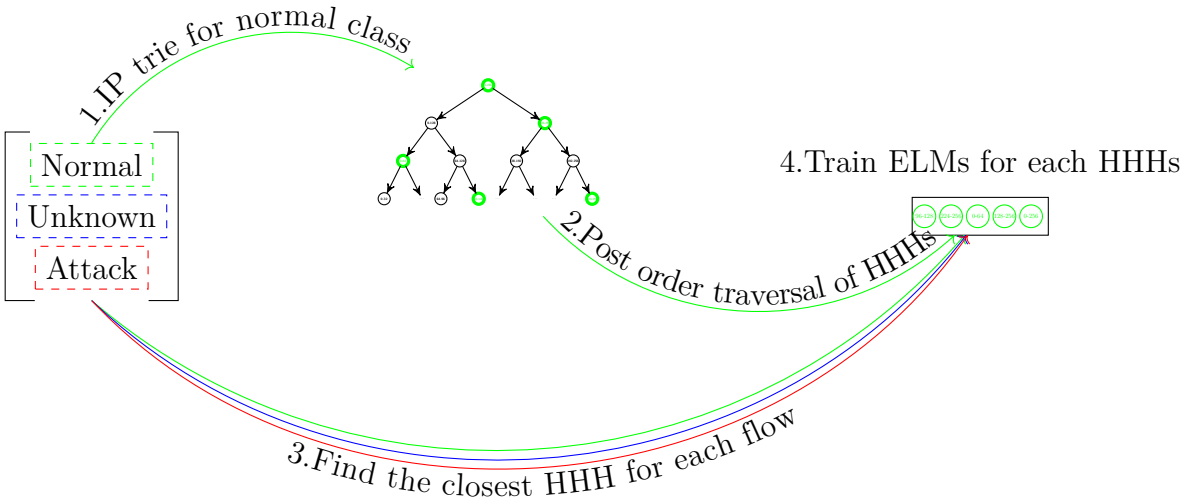


Figure 4.9: Training Mechanism of ELM+PT 2

4.6.3 Case 3: ELM+PT 2

This strategy is the extended version of the ELM+PT 1 case in order to find out the possible effects of modeling different trees for each class. This case is also a

batch learning algorithm and each day of the weekly dataset was randomly split into training and test dataset using a 3-fold cross-validation approach.

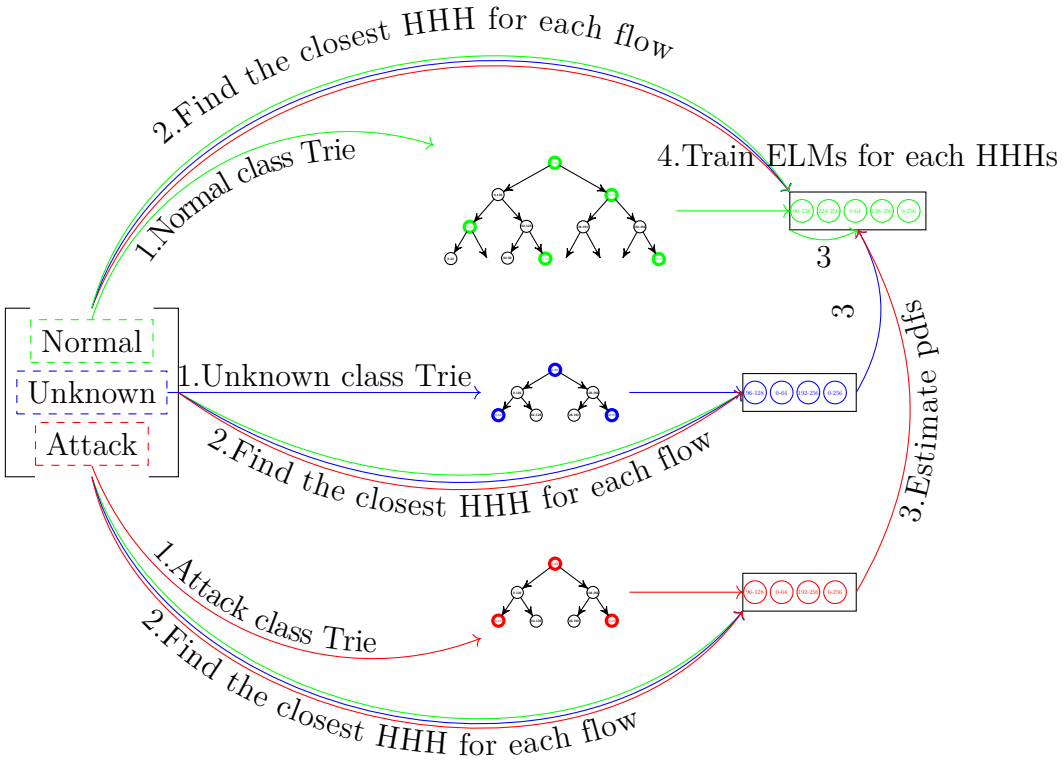


Figure 4.10: Training Mechanism of ELM+PT 2

The training phase is illustrated in Figure 4.10. This version differs from previous cases in terms of the number of IPv4 tries. The proposed training part requires these steps:

1. 8-bit destination IPv4 address prefix trees (IP tries) are obtained separately for normal, unknown and attacks flows in the training data. After this step, different hierarchical heavy hitters are extracted and collected as lists with the post order traversal method. At the end of this step, there are three distinct HHH lists for different labels, and for each HHH, a lookup table is constructed by calculating and normalizing histograms.

2. For each sample flow in the training network, the closest normal, unknown and attack HHHs are found.

3. Since, there are three different lookup tables based on class tries, probability values of each flow are estimated from the closest HHHs' lookup table. For instance, supposing that the i-th flow vector $\mathbf{x}_i$ with a label y_i is the closest

training sample to the n-th normal, u-th unknown and a-th attack HHHs, then the estimated pdfs of this flow will be

$$\begin{aligned}
\tilde{\mathbf{x}}_{in} &= \left[f^1_{\mathbf{X}^1_n}(x^1_i) \quad \cdots \quad f^d_{\mathbf{X}^d_n}(x^d_i) \right]^T, \\
\tilde{\mathbf{x}}_{iu} &= \left[f^1_{\mathbf{X}^1_u}(x^1_i) \quad \cdots \quad f^d_{\mathbf{X}^d_u}(x^d_i) \right]^T, \\
\tilde{\mathbf{x}}_{ia} &= \left[f^1_{\mathbf{X}^1_a}(x^1_i) \quad \cdots \quad f^d_{\mathbf{X}^d_a}(x^d_i) \right]^T,
\end{aligned} \tag{4.36}$$

where $f^j_{\mathbf{X}^j_n}$ is the lookup table of the n-th normal HHH, $f^j_{\mathbf{X}^j_u}$ is the lookup table of the u-th unknown HHH and $f^j_{\mathbf{X}^j_a}$ is the lookup table of the a-th attack HHH for the j-th feature. In addition, the output label y_i was binarized and transformed into a vector $\mathbf{y}_i$, as in Equation 4.32.

4. Estimated probabilities of each sample flow are given as a learning input to train ELMs of the closest normal HHH. Unlike the previous cases, dimensions of the training input is $3 \times N \times d$ in this case. Under certain circumstances such as purely normal destination IPv4 HHHs, the training phase is performed with only estimations from the available closest heavy hitters. As a result, the input matrix for the learning part might reduce to $2 \times N \times d$ or even to $N \times d$ in purely normal destination IPv4s.

In the testing part, steps 4 and 5 are performed directly and the probability estimations are given as input to the ELM in order to make predictions.

4.7 Summary

This chapter has provided a detailed analysis of the proposed intrusion detection system as well as the mathematical methods used in the design. The proposed IDS first builds a hierarchical binary tree of the destination IPv4 addresses and obtains the most frequent traffic patterns with hierarchical heavy hitters algorithm as an initial learning model. Secondly, it converts features of each heavy hitter into a probability space by using histogram normalization. In the last part of the training, the estimated probability values are trained with a supervised learning method. Since the fundamental learning method of the proposed IDS is ELM, this chapter has reviewed the original ELM algorithm by providing design choices for a better performance. Two of ELM variants, OS-ELM and OC-ELM have also been presented. In addition to the mathematical review, this chapter has introduced different sequential and batch learning cases with three novel approaches. In the first case, which implements the on-line learning method, the model is continuously updated with OS-ELM binary classification to test the capability of sequential learning. In other cases, the proposed methodology uses batch ELM to build and predict different classes by using estimated probability densities of network flow. These batch cases were constructed to evaluate the efficiency of learning from the aggregated probability space.

The following chapter will evaluate the performance of the proposed intrusion detection system in the three different cases explained in this chapter.

Chapter 5

Results and Evaluation

This chapter provides an evaluation of the proposed solution presented in Chapter 4. Section 5.1 explains the experimental setup in detail. In section 4.6, the performance of three different cases (OS-ELM+PT, ELM+PT 1 and ELM+PT 2) is discussed in terms of the criteria defined in Chapter 2. Finally, the two most efficient cases are compared to the simple machine learning-based anomaly detection methods and other state-of-the-art intrusion detection systems presented in Chapter 3.

5.1 Experimental Setup

In this thesis, measurements were performed using the ISCX-IDS 2012 dataset [39]. As introduced in Chapter 3, this dataset contains different normal traffic profiles as well as attack scenarios. These intrusions include HTTP DoS with Slowloris, DDoS using botnets, brute force SSH as well as infiltrating the network from inside, such as SQL injection and Cross Site Scripting.

Three different cases from section 4.6 were developed based on the proposed IDS architecture:

1. Case 1 (OS-ELM+PT): This case uses one class OS-ELM and takes streaming data as input in order to imitate existing real-life IDSs. Semi-supervised learning by using only normal behavior was implemented in this sequential model. Predicted outliers were removed and normal predictions were used in the update mechanism.

2. Case 2 (ELM+PT 1): This batch-learning case was trained with the sampled normal, attack and unknown behavior; however the training part was implemented using only normal flows. The rest of the dataset was used for testing purposes. ELM was used as a supervised learning algorithm.

3. Case 3 (ELM+PT 2): Similar to the previous case, ELM+PT 2 also conducted a batch learning method and used supervised ELM. The difference is that, this case constructs all possible tries (Normal, unknown and attack tries) and trains ELMs of each HHHs.

For batch learning models, a k-fold cross validation approach was used, where k was fixed as 3 in both proposed models and other machine learning methods explained in Chapter 3. In the experiments, each day was tested separately to compute the detection rate for different attack scenarios.

Experiments were conducted on a 64bit Ubuntu 16.04 PC with 8GB of RAM and CPU of 2.70GHz. In addition, experiments and all resulting figures were performed using Python 2.7.11, numpy 1.11.0, scipy 0.17.1 and Argus 3.0.8.1. The implementation of the proposed algorithm takes the flow records file and case choice as an input. Rest of the parameters including k-fold, threshold number for extracting heavy hitters, histograms bins and other metrics were tuned automatically.

In order to assess the performance of the proposed IDS, predicted labels were compared to expected classes. The experimental results and evaluations for these cases are provided in the following subsections.

5.2 Evaluation of Different Models

This section provides experimental results and evaluations of the proposed cases explained in section 4.6. The performance of these procedures was assessed by observing the confusion matrix, time complexity and other criteria explained in section 2.7. In addition to these results, the proposed cases were also compared with the traditional machine learning methods applied to the ISCX-IDS 2012 dataset in section 3.4. The experimental results of OS-ELM+PT are presented separately since it applies a sequential learning method.

5.2.1 Case 1: OS-ELM+PT

This procedure trains and updates the model with sequentially observed data. As discussed in section 4.6, the initial training was performed by aggregating flows observed in the first hour. In the ISCX-IDS 2012 dataset, the beginning of each day was a mixture of both normal and unknown flows. Therefore, the model can learn a normal profile for each heavy hitter and obtain a proper distance measure for one class classification. Since there is only one trie constructed from normal flows and one class classification was implemented to define the boundary for the normal data, the unknown flows can be predicted as normal if they are inside this boundary. Furthermore, outliers were directly predicted as attack. Therefore, the number of predicted classes was reduced to two: normal and attack class.

After the initial training, the retraining of the model is done in every one hour. In the testing part, the proposed model predicted the label of each flow by calculating the distance to the separating hyperplane as explained in section 4.6. Outliers were labeled as attack and discarded from the update procedure. In the update part, estimated pdf statistics of each normal predicted flow were given as a sequential

input to the ELM method.

After applying OS-ELM+PT to each day, confusion matrices were computed for the testing and updating part of the day. Confusion matrices for June 13 and June 15 are given in Table 5.1. In these results, unknown flows were predicted as either normal or attack, as expected. However, the results conflict with the theoretical model in terms of updating the learning part and differentiating the normal and attack classes in the sequential data. For example, the one-class OS-ELM model fails to find an optimum boundary of the separating hypersphere for normal and attack classes, as theoretically explained in section 4.5. For further analysis, experiments were repeated with different thresholds causing a change in the number of heavy hitters in order to examine the possible effect of this change. As illustrated in Figure 5.1, the detection rate of attack flows is very low for various threshold values. In order to understand the reason behind this discrepancy, the properties of each day in the dataset were thoroughly investigated. One possible explanation for this conflict might be the estimated pdf values of different features. Since all numerical statistics were considered in the learning part, some of them might negatively affect the construction of the classification boundary around the normal flows.

		Predicted		
		Normal	*Unknown*	*Attack*
Actual	*Normal*	205923	0	3451
	Unknown	18712	0	23707
	Attack	2855	0	9599

(a)

		Predicted		
		Normal	*Unknown*	*Attack*
Actual	*Normal*	990314	0	328
	Unknown	55695	0	7146
	Attack	37156	0	236

(b)

Table 5.1: Confusion Matrices for Case 1 with June 13 (a) and June 15 (b)

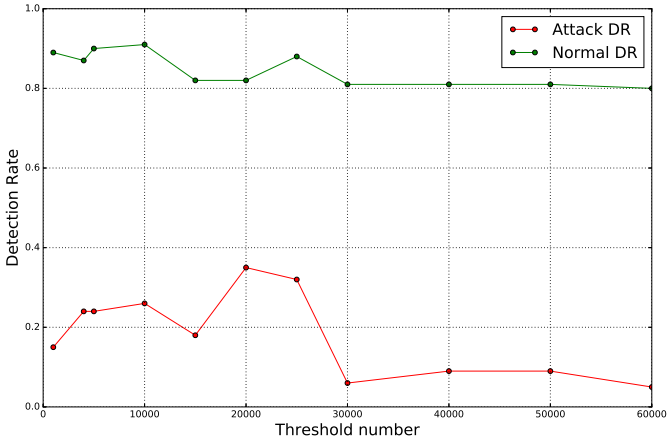
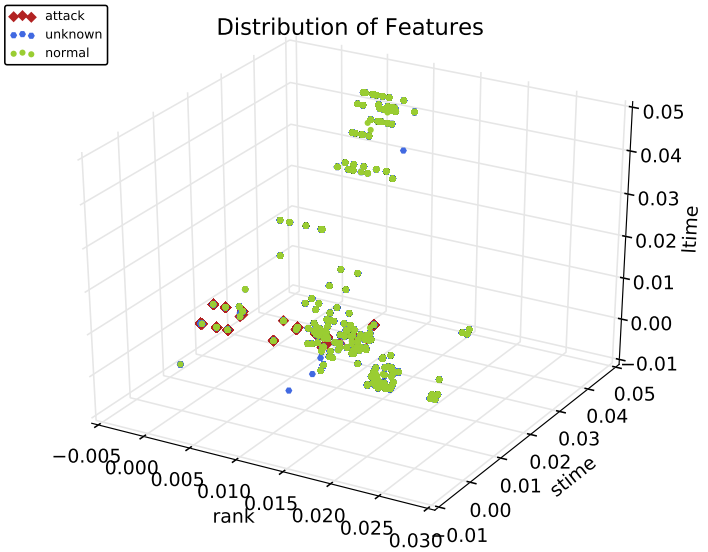
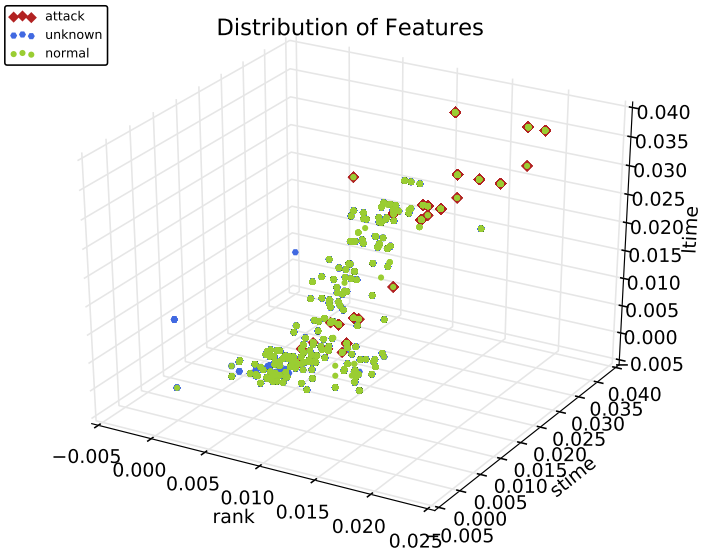


Figure 5.1: Detection Rate for June 13 with Different Threshold Values



(a)



(b)

Figure 5.2: Estimated PDFs of 3 Different Statistics for Heavy Hitter 192.x.x.x. in June 13 (a) and June 15 (b)

Figure 5.2 provides the estimated probability values of three different statistics for the only heavy hitter containing IPv4 addresses 192.x.x.x. These statistics were estimations from one complete day and saved for further analysis. As shown in

this figure, the trend in the data coincides with the possible explanation of the fail. Figure 5.2a shows that the attack data is almost in the center of the hypersphere containing normal flows. Furthermore, normal flows are spread in the *ltime* direction which leads to predicting many normal data samples as outliers. As can be observed in Figure 5.2b, a large number of attack and unknown flows are inside the normal hyperspace. Hence, most of the attack flows might be predicted as normal by checking these three features. If these 3 features only had been used for the proposed methodology, the results would have been much worse than that of Table 5.1.

Since all possible numerical features were used for the learning, some of them decreased the overall performance of the system. In order to avoid this degradation, feature selection algorithms can be deployed as a preprocessing method. However, extracting relevant features and excluding redundant feature subsets are demanding tasks, especially for this dataset. For example, it is hard to estimate the dependencies between variables for feature selection [114]. Moreover, this dataset contains unknown type flows which might pose highly non-linear dependencies among features. Also, streaming data can change the relevance of features and the previously selected features may not be suitable for the data anymore. Therefore, application of feature selection algorithms and elimination of irrelevant features are left as a future work.

The specific one class formulation used in the first case (OS-ELM+PT) can be extended to multi class classification. As stated in [115], multi class classification might be a more efficient implementation as it can handle the separation of different classes. Therefore, case 2 and 3 should perform better than this approach since they are able to learn characteristics of attack, normal and unknown classes.

5.2.2 Case 2 (ELM+PT 1) and Case 3 (ELM+PT 2)

After observing the poor performance of the first case, batch learning of ELM using the hierarchical heavy hitter aggregation concept was implemented. Therefore, the main focus of these cases is to assess the capability of the proposed model when trained with attack samples beforehand. In addition, tries were constructed for each class sample in order to examine the effect of the known classes. Similar to the OS-ELM+PT approach, these cases are applied to each day in the data set separately. In both of these cases, 1/3 of the flow records are randomly sampled for the training and the rest of it used as testing dataset. It should be noted that each training and test set contains approximately the same percentage of normal, attack and unknown classes in order to ensure that enough attack flows are also used in the training part. Although this homogeneous proportion method differs from real life data in terms of continuous network traffic property, this approach allows Case 2 and 3 to be evaluated in a similar way as the other state-of-the-art intrusion detection systems, since the approaches in [88]-[93] also split the data into training and testing with similar proportions.

Dataset		Basic ELM			ELM+PT 1			ELM+PT 2		
Actual class		Predicted class			Predicted class			Predicted class		
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 12	*Normal*	**222506**	1633	34	**219364**	4778	31	**220550**	3617	6
	Unkn.	16260	**18683**	4	10003	**25321**	3	9401	**25546**	0
	Attack	6	28	**3189**	1	3	**3206**	1	0	**3220**
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 13	*Normal*	**220024**	1012	112	**218468**	2594	86	**218261**	2786	101
	Unkn.	14842	**26009**	4328	8302	**34146**	2731	7105	**35349**	2725
	Attack	2423	597	**9833**	299	1801	**10753**	293	1829	**10731**
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 14	*Normal*	**317571**	667	128	**315977**	2263	126	**314485**	3837	44
	Unkn.	23736	**22161**	306	16161	**27667**	2375	10486	**33307**	2410
	Attack	5491	2757	**424**	491	2890	**5291**	63	3278	**5331**
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 15	*Normal*	**1026753**	621	6598	**1032760**	351	861	**1029481**	3903	588
	Unkn.	30502	**35775**	55	29477	**36802**	53	24096	**42193**	43
	Attack	3469	0	**33933**	409	0	**36993**	522	0	**36880**
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 16	*Normal*	**587867**	75775	-	**593402**	70240	-	**592039**	71603	-
	Unkn.	120504	**227410**	-	55675	**292239**	-	56037	**291877**	-
	Attack	-	-	-	-	-	-	-	-	-
		Normal	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*	*Normal*	*Unkn.*	*Attack*
June 17	*Normal*	**611587**	4	498	**612039**	40	10	**612043**	43	3
	Unkn.	6316	**13639**	18	6251	**13720**	2	6226	**13747**	0
	Attack	49	2	**3243**	10	0	**3284**	0	1	**3293**

Table 5.2: ISCX-IDS 2012 Results: Basic ELM, ELM+PT 1 and ELM+PT 2

Training and testing part of these cases were implemented as explained in section 4.6. Contrary to the OS-ELM+PT case, these methods do not include an update procedure. Therefore, predictions were averaged over 50 runs of experiments having 3-fold cross validation. Furthermore, confusion matrices were built for each day after obtaining predictions. Confusion matrices for each day and the comparison between the basic ELM, ELM+PT 1 and ELM+PT 2 are provided in Table 5.2. The table shows that the proposed method for ELM+PT 1 and ELM+PT 2 achieve better performance than the basic ELM. If June 14, which contains a DoS attack, is specifically compared, these methods have far less false positives than does the basic ELM. In addition, a further investigation of the mislabeled unknown data in these methods shows that those flows were observed from either heavily normal or attack traffic. Hence, these unknown flows might be falsely predicted due to the excessive traffic of a specific class. Nevertheless, misclassification rates of the normal and attack classes were quite low in contrast to the basic ELM. Thus, these results confirm that the basic ELM method can be improved by aggregating the data into heavy hitters and learning in a probability space, as proposed in this work.

After this simple analysis, average statistics were obtained for the last two cases: False positives, false alarms and detection rate for different classes as well as for various attack types were computed. In addition, the time complexity of each method was measured. These final results for these cases were compared to the performance of the best simple machine learning algorithms explained in Chapter 3. Table 5.3 shows this comparison and the best results for each criterion are written in bold

letters. As can be seen from the table, ELM+PT 1 and ELM+PT 2 achieve the best performance in most of the evaluation criteria. These two methods were found to be very effective in terms of detection rate, false positives and false alarms. This table clearly shows that these two methods detect HTTP DoS attacks much better than the other machine learning techniques, at the cost of a larger computational time.

		Basic ELM			Decision Trees		
ID	Class Name	False Pos.	False Alarm	Detec. rate	False Pos.	False Alarm	Detec. rate
1	Normal	-	0.03	0.93	-	0.04	0.96
2	Infiltration (SQL, CSS)	0.13	-	0.84	0.14	-	0.87
3	HTTP DoS	0.95	-	0.38	**0.35**	-	0.62
4	DDoS (using botnet)	0.08	-	0.84	0.02		**0.98**
5	Brute SSH	0.01	-	0.89	**0.00**	-	**1.00**
6	Unknown	-	-	**0.90**	-	-	0.74
		Average elapsed time (sec.) **5.32**			Average elapsed time (sec.) 26.12		

		Linear Regression			KNN		
ID	Class Name	False Pos.	False Alarm	Detec. rate	False Pos.	False Alarm	Detec. rate
1	Normal	-	**0.02**	0.91	-	0.04	0.96
2	Infiltration (SQL, CSS)	0.18	-	0.84	0.11	-	0.87
3	HTTP DoS	0.95	-	0.68	0.45	-	0.59
4	DDoS (using botnet)	0.06	-	0.34	0.06	-	0.91
5	Brute SSH	**0.00**	-	0.94	**0.00**	-	0.97
6	Unknown	-	-	0.84	-	-	0.83
		Average elapsed time (sec.) 22.72			Average elapsed time (sec.) 16.94		

		ELM+PT 1			ELM+PT 2		
ID	Class Name	False Pos.	False Alarm	Detec. rate	False Pos.	False Alarm	Detec. rate
1	Normal	-	**0.02**	0.96	-	**0.02**	**0.97**
2	Infiltration (SQL, CSS)	**0.08**	-	0.88	0.10	-	**0.90**
3	HTTP DoS	0.38	-	0.65	0.38	-	**0.70**
4	DDoS (using botnet)	**0.01**	-	**0.98**	0.02	-	**0.98**
5	Brute SSH	**0.00**	-	**1.00**	0.01	-	0.99
6	Unknown	-	-	**0.90**	-	-	0.89
		Average elapsed time (sec.) 21.07			Average elapsed time (sec.) 42.27		

Bold face: best value for different classes

Table 5.3: Average Results of ISCX-IDS 2012 Dataset: Basic ELM, ELM+PT 1, ELM+PT 2, Decision Trees, Linear Regression and KNN

The basic ELM consumes quite low CPU time in both training and testing since its parameters can be randomly assigned in the initialization process [59]. For this reason, it was expected that a simple ELM might run much faster than the other algorithms and the results were consistent with that expectation. It was also observed that the time complexity of the proposed methods were quite high. It appears that ELM+PT 2 has the worst time complexity as the pdf of each flow is estimated from 3 different heavy hitters and the dimension of the input data was quite big. However, the time elapsed for ELM+PT 1 was lower than both decision trees and linear regression. In addition, the overall performance of this case is quite satisfying and very close to the best statistics. This table confirms that these two cases can operate consistently better than the other simple algorithms with a trade-off between the time complexity and the performance.

A more detailed comparison of these cases with the basic ELM and decision trees

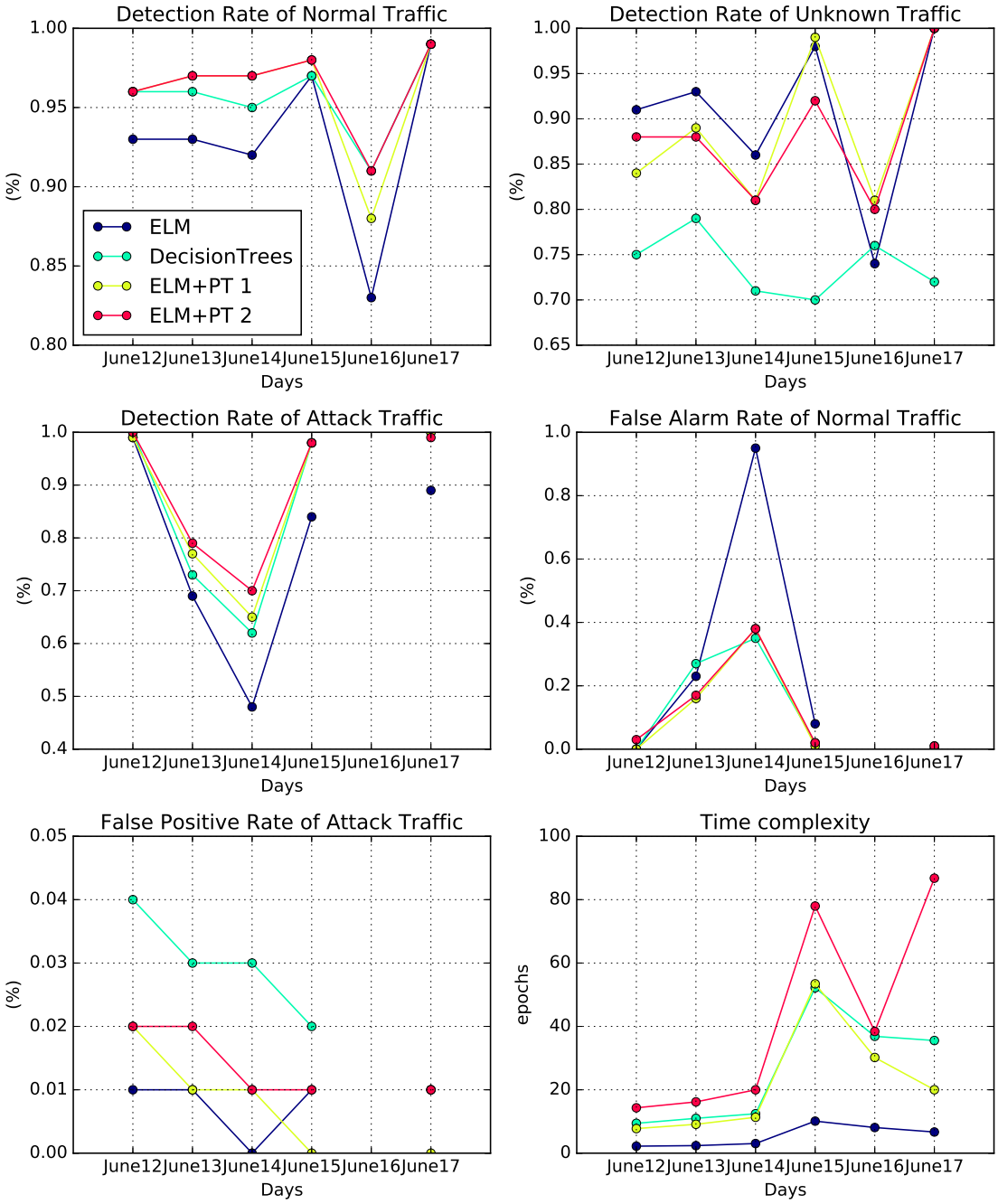


Figure 5.3: Comparison of Decision Trees, ELM, ELM+PT 1 and ELM+PT 2 for Precision, False Positive Rate, False Alarm Rate and Time Complexity

was made in order to strengthen the findings of ELM+PT 1 and ELM+PT 2. Since decision trees had provided the best results, compared to other simple machine learning classification techniques, ELM+PT 1 and ELM+PT 2 were compared with the decision trees by analyzing each day separately. The statistics for each day were plotted in Figure 5.3. As shown in this figure, the simulation results agree well with the average statistics shown in Table 5.3. ELM+PT 1 and ELM+PT 2 have generally higher precision and lower false alarm rate than the basic ELM and decision trees in each day. In addition, this figure clearly illustrates that time complexities of ELM+PT 1 and ELM+PT 2 were strongly dependent on the data size. For instance, the elapsed time of each algorithm were relatively close in the first three days. However, the difference between elapsed times are higher when the flow records contain more traffic. This outcome was expected since a heavily populated traffic tends to produce more heavy hitters. Accordingly, the pdf prediction for each heavy hitter consumes more time. Despite this trade-off, ELM+PT 1 and ELM+PT 2 generally achieve good learning performance under 20 seconds.

In the evaluation criteria, the confusion matrix, the detection and false alarm rates are chosen as main performance measures instead of other unifying measures, such as F1-score that can capture overall classifier performance. The main reason for this choice is the difference between the cost of false positives and false negatives. For instance, if an unknown flow is mislabeled in a heavily normal or attack traffic, then its cost should be lower than a misclassified attack or normal flow in the same time window. Especially on June 16, which is an attack-free day, even if all unknown flows are misclassified as normal, it should be considered as a good performance by only checking the detection rate of normal flows or the confusion matrix. However, F1-score might give worse results, since it calculates and adds the misclassification rate of unknown flows to the overall performance.

Since all cases were tested on a network traffic dataset representing only a small period of time with a few types of intrusions, evaluation results might not be the same with a more comprehensive and continuously updated real-life network traffic. However, gathering massive network traffic dataset that exactly represents the most current vulnerabilities and intrusions is quite hard. Therefore, the experimental results can be interpreted by comparing ELM+PT 1 , ELM+PT 2 and other leading approaches using the same dataset with similar evaluation metrics.

5.3 Evaluation and Comparison of the IDS to State-of-the-Art Approaches

The experiments in the previous section show that the last two cases have provided good learning results. However, the ELM+PT 2 approach may not be feasible with large datasets since its time complexity increases with the size of the dataset, as illustrated in Figure 5.3. Although ELM+PT 1 and ELM+PT 2 outperform simple machine learning techniques, it is important to compare the results of these cases to

other leading intrusion detection approaches.

Table 5.4 demonstrates a comparison in terms of average detection (DR) and false positive (FPR) rates between ELM+PT 1, ELM+PT 2 and state-of-the-art intrusion detection systems summarized in Chapter 3. It should be noted that other IDSs mostly used a subset of the dataset with selected features. Moreover, few of these supply per class detection or a false positive rate. For example, the EMD algorithm in [93] has only focused on the detection of DoS attacks. In [88], testing and evaluation data are sampled and aggregated from specific days. Nevertheless, average detection and false positive rates can still be compared to the findings of the leading approaches in order to show the efficacy of the proposed model. The table shows that ELM+PT 1 and ELM+PT 2 demonstrate consistently good performance by using the complete dataset without any sampling method.

Technique	Avg. DR	Avg. FPR
KMC+NBC [88]	0.98	0.02
Bagged-NB [89]	0.45	0.09
Boosted-NB [89]	0.43	0.09
AMGA2-NB [89]	0.94	0.07
Decision Tree Classifier with Snort Priority [90]	0.98	0.06
CAGE-MetaCombiner [91]	0.76	0.06
CAGE-MetaCombiner+ Specialized Ensemble [91]	0.83	0.09
EMD [93](low threshold, only on DoS attacks)	0.90	0.08
EMD [93](high threshold, only on DoS attacks)	0.44	0.01
Case 2 (ELM+PT 1)	0.90	0.09
Case 3 (ELM+PT 2)	0.91	0.09
Case 2 using only normal and attack flows (ELM+PT 1)	**0.994**	**0.016**
Case 3 using only normal and attack flows (ELM+PT 2)	**0.997**	**0.008**

Table 5.4: Comparison of Proposed Method Performance to Leading Approaches

Table 5.4 shows that K means clustering and Naïve Bayes Classifier in [88] as well as the method in [90] achieved the highest average detection rate. When the dataset description of [88] is analyzed, it can be seen that the authors performed experiments by selecting only attack and normal labeled incoming packets sampled from particular days for a particular host. Similarly, the authors in [89] randomly selected normal and attack instances to create a training and test subset for the evaluation of their multi-objective genetic algorithm. In [90], the training dataset consists of Snort alerts corresponding to only true attacks, and these alerts are randomly subdivided into learning, validation and testing datasets. Since the same sampling scenario of these methods can not be recreated, ELM+PT 1 and ELM+PT 2 were evaluated again without unknown flows in order to see the effect of these flows on the average detection rate as well as the average false positive rate.

The last two rows of Table 5.4 present ELM+PT 1 and ELM+PT 2 results when performed on the only attack and normal flows of the ISCX-IDS 2012 dataset. It is clear that both ELM+PT 1 and ELM+PT 2 have very low average false positive rates, as well as provide average detection rates much greater than before reported

in Table 5.4. These results confirm that ELM+PT 1 and ELM+PT 2 outperform other leading approaches and improve the performance of the basic ELM.

In conclusion, the batch learning of estimated probability density functions computed from aggregated heavy hitters can significantly improve the simple ELM algorithm in intrusion detection systems, despite a trade-off between performance and time complexity. The time consumed can be decreased by reducing the input dimension, feature selection or applying an adaptive threshold for detecting heavy hitters. Possible improvements will be presented in the next chapter.

5.4 Summary

This chapter has provided an evaluation of the proposed intrusion detection system based on different cases explained in the previous chapter. Using the experimental setup and tuned parameters, the performance of the first case, where the model is continuously updated, was found to be the worst of the three cases. Unfortunately, the results for this case conflicted with the theoretical model, suggesting that this poor performance may have been attributed to feature characteristics. One future improvement for this case would be to select the most important features. The findings of the second and third case, where the learning was implemented in off-line mode, demonstrated that the off-line version of the proposed method outperforms the most commonly used simple machine learning techniques for intrusion detection. The results also revealed that the proposed system achieves a very good detection rate while preserving a lower rate of false positives and false alarms. In addition, the computational time of the proposed method is in an acceptable range, in that it matches the other traditional machine learning methods. However, it can be inferred that the time complexity of the proposed system is highly dependent on the size of the dataset. Average results of the last two cases have been compared with the leading approaches summarized in Chapter 3. The results showed that the proposed method is one of the most efficient detection methods when applied to the ISCX-IDS 2012 dataset. The following chapter will conclude the thesis by summarizing the key points as well as suggest possible future work.

Chapter 6

Summary and Conclusion

In this thesis, the features of a sample network traffic dataset have been modeled with probability density functions relating to the hierarchical heavy hitters concept in order to improve detection accuracy of the attacks. The developed intrusion detection system provides probability estimation of feature statistics and uses these values to learn different normal and attack behaviors. Clustering network statistics and learning from the probabilistic space of the network data features is the main contribution of the thesis.

Instead of outdated benchmark datasets, the proposed method was evaluated with a relatively new ISCX-IDS 2012 dataset created by the Canadian Institute for Cybersecurity. Since this dataset has the complete capture of a one week network traffic and includes more real life attack scenarios, evaluations were done on this dataset to show the effectiveness of the approach in a more realistic way. As few researchers have studied the ISCX-IDS 2012 dataset, a preliminary work consisting in data analysis and preprocessing was implemented in order to confirm the usability of this dataset. In addition, the dataset was converted into flow level data with the Argus software tool, since the proposed method is based on flow analysis.

Three novel approaches were proposed as different cases. One of them captures the characteristics of the sequential normal network traffic and can update the model without being fully retrained. Major deviations from the normal profile were labeled as outliers, and the update mechanism continuously changes the definition of normal profile. The second and the third approaches were constructed in an off-line fashion in order to respond to the challenges of the first approach. The proposed system for those approaches achieved 90.0 percent of the detection accuracy on the ISCX-IDS 2012 dataset. The experimental results confirmed that these approaches made a significant improvement to the simple extreme learning machine method. The proposed method in the last two approaches also outperformed simple machine learning methods including decision trees, k nearest neighbors and regression models. Furthermore, the proposed detection system achieved a better performance in comparison with other state-of-the-art approaches evaluated on the ISCX-IDS 2012 dataset. Although the time complexity of the solution is highly

dependent on the data size, measured computational time is still satisfactory and can be deployed as a learning mechanism in more complex intrusion detection systems.

In conclusion, results suggest that modeling the probability density functions of the aggregated network statistics can be applied as an intrusion detection mechanism and can possibly cooperate with other network behavior analysis tools for DoS and DDoS defense.

Future Work

As presented in Chapter 5, the first case achieves lower performance than other traditional methods and predicts unknown flows as either normal or attack class. One reason for this discrepancy might be due to characteristics of different features. As plotted in Figure 5.2, estimated probability values of some features in the attack class can resemble the normal behavior. Since these features can negatively affect the overall performance, they might be excluded from the input data. Therefore, feature selection methods for streaming data might be incorporated into the first case. Another possible future work can be adding a forgetting factor for continuous learning. This can fix the size of the tree in a reasonable way and save memory as well as forcing only the most recent normal behavior to be used for probability density estimation, as recently observed source and destination addresses are most likely to connect again soon.

Although ELM+PT 1 and ELM+PT 2 achieve a high detection rate, the time and memory complexity for large datasets might be high. Therefore, some future computational improvements can be included to the proposed method. For instance, appropriate feature selection methods can be deployed in order to decrease the size of the input and improve the learning speed. In addition, a variable threshold can be incorporated in the heavy hitter extraction to limit the size of the trie and the number of clusters.

Bibliography

[1] A. Patcha and J. M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[2] O. Barayas. (2014) How the Internet of Things Is Changing the Cybersecurity Landscape. [Online]. Available: https://securityintelligence.com

[3] R. P. Hartwig and W. Claire, "Cyber risk: Threat and opportunity(white paper)," Insurance Information Institute, Tech. Rep., 10 2015.

[4] C. Ventures. (2016) Cybersecurity Economic Predictions: 2017 to 2021. [Online]. Available: http://cybersecurityventures.com/cybercrime-infographic/

[5] D. Sequeira, "Intrusion prevention systems: security's silver bullet? (white paper)," *Business Communications Review*, vol. 33, no. 3, pp. 36–41, 2003.

[6] "Why IPS Devices and Firewalls Fail to Stop DDoS Threats (white paper)," Arbor Networks, Inc, Tech. Rep., 2015.

[7] F. P. Stanley, "Intrusion detection and response for system and network attacks," Master's thesis, Iowa State University, USA, 2009.

[8] R. Bace and P. Mell, "NIST special publication on intrusion detection systems," DTIC Document, Tech. Rep., 2001.

[9] D. Barbará, J. Couto, S. Jajodia, and N. Wu, "ADAM: a testbed for exploring the use of data mining in intrusion detection," *ACM Sigmod Record*, vol. 30, no. 4, pp. 15–24, 2001.

[10] R. Heady, G. F. Luger, A. Maccabe, and M. Servilla, *The architecture of a network level intrusion detection system.* University of New Mexico. Department of Computer Science. College of Engineering, 1990.

[11] V. Jaiganesh, S. Mangayarkarasi, and P. Sumathi, "Intrusion detection systems: A survey and analysis of classification techniques," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 4, pp. 1629–1635, 2013.

[12] H. Alene, "Graph Based Clustering for Anomaly Detection in IP Networks," Master's thesis, Aalto University, Finland, 2011.

[13] H. Kozushko, "Intrusion detection: Host-based and network-based intrusion detection systems," *Independent study*, 2003.

[14] J. Corsini, "Analysis and evaluation of network intrusion detection methods to uncover data theft," Ph.D. dissertation, Napier University, 2009.

[15] A. Orebaugh, S. Biles, and J. Babbin, *Snort Cookbook: Solutions and Examples for Snort Administrators*. O'Reilly Media, Inc., 2005.

[16] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[17] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[18] K. Ilgun, R. A. Kemmerer, and P. A. Porras, "State transition analysis: A rule-based intrusion detection approach," *IEEE transactions on software engineering*, vol. 21, no. 3, pp. 181–199, 1995.

[19] K. Sequeira and M. Zaki, "ADMIT: anomaly-based data mining for intrusions," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 386–395.

[20] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2. IEEE, 2002, pp. 1702–1707.

[21] W. Li, "Using genetic algorithm for network intrusion detection," *Proceedings of the United States Department of Energy Cyber Security Group*, vol. 1, pp. 1–8, 2004.

[22] J. Gómez, C. Gil, N. Padilla, R. Baños, and C. Jiménez, "Design of a snort-based hybrid intrusion detection system," in *International Work-Conference on Artificial Neural Networks*. Springer, 2009, pp. 515–522.

[23] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: techniques, systems and challenges," *Computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

[24] D.-Y. Yeung and Y. Ding, "Host-based intrusion detection using dynamic and static behavioral models," *Pattern recognition*, vol. 36, no. 1, pp. 229–243, 2003.

[25] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2014.

[26] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on computers*, vol. 51, no. 7, pp. 810–820, 2002.

[27] "Detecting Hackers (Analyzing Network Traffic) by Poisson Model Measure," Simon Fraser University, School of Engineering Science, Tech. Rep., 2004.

[28] J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo, "Stochastic protocol modeling for anomaly based network intrusion detection," in *Proceedings of the first IEEE International Workshop on Information Assurance*. IEEE, 2003, pp. 3–12.

[29] D. Heckerman, "A tutorial on learning with Bayesian networks," in *Learning in graphical models*. Springer, 1998, pp. 301–354.

[30] S. Y. Lim and A. Jones, "Network anomaly detection system: The state of art of network behaviour analysis," in *Convergence and Hybrid Information Technology, 2008. ICHIT'08. International Conference on*. IEEE, 2008, pp. 459–465.

[31] L. A. Chappell, *Wireshark network analysis: the official Wireshark certified network analyst study guide*. Protocol Analysis Institute, Chappell University, 2010.

[32] V. Jacobson, C. Leres, and S. McCanne, "TCPDUMP public repository." [Online]. Available: http://www.tcpdump.org

[33] C. E. Perkins, *IP mobility support for IPv4, revised*. Internet Engineering Steering Group, 2010.

[34] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for IP flow information export (IPFIX)," Tech. Rep., 2004.

[35] M. Lucas, *Network flow analysis*. No Starch Press, 2010.

[36] D. Plonka, "FlowScan: A Network Traffic Flow Reporting and Visualization Tool," in *LISA*, 2000, pp. 305–317.

[37] M. Thomas, L. Metcalf, J. Spring, P. Krystosek, and K. Prevost, "Silk: A tool suite for unsampled network flow analysis at scale," in *Big Data (BigData Congress), 2014 IEEE International Congress on*. IEEE, 2014, pp. 184–191.

[38] C. Bullard, "Argus, Auditing Network Activity." [Online]. Available: https://www.qosient.com/argus/index.shtml

[39] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *Computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[40] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, 2003, pp. 25–36.

[41] G. Gu, P. Fogla, D. Dagon, W. Lee, and B. Skorić, "Measuring intrusion detection capability: an information-theoretic approach," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 90–101.

[42] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Internet traffic behavior profiling for network security monitoring," *IEEE/ACM Transactions On Networking*, vol. 16, no. 6, pp. 1241–1252, 2008.

[43] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval*. ACM press New York, 1999, vol. 463.

[44] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the resource consumption of network intrusion detection systems," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 1. ACM, 2008, pp. 437–438.

[45] A. Kalliola, Y. Miche, I. Oliver, S. Holtmanns, B. Atli, A. Lendasse, K.-M. Bjork, A. Akusok, and T. Aura, "Learning flow characteristics distributions with elm for distributed denial of service detection and mitigation," in *Proceedings of ELM-2016*. Springer, 2018, pp. 129–143.

[46] T. F. Lunt and R. Jagannathan, "A prototype real-time intrusion-detection expert system," in *Security and Privacy, 1988. Proceedings., 1988 IEEE Symposium on*. IEEE, 1988, pp. 59–66.

[47] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[48] S. Kumar and E. H. Spafford, "A pattern matching model for misuse intrusion detection," Purdue University, Department of Computer Science, Tech. Rep., 1994.

[49] N. Hubballi and V. Suryanarayanan, "False alarm minimization techniques in signature-based intrusion detection systems: A survey," *Computer Communications*, vol. 49, pp. 1–17, 2014.

[50] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles, "HIDE: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification," in *Proc. IEEE Workshop on Information Assurance and Security*, 2001, pp. 85–90.

[51] K. Das, J. Schneider, and D. B. Neill, "Anomaly pattern detection in categorical datasets," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 169–176.

[52] I. Kang, M. K. Jeong, and D. Kong, "A differentiated one-class classification method with applications to intrusion detection," *Expert Systems with Applications*, vol. 39, no. 4, pp. 3899–3905, 2012.

[53] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[54] C. Zhang, G. Zhang, and S. Sun, "A mixed unsupervised clustering-based intrusion detection model," in *Genetic and Evolutionary Computing, 2009. WGEC'09. 3rd International Conference on.* IEEE, 2009, pp. 426–428.

[55] S. Chebrolu, A. Abraham, and J. P. Thomas, "Feature deduction and ensemble design of intrusion detection systems," *Computers & security*, vol. 24, no. 4, pp. 295–307, 2005.

[56] G. Wang, J. Hao, J. Ma, and L. Huang, "A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering," *Expert Systems with Applications*, vol. 37, no. 9, pp. 6225–6232, 2010.

[57] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on.* IEEE, 2010, pp. 408–415.

[58] Y.-S. Goh and E.-C. Tan, "An integrated approach to improving back-propagation neural networks," in *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994.* IEEE, 1994, pp. 801–804.

[59] C. Cheng, W. P. Tay, and G.-B. Huang, "Extreme learning machines for intrusion detection," in *Neural networks (IJCNN), the 2012 international joint conference on.* IEEE, 2012, pp. 1–8.

[60] W. Srimuang and S. Intarasothonchun, "Classification model of network intrusion using Weighted Extreme Learning Machine," in *Computer science and software engineering (JCSSE), 2015 12th international joint conference on.* IEEE, 2015, pp. 190–194.

[61] J. M. Fossaceca, T. A. Mazzuchi, and S. Sarkani, "MARK-ELM: Application of a novel Multiple Kernel Learning framework for improving the robustness of Network Intrusion Detection," *Expert Systems with Applications*, vol. 42, no. 8, pp. 4062–4080, 2015.

[62] A. Kalliola, T. Aura, and S. Šćepanović, "Denial-of-service mitigation for internet services," in *Nordic Conference on Secure IT Systems.* Springer, 2014, pp. 213–228.

[63] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding DDoS mitigation and traffic management with software defined networking," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 248–254.

[64] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Towards Generating Real-life Datasets for Network Intrusion Detection," *IJ Network Security*, vol. 17, no. 6, pp. 683–701, 2015.

[65] M. Lichman, "UCI Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 2013.

[66] "NSL-KDD dataset," University of New Brunswick, Canadian Institute for Cybersecurity. [Online]. Available: http://www.unb.ca/cic/research/datasets/nsl.html

[67] "Darpa Intrusion Detection Evaluation," Massachusetts Institute of Technology, Cambridge, MIT Lincoln Laboratory, 2000. [Online]. Available: https://www.ll.mit.edu/ideval/index.html

[68] "DEFCON Capture the Flag Contest traces," The Shmoo Group, 2011.

[69] "CAIDA Data - Overview of Datasets, Monitors, and Reports," 2011. [Online]. Available: http://www.caida.org/data/overview/

[70] "UNIBS: Data sharing," 2009. [Online]. Available: http://netweb.ing.unibs.it/~ntw/tools/traces/

[71] L. Loris, "A new DOS Perl Programm," https://github.com/llaera/slowloris.pl, 2013.

[72] G. Kumar, "Denial of service attacks–an updated perspective," *Systems Science & Control Engineering*, vol. 4, no. 1, pp. 285–294, 2016.

[73] Obnosis, "QuickTools for PLUG Hackfest demonstrations and presentations," https://github.com/obnosis/QuickTools, 2012.

[74] J. Shlens, "A tutorial on principal component analysis," *arXiv preprint arXiv:1404.1100*, 2014.

[75] E. Alpaydin, *Introduction to Machine Learning*. MIT press, 2014.

[76] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[77] I. Narsky and F. C. Porter, "Linear and Quadratic Discriminant Analysis, Logistic Regression, and Partial Least Squares Regression," *Statistical Analysis Techniques in Particle Physics: Fits, Density Estimation and Supervised Learning*, pp. 221–249.

[78] H. Zhang, "Exploring conditions for the optimality of naive Bayes," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 02, pp. 183–198, 2005.

[79] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[80] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[81] T. M. Mitchell, *Machine learning. WCB*. McGraw-Hill Boston, MA, 1997.

[82] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics Springer, Berlin, 2001.

[83] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[84] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[85] B. Ripley, "Neural networks and pattern recognition," *Cambridge University*, 1996.

[86] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[87] C. M. Bishop, "Pattern recognition," *Machine Learning*, vol. 128, pp. 1–58, 2006.

[88] W. Yassin, N. I. Udzir, Z. Muda, M. N. Sulaiman *et al.*, "Anomaly-based intrusion detection through k-means clustering and naives bayes classification," in *Proc. 4th Int. Conf. Comput. Informatics, ICOCI*, no. 49, 2013, pp. 298–303.

[89] G. Kumar and K. Kumar, "Design of an evolutionary approach for intrusion detection," *The Scientific World Journal*, vol. 2013, 2013.

[90] A. Ammar *et al.*, "A decision tree classifier for intrusion detection priority tagging," *Journal of Computer and Communications*, vol. 3, no. 04, p. 52, 2015.

[91] G. Folino, F. S. Pisani, and P. Sabatino, "A distributed intrusion detection framework based on evolved specialized ensembles of classifiers," in *European Conference on the Applications of Evolutionary Computation*. Springer, 2016, pp. 315–331.

[92] P. Foremski, C. Callegari, and M. Pagano, "Waterfall: rapid identification of ip flows using cascade classification," in *International Conference on Computer Networks*. Springer, 2014, pp. 14–23.

[93] Z. Tan, A. Jamdagni, X. He, P. Nanda, R. P. Liu, and J. Hu, "Detection of denial-of-service attacks based on computer vision techniques," *IEEE transactions on computers*, vol. 64, no. 9, pp. 2519–2533, 2015.

[94] K. K. Vasan and B. Surendiran, "Dimensionality reduction using Principal Component Analysis for network intrusion detection," *Perspectives in Science*, vol. 8, pp. 510–512, 2016.

[95] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, pp. 464–475.

[96] F. M. Liang, "Word hy-phen-a-tion by com-put-er," Calif. Univ. Stanford. Comput. Sci. Dept., Tech. Rep., 1983.

[97] S. Sahni, *Data Structures, Algorithms and Applications in Java*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1999.

[98] V. Srinivasan and G. Varghese, "Faster IP lookups using controlled prefix expansion," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1. ACM, 1998, pp. 1–10.

[99] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[100] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in streaming data," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 4, p. 2, 2008.

[101] R. F. Hoskins, *Delta functions: Introduction to generalized functions*. Elsevier, 2009.

[102] C. Tomasi, "Lecture notes in Computational Modeling for the Sciences," Spring 2007.

[103] W. Härdle, M. Müller, S. Sperlich, and A. Werwatz, *Nonparametric and semiparametric models*. Springer Science & Business Media, 2012.

[104] H. Shimazaki and S. Shinomoto, "A method for selecting the bin size of a time histogram," *Neural computation*, vol. 19, no. 6, pp. 1503–1527, 2007.

[105] G.-B. Huang, L. Chen, C. K. Siew *et al.*, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[106] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "OP-ELM: optimally pruned extreme learning machine," *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 158–162, 2010.

[107] Y. Miche, M. Van Heeswijk, P. Bas, O. Simula, and A. Lendasse, "TROP-ELM: a double-regularized ELM using LARS and Tikhonov regularization," *Neurocomputing*, vol. 74, no. 16, pp. 2413–2421, 2011.

[108] C. R. Rao and S. K. Mitra, "Generalized inverse of matrices and its applications," 1971.

[109] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on neural networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[110] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU Press, 2012, vol. 3.

[111] Q. Leng, H. Qi, J. Miao, W. Zhu, and G. Su, "One-class classification with extreme learning machine," *Mathematical problems in engineering*, vol. 2015, 2015.

[112] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[113] Z. Zhang and X. Zhu, "SVC-based multivariate control charts for automatic anomaly detection in computer networks," in *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*. IEEE, 2007, pp. 56–56.

[114] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *arXiv preprint arXiv:1601.07996*, 2016.

[115] M. Aly, "Survey on multiclass classification methods," *Neural Netw*, pp. 1–9, 2005.

Appendix A: Generating Profiles

Field	Description (GB)
appName	transport layer protocols such as TCP and UDP.
totalSourceBytes	total bytes sent from source to destination in a single profile
totalDestinationBytes	total bytes sent from destination to source in a single profile
totalDestinationPackets	total packets sent from source to destination in a single profile
totalSourcePackets	total packets sent from destination to source in a single profile
sourcePayloadAsBase64	actual message sent from source to destination and decoded as bytes
sourcePayloadAsUTF	actual message sent from source to destination and decoded as textual format
destinationPayloadAsBase64	actual message sent from destination to source and decoded as bytes
destinationPayloadAsUTF	actual message sent from destination to source and decoded as textual format
direction	Flow direction: local-to-local(L2L), local-to-remote (L2R), remote-to-local(R2L), remote-to-remote (R2R)
sourceTCPFlagsDescription	from source to destination TCP control bits
destinationTCPFlagsDescription	from destination to source TCP control bits
source	sender IP in IPv4 format
protocolName	communication protocol as textual format
sourcePort	port number from source to destination
destination	receiver IP in IPv4 format
destinationPort	port number from destination to source
startDateTime	communication profile starting time in YY/MM/DD
stopDateTime	communication profile ending time in YY/MM/DD
Tag	Normal or Attack profile

Appendix B: Argus Flow Level Numerical Features

Feature Name	Type	Description
rank	D	ordinal value of the output flow record
stime	C	start time
ltime	C	stop time
seq	D	Argus sequence number
dur	C	total record duration
runtime	C	total active flow run time
mean	C	average duration of aggregated records
stddev	C	standard duration of aggregated records
sum	C	total accumulated duration of aggregated records
saddr4	D	source IP addres (MSB)
saddr3	D	source IP addres
saddr2	D	source IP addres
saddr1	D	source IP addres (LSB)
daddr4	D	destination IP address (MSB)
daddr3	D	destination IP address
daddr2	D	destination IP address
daddr1	D	destination IP address (LSB)
proto	D	transaction protocol
sport	D	source port number
dport	D	destination port number
stos	C	source TOS byte value
dtos	C	destination TOS byte value
sdsb	C	source diff serve byte value
ddsb	C	destination diff serve byte value
sttl	D	src to dst TTL value
dttl	D	dst to src TTL serve byte value
shops	D	estimate number of IP hops from src to this point
dhops	D	estimate number of IP hops from dst to this point
sipid	D	source IP identifier
dipid	D	destination IP identifier
autoid	D	auto generated identifier(mysql)
nstroke	D	number of observed strokes
snstroke	D	number of observed strokes from src to dst

dnstroke	D	number of observed strokes from dst to src
pkts	D	total transaction packet count
spkts	D	src to dst packet count
dpkts	D	dst to src packet count
bytes	D	total transaction bytes
sbytes	D	src to dst transaction bytes
dbytes	D	dst to src transaction bytes
appbytes	D	total application bytes
sappbytes	D	total application bytes from src to destination
dappbytes	D	total application bytes from dst to source
pcr	C	producer consumer ratio
loss	D	pkts transmitted or dropped
sloss	D	source pkts retransmitted or dropped
dloss	D	destination packets retransmitted or dropped
retrans	D	pkts retransmitted
sretrans	D	source pkts retransmitted
dretrans	D	destination pkts retransmitted
sgap	D	source bytes missing in the data stream
dgap	D	destination bytes missing in the data stream
rate	C	pkts per second
srate	C	source pkts per second
drate	C	destination pkts per second
swin	D	source TCP window advertisement
dwin	D	destination TCP window advertisement
stcpb	D	source TCP base sequence number
dtcpb	D	destination TCP base sequence number
tcprtt	C	TCP connection setup round trip time
synack	C	TCP connection setup time between SYN and SYN_ACK
ackdat	C	TCP connection setup time between SYN_ACK and ACK
offset	D	record byte offset in file or stream
smeansz	C	mean of the flow packet size transmitted by the src
dmeansz	C	mean of the flow packet size transmitted by the dst
C-Continuous, D-Discrete		