Multi-timeline Based Real-time Anomaly Detection in Network Traffic

Kriangkrai Limthong

Doctor of Philosophy

Department of Informatics,
School of Multidisciplinary Sciences

The Graduate University for Advanced Studies (SOKENDAI)

September 2015

A dissertation submitted to
The Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies (SOKENDAI)
In partial fulfillment of the requirements for
The degree of Doctor of Philosophy

Supervisor:

Kensuke FUKUDA	National Institute of Informatics, SOKENDAI

Advisory Committees:

Shigeki YAMADA	National Institute of Informatics, SOKENDAI
Yusheng JI	National Institute of Informatics, SOKENDAI
Michihiro KOIBUCHI	National Institute of Informatics, SOKENDAI
Toshiharu SUGAWARA	Department of Computer Science and Communications Engineering, Waseda Unviersity

Abstract

The volume of traffic in both core and access networks has exponentially increased every year over the past few decades. The computer attacks also have increased in sophisticated techniques to evade existing intrusion detection systems. It is rather difficult for daily network operators and administrators to inspect every single packet or flow for discovering anomalies. Therefore, the need to automatically detect attacks and unusual incidents in computer networks is of crucial importance for nowadays operations.

An effective system that could expeditiously detect a broad range of anomalies would enable administrators to prevent serious consequences of anomalies related to network security, availability, or reliability. For over a decade, many researchers have been studying to improve techniques for anomaly detection by proposing and applying plenty of methods from simple to sophisticated ones. Unfortunately, most of the studies are batch processing techniques, and many of them are not fairly flexible to detect a vast variety of anomalies caused by threats or accidents.

In this study, we proposed a detection system using microscopic to macroscopic designs for real-time anomaly detection. The key idea of the proposed system is that the system learns network traffic from multiple timelines rather than a single timeline of input data employed by most conventional detection systems. The advantages of the proposed system are 1) improving on detection performance over the single timeline, 2) flexibility in applying the proposed system to various types of networks or protocols, 3) robustness to incorrect training data or manipulating data by attackers, 4) performance improvement with weighted multiple timelines, and 5) real-time detectability for anomalies caused by threats or accidents. We also performed a series of experiments to examine the proposed system by employing three standard machine learning algorithms, namely multivariate normal distribution, k-nearest neighbor, and one-class support vector machine. In our experiments, we extracted nine key features on account of several selected attacks from a testbed data set. We examined capabilities of the proposed system in many aspects including detection performance, robustness, learning rate, time consumption, different volume of background traffic, time of anomaly occurrence, and weighting for old data.

The experimental results show that the proposed system with machine learning algorithms effectively detected several of anomalies caused by threats or accidents. Our experiment also indicates that the multi-timeline technique outperforms both conventional real-time and a combination of single and multi-timeline. The proposed system shows a robust capability to learn from incorrect training data or manipulating data by attackers. Moreover, two of the three algorithms with the proposed system could learn from training data in reasonable time. The proposed system can not only enable network administrators to detect novel types of attacks but can be used to identify abnormal behavior of their networks in real time as well.

Acknowledgements

I would like to express my gratitude to everyone for their encouragement, enthusiasm, support, and guidance during my study as a doctoral student at the Graduate University for Advanced Studies (SOKENDAI).

First and foremost, I would like to gratefully and sincerely thank my supervisor, Kensuke Fukuda, whose support, encouragement, and supervision from the preliminary to the concluding level enabled me to develop not only an understanding of the subject, but also being a good researcher. This dissertation would not have been possible without his assistance.

I also would like to express a special thanks to my other committee members, Shigeki Yamada, Yusheng Ji, Michihiro Koibuchi, Motonori Nakamura, and Toshiharu Sugawara for their valuable time discussing and making helpful comments regarding my study.

I would like to gratefully acknowledge the funding from the Faculty Members Development Scholarship Program of Bangkok University, Thailand that provided the necessary financial support for this research and other expenses.

I would like to thank my parent, Somnuk and Aramsri Limthong, for all their love and encouragement in a number of ways through the entire period. I also owe my loving thanks to Yuri Yamazaki. Without her encouragement and understanding, it would have been impossible for me to finish this work.

I am indebted to my many colleagues in both of the National Institute of Informatics (NII) and Thailand to support and cheer me up all times when I needed them during my stay in Japan. I cannot forget to express appreciation to all of the staff who work in the NII office, especially Ayako Maeda, Mina Hiraiwa, Mio Takahashi, Mayumi Tsubaki, Mizuki Matsuoka, and Miyuki Kobayashi because their rich patience and enthusiasm make the NII a great place.

Finally, for any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own.

List of Original Publications

International Transactions and Journals

1. **Kriangkrai Limthong**; Kensuke Fukuda; Yusheng Ji; Shigeki Yamada, "Unsupervised learning model for real-time anomaly detection in computer networks," *IEICE Transactions on Information and Systems*, vol.E97-D, no.8, pp.2084-2094, August 2014

International Conference Proceedings

1. **Kriangkrai Limthong**; Kensuke Fukuda; Yusheng Ji; Shigeki Yamada, "Impact of Time Interval on Naive Bayes Classifier for Detecting Network Traffic Anomalies," *Computer Science and Engineering Conference (ICSEC), 2011 International Conference on*, pp.1-6, 7-9 September 2011.

2. **Kriangkrai Limthong**; Pirawat Watanapongse; Kensuke Fukuda, "A wavelet-based anomaly detection for outbound network traffic," *Information and Telecommunication Technologies (APSITT), 2010 8th Asia-Pacific Symposium on*, pp.1-6, 15-18 June 2010.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer and network security, i.e. cyber security are critical issues for all participants in information industries, because almost all daily activities rely on computers and the Internet nowadays. The principle objective of cyber security is protection of information and properties from theft, corruption, or natural disaster, while allowing the information and properties to remain accessible and reliable to its intended and legitimated users. However, a report from the Computer Emergency Response Team Coordination Center (CERT/CC) [1] in 2003 indicates an exponential increase in the number of security incidents every year from 1998 to 2003 as shown in Figure 1.1. This report strongly suggests that every single system connected to the Internet highly confront various and sundry threats of cyber security, including attacks, viruses, worms. The result of this report also shows a massive warning sign that all computers and network systems connected to the Internet presently involve various high relative risks of cyber security.

Not only a dramatic increase in security incidents arises on the Internet, but there also has been a steady growth of sophisticated techniques to evade detection schemes and prevention systems. Meanwhile, a number of attackers do not need in-depth technical knowledge in order to carry out such sophisticated attacks. According to studies by John Mchugh [2] and Howard F. Lipson [3], both current trends in attack sophistication and intruder technical knowledge are graphically represented as shown in Figure 1.2. In this figure, the dots along the attack sophistication line show inventive attack techniques discovered between 1990 and early 2010. Unfortunately, there is no 100-percent guarantee that some of these or novel attacks would not happen to the systems even if many systems can detect these attacks. As a consequence, the protection against a broad range of computer attacks is of crucial importance.

In addition to different types of computer attacks caused by human inten-

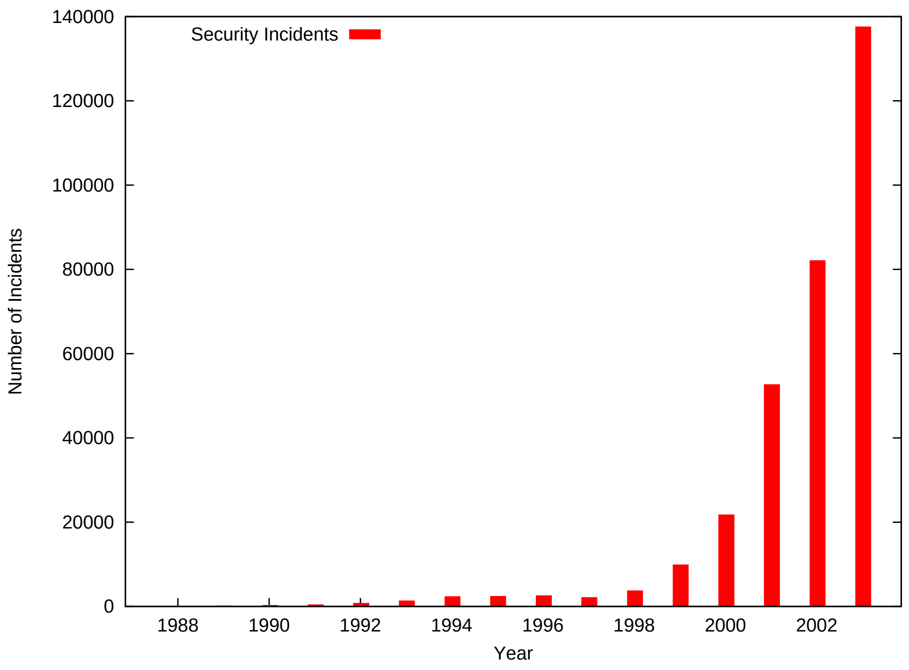

Figure 1.1: Security incidents on the Internet from 1988 to 2003.

tion, there is another kind of unusual incident in computer systems caused by accidents such as power outages, misconfigurations, and flash crowds. These unusual incidents have also produced several adverse effects on availability and reliability of computer systems. Therefore, an effective technique for anomaly detection does not mean only to perceive computer threats, but to perceive unusual incidents caused by accidents as well. In our context, anomaly detection is generally different from intrusion detection. Anomaly detection has to cover a number of unusual incidents caused by attacks or accidents from inside or outside; however, intrusion detection only focuses on attacks or threats from outside of the network system.

There are several issues that make anomaly detection in computer networks much more difficult than those in other domains, such as fraud detection, fault detection, system health monitoring. One of the major issues is the high growth rates of the Internet traffic which network operators and administrators face difficulties to detect anomalies. A study by K. G. Coffman and A. M. Odlyzko [4] suggested that the volume of the Internet traffic double every three months in early decades. As a result, it is quite difficult for day-to-day operators or administrators to manually inspect every single packet or single flow that pass through their own networks. The operators

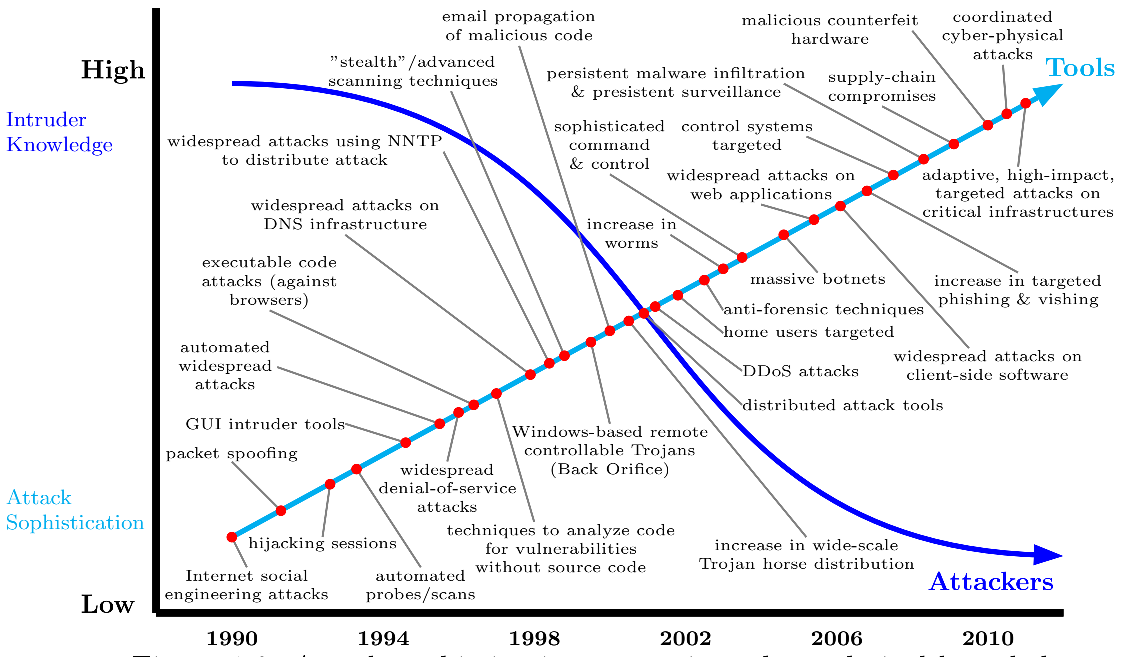

Figure 1.2: Attack sophistication versus intruder technical knowledge.

need a fully automatic system to perform such network inspection and detection tasks.

The next issue making detection more difficulty is that detecting anomalies in computer networks really depends on a variety of factors in particular situation. As an obvious example of these factors, we show a simple situation which might occur in a network system. Suppose that monitoring traffic data in a small office shows that someone is surfing the Internet from inside during office hours, we easily classify the data in this situation as a normal behavior or a normal class. On the other hand, if exactly the same situation occurs at midnight of the day when usually nobody is working at that moment and such behavior had never happened before, it is quite difficult for network administrators to clearly define this situation as normal or abnormal behavior. From this example, classifying data in this network traffic depends on conditions at a particular time and in a particular place, referred to as contextual anomalies.

Anomalies in computer networks have been defined as contextual anomalies or conditional anomalies [4], because a data instance is anomalous in a specific context but might be normal otherwise. The contextual anomalies have been found on data related to position, location, and time, such as spatial data or time-series data. Contextual anomalies have been commonly studied in spatial data [5, 6] and time-series data [7, 8]. Figure 1.3 clearly

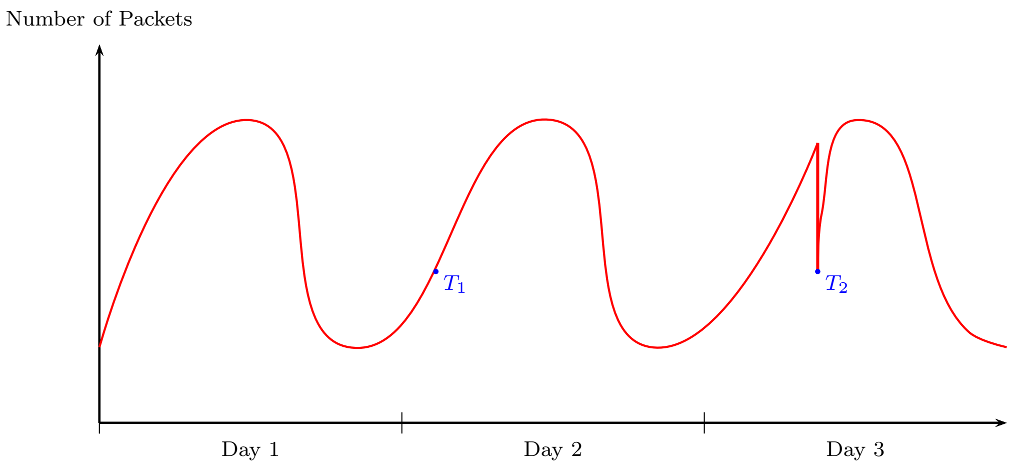

Figure 1.3: An example of anomaly in time-series data.

shows an example of contextual anomaly for time-series data of a computer network that shows the number of packets over the last few days. The lower number of packets might be normal during early morning at time T_1 in that network, but the same number of packets during noon at time T_2 on another day would be an anomaly.

Last but not least, a big problem of anomaly detection in computer networks is that attackers or intruders make a large amount of effort to imitate normal traffic in order to evade or influence detection systems, especially when attackers perceive the detection scheme used in the target system. However, such imitation or manipulation rarely occurs in other domains of anomaly detection. In other domains, such as medical care for detecting disease outbreaks [9] or quality control in factories [10], almost all anomalies in these problem domains are caused by nature or accidents, there is no human intention of causing anomalies. Most of the anomalies in computer network, however, are likely caused by human intention, and they could sometime caused by accidents. With freely and available sophisticated attack tools, attackers can effortlessly probe or scan the security system in order to discover the network structure and detection technique, then they also can imitate normal traffic to hide from the detection system. Therefore, the real challenge of our study is not only expeditiously detecting a wide range of anomalies but also developing a robust system that is not easily evade or easily influenced by attackers.

Over past decades, researchers have proposed a large number of various techniques to detect anomaly in computer networks, from simple techniques to sophisticated ones. These techniques have been categorized into two fundamental categories: signature-based techniques and statistical-based techniques [11, 12]. Some studies offered combination of signature-based and statistical-based techniques known as hybrid approach [13]; however, this approach is still based upon signature-based and statistical-based techniques. Before an introduction to anomaly detection using machine learning techniques, we discuss the advantages and disadvantages of signature-based techniques and statistical-based techniques.

Signature-based techniques find network traffic for a series of bytes, packet sequences, or network flows known to be anomalous. A key advantage of this detection method is that signatures are easy to develop and understand if target network traffic behavior is well-known. For example, one uses a signature that looks for particular strings within an exploit payload to detect attacks that are attempting to exploit a particular buffer-overflow vulnerability. The alarms generated by a signature-based system can easily indicate what caused the alert. Moreover, pattern matching can be performed very quickly on modern systems so the amount of power needed to perform these checks is minimal for a limited set of signatures. For example, if the systems intend to protect only communicate via DNS, ICMP and SMTP, all signatures related to other protocols can be removed.

Signature engines also have their disadvantages. Due to they only detect known attacks, a signature must be created for every single attack, and other novel attacks cannot be detected. Signature engines are also likely to suffer from false positives because they are generally based on regular expressions and string matching. Both problems of signature mechanisms are only search for strings within packets or flows over the transmission line.

Although signatures work well against attacks with a fixed behavioral pattern, they do not work well against mixtured attack patterns created by a human or a worm with self-modifying characteristics. Detection is more complicated by advancing exploit technology that allows malicious users to conceal their attacks behind the NOP generators, payload encoders and encrypted data channels. The overall ability of a signature engine to scale against these changes is badly injured by the fact that a new signature must be created for each variation; as the rule set grows, the performance of engines unavoidably declines. This is the clearly reason that many signature-based systems require high-end hardware and rich resources.

Eventually, the signature-based techniques are considered as an arm race between attackers and signature developers, how speed at which new signatures can be developed and applied to the system.

The statistical-based techniques, however, are based on the concept of a baseline for network behavior. This baseline is a description of accepted network behavior which is learned or specified by the network administrators, or both. Unusual incidents or anomalies are caused by any behaviors that fall outside the predefined or accepted model of behavior. A crucial part of statistical-based techniques is the capability to inspect protocols at all layers. For every protocol monitored, the engine must has the ability to decode and process the protocol to understand its goal and the payload. This inspection process is computationally expensive at first, but it allows the system to scale as the rule set grows and alert with fewer false positives when variances from the accepted behaviors are detected.

A disadvantage of statistical-based techniques is the difficultly of defining rules. Each protocol being analyzed must be defined, implemented and tested for accuracy. Moreover, detailed knowledge of normal network behavior must be constructed for accurate detection. On the other hand, once a protocol has been built and a behavior defined, the engine can scale more quickly and easily than the signature-based techniques because a new signature does not have to be created for every attack and potential variant. Another downside of statistical-based techniques is that malicious incidents that fall within normal usage patterns is not detected.

However, statistical-based techniques have an advantage over signature-based techniques in that a new attack for which a signature does not exist can be detected if it falls out of the normal traffic patterns. The best example of this is how such systems detect new automated virus spreading. When a new system is infected with a virus it usually starts scanning for other vulnerable systems at an abnormal rate flooding the network with malicious traffic, thus triggering a TCP connection or bandwidth rule.

Machine learning is one of the several techniques have been proposed by researchers to solve anomaly detection problem. We could consider machine learning to be a statistical-based technique, which has high capabilities to learn automatically to recognize complex patterns and make intelligent decisions on the basis of data [14]. There are two fundamental types of machine learning that can be applied for network traffic anomaly detection: supervised learning and unsupervised learning [15]. The distinction between these two types is drawn from how algorithms learn to classify data.

Supervised learning is the machine learning technique of inferring a function from labeled training data. The training data consist of a set of training examples. Each example is a pair consisting of an input value (normally called a feature vector) and a desired output value. A supervised learning algorithm analyzes training data and produces an inferred function, which is commonly called a classifier. This technique has been well studied and could cover and detect a wide range of network anomalies [16]. The general assumption of supervised learning for anomaly detection is that the anomalous traffic is statistically different from normal traffic. Many studies have been proposed and have applied several algorithms based on this assumption, such as the Bayesian network algorithm [17], k-nearest neighbor algorithm [18], and support vector machine algorithm[19]. Unfortunately, detection performance and other key aspects of these algorithms has not been compared. The main problem of applying supervised learning to network traffic is collecting traffic as training data.

Contrary to supervised learning, the unsupervised learning is the machine learning technique that takes a set of unlabeled training data as input, and then attempts to find hidden structure in the data. The unsupervised learning is closely related to the problem of density estimation in statistics. Before detecting anomalies, many researchers collect data for a certain amount of time, one day for example, and then clustered data into several groups. After that they can detect anomalies on the basis of the assumption that major groups are normal traffic and minor groups are anomalous traffic [20, 21]. Unfortunately, this assumption is not true in many cases, especially when we focus on traffic occur in a short period as real-time system. Examples of such cases are distributed denial of service attacks (DDoS), viruses or worms spreading, and flash crowds. In these examples, the amount of anomalous traffic can be larger than those of normal traffic, and as a result, anomalous traffic composes of a major group. In this case, learning algorithms will misclassify anomalous traffic as normal traffic and vice versa. In other cases, outages and misconfigurations for example, although no anomalous packet occurs, an unusual reduction in normal traffic also indicates that an unexpected incident arises. All of these unusual incidents cannot be generally detected by using the unsupervised learning as clustering techniques.

Machine learning methods for general purposes are not enough to detect anomalies in network traffic. There are two compelling reasons why we have to modify machine learning methods for network traffic anomaly detection. The first reason is that classifying traffic data depends on many factors, particularly on time and location conditions, but most other domains of anomaly detection are time and location independent. The second reason is that attackers place a large amount of effort into imitating data to evade or influence detection systems in computer networks, but this imitation rarely occurs in other domains of anomaly detection. Therefore, we cannot directly apply machine learning methods for general purposes to detect anomalies in network traffic.

One of the serious obstacles to detect anomalies in computer networks is that most of the existing and proposed methods utilize batch processing. This means that the methods have to collect traffic data for certain amount before examine whether data contain anomalies or not. The main disadvantage of batch processing methods is an unacceptable delay between time that anomalies occur and time that operators be notified. Real-time anomaly detection must guarantee response within strict time constraints, mostly in the order of less than a minute and sometimes a second.

A variety of anomalies in computer and network systems has adversely affected security, availability, or reliability. A lack of one or more of these crucial issues could make a business lose a large amount of revenue, and could damage corporate image or reputation in some ways. An effective system that could quickly and accurately detect such attacks or accidents would be able to prevent serious damages to computer systems and their own business. Consequently, the prospect of real-time anomaly detection in computer and network systems is really attractive and crucial for information industries.

1.1 Motivation

Anomalies in computer networks have been changed dramatically in the past decades. Modern attacks and abnormal behavior of network users are richly diverse with various characteristics. There is no single solution to detect all types of attacks and unusual incidents for dissimilar network environment. Most existing methods for network anomaly detection are specific to each particular anomaly or particular protocol, and most of them have assumed the batch processing technique. Therefore, we do require a general system which could expeditiously detect a broad range of attacks and misuse incidents in computer network as possible as real-time processing. The system should have flexible capabilities to easily adapt for a specific anomaly, protocol, etc. or even particular network environment.

Machine learning has the ability of the computer to learn from the previous experience or history, and performs better for a given task, as past behavior resembles future one. It is an artificial intelligence (AI) technique that provides computers with the ability to learn without being explicitly programmed. The machine learning technique has high capabilities to learn and classify data automatically, so it could be applied to detect a variety of anomalies in network traffic. There are also a hundred of machine learning algorithms from simple to sophisticated one, which have been developed more than a half century.

The machine learning techniques for general purposes are not suitable for real-time anomaly detection in computer network. We hypothesize that the specific representation of input data for computer networks should perform anomaly detection with a better performance than those of input data for general tasks. The representation of input data should tolerate any kind of noise, data imitation by attackers or misbehavior, and should be difficult for attackers and intruders to hide themselves from the detection system. The representation of input data should be able to perform such detection task in real time so that the operation could receive notification of anomalies as fast as possible, in our sense the time between anomalies occur and notification would be equal to or less than a minute.

We concentrate our study and experiments on how to represent input data rather than detection algorithms. The representation of input data is a higher level so that we could employ any algorithm of machine learning techniques. One of the main issues is the intrinsic characteristics of data representation that provide flexibility of capabilities to detect specific anomalies as well as general anomalies. For example, we could focus on particular addresses, ports, or protocols for certain anomalies without modification in learning algorithm. These all above are our motivation for conducting the research experiments in this thesis.

1.2 Problem Statement

The problem in this study is *"how to detect network traffic anomalies caused by threats or accidents in real time and attackers hardly evade or manipulate the detection system"*. The word *real time* in our context means that the detection system produces an alert after anomaly occurrence in a minute or less. We assume that these anomalies in network traffic are caused by threats or accidents, and a history of attack-free (or mostly attack-free) traffic is available from the network system that we are monitoring. We also assume that our system will be a part of a more comprehensive detection system which also employs hand-coded rules, such as anti-virus or firewall systems which prevent abnormal behavior and misuse of network traffic.

For specified problems, we intend to address the remaining problems from previous studies as follows:

1. Most of previous studies can not detect anomalies in real time.
2. Attackers can easily evade or manipulate detection system if they perceive the employed detection technique.
3. Most of prior work can detect anomalies caused by only threats, but anomalies caused by accidents also produce harmful effects on network systems.

These all three problems cannot be solved by a single solution or a single technique. Therefore, the primary aim of our study is to address the three problems by using a single detection system.

1.3 Contributions

Our study makes two distinct contributions for the field of network security as follows:

1. **The multi-timeline system for real-time anomaly detection**: it is a detection system by using multi-timeline representation from passing traffic as input information. This system does not require any labeled data to detect anomalies in network traffic. Our study proves that the proposed system is versatile and automatically detect anomaly caused by threats or accidents in real time with promising performance. The detection system also provides flexibility of capabilities to focus on particular logical boundary of network system or particular anomalies.

2. **Comparison of three learning algorithms**: we applied three learning algorithms with the multi-timeline system in order to examine the capabilities of the proposed system in different aspects. These are well-know learning algorithms that have been used in various detection problems.

There are many challenges and issues that our study intends to cover for the anomaly detection problems in network traffic which other studies cannot cover them all. The main differences are as follows:

1. We propose a detection system rather than detection algorithm. The main advantage of proposing a detection system is that we can apply any algorithm, and it is easy to change from an algorithm to any other algorithm without modification or with tiny adjustment.

2. Our study focuses on real-time scheme while nearly all other studies have been operated by offline or batch scheme.

3. We intend to detect contextual and point anomalies, while other studies can detect either collective or point anomalies. The differences between contextual, collective and point anomalies will be explained in the next chapter.

4. The multi-timeline detection system provides capabilities to discover anomalies caused by threats or accidents, while other studies can detect anomalies caused by threats only.

5. Our detection system does not need any modification of network system, while some others require protocol modification; some have to change network equipment or architecture.

6. The robustness of proposed system is one of the key issues in our study, especially from incorrect training data, and manipulation or poisoning by attackers. Other studies, however, have discussed robustness when attackers realize the detection technique in the target system.

Table 1.1 summarizes the main issues covered by our study, which are different from other studies of anomaly detection in network traffic.

Table 1.1: Different issues between our study and conventional techniques.

Issue	Our study	Conventional techniques
Proposal	Detection system	Algorithms or techniques
Scheme	Real-time	Offline or batch
Anomaly	Contextual and point	Collective or point
Caused by	Threats and outages	Only threats
Modification	None	Protocol or equipment
Flexibility	For various types of protocols	None
Robustness	From incorrect training data and imitation	None

1.4 Dissertation Outline

The remaining chapters of this dissertation are organized as follows:

- **Chapter 2** provides background knowledge on a variety of different anomalies in computer networks, and characterizes existing techniques for anomaly detection in network traffic.

- **Chapter 3** presents some general design considerations and introductory design guidelines for applying the multi-timeline detection technique to network systems.

- **Chapter 4** explains materials and methods for our experiments, including data preparation, data representation, preprocessing steps, learning algorithms, and evaluation matrices.

- **Chapter 5** describes data sources including normal traffic and anomaly traffic and explains how to create experimental network traffic in our study.

- **Chapter 6** shows the results of each individual experiment including comparison of different interval values, performance comparison between features, no packet situations, learning rates, time consumption, different volume of network traffic, time of anomaly occurrence.

- **Chapter 7** discusses capabilities to detect anomalies in computer networks by using the multi-timeline detection system with machine learning algorithms. We also describe the steps required to apply the proposed system to real network environments. We further point out the limitation of our detection system and suggest some possible solutions.

- **Chapter 8** concludes our study and gives some outlines of future work.

Chapter 2

Literature Review

This chapter first provides general types of anomalies and its nature, then explains the primary causes of anomalies in computer networks. Next, we present several detection techniques employed in the past studies, and point toward some advantages and disadvantages of these techniques. After that we introduce key concepts of machine learning for anomaly detection in general, and indicate why machine learning techniques gain more advantages over conventional techniques. Finally, We review representation of input data for anomaly detection by using machine learning algorithms.

2.1 Types of Anomalies

An important aspect of anomaly detection techniques is that we have to understand distinctive characteristics of target anomalies. Anomalies are generally classified into three main categories as follows [12]:

Point anomaly is an individual data instance considered as anomaly with respect of the remainder of data. This is the simplest type of anomaly, and many detection techniques of prior studies have been focusing on this type of anomaly. We show an example of point anomalies in Figure 2.1. In the figure, assume that we collected data with two features, x and y axes are represented as feature values of data points or feature vector. We clearly notice that the majority of data are distributed around on the top-right corner of this graph. The points p_1 and p_2 are located far away from the majority group. Therefore, these two points have been classified as point anomalies because they are different from normal or major data points.

As a real example, consider anomaly detection in network traffic, let the data set correspond to an individual packet that pass through a router. Let

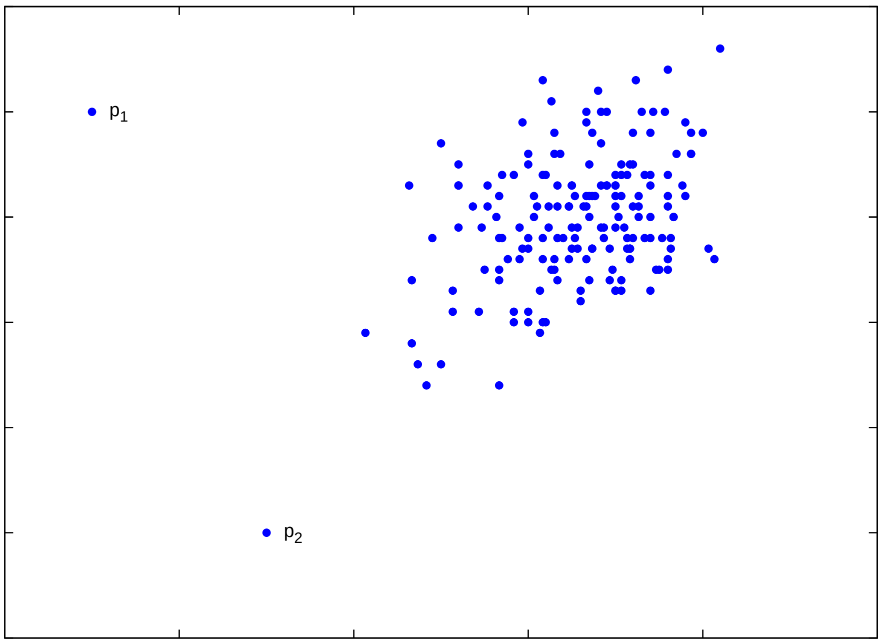

Figure 2.1: An example of point anomalies in a two-dimensional data set.

us assume that the data is defined using only packet size as a feature. A packet whose packet size is very large or very small compared to normal size of packets for the same protocol will be a point anomaly.

Contextual anomaly is a data instance considered as anomaly in a specific context but migh be normal in other contexts. We could refer to this type of anomaly as conditional anomalies [22]. This type of anomaly has been found on data related to position, location, and even time-series data like network traffic. Contextual anomalies have been commonly studied in time-series data and spatial data. Each data instance of a context is defined by the following two sets of attributes.

- Contextual attributes. The contextual attributes define the context (or neighbor) of that instance. For example, in time-series data of network traffic, time is a contextual attribute that determines the position of an instance on the entire sequence.
- Behavioral attributes. The behavioral attributes describe the noncontextual characteristics of an instance. For example, a spatial data shows the average traffic of the entire network; however, the volume of traffic at any location is a behavioral attribute.

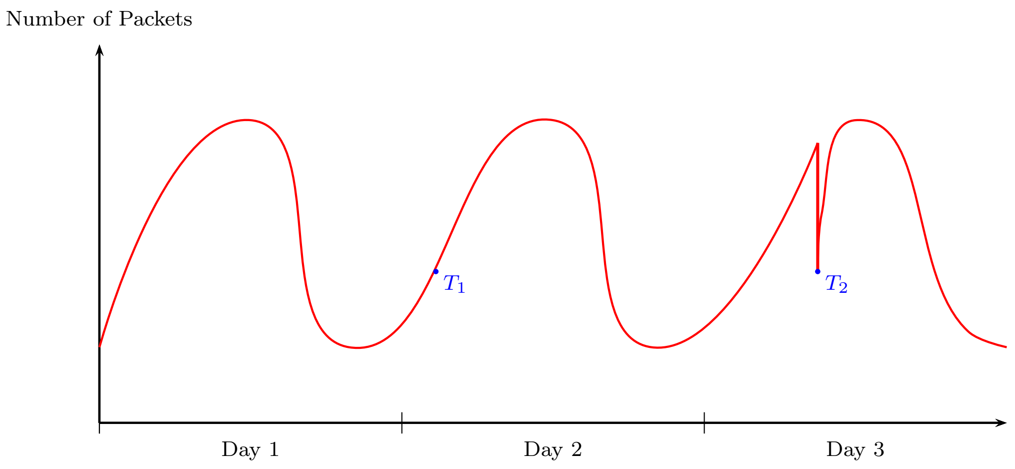

Figure 2.2: An example of contextual anomaly in network traffic.

We could categorize anomalies in network traffic as contextual anomalies because traffic behavior is different in regard to network locations and a particular time. Figure 2.2 shows one such example of contextual anomaly in network traffic, where x axis represents a time and y axis represents the number of packets. The lower number of packets might be normal during early morning at time T_1 but the same number of packets during noon at time T_2 on the following day would be an anomaly. In many cases of network traffic, defining a context is straightforward, and thus applying a contextual anomaly detection technique makes sense.

Collective anomaly is a collection of related data instances considered as anomaly in respect of the rest of data set. An individual data instance in a collective anomaly may not be anomaly by itself, but a collection of them is a collective anomaly. An example of collective anomalies in network traffic is as shown below:

...`http-web, buffer-overflow, ftp-login, smtp-mail, ssh, http-web, smtp-mail,` **ftp-login, ftp-login, buffer-overflow, ssh**`,` `smtp-mail`...

The highlighted series of packet (ftp-login, ftp-login, buffer-overflow, ssh) correspond to a denial of service attack (DoS) from a remote machine followed by connecting to the target computer via the secure shell protocol (ssh). It should be noted that this packet sequence is an anomaly, but the individual packets are normal packets when they occur in other sequences. Collective anomalies have been studied for several types of data, such as sequence data [23, 24], graph data [25], and spatial data [6].

Note that point anomalies can occur in any data set, while collective anomalies can occur only in data sets in which data instances are closely related to each other. Contextual anomalies, however, are subject to availability of contextual attributes in the data set. In addition, a point anomaly or a collective anomaly can be a contextual anomaly if analyzed in respect of a context. Therefore, we can transform a point anomaly or collective anomaly detection problem to a contextual anomaly detection problem by corresponding to the context information. In our study, we assume that all anomalies in network traffic are point and contextual anomalies, because the techniques used for detecting collective anomalies are very different than those two types of anomalies.

Many situations and behaviors of users have caused anomalies in network traffic, both directly and indirectly. Many anomalies result from human intention; however, some anomalies results from unintentional situations. Therefore, we can classify the primary causes of anomalies in network traffic under two groups, first group caused by threats and the other group caused by accidents.

Threats refer to any situation from human intention that potentially causes serious harm to a target network. Threats can include everything from viruses, worms, Trojans, back doors, and all types of attacks from malicious users including inside and outside users. For example, an intruder can use different scanning techniques to gain information about a target network, and try to break into the network. Network anomalies caused by threats have been studied more than a decade; however, many studies have been focusing on specific threats or specific network protocols. Unfortunately, they have to improve or modify these detection techniques all the time after discovering a new threat or releasing a new network protocols.

Accidents are unexpected and undesirable events, some anomalies are not harmful to network systems but most of them result in serious damage to network systems. Examples of network anomalies caused by accidents are outages, hardware failures, misconfigurations, and flash crowds. Although some of these incidents are not harmful, these are reflected in a lack of reliability, availability, or security in the network system. If these incidents occur frequently, they highly damage corporate image and reputation. Many studies separates detection techniques for anomalies caused by accidents from those for anomalies caused by threats. In this study, however, we intend to propose a system solution which has a capability to detect anomalies caused by both threats and accidents.

2.2 Anomaly Detection Techniques

In a few decades, researchers have proposed various methods from simple techniques to sophisticated ones for anomaly detection and intrusion detection. Instrusion detection refers to detection of malicious activities from threats by human intention [26], while anomaly detection refers to detection of anomalous activities from both threats and accidents. From these definition, intrusion detection is a bit different from anomaly detection; however, many researchers use these two terminologies interchangeably. Most of prior studies mainly focused on intrusion detection techniques, unfortunately they rarely try applying the techniques for anomalies caused by accidents.

The key characteristic of anomalies in computer networks is the huge volume of traffic. The detection techniques need to be computationally efficient to handle the large size of input data. Moreover, network traffic typically comes in a stream fashion and therefore detection techniques require online analysis rather than offline analysis. Another issue is that labeled data correspond to normal traffic is usually available, while labeled data for anomaly and intrusion are not avalable. All these issues cause anomaly detection in computer networks unique and quite different from those in other domains.

A study by Denning [27] classified detection systems into host-based and network-based detection systems. The host-based detection systems focus on anomalous behavior at particular machine, while network-based detection systems pay attention to deviant traffic over the network system.

2.2.1 Host-based Detection Systems

These detection systems deal with anomalies along traces at operating system level. The anomalies are in the form of unusual subsequences (collective anomalies) of the traces. Such unulual subsequences indicate malicious programs, unauthorized behavior and policy violations, for example. Although all traces contain events belonging to the same order, the co-occurrence of events is the key factor in discriminating between normal and anomaly behavior. Unfortunately, point anomaly detection techniques are not suitable in this domain. The techniques need to model the sequence data or compute similarity between sequences. A study by Snyder et al. [28] conducted a survey of different techniques used for this problem. Forrest et al. [23] and Dasgupta and Nino [29] revealed comparative evaluations of anomaly detection for host-based detection systems. Table II shows some other anomaly detection techniques used in this domain.

Table 2.1: Examples of anomaly detection techniques for host-based detection systems.

Detection technique	References
Statistical technique using histograms	Forrest et al. [30, 23], Gonzalez and Dasgupta [31], Dasgupta et al. [29, 32]
Mixture of models	Eskin [33]
Neural networks	Gosh et al. [34]
Support vector machines	Hu et al. [35], Heller et al. [36]
Rule-based systems	Lee et al. [37, 38, 39]

2.2.2 Network-based Detection Systems

These detection systems deal with anomalies in network traffic. The anomalies generally occur as abnormal patterns (point anomalies) among network data and occur as anomalous subsequences (collective anomalies) [40, 41]. Due to computer network connected to the rest of the world via the Internet, these anomalies mainly cause by outside attackers who intend to gain unauthorized access to the network for information theft or to attack the network. Available network data for detection systems can be at different levels of granularity, for example, packet level traces, flow level data, and so forth. The network data has a temporal aspect associated with it but most of detection techniques typically do not explicitly handle the sequential aspect. The network data also contain high dimensional with a mix of categories as well as continuous attributes. A challenge faced by anomaly detection techniques in this domain is that the nature of anomalies keeps changing over time as the intruders adapt their network attacks to evade the existing detection systems. Some anomaly detection techniques used in this domain are shown in Table 2.2.

Although network-based detection systems have been applied a broad range of detection techniques, according to survey researches [2, 11, 12], we can categorize anomaly detection techniques for network traffic into two major groups: signature-based and statistical-based methods.

Signature-based methods monitor and compare packets or traffic flows with predetermined attack patterns known as signatures. These techniques are simple and efficient to process data in computer networks, and achieve high accuracy with a low false detection rate. There are many commercial systems that conform to an ideal of signature-based methods, for example Snort [42, 43, 44], Suricata [43, 44], Bro [45], RealSecure, and Cisco Secure IDS. However, comparing a massive number of network packets or traffic flows with a large set of signatures is a time consuming task and it has limited predictive capabilities. One of the main disadvantages is that the signature-based methods cannot detect new or undefined attacks which are not included in signatures [46], so administrators have to frequently update signatures on the detection system. In addition, these techniques cannot detect anomalies caused by some internal operations, such as outages or misconfigurations, which are cannot defined as signatures.

Statistical-based methods [33, 47, 48, 49] can learn behavior of network traffic and possibly detect undiscovered anomalies and unusual incidents, especially ones caused by accidents. Many researchers have studied on particular techniques, for instance, the statistical profiling using histograms [50], parametric statistical modeling [40], non-parametric statistical modeling [51], a rule-based system [52], a clustering-based technique [53], and a spectral technique [54]. All these techniques are straightforward, but selecting appropriate parameters and threshold values for classification is still difficult, especially when network infrastructures have been changes. Another disadvantage of this technique is that some need a particular period of time for learning process before detecting anomalies in real environments.

Machine learning is one kind of the statistical-based techniques which has high capabilities to automatically recognize complex patterns, and make intelligent decisions on the basis of data [14]. There are two fundamental types of algorithms in machine learning: the unsupervised algorithm and supervised algorithm [15].

The unsupervised algorithm is a machine learning technique that takes a set of unlabeled data as input and cluster data. We could detect anomalies on the basis of the assumption that major groups are normal traffic and minor groups are anomalous traffic [20]. Unfortunately, many cases are not true in a certain period, such as distributed denial of service attacks (DDoS), viruses or worms spreading, and flash crowds. From these examples, the amount of anomalous traffic is normally larger than those of normal traffic. In other cases, outages and misconfigurations for example, although no

Table 2.2: Examples of anomaly detection techniques for network-based detection systems.

Detection technique	References
Statistical technique using histograms	NIDES Anderson et al. [55, 56], EMERALD Porras and Neumann [57], Yamanishi et al. [58, 50]
Parametric statistical models	Gwadera et al. [41, 40], Tandon and Chan [59]
Nonparametric statcistical models	Chow and Yeung [51]
Bayesian networks	Siaterlis and Maglaris [60], Sebyala et al. [61]
Neural networks	HIDE Zhang et al. [62], NSOM Labib and Vemuri [63]
Support vector machines	Eskin et al. [33]
Rule-based systems	ADAM Barbara et al. [52, 64, 17], Qin and Hwang [65]
Clustering based	ADMIT Sequeira and Zaki [53], Otey et al. [66]
Nearest neighbor based	MINDS Ertoz et al. [67]
Spectral	Lakhina et al. [68], Thottan and Ji [54], Sun et al. [69]
Information theoretic	Lee and Xiang [70], Noble and Cook [25]

anomalous packet occurs, an unusual decline in normal traffic also indicates an unexpected incident arising. Therefore, the unsupervised algorithm as a clustering technique is not suitable for these types of anomalies.

In contrast to the unsupervised algorithm, the supervised algorithm can cover and detect a wide range of network anomalies [16]. The basic assumption of supervised algorithm is that the anomalous traffic is statistically different from normal traffic. Many studies have been applied several algorithms based upon this assumption, such as the Bayesian network algorithm [17], the k-nearest neighbor algorithm [18], the support vector machine algorithm [19]. Nevertheless, the performance of these algorithms for real-time detection has not been compared with the same data set.

Many previous studies of supervised algorithms used packet-based or connection-based features, which have a scalability problem when the number of packets or connections increases. However, the interval-based features can possibly solve this problem [71]. For example, suppose we have network traffic including 10 packets for 10 seconds, if we apply packet-based features and the processing time for 1 packet is 1 unit, the processing time of packet-based features will be 10 units. When the number of packets increases to

1,000 packets for 10 seconds, the processing time also rises to 1,000 units as well. However, if we apply interval-based features and the processing time for 1 second is 1 unit, the processing time of interval-based features are only 10 units, regardless of the number of packets.

Another problem with the packet-based or connection-based features is that, the same as the unsupervised algorithm, they cannot detect some incidents. Although the packet-based features can distinguish between normal packets and anomalous packets, they cannot detect an unexpected incident that does not have any anomalous packet, such as outages and misconfigurations. While the interval-based features have been shown to be able to detect unusual incidents that do not have anomalous packets [72]. The question remains whether interval-based features are suitable for each particular type of anomalies. Thus, in this study, we also investigated which interval-based features are practical for particular types of anomalies.

2.3 Fundamental of Machine Learning for Anomaly Detection

To illustrate the basic concept of machine learning in which the anomaly detection has been involved, let us first consider a simplified example. Suppose that a network operator wants to automate the process of detecting a packet by using two features: source port and destination port number. Now we represent these two features for detecting anomalies in the test packet, x_1 for the source port and x_2 for the destination port. If we ignore how these features might be measured in practice, we realize that the feature extractor has thus reduced characteristics of the test packet to a data point or feature vector $\mathbf{x}$ in a two-dimensional feature space, where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{2.1}$$

The first step is to plot all measurements on a two-dimensional feature space as shown in Figure 2.3, where the horizontal axis represents the source port x_1 and the vertical axis represents destination port x_2. For the next step, an algorithm learns from all of the data points to locate the likelihood of normal packets, and then forms the decision boundary between normal and abnormal region. The decision boundary, the dash line in Figure 2.3 for example, can represented by a decision function to discriminate between two classes. The decision function is a mathematical function which takes a dataset as input and gives a decision (a test data point) as output. For

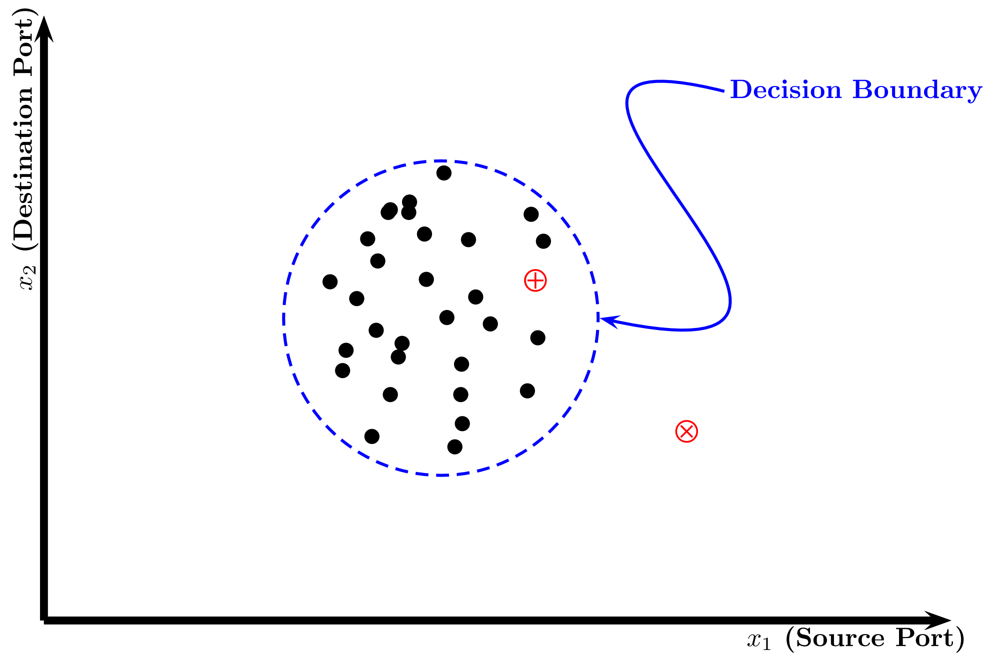

Figure 2.3: Feature space of network packet.

instance, if a test data point falls into the normal region, $\oplus$ mark for example, we classify the test data as normal class. On the other hand, if a test data point falls into the outside of normal region $\otimes$ mark for example, we classify it as anomaly class.

All above procedures are the simple version with two features to detect anomaly in the network packet. For the machine learning, we generally add more features up to n features, and then the algorithm can learn from prior data with all provided features. However, we define the probabilistic structure of data for the learning algorithm as representation of input data. There is a wider perspective of data representation in different problem domains, especially in network traffic. The representation of input data is one of the significant effects on detection performance beside the features and learning algorithms.

In the following section, we review existing representation of input data for anomaly detection in network traffic and point out important issues of these representation. Note that we mainly focus on the learning and detecting processes rather than post-processing after the anomalies have been detected. To simplify the explanation of the data representation in the following section, we assume that a data instance occurs in every interval on a timeline for one day. An instance at time $t = x$ is represented by $\mathbf{x}$, and when we consider more than one timeline, we represent the instance $\mathbf{x}$ on the present timeline p with $\mathbf{x}^p$, on the timeline number 1 with $\mathbf{x}^1$ and so on. The ultimate goal of anomaly detection in network traffic is to specify a decision function $g(\mathbf{x})$ that can classify a test instance $\mathbf{x}$ into either the normal class or anomaly class. Therefore, we can define the task of anomaly detection as a binary classification problem [73], and we can evaluate performance of $g(\mathbf{x})$ with a measurement for binary classification problems.

2.4 Representation of Input Data

From previous studies, we have categorized detection techniques based on how to create a decision boundary between normal and anomaly traffic. We simply grouped representation of input data from previous work into three categories as follows:

2.4.1 Manual-based Representation

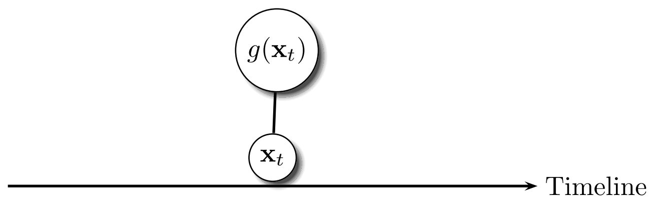

Figure 2.4: Manual-based representation.

The first is the simplest and most straightforward representation of input data to classify data. As shown in Figure 2.4, we have a single timeline, which flows from the left to the right side. Suppose that an only one instance $\mathbf{x}$ occurs on the timeline at time t represented by $\mathbf{x}_t$, and we intend to classify the instance $\mathbf{x}_t$ under either normal class or anomaly class. We input the information of $\mathbf{x}_t$ into a decision function $g(\mathbf{x}_t)$ to perform a task of classification. The figure depicts a detecting connection by a bold line between the instance $\mathbf{x}_t$ and decision function $g(\mathbf{x}_t)$. The question is how to define such the decision function $g(\mathbf{x}_t)$.

A simple way to create the decision function $g(\mathbf{x}_t)$ is to let the function be manually specified by anomaly experts, or define by

$$g(\mathbf{x}_t|\text{expertise information}). \tag{2.2}$$

We name this representation of input data as "manual-based representation". The expert could define the decision function for normal class, instances conform to defined patterns are classified as normal class, firewall systems have such function for example. The expert could define the decision function for anomaly class as well, instances conform to defined patters are classified as anomaly class, for instance, anti-virus software, signature-based intrusion detection and firewall contain such function. In addition, the decision function could be specified by both normal and anomaly class function. Many commercial products behave like this representation including Snort [42], Bro [45], NetSTAT [74], RealSecure, and Cisco Secure IDS.

The advantages of this data representation are simple, straightforward, and it could immediately detect anomalies after installation. Detection performance of this representation depends on the defined function by expert, so network administrators have to keep the function up to date. Most of existing systems with this data representation are great performance; however, this representation is not a flexible solution, and it is quite difficult to detect novel and variation of anomalies that are not defined in the decision function.

2.4.2 Batch Representation

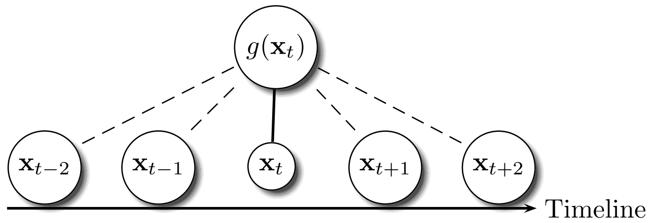

Figure 2.5: Batch representation.

For this representation, we suppose that five instances, $\mathbf{x}_{t-2}, ..., \mathbf{x}_{t+2}$ occur on a single timeline from $t-2$ to $t+2$ sequentially, as shown in Figure 2.4.2. We intend to classify a test instance whether it is an anomaly instance or not by using the decision function, the test instance is $\mathbf{x}_t$ for example. The decision function $g(\mathbf{x}_t)$ can use information from the rest of instances, $\mathbf{x}_{t-2}$, $\mathbf{x}_{t-1}$, $\mathbf{x}_{t+1}$, and $\mathbf{x}_{t+2}$ (it may include the test instance $\mathbf{x}_t$ as well), or define by

$$g(\mathbf{x}_t|\mathbf{x}_{t-2}, \mathbf{x}_{t-1}, \mathbf{x}_{t+1}, \mathbf{x}_{t+2}), \tag{2.3}$$

Here $\mathbf{x}_{t-2}$ to $\mathbf{x}_{t+2}$ are feature vectors from $t-2$ to $t+2$ during the day. As shown in the figure, we depict the bold line to represent a test connection between the test instance $\mathbf{x}_t$ and the decision function. We also draw the dash lines to represent learning connection between the rest of instances and the decision function. In this figure, we assume that the decision function does not have learning connection with the test instance $\mathbf{x}_t$.

To automatically generate the decision function $g(\mathbf{x}_t)$, some studies have proposed what we call the "batch representation". Generally, a learning algorithm of this representation generates the decision function $g(\mathbf{x}_t)$ by learning from other instances rather than the test instance, for example from $\mathbf{x}_{t-2}, \mathbf{x}_{t-1}, \mathbf{x}_{t+1}$, and $\mathbf{x}_{t+2}$. Examples of studies conformed to this representation are [75] in which the algorithm learns from an entire data set, and [76] in which the algorithm requires a certain amount of data before detecting an individual or group of test instances. The previous studies also show that plenty of learning algorithms have been applied to this representation; those algorithms are simply classified as clustering and classification techniques.

One of the advantages of this representation is that we can apply a various of learning algorithms from simple to sophisticated one. This representation, however, is highly suitable for offline mode or detecting at the end of the day rather than online mode or real-time anomaly detection. The main reason is that this representation requires entire or a certain amount of data including instances occur after the test instance to generate the decision function and use this function to detect anomalies. In real-time anomaly detection, we cannot acquire information about instances after the test instance, and we only have learning connection between instances before the test instance.

2.4.3 Real-time Representation

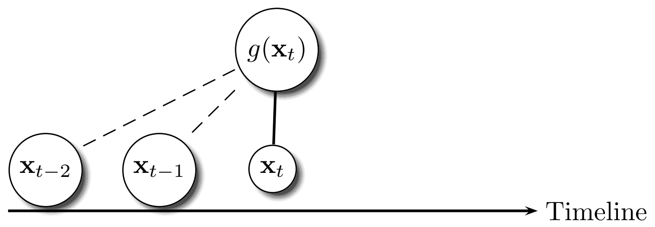

Figure 2.6: Real-time representation.

Due to the intrinsic characteristic of the batch representation explained in the previous subsection, we get rid of the connection between instances after the test instance, then we obtain a new input data named "real-time representation". As shown in Figure 2.6, the bold line represents a test connection between the test instance $\mathbf{x}_t$ and the decision function $g(\mathbf{x}_t)$, define by

$$g(\mathbf{x}_t | \mathbf{x}_{t-2}, \mathbf{x}_{t-1}), \tag{2.4}$$

Here $\mathbf{x}_{t-2}$ and $\mathbf{x}_{t-1}$ are feature vectors that occur before the test interval, and the decision function only has learning connection to instances before the test instance. Note that the decision function can learn from the information of test instance, but in this figure we assume that it does not. Obviously, this data representation is more suitable for real-time anomaly detection than the batch representation.

The decision function in this data representation still automatically generated by learning algorithm. An example of real-time representation is the Next-Generation Intrusion Detection Expert System (NIDES) [77, 56]. The NIDES was one of the few intrusion detection systems of its generation that could operate in real time for continuous monitoring of user activity or could run in an offline mode for periodic analysis. The other interesting example is the real-time intrusion detection for ad hoc networks (RIDAN) system [78], which combined real-time with manual-based representation.

This data representation has been used by real-time detection systems in many other areas. However, there are three key issues under our consideration for applying the real-time representation to computer networks. The first issue is that the data from this representation are easily imitated or manipulated by attackers to evade or to compromise detection systems, because prior data have a strong influence on the decision function of the test instance. The second issue is how to classify the present instance when there is no prior data, instance. The last issue is what amount of prior data is sufficient to generate a trustworthy classification function.

Table 2.3: Comparison of representation of input data in network anomaly detection.

Criteria	Manual	Batch	Real-time
Automation	No	Yes	Yes
Real-time detection	Yes	No	Yes
Flexibility	No	Yes	Yes
Robustness	No	No	No

Table 2.3 shows a comparison of representation of input data for anomaly detection in network traffic. Automation denotes that system do not have to change configuration or add information into a signature database after discovering new type of anomaly, but manual representation is a non-automation system. Real-time indicates that system can detect anomalies less than a minute after anomaly occur. Batch representation cannot detect in real time because the system needs to process whole data at the end of day. Robustness means that attackers hardly manipulate or evade the sys-

tem. For all three representation, attacker can easily manipulate or evade the system if they know detection technique employed in the system.

Chapter 3

Proposed System Design

This chapter introduces the multi-timeline representation and provides general design guidelines for applying it to network systems. The specific design for our experements, however, will be explained in the next chapter. For general design, we consider anomaly detection system as both microscopic design and macroscopic design. Here both designs work together, and the microscopic design has been included as a part of macroscopic design. In Chapter 3.2, we explain our microscopic design named detector module, how it works and describe a major role of multi-timeline representation of input data in this module. We go into detail of anomaly detection device as macroscopic design in Chapter 3.3 and explain how to combine different detector modules for concurrent operation. In the last section, we discuss issues of system design and point out some design considerations.

3.1 Multi-timeline Representation of Input Data

The important issues of the real-time representation of input data, which explained in the previous chapter, suggest that the detection system is much easy to be compromised by attackers or intruders. On the other hand, the attackers who know the mechanism of detection system can easily conceal their attacks or evade monitoring systems. To solve these issues, we propose a representation as shown in Figure 3.1. We use information on network traffic from past timelines as learning data for the decision function. This technique hardens the detection system and it is quite difficult for attackers to conceal their attacks or evade monitoring systems. However, we should select past timelines that correlate with the present or detecting timline. For example, we should use weekdays timelines only as learning data to detect anomalies on weekdays, because behavior of weekdays traffic is different from those of weekends traffic.

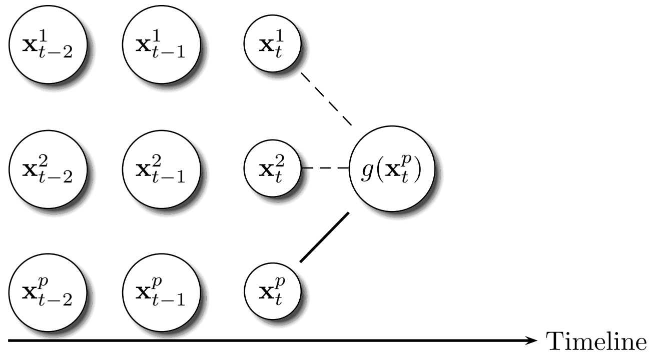

Figure 3.1: Multi-timeline representation of input data.

The multi-timeline representation as shown in Figure 3.1 is different from the single timeline. Suppose we intend to classify the test instance $\mathbf{x}^p_t$ on the present timeline and have three timelines, 1, 2, and p with the same length for representing day one, day two, and the present day. We also depict intervals on day one at time $t-2,...,t$ with $\mathbf{x}^1_{t-2}$, $\mathbf{x}^1_{t-1}$,and $\mathbf{x}^1_t$, and so forth on others timelines. The algorithm uses multiple timeline from the past timelines as input at the same interval of test instance $\mathbf{x}^p_t$. The learning algorithm also generates the decision function $g(\mathbf{x}^p_t)$ by learning from the same interval as the test instance. In this figure, for the test instance $\mathbf{x}^p_t$, we show a detecting connection has been represented by the bold line, and learning connection by dash lines. The algorithm generates the decision function for $\mathbf{x}^p_t$ by learning from time intervals t from day one and day two, which are represented by $\mathbf{x}^1_t$ and $\mathbf{x}^2_t$, respectively. The number of past timelines for decision functions depends on the number of historical traffic stored in the system, but it affects processing time of learning process. We will show empirical results for the suitable number of past timelines and effect on prossessing time later.

We can present a difference of decision function between single timeline and multi-timeline representation as follows. The decision function for single timeline is

$$g(\mathbf{x}^p_t|\mathbf{x}^p_1, \mathbf{x}^p_2, ..., \mathbf{x}^p_{t-1}), \tag{3.1}$$

and decision function for multi-timeline is

$$g(\mathbf{x}^p_t|\mathbf{x}^1_t, \mathbf{x}^2_t, ..., \mathbf{x}^{p-1}_t). \tag{3.2}$$

Here, for single timeline, the decision function $g(\mathbf{x}^p_t)$ contains feature vectors $\mathbf{x}^p_1, \mathbf{x}^p_2, ..., \mathbf{x}^p_{t-1}$ as input (all feature vectors before the test interval over the present timeline p). For multi-timeline, $g(\mathbf{x}^p_t)$ is decesion function for the test feature vector x at interval t over the present timeline p, and contains feature vector $\mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}$ as input (feature vectors at the same interval t over past timelines). We assume that input vectors of these two decision functions are attack-free or almost attack-free, because detection performance of these two functions mainly rely upon input vectors. If input vectors contain more attacks, detection performance of decision functions would be decreased by attack input.

The number of decision functions $g(\mathbf{x}_t)$ along a timeline is a major difference between the real-time representation and multi-timeline representation. For real-time representation, the learning algorithm generates only one decision function along a timeline. This decision function could be an updatable function on account of network traffic occurred before the test data. As a result, one decision function produces a vulnerable point that attackers can easily conceal attack traffic or manipulate the detection system for evasion. For multi-timeline representation, the number of decision functions is exactly equal to the number of intervals for a single timeline. Each interval contain an independent decision function from other interval. Therefore, the multi-timeline representation is quite difficult for attackers to conceal attack traffic or manipulate the detection system.

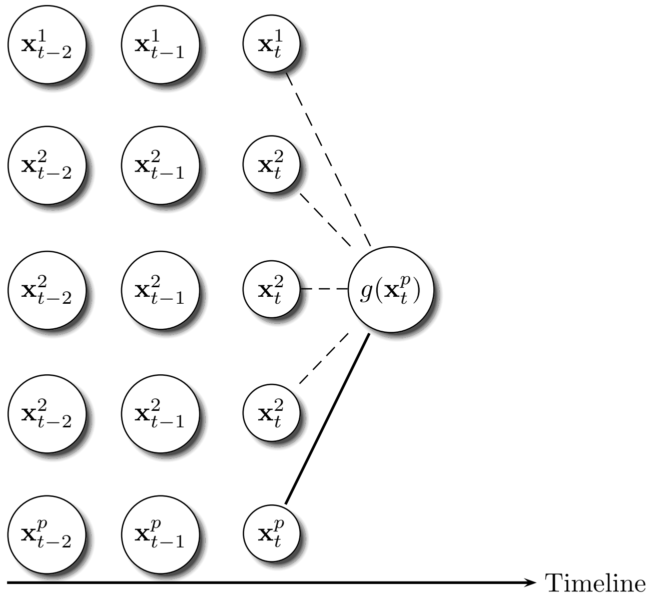

Figure 3.2: Multi-timeline representation after applying a weight value 3 to the timeline 2.

Weighting techniques could be applied to the multi-timeline representation. Weighting is a technique for telling learning algorithm which data is more significant than others, so that the algorithm should considerably employ them to generate the decision function. Suppose we have three timelines as shown in Figure 3.1, where timeline 1, 2, and p represent day one, day two, and the present day sequentially. A simple way to weight a new timeline, timeline 2, over an old timeline, timeline 1, is replicating the timeline 2 as much as the number of weight. If a weight value 3 has been applied to timeline 2, then we replicate the timeline 2 to 3 timelines as shown in Figure 3.2. As a result, the timeline 2 has an influence 3 times on the decision function $g(\mathbf{x}_t)$ more than the timeline 1 which contains only a single timeline. From this figure, we can define the decision function as

$$g(\mathbf{x}_t^p|\mathbf{x}_t^1, \mathbf{x}_t^2, \mathbf{x}_t^2, \mathbf{x}_t^2). \tag{3.3}$$

Conversely, if we concern about an old timeline which far from the present timeline, we can weight timeline 1 over timeline 2 as the same fasion explained.

Another techniques that could be applied to the multi-timeline representation is the Synthetic Minority Over-sampling Technique or SMOTE [79]. This technique applies a combination of over-sampling the normal class and under-sampling the normal class and can achieve better classifier performance. The SMOTE approach has been proposed for classifying imbalanced datasets, which contains normal examples with only a small percentage of abnormal examples. Nearly all datasets of network traffic are imbalanced data, so that we could apply this approach to our milti-timeline representation as well.

Table 3.1: Comparison of conventional representation to multi-timeline representation of input data.

Criteria	Manual	Batch	Real-time	Multi-timeline
Automation	No	Yes	Yes	Yes
Real-time detection	Yes	No	Yes	Yes
Flexibility	No	Yes	Yes	Yes
Robustness	No	No	No	Yes

Table 3.1 shows a comparison of representation of input data for anomaly detection in network traffic. For automation, we measure whether the detection system need to change configuration or not. For real-time detection, we measure that the detection system can alert after anomalies occur less than a minute. For flexibility, we observe that we can apply more than one algorithms to detection system without changing main configuration. For

robustness, we measure that variation of detection performance of system less than 3% when 10% of anomaly traffic be contaminated in training data.

Here, we explain an example of the robustness criterion. Equation 3.1 represents a decision function for single timeline. Assume that we use the number of packets as a single feature for detection system and attackers contaminate all packets before test data at time t as

$$g(\mathbf{x}_t|100, 100, ..., 100), \tag{3.4}$$

where 100s are the number of contaminated packets from attackers, while the number of packets in normal traffic is 10 packets, for example. In this case, if test data $\mathbf{x}_t = 100$, the system detect test data as normal instead of anomalous traffic. However, the decision function for multi-timeline as Eq.3.2 did not use all packets before test data on the present timeline. Therefore, contaminated packets have no effect on decision function for multi-timeline.

Another example of robustness, assume that an outage occur and inbound input contains no packet at all. We can rewrite Eq.3.1 as

$$g(\mathbf{x}_t|0, 0, ..., 0), \tag{3.5}$$

where 0s are the number of packets from inbound input, while the number of packets in normal traffic is 10 packets. In this case, if test data $\mathbf{x}_t = 0$, the system also detect test data as normal instead of anomalous traffic. However, the decision function for multi-timeline as Eq.3.2 did not use all packets before test data on the present timeline. Therefore, no packet on present timeline have no effect on decision function for multi-timeline.

Changing the representation from horizontal to vertical, as the multi-timeline representation does, gains several advantages. The first advantage is that it is quite difficult for attackers to imitate or manipulate data to evade detection system when we set the distance between timelines to be sufficiently long. In Figure 3.1 for example, assume the length of every timeline is one day long, such that the distance between $\mathbf{x}_t^p$ and $\mathbf{x}_t^2$ is one day and the distance between $\mathbf{x}_t^p$ and $\mathbf{x}_t^1$ is two days. It seems impossible for attackers to go back to the past and manipulate data from yesterday or two days ago. The second advantage is that even if attackers imitate data on the present timeline, this action has no influence on the decision function for the following instances. In Figure 3.1 for example, attackers may create imitated instance $\mathbf{x}_{t-2}^p$ and $\mathbf{x}_{t-1}^p$ like normal instances, but it does not influence the decision function for instance $\mathbf{x}_t^p$, because $g(\mathbf{x}_t^p)$ is generated from $\mathbf{x}_t^1$ and $\mathbf{x}_t^2$, which occur at the same time t on past timelines. The last advantage is that our analysis suggests that the multi-timeline representation (with some learning algorithms) highly tolerates incorrect training data, because there is a very low possibility that incorrect data can occur at the same time on different timelines. Therefore, we have conduct a series of experiments to examine these key aspects and other aspects of the multi-timeline representation.

Time complexity of multi-timeline is also change from single timeline, especially for training process. Time complexity of single timeline depends on the number of intervals and number of timeline as $O(mn)$, where m the number of training data (days) and n is the number of interval. On the contrary, Time complexity of multi-timeline depends on the number of timeline only $O(m)$.

Representation of input data plays a crucial part in our design for anomaly detection because the classifier in this system heavily relies upon the representation of input data. The input data represents data of network traffic in order to generate an effective classifier from historical or training data. It means that the representation of input data is mainly associated with the learning process rather than detecting process. In Chapter 2, we have shown three conventional representation of input data and proposed a representation named multi-timeline to enhance capabilities of detection system.

We provide a short summary of three conventional and one proposed representation of input data in Chapter 2 as follows:

- Manual-based representation depends entirely on knowledge of technical experts in network traffic anomaly. Although this system could perform detection in real time, it cannot automatically discover new types of anomalies.

- Batch representation have been developed to automatically detect a new type of anomaly. This system employs batch processing techniques, so it requires entire data and causes many problems to apply this representation of input data for real-time detection.

- Real-time representation has the ability to automatically detect a new type of anomaly and can be applied for real-time detection. However, the main problem of this representation is that attackers can easily evade the detection system or imitate normal traffic if they realize the detection mechanism.

- Multi-timeline representation is our proposed input data to resolve the main problem of real-time representation. This representation keeps capabilities to automatically discover new anomalies and can operate in real time. Moreover, it is a robust input data from unlabeled training data so we do not require any preprocessing of training data.

These input data are representation of network traffic, so they cannot process data by themselves. Therefore, we require a learning algorithm utilizing the representation of input data to produce a classifier in training process and to classify test data in detecting process. We design the input data in general so that one can pick any machine learning algorithm to work with multi-timeline representation. In our design, a key role of learning algorithm is making connections among training data, test data, representation of input data, and classifier. Additionally, we provide some learning algorithms for our experiments in Chapter 4.

Note that closer to our approach, many studies have proposed a technique known as the multivariate technique. For example, a study by Lakhina et al. [80] shows that the principal component analysis (PCA) can be used for separating the high-dimentional space from different network traffict measurements into subspaces, after that they can perform anomaly detection from these disjoint subspaces. Another study by Nychis et al. [81] proposed an entropy-based technique for anomaly detection in network traffic. The main idea of this approach is that they analyze the power of multiple traffic distributions, including the number of addresses, ports, and flow sizes, then they use the entropy of these distributions in time series to detect anomalies in network traffic. The last example has been studied by Kanda et al.[82]. They proposed a combination of sketchs, PCA, and entropy-based technique to detect anomalies in time series of network traffic. These techniques as the multivariate technique employ multiple values or multiple features of time series and compare a test data to other time series. The multivariate technique is still based on batch representation because they still need entire network traffic for a certain period. The multi-timeline representation, however, compares a test data to historic network traffic rather than the current one, and do not need entire data for detection.

3.2 Detector Module

We name the smallest part of our system as detector module. A detector module consists of four primary functions, namely feature extraction, feature scaling, weighting process and classifier as shown in Figure 3.3. The module has two outside connections, one is between network traffic and feature extraction as an input and another is between classifier and alarm system as an output. We can divide data flows inside a detector module into two flows, one flow for learning process and another flow for detecting process. Feature extraction and feature scaling are common functions, but representation of input data and classifier differently operate for learning and detecting process. We also apply the multi-timeline detection system to take the major role of representation of input data, it is a crucial part to generate an efficient classifier for anomaly detection. To understand functionality of detector module, we explain each function in the following subsections.

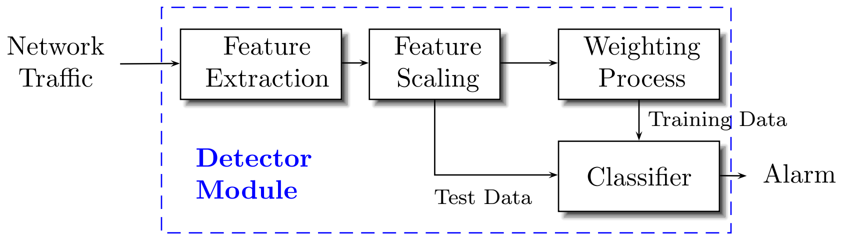

Figure 3.3: Process connections and data flows of detector module.

3.2.1 Feature Extraction

The main function of feature extraction is selecting important features of network traffic. Extracted features should directly or indirectly associate with expected anomalies. In our experiments, we can extract feature of network traffic by using information on packet header and aggregate packets or flows on interval basis. Table 3.2 shows fundamental features of network traffic by aggregating information of packet header. However, we can create variations of these features by combing more features together, for example, the number packets per flow over an interval could be derived from ratio between the Packet and Flow feature. Moreover, we also can extract more specific protocols or ports based on these features, such as the number packets of HTTP port 80 or the number of flows of ICMP, and so forth.

Table 3.2: Fundamental features of network traffic by aggregating information of packet header.

Feature	Description
Packet	Number of packets
Byte	Sum of packet sizes
Flow	Number of flows
SrcAddr	Number of source addresses
DstAddr	Number of destination addresses
SrcPort	Number of source ports
DstPort	Number of destination ports

On the contrary, feature representation in nominal value, such as binary values of 0 and 1, is another way for feature extraction rather than aggregating information of packet header. For example, we indicate that IP addresses or port numbers appear on an interval with 1 and does not appear with 0, so the classifier should consider IP addresses or port numbers which does not appear before as anomalies. Therefore, output from feature extraction could be mixed between features with real value and features with nominal value. Our experiments apply only the aggregation technique so all features are represented in real values.

One of the feature extraction parameters is interval value δ that affects detection performance, resources, and time consumption of multi-timeline system. If we set a short interval value, the system could detect short time anomalies; however, we have to reserve more storage for each interval information and take processing unit more often. Conversely, if we set a long interval value, we could reduce storage for each interval information; however, the system hardly detect a short time anomaly. The interval value also has an effect on time consumption, because it depends on the number of packets in each interval. If we set a short interval value, the number of packet in each interval less than those of a long interval value. Moreover, the time between anomaly occur and alert rely on time interval and processing time after that as

$$T(\mathbf{x}) = \delta + d(\mathbf{x}), \tag{3.6}$$

where $T(\mathbf{x})$ is total processing time when an anomaly occur at interval x, δ is an interval value, and $d(\mathbf{x})$ is detecting time processed at interval x.

3.2.2 Feature Scaling

Feature scaling is a process attempts to standardize a wide range of feature values to the same range. This process may be unnecessary for other techniques for anomaly detection in network traffic; however, feature scaling or feature normalization is indispensable for the multi-timeline detection system that relies mainly upon machine learning algorithms. Most learning algorithms will not function properly without feature scaling.

Even though many scaling functions have been proposed [83], the common goal of scaling functions is to independently normalize each feature component to the [0,1] or [-1,1] range. We highly recommend the following two scaling functions that would be appropriate for real-time anomaly detection. We could normalize feature values by using the max value as

$$x' = \frac{x}{\max(x)}, \tag{3.7}$$

or by using the range of feature as

$$x' = \frac{x - \mu}{\max(x) - \min(x)}, \tag{3.8}$$

where x' is the normalized value derived from x an original value, μ is the average value, $\max(x)$ and $\min(x)$ are maximum and minimum values of x respectively. These two scaling functions have the same time complexity as $O(q)$ where q is the number of feature values, so these would take a very short time of computation for real-time systems.

3.2.3 Weighting Process

Weighting is a mandatory process for the multi-timeline detection module, especially during the learning process. This module guides the learning algorithm to generate a decision function which relies on particular timelines. Consequently, network operators could weight on one or more timelines to bias the decision function of detection system. For example, recent timelines should have a strong influence on the classifier than other older timelines. There are various weighting techniques for different purposes that could be plug into this module, such as by performing a sum, integral, average or even calculus [84]. In our experiments, we adapted a gradual weighting function to recent timelines and we will describe more details of our weighting function in Chapter 4, System Implementation.

3.2.4 Classifier

The major role of classifier is distinguishing between normal and anomaly in network traffic. The classifier in detector module is created by an algorithm using representation of input data and training data during learning process. As a result, effectiveness of classifier depends highly upon three factors: learning algorithm, representation of input data, and training set. In detecting process, the classifier receives a test data as an input, then produces a label of test data as an output to alarm system. One of our purposes of multi-timeline learning is to enhance several capabilities of detector module. In addition, the detector module could be applied in various schemes to detect different types of anomalies in network systems.

3.3 Homogeneous Detector Device

An anomaly detection device contains a number of detector modules instead of a single module. Though a single detector module could discover a new type of anomaly, it is better to consolidate two or more detector modules for anomaly detection.

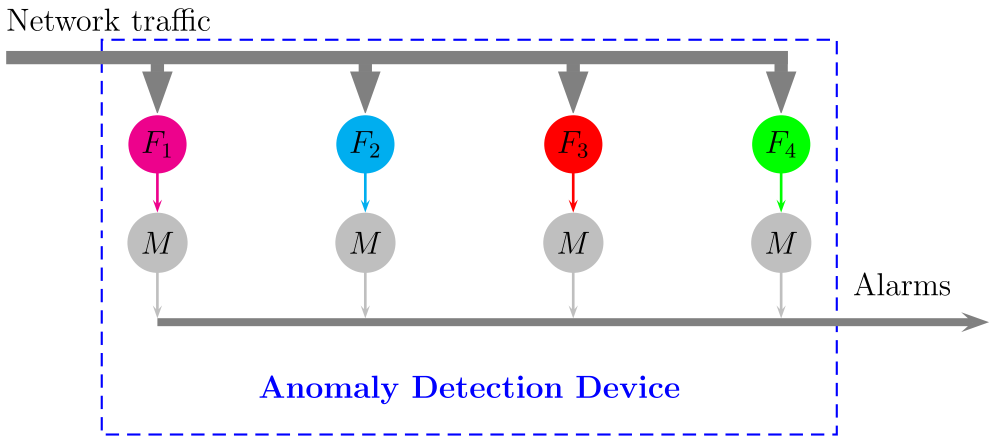

Figure 3.4: Example of anomaly detection device using homogeneous detector modules.

An anomaly detection device using homogeneous detector module contains exactly the same type of detector modules as shown in Figure 3.4. The same type of detector module as M in the figure could be the same learning algorithm, representation of input data, and feature space. The main input of this device is all network traffic through a certain point, such as a router or switch in the network. Additional parts placed between main network traffic and detector modules are traffic filters, for example F_1-F_4 in the figure. The main function of these filters is filtering network traffic based on many types of criteria, such as network protocol, port, IP address, MAC address and so on. Each detector module will learn and create an own classifier based on each criteria of the related filter. It is somewhat similar to home security systems that we install a fire detector for each room in a house. In the case of anomaly, each detector module could raise an alarm signal with detector ID, so that operators could be notified and inspect traffic relating to such anomaly.

The positive aspect of homogeneous detector module is that we could filter only essential network traffic and do not require all network traffic for monitoring. Moreover, the device is fairly adaptable and flexible because network operators can adjust the device via filters alone. However, the negative aspect of homogeneous detector module is that we require traffic filters and the device has to pay the extra cost for these additional parts. Another drawback of multiple detector is how to judge when the decisions of multiple detectors are different. Our results show that each type of attack contains it own characteristics and alarms by using different sets of features, Back attack by using f_1 and f_2, PortSweep attack by using f_7 for example. Therefore, we might keep these records to identify types of attacks later when multiple detectors have been alerted.

In our design, the majority vote [85] could be applied in the system; however, if each module takes care of one feature the majority vote will fail for detecting a specific anomaly. In this case, one module only answers correctly and the rest of them fails. However, this technique could be useful for traceback the original of anomalies by deploying multi-timeline detection devices in multiple networks. We can use the majority vote from the multiple networks to find the probability of attack path when devices alert with the same set of features, similar to the conventional IP traceback technique [86].

3.4 Design Consideration

This final section provides a step-by-step guide to apply our design for a real network environment, and gives some considerations for each step. Although we start from a microscopic design then expand to a macroscopic design, applying our detection devices for real network environment we have to completely reverse these all processes. The following steps indicate standard procedures for employing our designs in real network systems.

First, designers need to contemplate where the locations are necessary for network operators to carefully monitor traffic. Global and local location strategies should be applied in this first step. For global location strategy, ones might deploy a couple of detection devices near a gateway to the Internet, but the main concern is that a huge volume of network traffic could slow detection devices down. As a result, detection devices should be hardware-based devices rather than software-based devices. However, ones would apply software-based devices if they do not require real-time anomaly detection. For local location strategy, ones have to decide which particular areas, servers, or workstations are important for monitoring. Even though software-based devices could be applied, the number of devices results in the total cost of detection system. Like home security systems, we choose precise locations in this step before installing security cameras or several types of detectors.

The following step is detection device design. Designers need to individually design detector modules for each detection device or the same type of partial areas in network system. For example, ones would apply homogeneous detector modules to detection device connected with the main router and monitor only TCP, UDP, and ICMP. Ones could apply hybrid detector modules to a software-based detection device installed at a workstation or server. The main consideration in this step is that detector modules could be altered when system requirements have been changed. For example, at the beginning detector modules monitor the entire network traffic, then they would be altered to monitor TCP, UDP, and ICMP. Therefore, we proposed anomaly detection device that is ready to get detector modules altered. Similar to home security systems, we assign various types of security cameras or detectors to suit each designed location in this step.

Finally, we need to select features, learning algorithm, representation of input data, and other parameters for each detector module. We personally recommend that the multi-timeline representation is highly suitable for real-time detection device. In the next chapter, however, we examine all factors that might have an effect on the multi-timeline representation, namely feasible features for particular types of anomalies, learning algorithm, and other practical parameters for each learning algorithm. After our experiments, we provides discussion and guidance to apply the multi-timeline representation to anomaly detection module in Chapter 7.

Chapter 4

System Implementation

In this chapter, we will explain all of implementation used and related to our experiments. In Chapter 4.1, we describe network features used in our experiments and explain how to normalize these features to the same scale. We explain three learning algorithm applied to our experiments in Chapter 4.3, namely multivariate normal distribution, k-nearest neighbor, and one-class support vector machine.

Main tools for our experiments can be separated into three groups: programming tools for dealing with the packet level, a numerical language and libraries for computations. We used the `C++` on Unix platform as a main programming language with Libpcap [87] to read a packet header and aggregate network traffic into different features. For numerical computations, we applied the GNU Octave [88] as a high-level programming language to deal with a number of matrices. Table 4.1 shows programing languages, version of language, and lines of code for each step in the proprosed system.

Table 4.1: Programing language for each step in the proposed detection system.

	Extraction	Scaling	Weighting	Classifier		
				MND	KNN	OSVM
Language	`C++`	Octave	Octave	Octave	Octave	LibSVM
Version	4.8.4	3.8.1	3.8.1	3.8.1	3.8.1	3.12
Lines of code	483	24	41	69	53	27

4.1 Features Extraction and Feature Scaling

In this section, we explain what features have been extracted and how to scale different ranges of these feature values. For our experiments, there are two steps actively involved with features of network traffic: feature extraction and feature scaling. The feature extraction is a process of transforming the input data into the set of features in order to perform the detection task, using this reduced representation instead of the full size input. The feature scaling is a method used to normalize the range of independent variables or features of data, also known as data normalization, it is performed during the data preprocessing step.

Table 4.2: Features of network traffic on an interval basis.

$f\#$	Feature	Description
f_1	Packet	Number of packets
f_2	Byte	Sum of packet sizes
f_3	Flow	Number of flows
f_4	SrcAddr	Number of source addresses
f_5	DstAddr	Number of destination addresses
f_6	SrcPort	Number of source ports
f_7	DstPort	Number of destination ports
f_8	ΔAddr	|SrcAddr − DstAddr|
f_9	ΔPort	|SrcPort − DstPort|
f_{all}	All	Combined all features

In the feature extraction step, we extracted nine features as listed in Table 4.2 during packet aggregation for every single interval. In this table, we list the feature abbreviation for our experiments, feature name, and description. Note that the ΔAddr (f_8) is the absolute number of difference between the number of source addresses and the number destination addresses, and the ΔPort (f_9) is the absolute number of difference between the number of source ports and the number of destination ports. We represent feature vector f_{all} at time interval x as

$$\mathbf{x}_{f_{all}} = \begin{bmatrix} x_{f_1} \\ x_{f_2} \\ \vdots \\ x_{f_9} \end{bmatrix}. \tag{4.1}$$

All of the features are carefully selected to account for characteristics of the selected anomalies listed in Table 5.1. We assume that input vectors as training data for learning process are attack-free or almost attack-free, because detection performance of the proposed system mainly rely upon input vectors. If input vectors contain more attacks or contamination, detection performance of the proposed system would be decreased by attack or contamination input.

In the feature scaling step, it is one of the important step for data preprocessing in our experiments. The feature scaling makes the values of each feature in the data have the common range in [0,1]. Selecting the target range depends on the nature of the data; however, we chose the simple way to compute the scaled value. Due to a time constraint, especially in detection process, we have to minimize computational time for feature scaling step by using only the maximum value. We scaled features according to

$$\hat{x}_{i,j} = \frac{x_{i,j}}{\max_j(x_{i,j})}, \forall_{i=1\ldots f} \wedge \forall_{j=1\ldots m}, \tag{4.2}$$

where $\hat{x}_{i,j}$ is a scaled feature value, $\max_j(x_{i,j})$ is the maximum value of the data in the i-th feature, m is the number of samples in the training data, and f is the number of the feature.

4.2 Weighting Process

Weighting is one of the processes in multi-timeline detector module as explained in Chapter 3. This process will guide the learning algorithm to generate a decision function which relies on particular timelines. Consequently, network operators could weight on one or more timelines to bias the decision function of detection system. Figure 4.1 shows the weighting process in our detector module during the learning process.

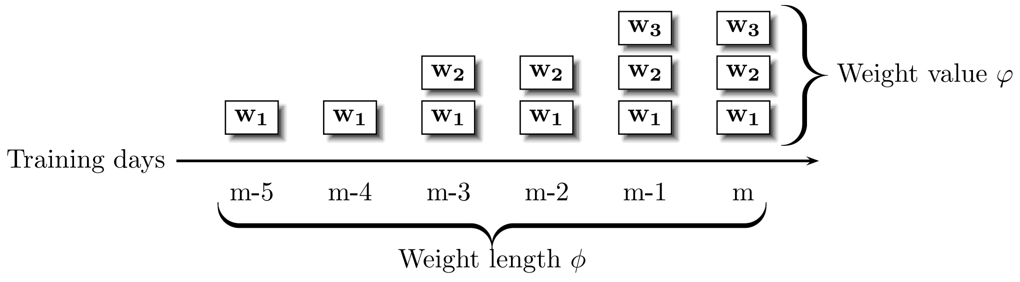

Figure 4.1: An example of weighting process with weight length $\phi = 6$ and weight value $\varphi = 3$.

In our implementation, we conducted experiments on the detector module both with and without the weighting process. There are two parameters for weighting process, namely weight length ϕ and weight value φ, where the weight length is the number of weighted training days and weight value is the maximum number of replication. We also made an assumption that recent data contains more useful information than older ones. To clarify, Figure 4.1 shows an example when m is the number of training days, and setting weight length to 6 and weight value to 3. As a result, we add a weight from day m to $m-5$ where weight 3 added to m and $m-1$, weight 2 added to $m-2$ and $m-3$, weight 1 added to $m-4$ and $m-5$ sequentially. For weight value, we replicate training samples as the number of weight value, for example, we replicate samples 3 times for weight value 3 and so on. We call this technique as linear gradual weighting, used in a study by Sousuke Amasaki and Chris Lokan [89]. We could be modified by reducing the weight value exponentially called exponential gradual weighting. In our experiments, however, we apply only linear gradual weighting technique. The weighting process could be disabled by specifying the weight value equal to 1 or setting all timeline in training data with the same weight value.

4.3 Learning Algorithms

In this section, we explain three learning algorithms selected for working with the multi-timeline technique in our experiments. We employed three standard and well-known algorithms of machine learning, namely the multivariate normal distribution, k-nearest neighbor, and one-class support vector machine, to work with our learning model. Even if, the multivariate normal distribution and k-nearest neighbor are not sophisticated algorithms, both algorithms are in the list of top 10 algorithms in data mining as well as the support vector machine [90]. We explained details of all these three learning algorithms as follows:

4.3.1 Multivariate Normal Distribution (MND)

Before explaining about multivariate normal distribution, let us begin with the one-dimensional or the univariate normal distribution (Gaussian density), is defined by

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right], \tag{4.3}$$

where μ is the mean value of the random variable x (an average), is defined as

$$\mu = E[x] \equiv \int_{-\infty}^{+\infty} xp(x)dx, \tag{4.4}$$

where $E[x]$ denotes the mean (or expected) value of a random variable x. The parameter σ^2 is the expected squared deviation or variance of a random variable x, is defined as

$$\sigma^2 = E[(x-\mu)^2] \equiv \int_{-\infty}^{+\infty} (x-\mu)^2 p(x)dx. \tag{4.5}$$

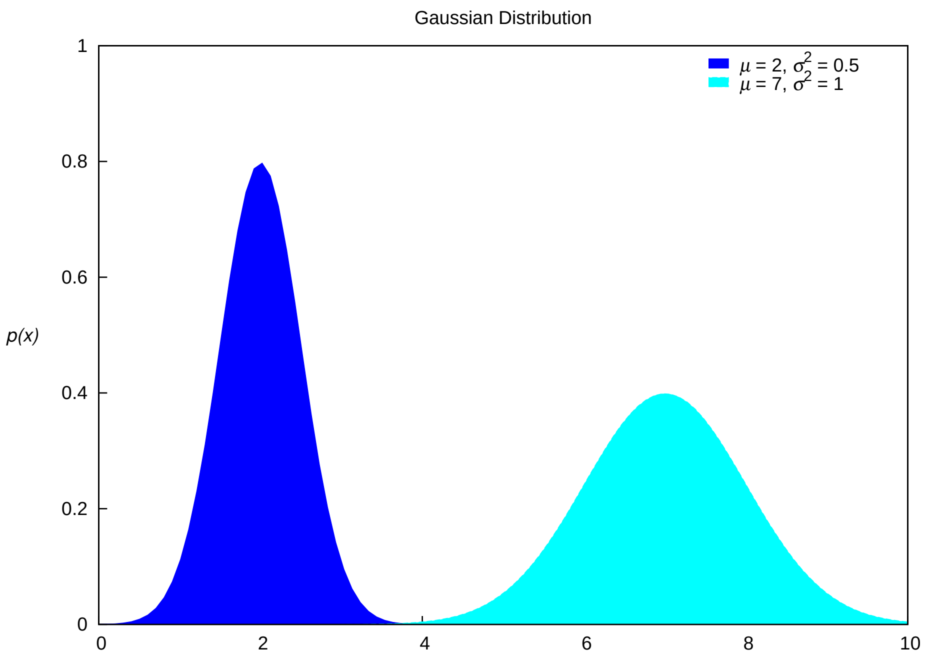

Figure 4.2: Gaussian probability density function, (left) $\mu = 2$ and $\sigma^2 = 0.5$, (right) $\mu = 7$ and $\sigma^2 = 1$

.

The univariate normal distribution is completely specified by two parameters, mean μ and variance σ^2. Figure 4.2 (left) shows the graph of the univariate normal distribution or Gaussian function for $\mu = 2$ and $\sigma^2 = 0.5$, and Figure 4.2 (right) shows the case for $\mu = 7$ and $\sigma^2 = 1$. Both are symmetric graph and always centered at μ; in addition, the larger value of the variance cause the wider of the graph as shown in the figure.

The MND is a generalization of the Gaussian or normal probability density in high dimensions [91]. In f dimensions, the probability density function of MND is written as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{f/2} |\mathbf{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1} (\mathbf{x}-\boldsymbol{\mu}) \right], \tag{4.6}$$

where $\mathbf{x}$ is a f-component column vector, $\boldsymbol{\mu}$ is the f-componet mean vcctor, $\mathbf{\Sigma}$ is the f-by-f covariance matrix, $|\mathbf{\Sigma}|$ and $\mathbf{\Sigma}^{-1}$ are its determinant and inverse, respectively. The covariance matrix $\mathbf{\Sigma}$ is defined as

$$\mathbf{\Sigma} = E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T], \tag{4.7}$$

where $(\mathbf{x}-\boldsymbol{\mu})^T$ denotes the transpose of $(\mathbf{x}-\boldsymbol{\mu})$. It seems obvious that for a single dimension or $f = 1$, the multivariate Gaussian coincides with the univariate normal distribution.

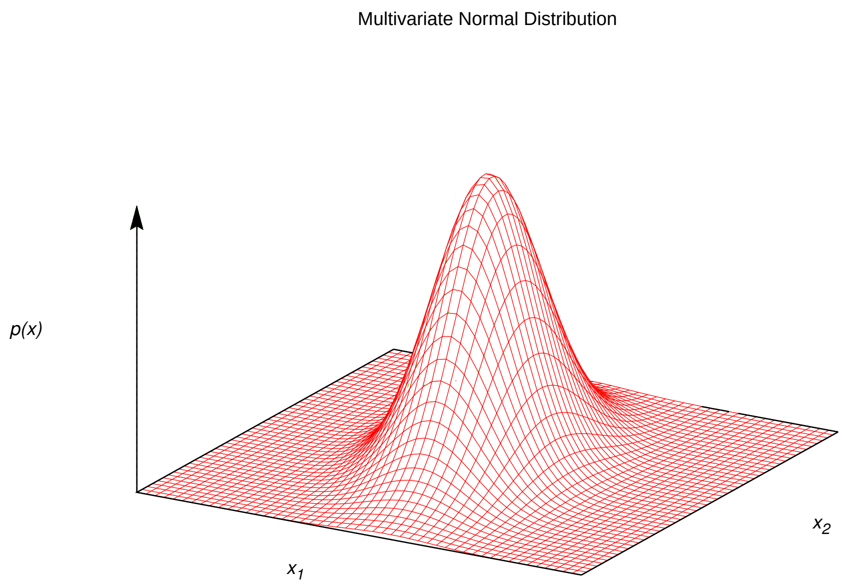

Figure 4.3: Multivariate normal distribution.

To get a sense on what the multivariate Gaussian looks like, let us focus on a case in the two-dimensional space for visualization. Figure 4.3 shows an example of a two-dimensional Gaussian probability density function. The graph contains two features, x_1 and x_2 represented by x-axis and y-axis respectively, while the z-axis indicates $p(\mathbf{x})$, and corresponds to the covariance matrix

$$\mathbf{\Sigma} = \begin{bmatrix} {\sigma_1}^2 & 0 \\ 0 & {\sigma_2}^2 \end{bmatrix}. \tag{4.8}$$

The shape of this graph are controlled by the covariance matrix $\mathbf{\Sigma}$. Figure 4.3 has the $\mathbf{\Sigma}$ with ${\sigma_1}^2 \ll {\sigma_2}^2$, so the graph is elongated along the x_2 direction.

To classify a test instance $\mathbf{x}$ in our experiments, we specified a threshold ε which have a range of value between 0 and 1. If $p(\mathbf{x})$ is less than ε, we will decide that the test instance $\mathbf{x}$ is anomaly, because the test instance gives a lower probability compared to normal instances. Otherwise, we decide that the test instance $\mathbf{x}$ is normal, because the test instance gives a probability in a boundary of normal instances. For our experiments, however, we generalize the ε to the adaptive threshold, is specified as

$$\varepsilon = \frac{1}{(2\pi)^{f/2} \left|\mathbf{\Sigma}\right|^{1/2}} \exp\left(-\frac{1}{2}\rho^2\right), \tag{4.9}$$

where ρ is a parameter to obtain the proportion of maximum probability. Smaller values of ρ produce higher value of $p(\mathbf{x})$ than larger values of ρ. In our experiments, we varied values of ρ between 2 and 4 on a linear scale to determine the best detection performance. Therefore, we can define the classification function of test data $\mathbf{x}$ as

$$g(\mathbf{x}) = \begin{cases} \text{anomaly}, & \text{if } p(\mathbf{x}) < \varepsilon; \\ \text{normal}, & \text{otherwise.} \end{cases} \tag{4.10}$$

The main reason for selecting MND is that this learning algorithm is one of the linear classifiers. We need a simple learning algorithms to work with our learning model, because the simpler algorithm we used, the faster computational time we obtain. Although data in real network environments may never come with the normal distribution, the MND provides a robust approximation and has many nice mathematical properties. Furthermore, because of the central limit theorem, many multivariate statistics converge to the normal distribution as the sample size increases.

4.3.2 *k*-Nearest Neighbor (KNN)

The k-nearest neighbor is one of the learning algorithms, which is very simple to understand but works incredibly well in practice. The KNN is an instance-based learning algorithm for classifying data instances based on the closest learning samples in the f-dimensional space [92]. It is quite different from the MND, which is based upon probability density of random variable.

To get better known about KNN, let us show an example of two-dimensional instances as shown in Figure 4.4. Suppose we plot training instances on a two-dimensional plane as shown in Figure 4.4 (left). The training instances have been labeled with one of the two classes, square class and triangle class. The test sample ($\otimes$ mark) should be classified either to the square class or the triangle class. If we set $k = 3$ (solid line circle as a boundary of 3-nearest neighbor), the algorithm assigns the test sample to the triangle class because there are 2 triangles and only 1 square inside the inner circle. If we set $k = 5$ (dashed line circle as a boundary of 5-nearest neighbor), the algorithm assigns the test sample to the square class because the number of squares is greater than the number of triangles inside the outer circle.

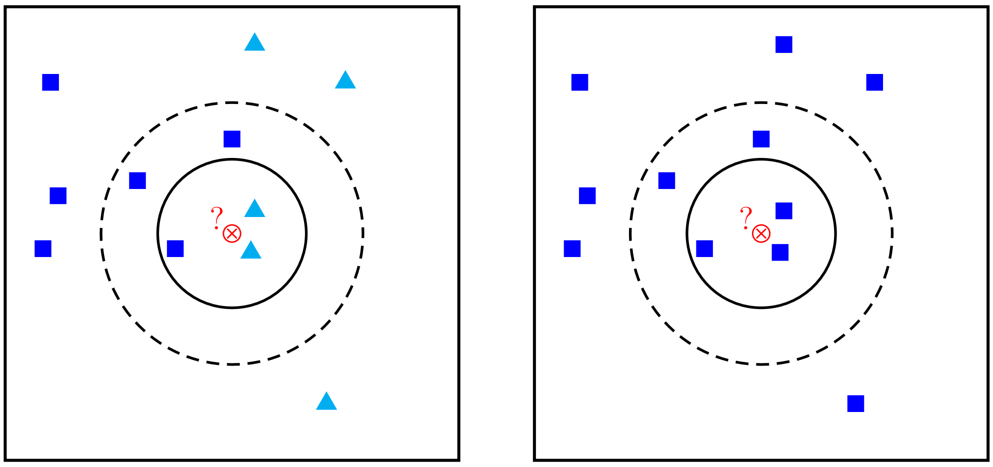

Figure 4.4: Examples of KNN: (left) the original KNN, (right) our modified KNN.

The original KNN, we need to define two parameters, the constant k and distance metric. In our experiments, however, we added another parameter, the distance D, in order to apply KNN for unlabeled training data. To understand our modified KNN, let us explain by using Figure 4.4 (right). Suppose we have only one square class, and intend to classify the test sample ($\otimes$ mark) whether it is in the square class or not. If we set $k = 5$ and $D = d_1$ (solid line circle), the algorithm does not assign the test sample to the square class because there are only 3 squares within the distance d_1. On the other hand, if we set $k = 5$ and a longer distance $D = d_2$ (dashed line circle), the algorithm assigns the test sample to the square class because there are 5 squares within the distance d_2. Therefore, there are 3 parameters used to classify the test data in our experiments, namely the constant k, the distance d, and distance metric.

For the distance metric, the Euclidean distance [93] is commonly used for many classification problems. In our experiments, the nearest neighbors of the data are also defined by standard Euclidean distances. More precisely, suppose test data $\mathbf{x}$ comprising f features be described by the feature vector $[x_1 \ldots x_f]^T$, where x_i denotes the value of the i-th feature of data $\mathbf{x}$, and T denotes the transpose of column vector, which is the simple way to write a column vector in a single row. The Euclidean distance between two instances $\mathbf{x}$ and $\mathbf{y}$ is the length of the line segment connecting them, given by

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{f} (x_i - y_i)^2}. \tag{4.11}$$

Even if we employ the Euclidean distance as the distance metric for our experiments, there are many other possible distance metrics. The following distance metrics are well studied:

- Mahalanobis distance [94]
- Manhattan distance [95]
- Minkowski distance [96]
- Canberra distance [97]

These metrics provide a different technique to measure between two instances in a high dimensional space. Dissimilar distance metrics also have advantages and disadvantages for specified tasks. Therefore we need to carefully select distance metric for our purpose.

To classify a test instance in our experiments, we specified the constant parameter $k = 3$ and varied the distance parameter. Thanks to the feature scaling step, we can vary the parameter D on a logarithmic scale between 10^{-6} and 10^{0} for selection of the best detection performance. We defined the classification function of test data $\mathbf{x}$ as

$$g(\mathbf{x}) = \begin{cases} \text{anomaly}, & \text{if amount of training data nearest} \\ & \text{to } \mathbf{x} \text{ is less than } k \text{ in the distance } D; \\ \text{normal}, & \text{otherwise.} \end{cases} \tag{4.12}$$

We selected the KNN to work with the multi-timeline model because this algorithm do not need an explicit learning phase. In other word, all the training data is needed in the detecting phase, so time consumption in learning phase is a constant value. Moreover, the modified KNN used in our experiment do not require all training data in learning phase because if we find the k training instances in range of D distances, we classify the test instance as a normal immediately, we do not need all training instances for classification. In this manner, this algorithm takes a short detection time for new instance. For this reason, the KNN is highly suitable for real-time anomaly detection in network traffic. In addition, the k value of KNN allows the classifier to tolerate noisy data; network traffic is one of the data which contain lot of noise.

4.3.3 One-class Support Vector Machines (OSVM)

One-class support vector machines first introduced by Schölkopf *et al.* [98] is a variation of the standard support vector machines (SVM) algorithm. To better understand how OSVM works, we start with the concept of SVM algorithm as shown in Figure 4.5.

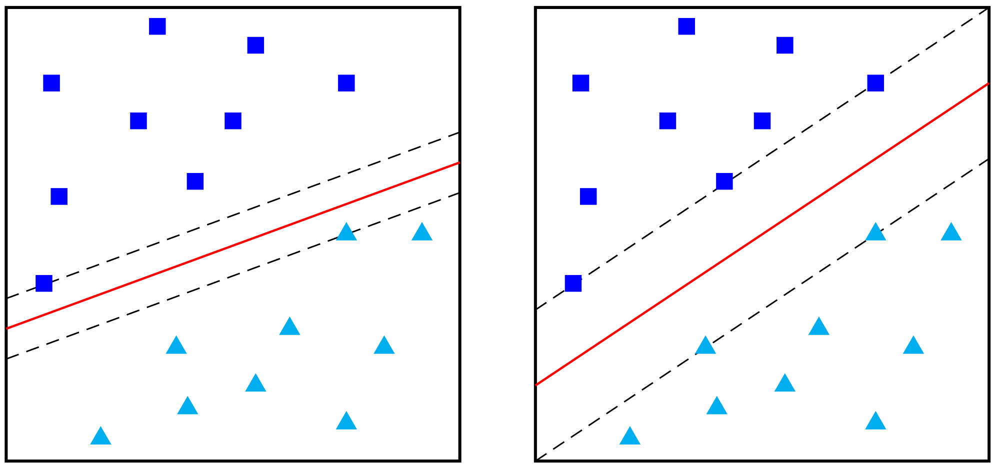

Figure 4.5: Examples of SVM algorithm: (left) smaller margin, (right) larger margin.

We briefly introduce the basics necessary concepts of SVM rather than mathematical concepts. Suppose that we have input data consisting of two classes, square class and triangle class. We plot all input instances on a two-dimensional plane as shown in Figure 4.5, it is obvious that both classes are separable from each other by a single decision line. However, there are many possible decision lines which can separate these two classes. Figure 4.5 on the left shows a decision line with smaller margins between two classes, but on the right shown another decision line with larger margins. First main concept of SVM is that the algorithm tries to find the line which is separate two classes with the maximum margin.

However, since input data is often not linearly separable, the SVM introduces the notion of a "kernel induced feature space" which casts the data into a higher dimensional space where the data is separable. For example, as shown in Figure 4.6 on the left, the original feature space cannot linearly separate between the square class and triangle class. Therefore, the SVM algorithm transforms original input data to a new feature space, as shown in Figure 4.6 on the right, which can separate two classes by a linear function as shown in Figure 4.6. The SVM utilizes a kernel function to transform original input data to a new feature space.

More precisely, a classification task usually involves separating data into training and testing sets. Each instance in the training set contains one target value (i.e. the class labels) and several attributes (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Given a training set of instance-label pairs $(\mathbf{x}_i, y_i), i = 1..., l$ where $\mathbf{x}_i \in \mathbb{R}^f$ and $\mathbf{y} \in 1, -1^l$, the SVM [99, 100] requires the solution of the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{4.13}$$

Here training vectors $\mathbf{x}_i$ are mapped into a higher (maybe infinite) dimensional space by the function ϕ. SVM finds a linear separating hyperplane with the maximal margin in this higher dimensional space. The constant value $C > 0$ is the penalty parameter of the error term. Furthermore, $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i{}^T\phi(\mathbf{x}_j)$ is called the kernel function. Although various kernels have been introduced by many researchers, the four basic kernels have been used in SVM are

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i{}^T\mathbf{x}_j$.
- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma\mathbf{x}_i{}^T\mathbf{x}_j + r)^d, \gamma > 0$.
- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$.
- sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma\mathbf{x}_i{}^T\mathbf{x}_j + r)$.

Here γ, r, and d are kernel parameters. The underlying mathematical theory behind the SVM algorithm are explained in [101].

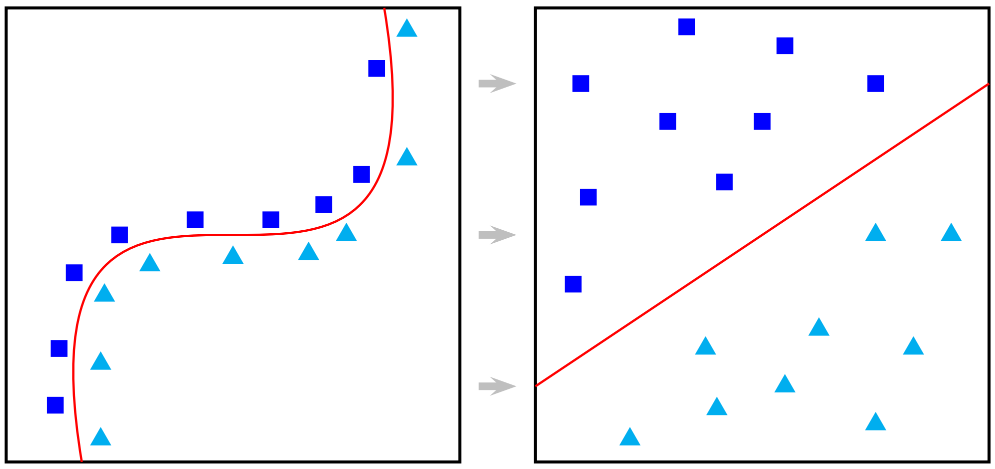

Figure 4.6: Transformation from the original input data to a new feature space.

The OSVM is a little different from original SVM, but the main concepts are still the same as SVM. The difference is that the training data for OSVM contain only one class, while those for SVM contain two or more classes. The main idea is that the OSVM maps unlabeled input data into a high dimensional space via an appropriate kernel function, and then attempts to find hyperplanes that separate the input data with maximum margin.

To classify test data in our experiments, we defined the decision function according to Muller *et al.* [102] as follow:

$$h(\mathbf{x}) = \mathrm{sign}((\boldsymbol{\omega}.\Phi(\mathbf{x})) - \rho) \tag{4.14}$$

where $h(\mathbf{x})$ is positive for most samples $\mathbf{x}_i$ contained in the training set and negative for the opposite, where $\boldsymbol{\omega}$ is a normal vector of hyperplane, Φ is a kernel function, and ρ represents a penalized value in the object function. Therefore, we define the classification function of test data $\mathbf{x}$ as

$$g(\mathbf{x}) = \begin{cases} \text{anomaly}, & \text{if } h(\mathbf{x}) = -1; \\ \text{normal}, & \text{if } h(\mathbf{x}) = +1. \end{cases} \tag{4.15}$$

In our experiments, we used the LIBSVM [103] tool with a radial basis function (RBF) as an appropriate kernel. We used default values of standard parameters from this tool for all experiments; however, we varied the *nu* and *gamma* parameters on a logarithmic scale between 10^{-3} and 10^{0} for selection of the best detection performance.

There are several reasons that we employed the OSVM as a learning algorithm with our learning model. First, the SVM is one of the new and popular classification techniques, especially for binary classification, and considered easier to use than other sophisticated algorithms, such as neural networks. Second, in other domains, results by using SVM are often highly satisfactory, so it might give reasonable results in task of anomaly detection in network traffic as well. Last but not least, the SVM is a flexible algorithm whose kernel function and other parameters can be tuned for own purpose. However, the SVM has a problem of over-fitting from optimizing the parameters to model selection as explained in [104].

Chapter 5

Datasets and Performance Metrics

In this chapter, we first explain how to acquire traffic from a reliable campus network and how to extract anomalies from a testbed data set, then we explain the process of data preparation in Chapter 5.1. Next in Chapter 5.2, we describe how to represent network traffic at a particular time interval by using a feature matrix, which is fast and easy for algorithms to learn and test instances. Finally, in Chapter 5.3, we explain the single value measure used in our study to evaluate detection performance of three learning algorithms.

5.1 Data Sources and Preparation

In this section, we explain about collecting network traffic and process of data preparation for experiments. We acquired data for our experiments from two separated sources, one is a restricted campus network as normal traffic and the other is a testbed data set as anomalous traffic.

For normal traffic, the primary source of raw data is from an edge router of the Internet service center in Kasetsart University, Thailand. This center provides computer clients for all university members, especially for students and researchers, to access the Internet and various local services. There are approximately 1,300 users every day among 157 computer clients, and the service hours of this center are open between 8:30 and 24:00 on weekdays, but close on weekends. Administrators installed anti-virus, personal firewall, and appropriate software for ordinary users in each client, and users cannot modify these software or install any other software by themselves. At the end of every day, all clients automatically erase any modification and the installed software and operating system are returned back to the initial state.

Administrators of this center guarantee that no client contains any malicious or attack software.

We collected real traffic from this campus network for 3 months and then manually selected a total of 55 days according to official statistics that indicate normal behavior of usage. For example, the report of official statistics indicates that there are fairly low usages during midterm and final examination periods. Consequently, we can assume that all of the network traffic collected from this source could contain a small number of anomalies, but such tiny anomalies is negligible.

For anomalous traffic, we manually selected several types of anomalies from a well-know testbed data set instead of simulating anomalies by ourselves. We extracted only packets associated with attacks from the testbed of DARPA Lincoln Lab in Massachusetts Institute of Technology [105, 106, 107]. The testbed contains both normal and anomalous traffic, and was provided for researchers to evaluate intrusion detection systems. However, we used only anomalous traffic because both types of traffic are from simulation, not from real behavior of usage network, so we better collected normal traffic from the real network environment rather than using from simulation. Even if the testbed from DARPA Lincoln Lab has been provided in order to evaluate intrusion detection systems more than a decade, the recent study by C. Thomas et al. [108] concluded that this testbed can be used for evaluation in the present scenario as well. The other crucial reason is that machine learning algorithms practically learn from background traffic. Therefore, the more realistic network traffic, the more realistic and accuracy of algorithms.

We selected five types of anomalies from the the testbed as follows:

1. *Back* attack is a denial of service attack against the Apache web server, where a client requests a URL containing many backslashes.

2. *IpSweep* attack is a surveillance sweep performing either a port sweep or ping on multiple IP addresses.

3. *Neptune* attack is a SYN flood denial of service attack on one or more destination ports.

4. *PortSweep* attack is a surveillance sweep through many ports to determine which services are supported on a single host.

5. *Smurf* attack is an amplified attack using ICMP echo reply flood.

Essential characteristics of these selected attacks are listed in Table 5.1. In the first column, we indicate sources from the testbed and types of anomalies for each instance. The *Back* and *IpSweep*, each attack contained two instances, while the *Neptune*, *PortSweep*, and *Smurf*, each attack contained three instances. In the next five columns, we show primitive characteristics of each instance: the number of source addresses, the number of destination addresses, the number of source ports, the number of destination ports, and the total amount of attack packets. Next, the average packet size (Bytes) and duration (Seconds) of each instance are shown in the seventh and eighth columns. Lastly, the average number of attack packets per second and percentages of each instance in one day long are shown in the last two columns.

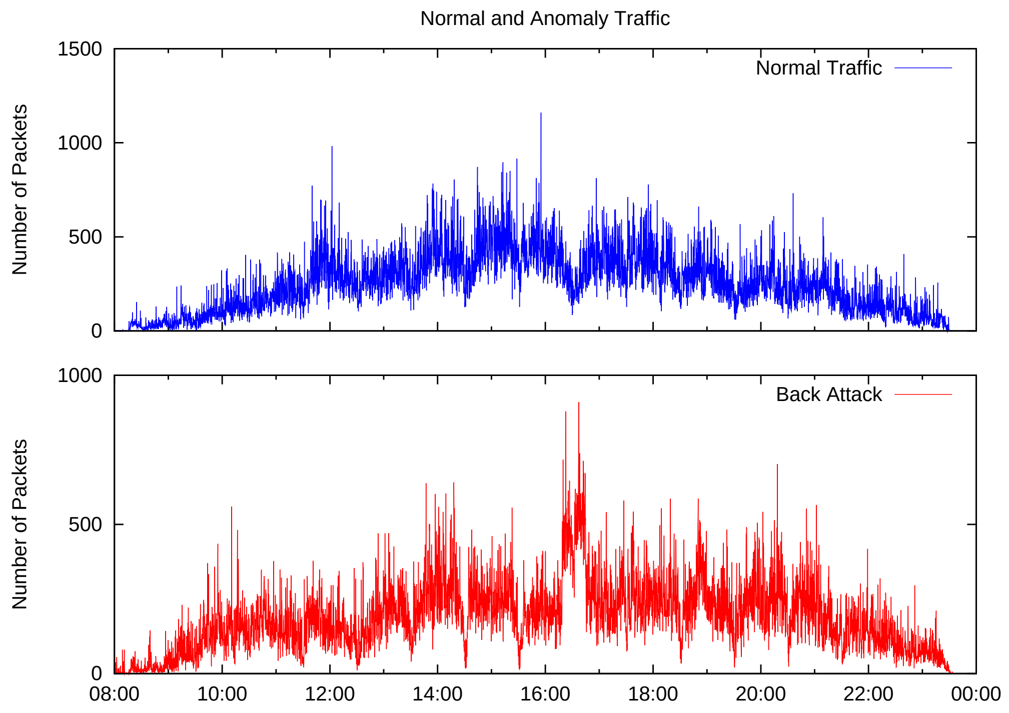

Figure 5.1: Examples of network traffic in our experiments, (top) normal traffic in training data, (bottom) Back attack in test data.

For data preparation, we created training and test dataset from normal and anomalous traffic data. We divided the 55-day raw data of normal traffic collected from the reliable source into two data sets: one is a 39-day (≈70%) traffic data set and the other is a 16-day (≈30%) traffic data set. We used the 39-day data set as a learning data set for designated algorithms without any modification. Separately, we combined the 16-day traffic data set with each instances of anomalies to create a test data set for each individual type of anomalies. For example, we combined the 16-day traffic data set with two instances of the *Back* attack, as listed in Table 5.1, to produce a 32-day test data set for the *Back* attack, and so on and so forth. Therefore, we have a 32-day test data set for the *Back* and *IpSweep* attack, and a 48-day test data set for the *Neptune*, *PortSweep*, and *Smurf* attack individually.

We also measured the volume of normal traffic, which is the aggregated traffic volume of all packets from/to the computer center, and anomaly traffic in both datasets. The minimum, maximum, and average volume of normal traffic are approximately 496 bit/sec., 394 kbit/sec., and 13 kbit/sec. respectively. The average volume of Back, Ipsweep, Neptune, PortSweep, and Smurf are approximately 560 kbit/sec., 11 kbit/sec., 32 kbit/sec., 576 kbit/sec., and 8 Mbit/sec. respectively.

There are two main reasons that we separated the learning data set and test data set for individual types of anomalies. The first reason is that we need to control network traffic and examine detection performance for individual types of anomalies. The other reason is the purpose of performance evaluation, we do need to exactly identify which network packets related to normal or anomalous traffic. We will explain the measure for performance evaluation in Chapter 5.3. If we cannot exactly identify network traffic in experiments, we cannot evaluate detection performance in the experiments.

Figure 5.1 shows examples of aggregated network traffic for our experiments. The x-axis indicates time between 8:00 and 24:00, and the y-axis presents the number of packets per time interval δ at 10 seconds. The top demonstrates a one-day network traffic in the training data, while the bottom shows an instance of Back attack that occurs between 16:18 and 16:44. At that time, as a results, the number of packets has an abrupt change as shown in the bottom figure.

Table 5.1: Characteristics of selected attacks.

Source	No. of SrcAddr	No. of DstAddr	No. of SrcPort	No. of DstPort	No. of Packet	Average Packet Size (Byte)	Duration (sec.)	Packets/sec.	%Anomaly
Back									
Week 2 Fri	1	1	1,013	1	43,724	1,292.31	651	67.16	0.75
Week 3 Wed	1	1	999	1	43,535	1,297.29	1,064	40.92	1.23
IpSweep									
Week 3 Wed	1	2,816	1	104	5,657	60.26	132	42.86	0.15
Week 6 Thu	5	1,779	2	105	5,279	67.75	4,575	1.15	5.30
Neptune									
Week 5 Thu	2	1	26,547	1,024	205,457	60	3,143	65.37	3.64
Week 6 Thu	2	1	48,932	1,024	460,780	60	6,376	72.27	7.38
Week 7 Fri	2	1	25,749	1,024	205,600	60	3,126	65.77	3.62
PortSweep									
Week 5 Tue	1	1	1	1,024	1,040	60	1,024	1.02	1.19
Week 5 Thu	1	1	1	1,015	1,031	60	1,015	1.02	1.17
Week 6 Thu	2	2	2	1,024	1,608	60	1,029	1.56	1.19
Smurf									
Week 5 Mon	7,428	1	1	1	1,931,272	1,066	1,868	1,033.87	2.16
Week 5 Thu	7,428	1	1	1	1,932,325	1,066	1,916	1,008.52	2.22
Week 6 Thu	7,428	1	1	1	1,498,073	1,066	1,747	857.51	2.02

5.2 Data Representation

In this section, we explain how to represent network traffic, which is highly suitable for our learning model. To reduce computational time of algorithms, we aggregated packets and represented them as a single data instance on an interval basis. In this manner, learning algorithms also will be able to learn and detect anomalies; although, there is no packet at all in the network. We represented aggregated traffic at an interval with a q-dimensional vector known as a feature vector, where q is the number of features. The feature vector is a column matrix, a matrix consisting of a single column of q elements. We represent aggregated traffic at time t as

$$\mathbf{x}_t = \begin{bmatrix} x_{t_1} \\ x_{t_2} \\ \vdots \\ x_{t_q} \end{bmatrix}. \tag{5.1}$$

Here x_{t_1} is a scalar-valued random variable of first feature, x_{t_2} is a scalar-valued random variable of second feature, and so on until q^{th} feature. In our experiments, we aggregated packets and extracted features from network traffic, and then scale every feature variable to the value between 0 and 1, we will describe feature extraction and scaling in Chapter 4.1, so that we can rewrite Eq. 5.1 with

$$\mathbf{x}_t = \begin{bmatrix} x_{t_1} \\ x_{t_2} \\ \vdots \\ x_{t_q} \end{bmatrix} \in \mathbb{R}^q_{[0,1]}. \tag{5.2}$$

However, Eq. 5.2 is a feature vector for a particular interval, and we mainly used this $\mathbf{x}$ representation for all experiments in Chapter 6.

In programming, however, we use following representation because it is more effective representation than Eq.5.2. Along a whole day, we can break time into n intervals by setting size of interval δ. In our experiments, we varied the size of interval δ from 1 second to 60 seconds, so the number n of intervals in a single day are varied from 86,400 to 1,440 respectively. Therefore, we can generalize Eq. 5.2 for one-day long as

$$\mathbf{x} = \begin{bmatrix} x_{t_{(1,1)}} & x_{t_{(1,2)}} & \cdots & x_{t_{(1,n)}} \\ x_{t_{(2,1)}} & x_{t_{(2,2)}} & \cdots & x_{t_{(2,n)}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{t_{(q,1)}} & x_{t_{(q,2)}} & \cdots & x_{t_{(q,n)}} \end{bmatrix} \in \mathbb{R}^{q\times n}_{[0,1]}, \tag{5.3}$$

where q is the number of features and n is the number of intervals in one day. The Eq. 5.3 is for one day network traffic; however, we actually use a three-dimentional matrix to represent multiple day network traffic, but we did not show the equation here. We applied Eq.5.3 into each decision function or classify function in Eq.4.10, 4.12, and 4.15 as explained in Chapter 4.

5.3 Performance Evaluation

Before giving an explanation of the measure used in our study, we provide background and basic concepts of performance evaluation for a binary classifier problem. Because of anomaly detection in our context defined as a binary classification, there are a number of metrics to take a measurement of this problem.

Table 5.2: Confusion matrix for anomaly detection.

Detected Class	Actual Class	
	Anomaly	Normal
Anomaly	True Positive (TP)	False Positive (FP)
Normal	False Negative (FN)	True Negative (TN)

Let us consider a two-type classification problem, in which the outcomes are labeled either as anomaly or normal. There are four possible results from a binary classifier.

1. If the result from a classification is anomaly and the actual value is also anomaly, then it is called a true positive (TP).

2. If the result from a classification is anomaly but the actual value is normal, then it is called a false positive (FP).

3. If the result from a classification is normal and the actual value is also normal, then it is called a true negative (TN).

4. If the result from a classification is normal but the actual value is anomaly, then it is called a false negative (FN).

These four possible results can be formulated in a 2×2 contingency table called confusion matrix as listed in Table 5.2. Moreover, almost all measures for binary classification problems have been derived from these four values.

In our study, we define anomaly detection as a binary classification problem, so there are several measurements for such the problem [109]. The following list shows measures for evaluating the detection performance for all our experiments.

- Precision-Recall
- F-score

We provide definition of these measures, then we show advantages and disadvantages of each measures.

Precision-recall (PR) is a measure that gives more information than the ROC curve, particularly in unbalanced data. The precision is the fraction of retrieved instances that are relevant, while the recall is the fraction of relevant instances that are retrieved. For classification tasks, the precision and recall are defined as:

$$precision = \frac{TP}{TP + FP}, \tag{5.4}$$

$$recall = \frac{TP}{TP + FN}. \tag{5.5}$$

The recall in this context is referred to as the true positive rate or sensitivity, and precision is also referred to as positive predictive value. From anomaly detection point of view, the precision is the percentage of detected intervals that are actually anomalies, and the recall is the percentage of the actual anomalous intervals that are detected. Although, an article of Jesse Davis [110] shows that PR curves give more details than ROC curves, in our experiments, we do require an average between both precision and recall as a single measure to indicate classification performance. For this reason, we ignore the PR curves in our experiments.

F-score or F-measure [111] is defined as a harmonic mean of precision and recall:

$$F\text{-}score = 2 \times \frac{precision \times recall}{precision + recall}. \tag{5.6}$$

The F-score is in the range of 0 to 1, where 0 represents the worst classification rate, and 1 represents a perfect classification rate. The F-score is often used in many fields, including information retrieval for measuring search, document classification, query classification performance, and machine learning. Many earlier studies related to binary classification used the F-score as a single measure to evaluate their work. However, the F-score do not take the true negative rate into account, and it may be not preferable to assess the performance of a binary classifier

We evaluated detection performance of the proposed system and learning algorithms by using the F-score as a single measure. It returns a value between 0 and 1, where an F-score of 1 represents a perfect detection and 0 represents a worst detection or 100% miss. In our experiments, an acceptable F-score is 0.5 because this value derived from the precision and recall. It takes account of both false positive (FP) and false negative (FN) values. Therefore, half range of F-score 0.5 or higher values indicate that the proposed system accomplish in task of anomaly detection.

Chapter 6

Evaluation

In this chapter, we represent the main purpose of individual experiments, and show results from each of the experiments on the multi-timeline representation. Note that, real-time anomaly detection in network traffic is quite different from those utilized batch processing, offline techniques, or processing at the end of the day. Therefore, we focus on many key aspects of the multi-timeline representation rather than detection performance only; these key aspects include beneficial effects of features and parameters usage, learning curves, and time consumption. Consequently, we designed a series of experiments in order to examine that the multi-timeline representation contains flexible capabilities, and has a high probability of detecting various anomalies in network traffic in real time.

In this study, we grouped our examination into nine experiments as follows:

- Experiment 1: comparison of different interval values
- Experiment 2: comparison of detection performance between individual features and all features combination
- Experiment 3: detection performance of realtime and combination representation
- Experiment 4: no packet situations
- Experiment 5: learning curves
- Experiment 6: time consumption
- Experiment 7: different volume of background traffic
- Experiment 8: time of anomaly occurrence

- Experiment 9: weighting for old data

In some experiment, parameter setting yields subsequent results from a prior experiment. We explain objectives, procedures, and show examination results of each experiment in the following sections.

6.1 Experiment 1: Comparison of Different Interval Values

In our study, we break a single timeline of one day long into a series of intervals. The interval value or bin size is one of the indispensable parameters, which is an important factors affecting detection performance and time consumption of the multi-timeline representation. On the one hand, if we set the interval value with a long time, the number of intervals for one day is small and rarely use computational resources; however, it is quite difficult to identify which packets related to anomalies because single interval could contains a large number of packets. On the other hand, if we set the interval value with a short time, the number of intervals for one day is huge and frequently use computational resources; however, it is easier to identify which packets related to anomalies because single interval contains a smaller number of packets.

The interval value also produces a significant effect on real-time detection or notification time after anomalies have been discovered. For example, assume that we set the interval value at five minutes. It means that we need to aggregate packet across five minutes and then perform a task of anomaly detection, so the notification time when anomalies occurred takes approximately five minutes plus processing time of detection. However, if we set the interval value at a short time, the system could notify network operators faster than a long interval value. For these reasons, the interval value or bin size is one of the crucial factors not only for the multi-timeline representation, but for real-time detection as well.

The purpose of this first experiment is to identify the feasible and suitable interval value, which will be later applied into the rest of our experiments. In this experiment, we varied interval values δ = 1, 2, 4, 6, 8, 10, 20, 30, 40, 50, 60 seconds, and fixed the number of training days $\beta = 39$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p|\mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.1}$$

where p is varied upon the number of training days β. We firstly performed the experiment by employing individual features on individual algorithms,

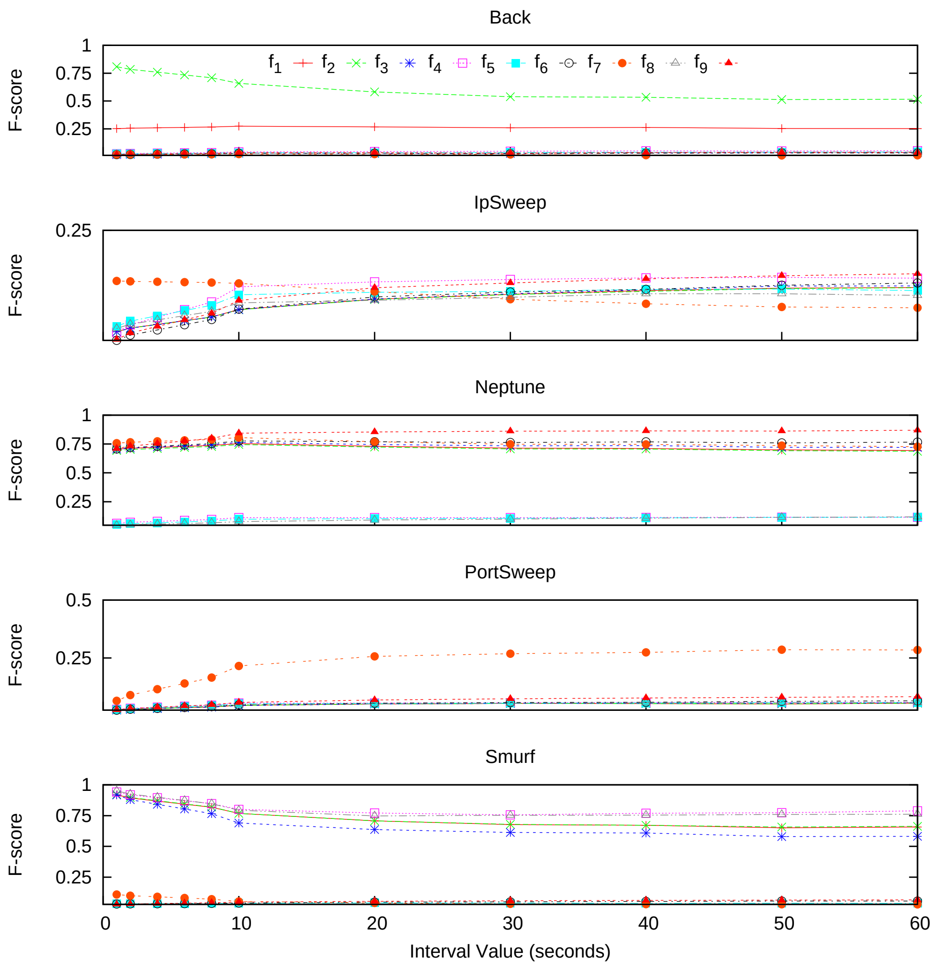

Figure 6.1: Detection performances of MND on different interval values by using individual features.

and then computed the F-score (Eq.5.6) for each single type of attacks. The results of this experiment are showed in Figure 6.1-6.3 by using MND, KNN, and OSVM respectively. Figure 6.4 provides the average detection performance with F-score as a summary for each algorithm over each type of attack. Note that we list the characteristics of selected attacks in Table 5.1, including the duration of target anomalies.

In the figures, the y-axis shows the F-score or detection performance, which has values between 0 and 1, the higher values of F-score indicate better detection performance than the lower values. The x-axis represents various interval values from 1 to 60 seconds. From top to bottom, each row depicts detection performance on different types of attacks, namely Back, IpSweep, Neptune, PortSweep, and Smurf as explained in Chapter 5.1. Different lines in the graph show detection performance by using different nine features as listed in Table 4.2.

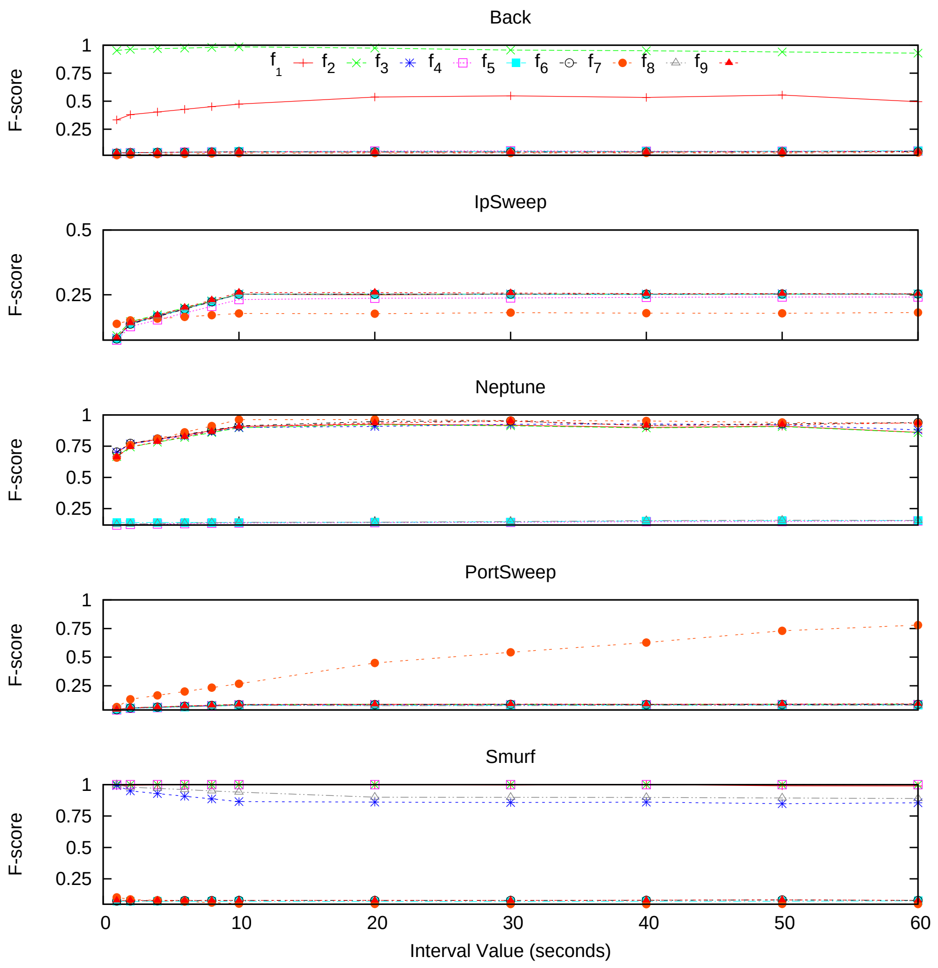

Figure 6.2: Detection performances of KNN on different interval values by using individual features.

The examination results by using MND on different interval values are shown in Figure 6.1. For Back attacks, the most effective feature is f_2 but detection performance declined when interval values increased; however, the next effective feature f_1 and the other features are steady along different internal values. For IpSweep attacks, performances of all features except f_7 are increased from 1 to 10 seconds, then slightly increased and remained steady from 10 to 60 seconds. For Neptune attacks, performances of f_{1-3}, f_{6-7}, and f_9 slightly increased close to F-score 0.75, and performances of

f_{4-5}, and f_8 are nearby F-score 0.1. For PortSweep attacks, performance of f_7 marked increased from 1 to 20 seconds, then remained steady until 60 seconds. For Smurf attacks, performances of f_{1-4} and f_8 noticeably decreased from 1 to 10 seconds, then slightly decreased until 60 seconds.

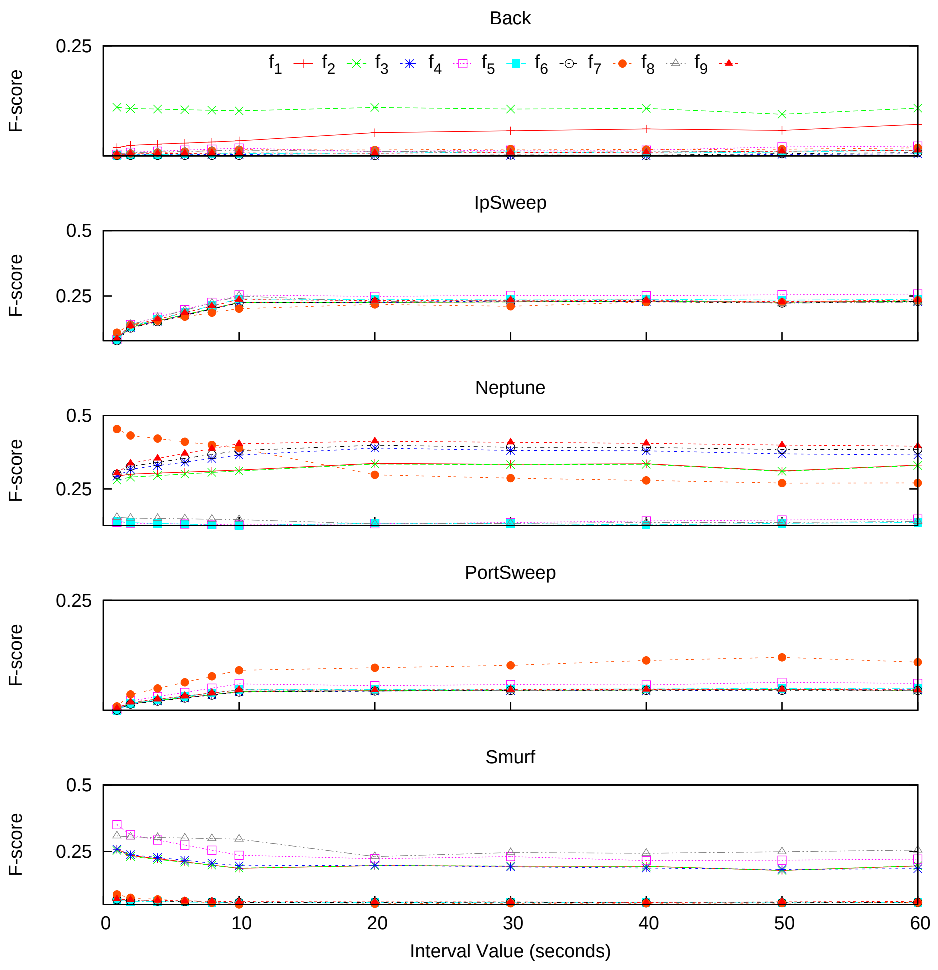

Figure 6.3: Detection performances of SVM algorithm on different interval values by using individual features.

We presented examination results by using KNN on different interval values in Figure 6.2. For Back attacks, best two effective features are f_2 and f_1 respectively; however, these slightly increased from 1 to 20 seconds and slightly decreased from 50 to 60 seconds. For IpSweep attacks, performances of all features except f_7 markedly increased from 1 to 10 seconds, then remained steady until 60 seconds. For Neptune attacks, performances of f_{1-3}, f_{6-7}, and f_9 grow from 1 to 10 seconds, and all seem higher than those by using MND. For PortSweep attacks, performance of f_7 steady increased when interval values were increased, while performances of other features were low and seem unchanged. For Smurf attacks, only performance of f_3 and f_8 slightly decrease from 1 to 10 seconds, f_{1-2} and f_4 remained at F-score 1, so as f_{5-7} and f_9 remained at F-score 0.1.

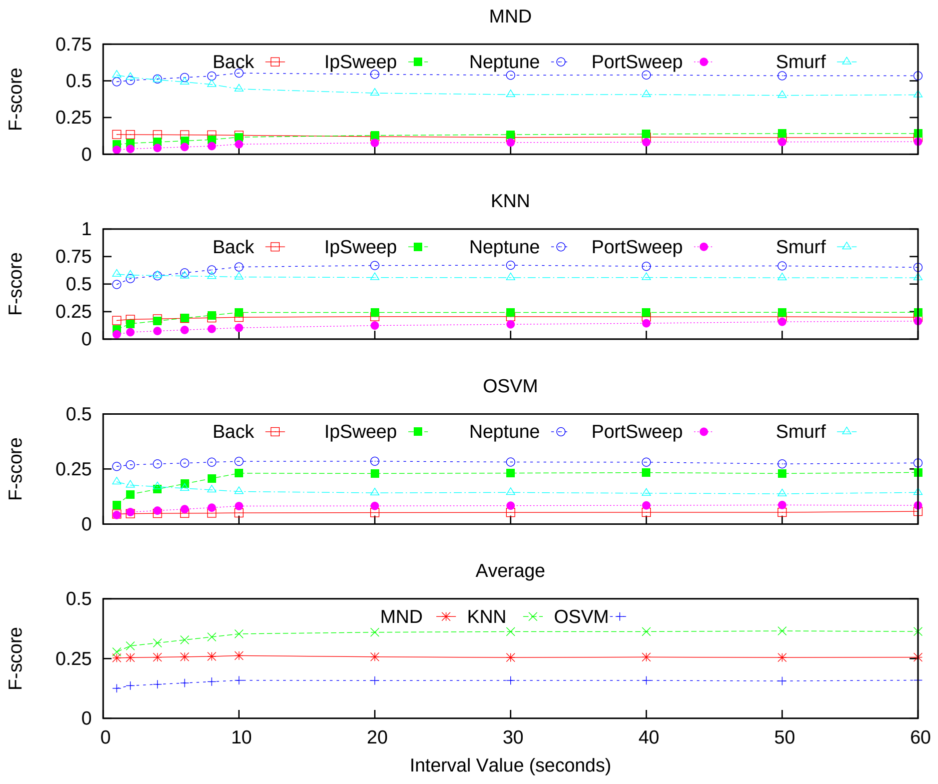

Figure 6.4: Average detection performances with different interval values using the MND, KNN, and OSVM.

The examination results in Figure 6.3 are from OSVM on different interval values. For Back attacks, only performance of f_1 gradually increased along interval values, while others have little changes. For IpSweep attacks, performances of all features markedly increased from 1 to 10 seconds, and then remained steady until 60 seconds. For Neptune attacks, only performance of f_7 slowly decreased from 1 to 20 seconds, and then seem unchanged until 60 seconds, while others slightly increased from 1 to 20 seconds or have a little change along interval values. For PortSweep attacks, performances of all feature increased from 1 to 10 seconds, and then little changed from 10 to 60 seconds. For Smurf attacks, performances of all features decreased from 1 to 10 or 20 seconds, and then remained steady to 60 seconds.

In Figure 6.4, the y-axis shows the F-score or detection performance, the x-axis represents interval values from 1 to 60 seconds. Three rows from the top depict detection performances by using MND, KNN, and OSVM respectively, different lines show performances on different types of attacks. The last row in this figure shows average performances of all three algorithms.

From top three rows, we can group detection performance lines along interval value into three group: first, performance lines increased from 1 to 10 seconds and remained steady to 60 seconds; second, performance lines decreased from 1 to 10 seconds and remained steady to 60 seconds; third, performance lines seem steady along all interval values. The last rows depict comparison of performance lines from three learning algorithms. It shows that all performance lines increased from 1 to 10 second and then have a little change between 10 and 60 seconds. Best detection performance lines are from KNN, MND, and OSVM respectively. The results in this experiment demonstrate that the feasible interval value for our data set is 10 seconds, so we decided to perform the rest of our experiments with interval value at 10 seconds.

6.2 Experiment 2: Comparison of Detection Performance between Individual Features and All Features Combination

Feature selection play a crucial role in anomaly detection not only for the purposed representation, but also for all other representation in anomaly detection and any other domains. In theory, an efficient feature for a particular type of anomaly may not be an efficient feature for another type of anomaly. In machine learning perspective, we can also combine two or more features up to an unlimited number of features to precisely detect or to cover a wider range of anomaly. However, the number of features eventually have an effect on time consumption over both learning and detecting process. A large number of combined features takes more computational resources and time than a small number of combined features.

Nearly in the all experiments, we extract nine features on interval basis as explained in Chapter 4.1. The number of combinations of these nine features can be 511 possibilities; therefore, we decided to perform experiments with individual features and a combination of all nine features instead. From this manner, we can obtain the results which one is an effective feature for a particular type of anomalies, and then we later selected these effective features for next experiments.

In this experiment, we use different feature vectors for a single and combined feature. For single feature, we represent a feature vector as

$$\mathbf{x}_t = \left[x_{t_f}\right] \in \mathbb{R}^{f}_{[0,1]}, \tag{6.2}$$

where f is a feature, such as $f = 1$ indicates f_1 or packet feature and so on. For combined all feature, we represent a feature vector as

$$\mathbf{x}_t = \begin{bmatrix} x_{t_1} \\ x_{t_2} \\ \vdots \\ x_{t_9} \end{bmatrix} \in \mathbb{R}^{9}_{[0,1]}, \tag{6.3}$$

that combine all 9 features, from f_1 to f_9.

The purpose of Experiment 2 is to investigate highly effective features for each individual type of attack. In this experiment, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 39$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p | \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.4}$$

where p is varied upon the number of training days β. First, we sequentially employed individual features, from f_1 to f_9, with a learning algorithm to detect each type of selected attack. Next, we used the combination of all nine features, and performed the same experimental process as individual features. We then compared detection performance of individual features and the combination of all nine features with the precision (Eq.5.4), recall (Eq.5.5), and F-score value (Eq.5.6). The results of this comparison are shown in Figure 6.5-6.7 in order of algorithms, MND, KNN, and OSVM respectively. Finally, we compared F-scores of all three algorithms on each type of attack, the results are shown in Figure 6.8.

In Figures 6.5-6.7, the y-axis shows precision, recall, and F-score, each has a value between 0 and 1, where 0 indicates the worst detection performance and 1 indicates the best detection performance. The x-axis represents individual features from f_1 to f_9, as listed in Table 4.2, and a combination of all features, f_{all}. In these figures from top to bottom, each row illustrates performances on different types of attacks: Back, IpSweep, Neptune, PortSweep, and Smurf respectively as explained in Chapter 5.1. Different vertical bars in the graph represent precision (P), recall (R), and F-score (F) values.

Detection performances of MND using different features are shown in Figure 6.5. For Back attacks, effective features are f_2 and f_1 respectively, but the F-score of f_{all} is relatively low, although the recall value of f_{all} is quite high. For IpSweep attacks, detection performances of individual features are quite low, while performance of f_{all} are relatively high. For Neptune attacks, f_{1-3}, f_{6-7}, and f_9 gain high detection performance, but the F-score of f_{all} is relatively low, although the recall value of f_{all} reached the highest point. For PortSweep attacks, only f_7 is a feasible feature, while f_{all} is not better than f_7, although the recall value of f_{all} is better than those of f_7. For Smurf attacks, f_{1-4} and f_8 are highly effective features, while the F-score of f_{all} is relatively low, although the recall value of f_{all} is high.

The results of KNN using different features are shown in Figure 6.6. For Back attacks, f_2 and f_1 are the same effective features as using MND;

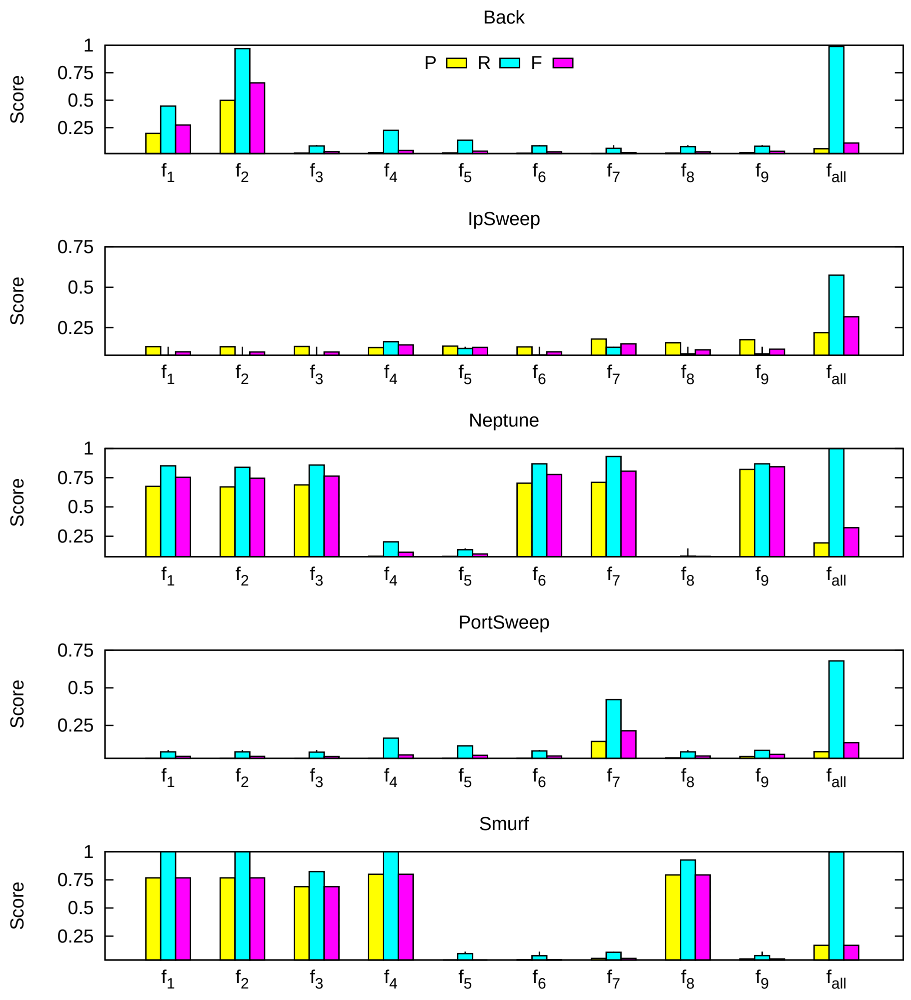

Figure 6.5: Precision (P), Recall (R), and F-score (F) of MND using different features.

however, the F-score of f_{all} is much more higher than the F-score result by using MND. For IpSweep attacks, although recall values of all individual features, excepting f_7, seem very high, F-score of all individual features are obviously low. For Neptune attacks, detection performances of f_{1-3}, f_{6-7}, and f_9 are very high, but the F-score of f_{all} slightly decrease from those by using individual features. For PortSweep attacks, the detection performance of f_7 is a little better than those of all other features. For Smurf attacks, detection performances of f_{1-4}, f_8, and f_{all} are very high, it is not the same result of MND that detection performance of f_{all} is quite low.

The results shown in Figure 6.7 are from OSVM using different features.

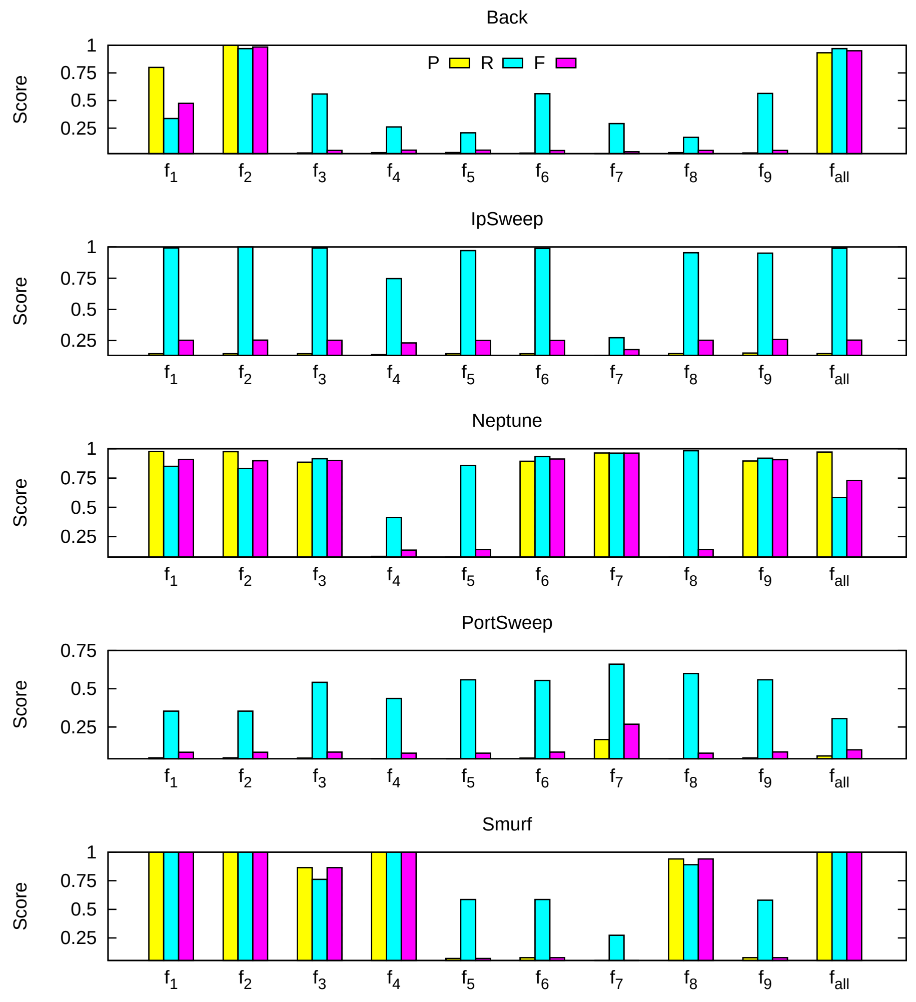

Figure 6.6: Precision (P), Recall (R), and F-score (F) of KNN using different features.

Obviously, recall values of OSVM seem pretty for all types of attacks, but F-scores are quite low by using both individual features and the combination of all features. For Back attacks, detection performances of f_2 and f_{all} are slightly higher than those of other features. For IpSweep attacks, there is no difference in detection performance among all individual features and the combination of all features. For Neptune attacks, effective features are f_{1-3}, f_{6-7}, f_9, and f_{all}, it is similar to MND and KNN. For PortSweep attacks, detection performance using f_7 is a little bit higher than using other features. For Smurf attacks, F-scores of f_{1-4}, f_8, and f_{all} are a little higher than those using f_{5-7} and f_9.

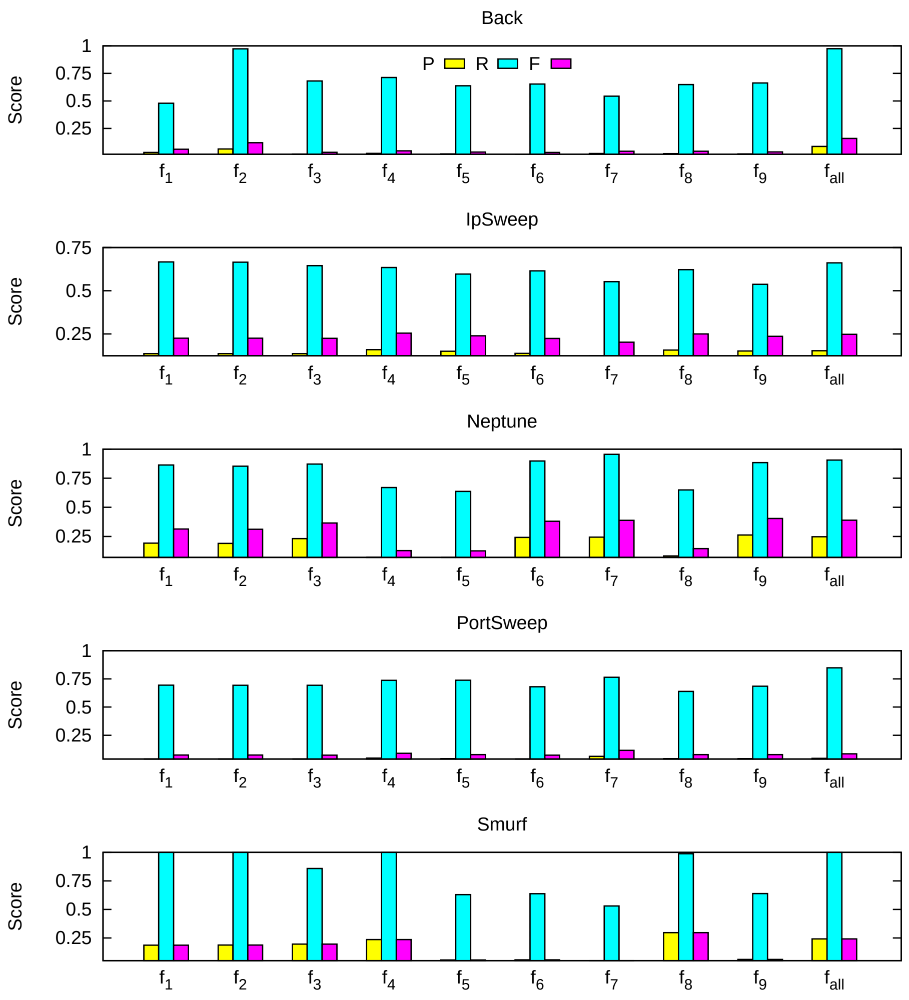

Figure 6.7: Precision (P), Recall (R), and F-score (F) of OSVM using different features.

Figure 6.8 provides the same details as those in Figures 6.5-6.7; however, we show only F-scores of all three algorithms for comparison. For Back attacks, the effective features are f_1, f_2, and f_{all} for all three algorithm, but KNN is the best algorithm, especially by using f_{all}. For IpSweep attacks, F-scores of MND are the lowest by using individual features, but is the highest by using f_{all}. For Neptune attacks, detection performances of MND and KNN quite higher than those of SVM, especially by using individual features. For PortSweep attacks, f_7 is the only effective feature for all algorithms, but the performance fairly dropped by using f_{all}. For Smurf attacks, f_{1-4}, f_8, and f_{all} are notable features, F-scores of KNN are the highest values of all three

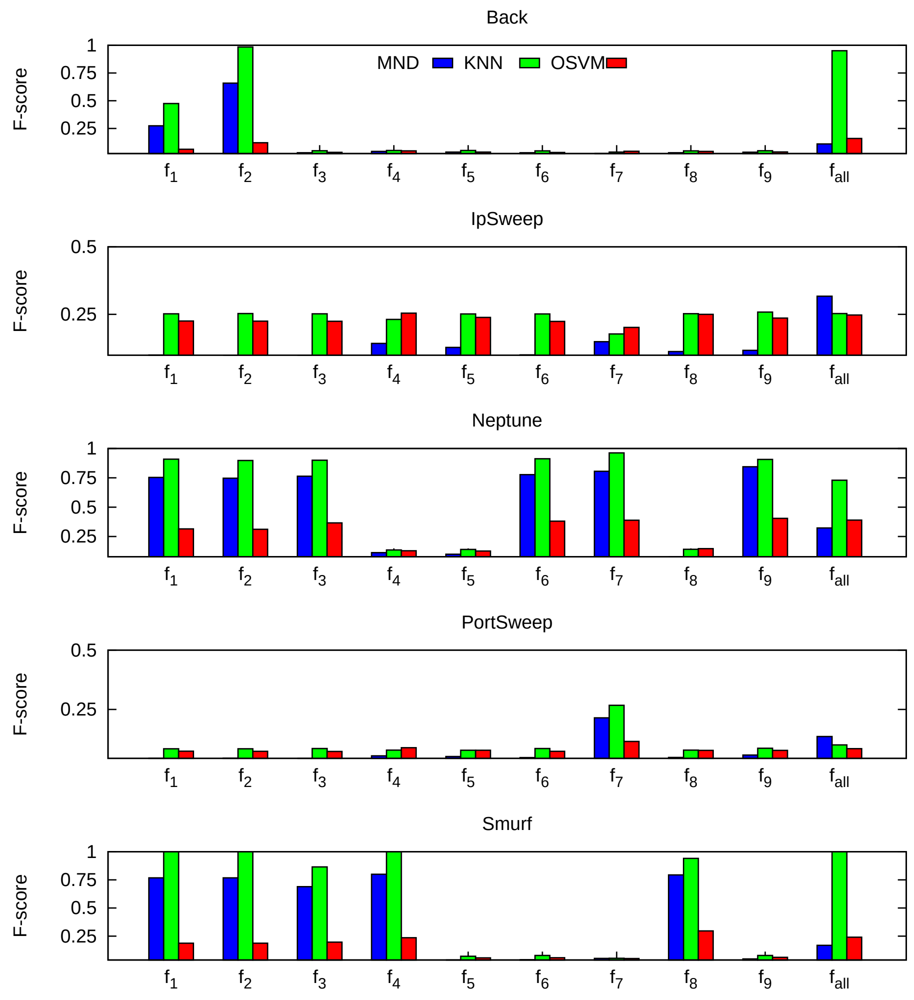

Figure 6.8: F-score comparison between MND, KNN, and OSVM with multi-timeline representation.

algorithms.

6.3 Experiment 3: Detection Performance of Realtime and Combination Representation

To compare detection performance of multi-timeline representation with another conventional representation, we also conducted additional experiments

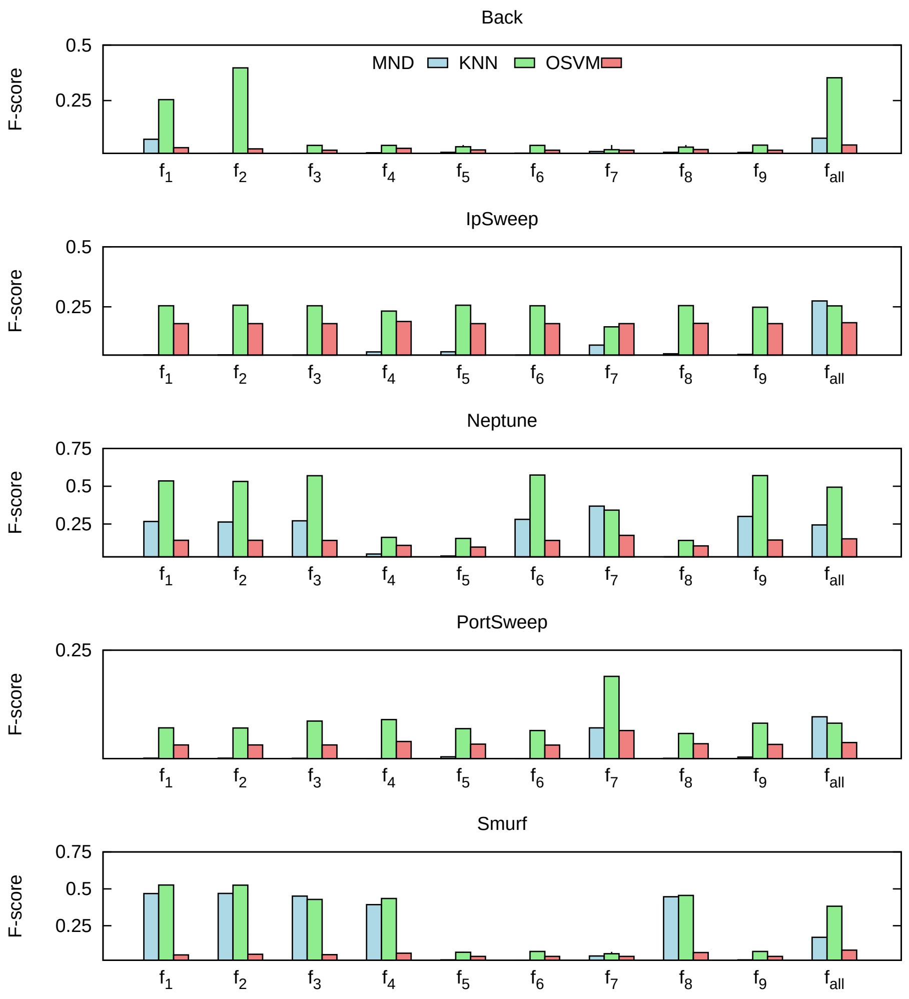

Figure 6.9: F-score comparison between MND, KNN, and OSVM with realtime representation.

by using the real-time representation as explained in Chapter 2.4.3. Note that we cannot compare multi-timeline representation to the manual-based representation and batch representation because of their limitations; the manual-based representation cannot detect anomalies which are not contained in database or discriminant function and batch representation cannot be applied in real time, for example. In this experiment, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 39$ days as multi-timeline representation. At present timeline p, we used decision function for real-time or single timeline representation

$$g(\mathbf{x}_t^p | \mathbf{x}_1^1, ..., \mathbf{x}_{8640}^1, ..., \mathbf{x}_1^{p-1}, ..., \mathbf{x}_{8640}^{p-1}, \mathbf{x}_1^p, \mathbf{x}_2^p, ..., \mathbf{x}_{t-1}^p). \tag{6.5}$$

The function detects a test data at time t for the present timeline p, $\mathbf{x}_1^1$ and $\mathbf{x}_{8640}^1$ indicate feature vectors of first and last interval for training day 1, and so far. In this case, we set the interval value $\delta = 10$, so the number of feature vector for one day is 8,640. The number of input in this function varied upon the number of interval and training days β. Detection results of real-time representation are shown in Figure 6.9, we show F-scores of real-time representation with same details as those in Figure 6.8. For Back attacks, the KNN outperformed other algorithms, especially by using f_1, f_2, and f_{all}; however, detection performances of all three algorithms are worse than those by using the multi-timeline representation. For IpSweep attacks, F-scores of all three algorithms by using the real-time representation are very similar to F-scores of our representation as shown in Figure 6.8. For Neptune attacks, the KNN gain higher detection rates than other algorithm, but quite lower detection rates than the multi-timeline representation. For PortSweep attacks, detection performances of all algorithms seem a little lower than those by using multi-timeline representation as shown in Figure 6.8. For Smurf attacks, effective features by using real-time representation and the multi-timeline representation are the same, but F-score of real-time representation are much lower than those of the multi-timeline representation.

We are also interested in combining single and multi-timeline representation. To this end, we conducted an additional experiment to measure detection performance of combination scheme. In this experiment, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 39$ days as multi-timeline representation. We also use the present timeline p as a training day for single timeline representation. Therefore, the decision function for this experiment is

$$g(\mathbf{x}_t^p | \mathbf{x}_1^p, \mathbf{x}_2^p, ..., \mathbf{x}_{t-1}^p, \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}). \tag{6.6}$$

The function detects a test data at time t for the present timeline p, from

$\mathbf{x}_1^p$ to $\mathbf{x}_{t-1}^p$ indicate feature vectors from single timeline representation, and from $\mathbf{x}_t^1$ to $\mathbf{x}_t^{p-1}$ indicate feature vectors from multi-timeline representation.

First, we sequentially employed individual features, from f_1 to f_9, with a learning algorithm to detect each type of selected attack. Second, we used the combination of all nine features, and performed the same experimental process as individual features. Finally, we compared F-scores of all three algorithms on each type of attack, the results are shown in Table 6.1.

We found that detection performances of the combination scheme are lower than those of multi-timeline representation; ; 1.08-2.11% for 10 minutes training data ($t = 60$), 4.97-6.34% for 1 hour training data ($t = 360$), and 9.82-18.31% for 1 day training data ($t = 8640$) for the single representation. From this experiment, we conclude that the detection performance of the combination scheme largely decreases for more input data in the single timeline. Thus, the multi-timeline representation outperforms both single and combination representation.

Table 6.1: Performance degradation of real-time and combination from those of multi-timeline representation.

t	Real-time	Combination
60	5.24-7.13%	1.08-2.11%
360	6.93-10.47%	4.97-6.34%
8640	15.71-23.53%	9.82-18.31%

6.4 Experiment 4: No Packet Situations

One of our main ideas for anomaly detection is that the system can detect anomalies caused by both attacks or unusual incidents. In terms of anomaly detection, the meaning covers far beyond intrusion detection, which is focus on attacks from outsiders only. Moreover, anomaly detection systems should have an essential characteristic which is difficult to be compromised or influenced by the outside world. Even if attackers deeply know about techniques or methods of detection system, it should be hardly possible for them to imitate normal traffic or evade the detection system. These characteristics are crucial points for not only batch processing but also for real-time processing.

In this third experiment, we examine learning algorithms with the multi-timeline representation to detect an anomaly caused by unusual incidents, and to test robustness of the multi-timeline representation. Therefore, we separately conducted two sub-experiments for different purposes. The purpose of the first sub-experiment is to investigate performance of three learning algorithms with the multi-timeline representation in detecting an unusual incident, which is no packets at all for the whole day. The purpose of the second sub-experiment is to examine robustness of the multi-timeline representation with incorrect data in the learning process. To perform these two sub-experiments, we simulated and applied no packet traffic to different data. We simulated no packet traffic in test data for the first sub-experiment, and combined original training data with no packet traffic for the second sub-experiment.

6.4.1 No Packet Incident in Test Data

For the first sub-experiment, we assumed that an outage happens in the network system; as a result, there is no traffic packet for one day long. At the beginning, we let algorithms learn from all training data as we performed in the previous two experiments, by using each individual feature and the combination of all features. We then attempted to detect this unusual incident (no traffic packet in test data) by using the same feature as performed in the learning process. In this experiments, we fixed the interval value $\delta = 10$ seconds and the number of training days $\beta = 39$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p | \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.7}$$

where p is varied upon the number of training days β. Here $\mathbf{x}_t^p$ is a zero vector, for no packet incident in test data experiment. Experiment results in this first sub-experiment are shown in Figure 6.10. First three rows from the top in Figure 6.10 show detection performances of MND, KNN, and OSVM respectively. The last row in this figure provides F-score comparison of all three learning algorithms over the no packet incident in test data.

For first three rows from the top in Figure 6.10, the y-axis shows precision (Eq.5.4), recall (Eq.5.5), and F-score (Eq.5.6), these three indicators have a value between 0 and 1, where 0 denotes the worst detection performance and 1 denotes the best detection performance. The x-axis represents individual features from f_1 to f_9, as listed in Table 4.2, and f_{all} the combination of all features. For the last row in Figure 6.10, the y-axis shows only F-score values in the same range between 0 and 1. The x-axis still represents f_1 to f_9, and f_{all}.

Each row in Figure 6.10 represents the following results. The first row from MND shows that precision scores are high for all features, but F-scores of

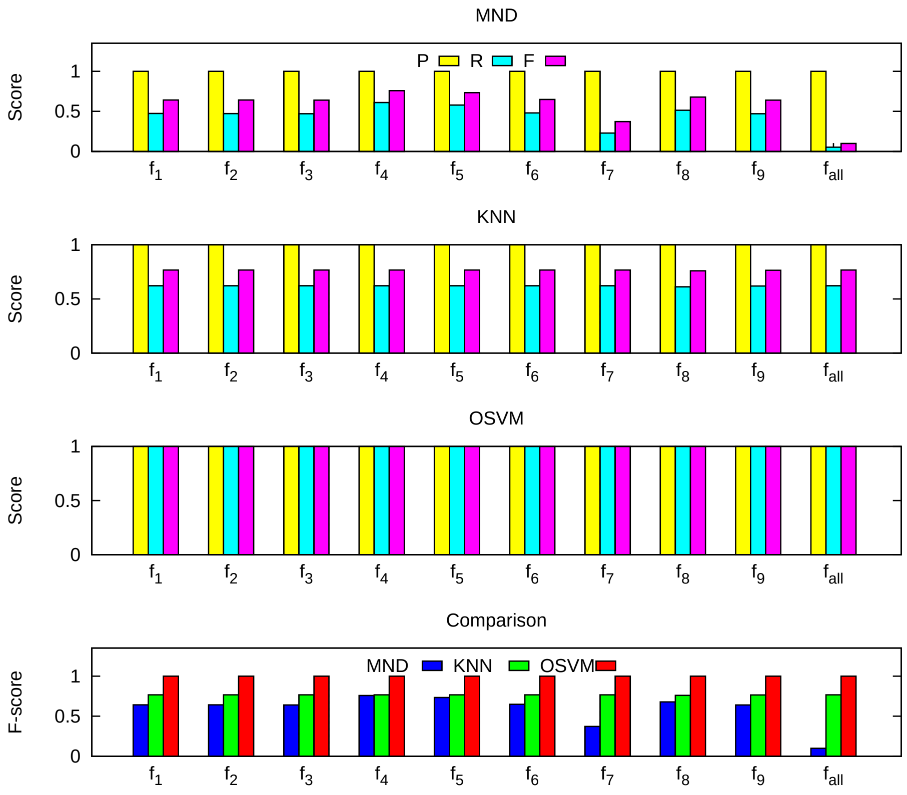

Figure 6.10: Detection performance of multi-timeline representation for one-day no packet incident as test data using the MND, KNN, and OSVM.

f_7 and f_{all} are relatively low. The second row from KNN shows that precision, recall, and F-score are nearly the same score for all features. The third row from OSVM shows that all precision, recall, and F-score are completely perfect for the no traffic packet in test data. Comparison in the last row shows that the efficient algorithms are OSVM, KNN, and MND respectively for all features, individual features and the combination of all features; however, the F-score of MND with the combination of all features (f_{all}) is the worst of all features and algorithms.

To compare results between using multi-timeline representation and real-time representation, we also performed the same experiments with real-time representation as early explained in Chapter 2.4.3. We show detection performance of real-time representation in Figure 6.11 with the same details as the multi-timeline representation shown in Figure 6.10. Each row in Figure 6.11 represents the following results. The first row from MND shows that precision scores are as high as those of the multi-timeline representation, but

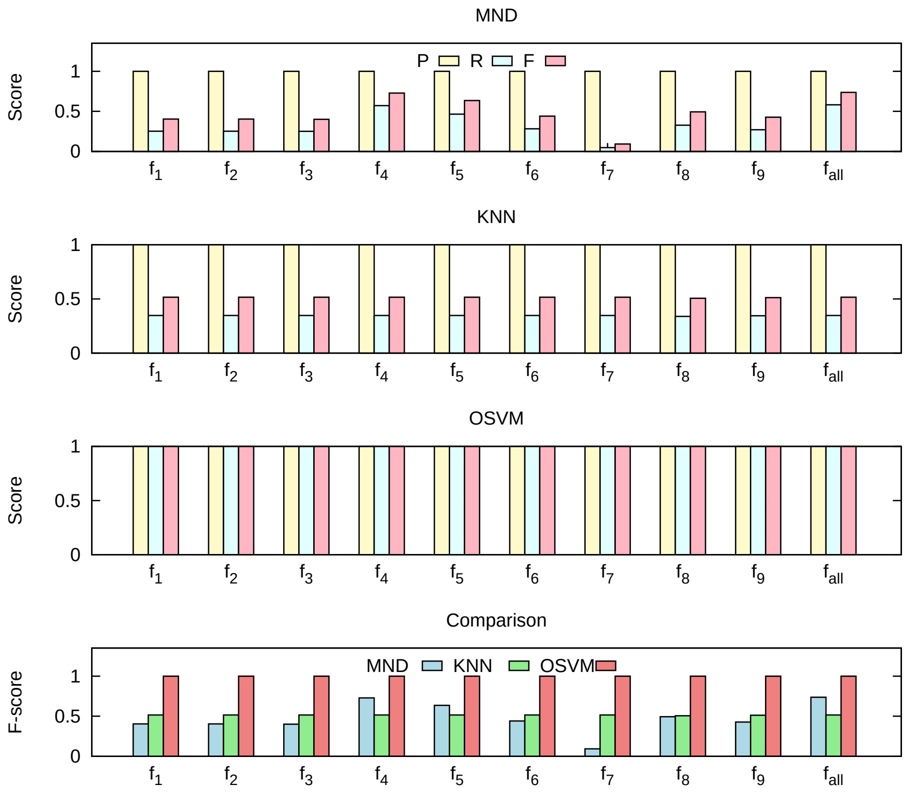

Figure 6.11: Detection performance on real-time representation for one-day no packet incident as test data using the MND, KNN, and OSVM.

only F-score of f_7 is relatively low. The second row from KNN shows that precision, recall, and F-score are nearly the same score for all features, but recall and F-score are quite lower than those of the multi-timeline representation. The third row from OSVM shows that precision, recall, and F-score are nearly perfect and very similar to results in Figure 6.10. We compared F-scores of MND, KNN, and OSVM for all features as shown in the last row. In this experiment, a highly effective algorithm is OSVM for both real-time representation and multi-timeline representation.

6.4.2 No Packet Incident in Training Data

For the second sub-experiment, we assumed that a network administrator accidentally combined the no packet incident into the original training data. Therefore, we can test and examine robustness of the multi-timeline representation in case of external influencing, including incorrect training data and any manipulation from attackers. At the beginning, we added the no packet incident for one day long into the original training data, and then we try to detect each type of attack with the same procedure in experiment 2, as explained in Chapter 6.2. After that we compared the results between using train data with no packet incident and the original train data without no packet incident as experiment 2. We then expanded the number of no packet incident from one day to three and five day long respectively into the original training data. In this experiments, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 39$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p|\mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}, 0, ..., 0), \tag{6.8}$$

where p is varied upon the number of training days β. Here the number of zero vector 0 at the end depend on the number of days of no packet incident. Finally, we computed relative percentage differences between using original training data with no packet incident and without no packet incident. We also compared these percentage differences between the multi-timeline representation and real-time representation as early explained in Chapter 2.4.3. All comparative results in this experiment are listed in Table 6.2-6.4.

Experiment results from MND, KNN, and OSVM are shown in Table 6.2-6.4 sequentially. For all three tables, the first column list types of attacks, and the next three columns represent percentage difference of the multi-timeline representation with no packet incident in training data for one day, three days, and five days long respectively. The next three columns represent percentage difference of the real-time representation with no packet incident in training data as well. The positive sign (+) of percentage indicated an increase, and negative sign (−) indicated a decrease of detection performance compared with original results from the experiment 2.

Table 6.2: Percentage difference in detection performance of MND between the multi-timeline and real-time representation.

Attack	Multi-timeline			Real-time		
	1 day	3 days	5 days	1 day	3 days	5 days
Back	-0.10%	-0.54%	-0.61%	-0.10%	+0.07%	+0.74%
IpSweep	-2.90%	-5.47%	-5.83%	-0.94%	-1.67%	-1.83%
Neptune	+0.60%	-0.50%	-1.16%	+0.49%	+1.20%	+1.92%
PortSweep	-1.07%	-2.37%	-2.60%	-0.36%	-0.75%	-0.87%
Smurf	+0.40%	+0.38%	+0.33%	+0.85%	+1.82%	+2.18%

Table 6.2, for the multi-timeline representation, shows that detection performance of MND declines when the number of no packet incident increase.

The downward trends in detection performances for all types of attacks are the same. Percentage differences for Back, IpSweep, PortSweep attacks are all negative, but performance for Neptune attacks is positive with no packet incident for one day long only. For Smurf attacks, there are all positive percentage but performance fall down when the number of no packet incident increased. For the real-time representation, this table shows detection performance is a small increase for Back, Neptune, and Smurf attack, but is a slight decrease for IpSweep and PortSweep. These results indicate that both the multi-timeline representation and real-time representation with MND are fairly robust from no packet incident in training data.

Table 6.3: Percentage difference in detection performance of KNN between the multi-timeline and real-time representation.

Attack	Multi-timeline			Real-time		
	1 day	3 days	5 days	1 day	3 days	5 days
Back	+0.15%	+0.41%	+0.41%	+0.16%	+0.45%	+0.45%
IpSweep	-0.01%	+0.01%	+0.01%	+0.01%	+0.04%	+0.04%
Neptune	+0.40%	+0.86%	+0.86%	+0.22%	+0.55%	+0.55%
PortSweep	+0.06%	+0.20%	+0.20%	+0.02%	+0.04%	+0.04%
Smurf	+0.01%	+0.04%	+0.04%	+0.38%	+0.86%	+0.86%

Table 6.3 shows that detection performances by using KNN between the multi-timeline representation and real-time representation are very similar. The detection performances of both representations has slightly improved when the number of no packet incident have been increased for all types of attacks. Detection performances with one day long of no packet incident are different from three day and five day long for all types of attacks. It seems that detection performances of KNN converge to a constant value for all type of attacks, although the number of no packet is rising. However, all percentages differences are a tiny change, less than 1 percent for both representations. This table indicates that both the multi-timeline representation and real-time representation with KNN are quite robust from no packet incident in training data.

Table 6.4 indicates that detection performances by using OSVM have many various trends when we added different numbers of no packet incident into the original training data. The table also shows that performances for Back, Neptune, and Smurf attacks increase when the number of no packet incident have been increased. While detection performance for IpSweep and PortSweep attacks randomly fluctuate. These happen similarly for both the multi-timeline and real-time representation.

Table 6.4: Percentage difference in detection performance of OSVM between the multi-timeline and real-time representation.

Attack	Multi-timeline			Real-time		
	1 day	3 days	5 days	1 day	3 days	5 days
Back	-0.36%	+1.33%	+1.73%	+0.28%	+0.34%	+0.34%
IpSweep	-1.12%	-3.14%	-0.32%	+0.01%	+0.15%	+0.03%
Neptune	-1.05%	+6.93%	+7.59%	+0.73%	+0.92%	+1.39%
PortSweep	-0.80%	-0.82%	-0.42%	-0.02%	+0.00%	-0.02%
Smurf	-1.78%	+4.25%	+6.86%	+1.17%	+1.80%	+2.23%

In conclusion for this section, we found two interesting points from these experimental results. The first point is that F-scores of OSVM for no packet incident in test data from the first sub-experiment are much higher than those for selected attacks as shown in Chapter 6.2. The main reason is that the OSVM is highly sensitive to noise, so nearly all of the predictions from OSVM are anomalies. Therefore, the OSVM produces high detection performance when a large portion of data is anomaly as shown in Figure 6.10 for the multi-timeline representation, and in Figure 6.11 for the real-time representation. These two figures also show that the multi-timeline representation by using MND and KNN outperform the real-time representation. The second point is that one of the intrinsic features of KNN is noise tolerance. For this reason, incorrect training data produce a relatively small effect on detection performance of KNN as shown in Table 6.3. While the MND has a negative effect as shown in Table 6.2 and the OSVM have a random effect from incorrect training data as shown in Table 6.4.

6.5 Experiment 5: Learning Curves

A learning curve is one of the graphical tools that show development of detection performance. A learning curve conventionally depicts improvement in performance on the vertical axis when there is alternation in another parameter on the horizontal axis, such as the number of training data or the number of iteration. One question in our concerns about the multi-timeline representation is that how many train data are fairly enough for learning algorithms to detect anomalies. Although the real-time anomaly detection does not require simultaneous learning with detecting, time period of learning process plays a crucial factor in multi-timeline representation. From machine learning perspective, it is absolutely true that the more training data acquired, the more detection performance. Therefore, we conducted this experiment to reveal learning curves of designated algorithms with multi-timeline representation.

The main objective of this experiment is to investigate how quickly all three algorithms learn from different size of training data to detect each type of selected attack. In this experiment, we fixed interval values $\delta = 10$ seconds and varied the number of training days β = 3, 9, 12,..., 39 days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p | \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.9}$$

where p is varied upon the number of training days β. First, we picked the best features of each learning algorithm for a particular type of attack, which gained the highest F-score (Eq5.6) from the experiment 2. Next, rather than learn from all 39-day training data, we let algorithms learn from a small amount number of training data, then added training data up until cover the whole training data. We started at training data for 3-day long and ended with training data for 39-day long, increasing the number of training data by 3 for each iteration. After that we let each learning algorithm detect specific type of attack with the same test data as those conducted in the experiment 2. We then plot learning curves of individual algorithms for a particular type of attack. Finally, we computed the average performance for each learning algorithm in order to compare overall learning curves of these three algorithms as shown in Figure 6.12.

In Figure 6.12, the y-axis shows the F-score or detection performance; of which 0 indicates the worst performance and 1 indicates the best performance. The x-axis indicates the number of training data over learning process, start from 3-day and end with all 39-day long of the training data. From top to bottom, the first three rows show learning curves of MND, KNN, and OSVM respectively. Different lines in the first three rows show learning curves on Back, IpSweep, Neptune, PortSweep, and Smurf respectively. The last row shows average learning curves of all three algorithms.

Results of this experiment show the following points. In the first row using MND, detection performances for Back, Neptune, and Smurf increase between 3-day and 9-day long of training data, and then have a small change, while detection performances for IpSweep and PortSweep seem to be a constant value along different size of training data. In the second row using KNN, only the detection performance for Neptune have a rapid increase from 3-day to 6-day long of training data, but one for other types of anomalies have a bit change along different size of training data. In the third row using OSVM, detection performances for all types of anomalies have some

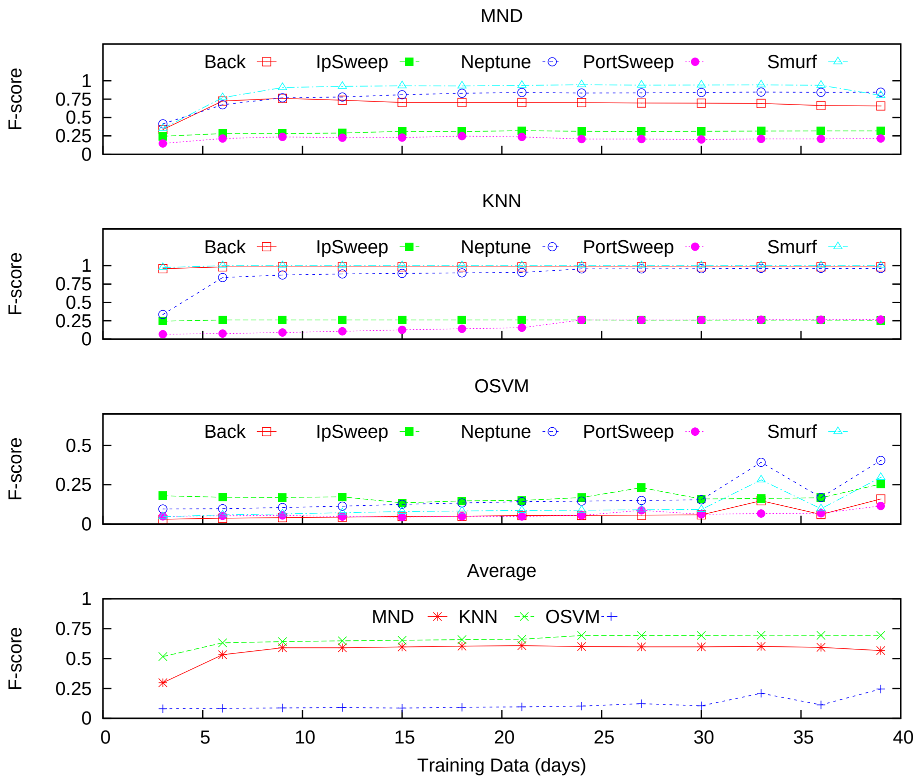

Figure 6.12: Learning curves of MND, KNN, and OSVM with different amounts of training data.

fluctuation from 24-day long of training data along; however, detection performances begin to improve after 30-day long of training data. The bottom row shows comparison between the three algorithms. The results indicate that detection performances of MND and KNN have a significant increase from 3-day to 9-day long then remain stable through 39-day long of training data. While the performance of OSVM increases after learning from 30-day long of training data.

6.6 Experiment 6: Time Consumption

Time consumption of learning algorithms working with the multi-timeline representation is one of the main issues of our investigation. For real-time anomaly detection, the detecting process requires a very short period of time to perform a function rather than the learning process. The definition of real-time detection in our context is that the system can detect and notify an occurrence of anomaly to network administrators in a few seconds or less than a minute. This means that the system has to perform tasks of detecting process less than a minute, while the learning process has no such restriction on computational time. Since we split the timeline for a single day into a series of intervals, the longest period in the detecting process between an occurrence of anomaly and notification is equal to the interval value plus processing time. The learning process, however, may take more than a minute or a longer time, because we could perform the learning process in advance.

The main purpose of this experiment is to measure time consumption of the three algorithms working with the multi-timeline representation for both learning and detecting process. In addition, we also can compare time consumption between all these algorithms. In this experiment, we measured time regardless the F-score or detection performance, so we can vary the number of training data and the number of features while we fixed the number of test data to one day long. We measured the time consumption during learning and detecting process with the following steps. First, we started with five features and five days of training data, we then measured time in both learning and detecting process. Next, we varied the number of features and the number of training data from 5 to 100, increasing the number by 5 for each iteration, in the meanwhile we measured time consumption for the both process as well. To do so, we simulate data by repeating and appending original data, features to themselves.

In this experiment, we fixed interval values δ = 1, 10, 20, 30, 40, 50, 60 seconds and varied the number of training days β = 5, 10, 15,..., 100 days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p|\mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.10}$$

where p is varied upon the number of training days β. We used these configuration for preprocessing, learning, and detecting processes.

6.6.1 Preprocessing

One of the key factors for real-time anomaly detection is time consumption, especially during preprocessing steps. Therefore, we conducted this experiment to measure time consumption of these steps. However, we performed this experiment without concern over detection performance or F-score. In this experiment, we varied three significant parameters which have a major effect on time consumption, namely the number of training data, the number of features, and the interval value. We altered the number of training data

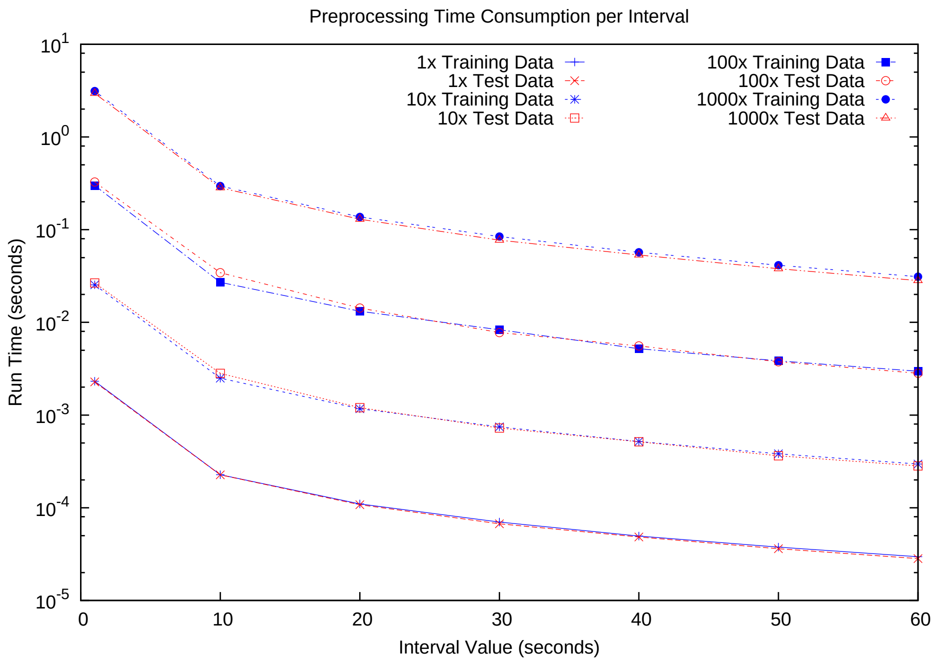

Figure 6.13: Time consumption per time interval in preprocessing processes.

and the number of features between 1 to 9, and the interval value from 1 to 60 seconds, with one day of test data during the test process.

Firstly, we measured the average time consumption of preprocessing steps for real training and test dataset in this experiments. Figure 6.13 shows time consumption per interval with 9 features over preprocessing steps for varying size of interval values in training data and test data between original (1x) and 1,000 times of background traffic (1000x). For the one-second interval and combined all nine features with original background traffic (1x), the average time consumption of preprocessing steps for training data is 2.32 milliseconds, while those for test data is 2.28 milliseconds per interval. For 1,000 times of background traffic (1000x), time consumption for training data is 3.13 seconds, while those for test data is 2.96 seconds per interval. These times consumption are measured from the end of each interval, therefore each interval spent 1.00232 and 1.00228 seconds for training and test data respectively. The minimum, maximum, and average packet of the experimental traffic are 1 packet/sec., 750 packets/sec., and 26.191 packets/sec. successively. Interval time consumption of preprocessing for training and test data seems very close, but the number of training data is more larger than those of test data. Eventually, preprocessing steps for training spend time depened on the number of training days as well.

6.6.2 Learning Process

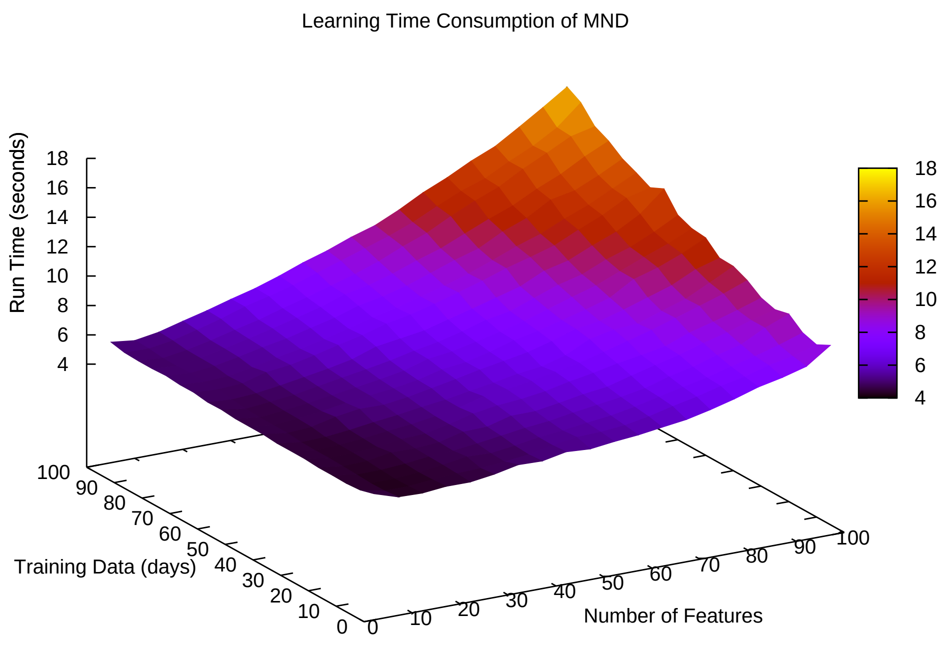

Figure 6.14: Time consumption of MND in learning process for varying size of training data and features.

We show time consumption result of MND over learning process in Figure 6.14. With five days of training data and five features, the MND took 4.88 seconds. When we varied the number of training data to one hundred days with fixed five features, the algorithm learned from all training data took 5.22 seconds. On the other hand, at five days of training data with one hundred features, the algorithm took 9.21 seconds. When we reached one hundred of both training data and features, the algorithm learned from training data took 16.81 seconds.

Interestingly, when we altered the interval value and done the same process with various number of training data and features, the graph of time consumption with different interval values are the same shape but the position of the graphs shifted up or down from the original at 10 seconds,

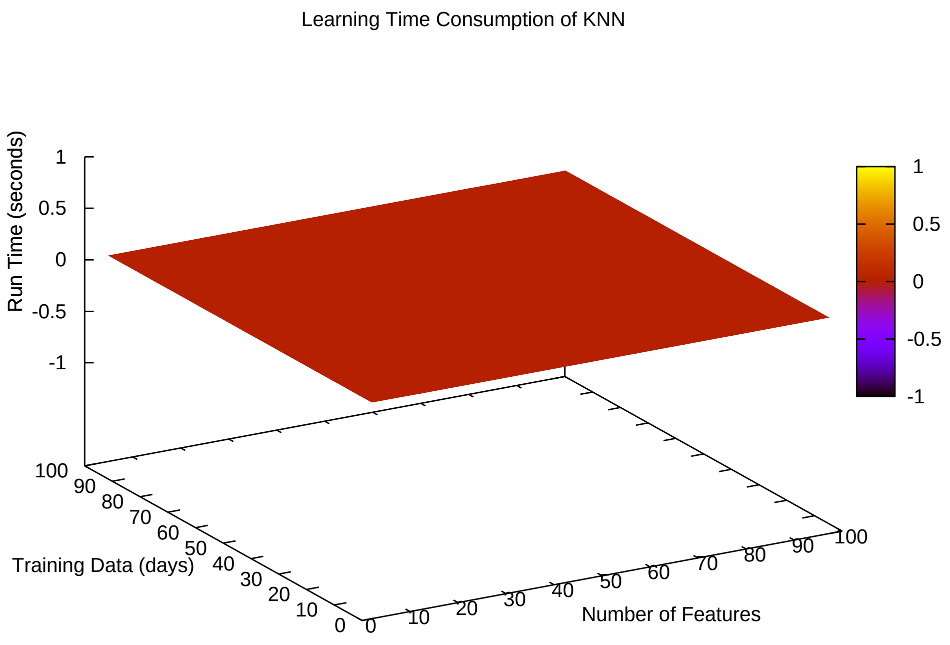

Figure 6.15: Time consumption of KNN in learning process for varying size of training data and features.

depended upon the interval value. If the interval value is smaller (1 second for example), the graph shift up from the original position. It means that the algorithm spent more time than the interval value as 10 seconds. If the interval value is larger (20, 30, 40, 50, and 60 seconds for example), the graph shift down from the original position. It means that the algorithm spent less time than the interval value as 10 seconds. However, we did not show results of different interval values here.

We show time consumption result of KNN on learning process in Figure 6.15. Interestingly, whether we varied a number of parameters during learning process, such as the size of training data, the number of feature, and the interval value, the KNN did not take computational time to process training data at all. The result from Figure 6.15 shows a horizontal plane at zero time consumption, no matter what the size of training data and the number of features are. Although we altered the interval with different values, the results are still as the original plane at the zero value, the graph did not shift up or down, it is different from results of MND and OSVM.

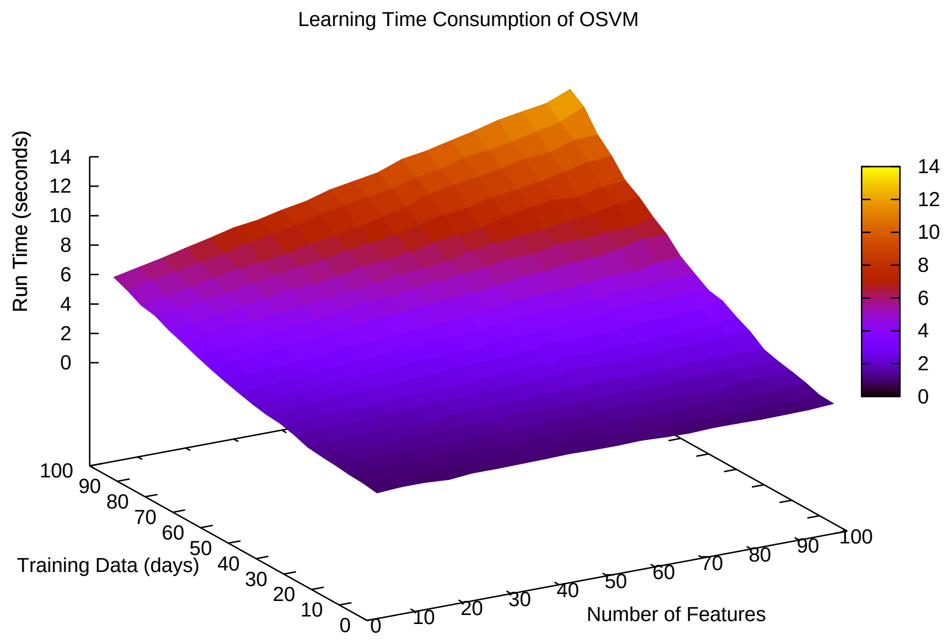

Figure 6.16: Time consumption of OSVM in learning process for varying size of training data and features.

We should recall about time measurement in our experiments. We measured time consumption over the learning process and detecting process alone, it does not include any other processes, which are before or after these steps, such as feature extraction process or feature scaling process. We will discuss the main reasons why the KNN does not take any time during the learning process in Chapter 7.

We show time consumption result of OSVM over training process in Figure 6.16. The algorithm took 0.83 seconds at five days of training data with five features. When we varied the number of training data to one hundred days but keep the number of feature at five features, the OSVM took 5.53 seconds. While we have the fixed number of training data at five days but varied the number of feature to one hundred features, the algorithm took 1.14 seconds. The maximum time consumption of OSVM is 12.53 seconds at one hundred days of training data with one hundred features. Obviously, time consumption of OSVM over training process depends upon both the number of training data and the number of features.

Moreover, for different interval values, the graph as shown in the figure shift up or shift down depended on interval value. The graph shift up when we choose smaller interval values, 1 second for example, and the graph shift down when we choose larger interval values, 60 seconds for example. However, we did not show shifted graphs for different interval values in Figure 6.16.

6.6.3 Detecting Process

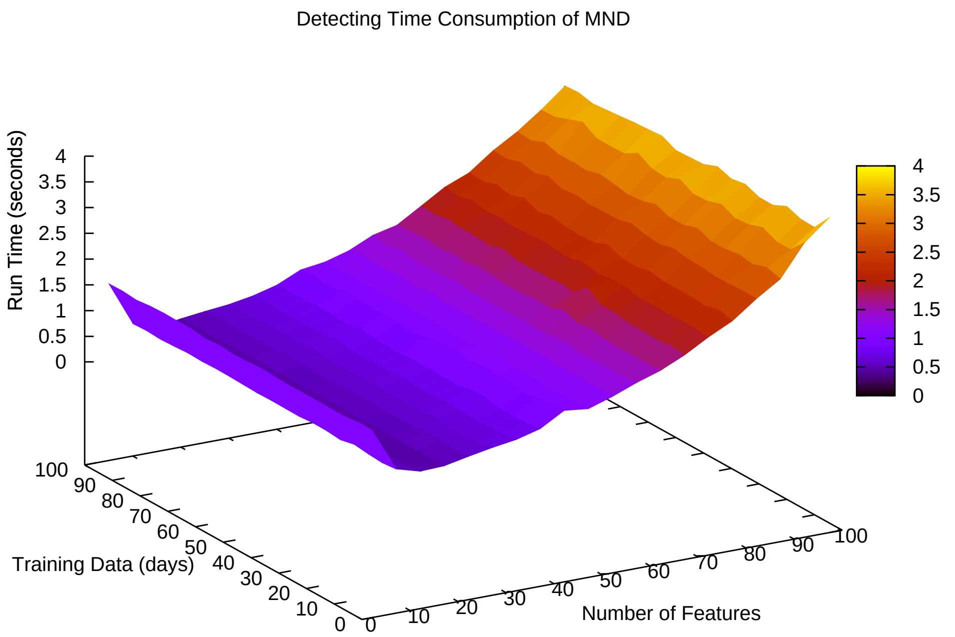

Figure 6.17: Time consumption of MND in detecting process for varying size of training data and features.

We show time consumption result of MND over detecting process in Figure 6.17. At five days of training data and five features, the MND took 1.44 seconds. When we varied the number of training data to one hundred days with five features, the algorithm detect anomalies in one day test data in 1.43 seconds, result is very close to that of five days of training data. After we varied the number to one hundred features with five days of training data, the algorithm took 3.90 seconds. Finally, when we used the maximum number with one hundred days of training data and one hundred features, the

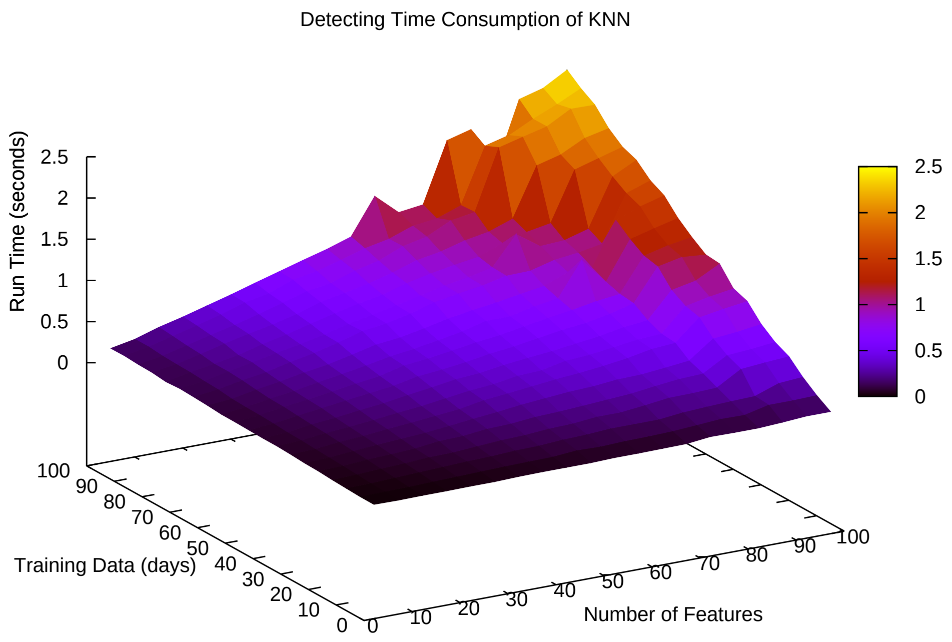

Figure 6.18: Time consumption of KNN in detecting process for varying size of training data and features.

algorithm took 3.63 seconds to detect test data for one day long. Obviously, no matter we varied the number of training data with the same number of features, it has a very small effect on the time consumption that the MND spend to detect one day test data.

In addition, when we altered the interval value, results are the same fashion as in the training process. The graph shift up when we set smaller interval values, and the graph shift down when we set larger interval values. In Figure 6.17, however, we did not show time consumption graphs of various interval values.

We show time consumption result of KNN during the detecting process in Figure 6.18. The time consumption over detecting process using the KNN is a very short period. The maximum time at one hundred days of training data and one hundred features is only 2.47 seconds, while the minimum time at five days of training data and five features is about 0.01 seconds. Figure 6.18 shows that the time consumption increases linearly along the number of training data and the number of features. The time consumption had

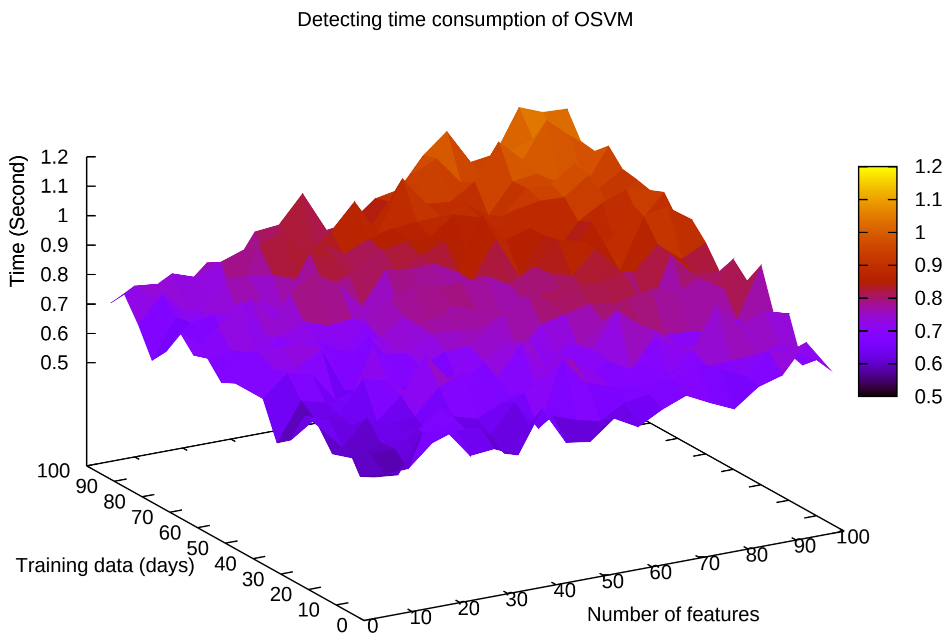

Figure 6.19: Time consumption of OSVM in detecting process for varying size of training data and features.

abrupt changes when the number of training data and number of features are greater than 70. Nevertheless, overall of time consumption by using KNN over detecting process is generally low, the algorithm took 0.12 seconds at one hundred days of training data with five features, and took 0.10 seconds at five days of training data with one hundred features.

We can see a clear distinction between the MND and KNN results during detecting process from Figure 6.17 and Figure 6.18. The MND is independent of the number of training data over detecting process, while the KNN depends on the number of training data.

We show time consumption result of OSVM over detecting process in Figure 6.19. Interestingly, this figure shows that the shape of time consumption by using OSVM is completely different from other results in this experiment. Even though the result show a random time consumption along x and y axis, the time consumption somehow gradually increased when we varied the number of training day and the number of feature toward larger values. During detecting process, the OSVM took 0.59 seconds by using five days of training data with five features, 0.67 seconds by using five days of training data with one hundred features, 0.68 seconds by using one hundred days of training data with five features, and 1.06 seconds by using one hundred days of training data one hundred features.

In addition, for different interval value, the graph also shifted up or down from the original position similar to the results by using MND and KNN. The graph shifted up by using smaller interval value; on the contrary it shifted down by using larger interval value than the original at 10 seconds. However, we did not show the results using different interval values in this figure. An effect of interval value on graph position is the same fashion as all graphs from previous results, except the graph of KNN during detecting process.

For all algorithems, the time between anomaly occur and alert rely on time interval and processing time after that as

$$T(\mathbf{x}) = \delta + d(\mathbf{x}), \tag{6.11}$$

where $T(\mathbf{x})$ is total processing time when an anomaly occur at interval x, δ is an interval value, and $d(\mathbf{x})$ is detecting time processed at interval x. We set the interval value $\delta = 10$ seconds for this experiment, so the number of interval for one day is 8,640. For MND, the average processing time for each interval in detecting process $T(\mathbf{x}) = 10 + (3.63/8640) \approx 10$ seconds. For KNN, the average processing time for each interval in detecting process $T(\mathbf{x}) = 10 + (2.47/8640) \approx 10$ seconds. For OSVM, the average processing time for each interval in detecting process $T(\mathbf{x}) = 10 + (1.06/8640) \approx 10$ seconds. These results imply that the total processing time for each interval in detecting process approximately equal to the interval value δ. They satified realtime of our definition that the system raises an alarm after anomalies occur less than 60 seconds.

6.7 Experiment 7: Different Volumes of Background Traffic

Volume of background or normal traffic is one of our concerns, because the normal traffic generally affects the performance of detection systems. In small network systems, the volume of anomalies traffic is normally larger than normal traffic and we are easy to notice them manually, so detection systems also easily perceive an occurrence of anomaly. In large network systems, however, the volume of anomalies traffic is relatively small compared to normal traffic, many of detection systems suffer from this situation. From early experiments, we found that the multi-timeline detection system can discover several anomalies with quite high of F-score (Eq.5.6) or detection performance. Therefore, we hypothesize that the detection performance of multi-timeline system should be declined when we employ the proposed system to different sizes of computer network.

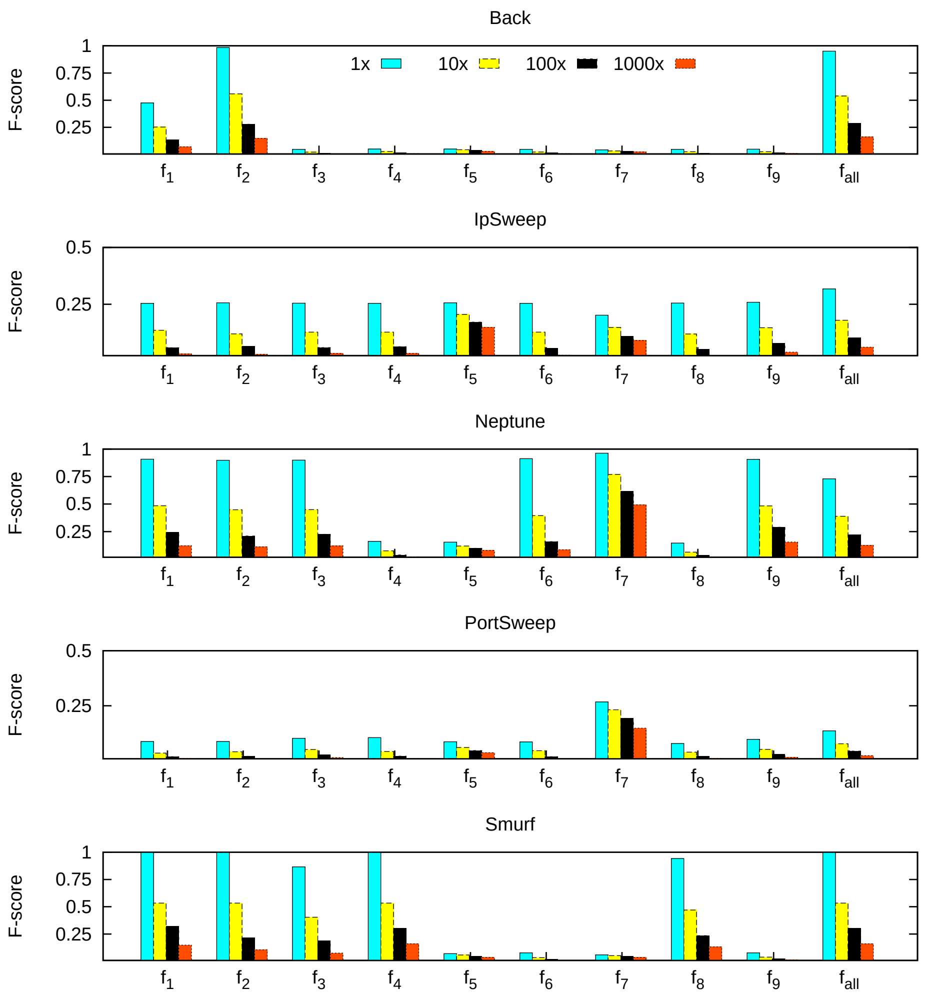

Figure 6.20: Comparison of F-score from the original background traffic (1x) to 1,000 times background traffic (1000x).

The main objective of this experiment is to examine the F-score or detection performance of the multi-timeline detection system on four different sizes of networks. In this experiment, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 5, 10, 15, ..., 100$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p|\mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.12}$$

where p is varied upon the number of training days β. First, we selected one by one feature for each type of anomaly similar to experiment 2, then carried out a detection task on the original data as explained in Chapter 5. The results from original data showed the highest F-score for each type of anomaly. Second, we generated a new background traffic based upon the original data by multiplying the volume of original data 10 times, then employed each feature for a single type of selected anomaly again. For all types of anomalies, we did not change the volume of traffic because we planned to compare F-scores of the same size of anomaly traffic over different volumes of background traffic. Next, we generated new network traffic by multiplying 100 and 1,000 times sequentially and measured the F-score for each several size of background traffic.

Detection performances of the multi-timeline system over different sizes of background traffic are shown in Figure 6.20. We found that the performance of multi-timeline detection system have been exponentially decreased when the volume of background traffic increase. However, F-scores by using the number of destination addresses (f_5) and the number of destination ports (f_7) are slowly decreased than those by using other features. The main reason is that both features ,the number of destination addresses and ports, have changed relatively small when the size of background traffic has been increased. Consequently, the multi-timeline detection system would be applied with any features in low-volume of network traffic, such as in access networks. On the other hand, we should employ features which does not be diverse in high-volume of network traffic, such as in core networks.

6.8 Experiment 8: Time of Anomaly Occurrence

For the multi-timeline detection system, we intend to investigate an hypothesis that the time of anomaly occurrence has an effect on detection performance. Normally, the volume of network traffic has change during a day, start at a small amount of traffic and gradually increase until reach the peak of the day then slowly decrease. We easily detect anomalies occurred at a small background traffic rather than those anomalies occurred at a heavy background traffic of the day. It means that the time of anomaly occurrence produces an effect on detection performance of the multi-timeline detection system. Therefore, we have to examine detection performance of the proposed system when anomalies occur at different time periods of the day.

In this experiment, we simulated that selected anomalies occur at the begin of each hour from 9am to 11pm. As a result, each type of anomaly appeared 15 times along one day of test data. We employed the top 3 features from the experiment 2 for each type of anomaly, and detect them one by one. We then evaluated the F-score (Eq.5.6) on different time perios that anomalies occure. The final outcomes from this experiment would reveal trends in F-score over time periods of a day that each anomaly occur.

In this experiment, we fixed interval values $\delta = 10$ seconds and the number of training days $\beta = 39$ days. We also used decision function for multi-timeline representation

$$g(\mathbf{x}_t^p | \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_t^{p-1}), \tag{6.13}$$

where p is varied upon the number of training days β.

Figure 6.21 shows F-scores of top 3 features detected each anomalies along time periods in one day. We found that the performance of multi-timeline detection system reverses variation in volume of network traffic. The detection performance of multi-timeline detection system quite high around the beginning and the end of the day, while the detection performance relatively low at the middle of the day. The main reason is that network traffic around the beginning and the end of the day are lower than those at the middle of the day. However, the performances of multi-timeline detection system at the middle of the day are small variation compared to the performance at the beginning and at the end of the day. From the results in this experiment, we can conclude that the time of anomaly occurrence has a small effect on the performance of multi-timeline detection system.

6.9 Experiment 9: Weighting for Old Data

We conduct the last experiment to explore detection performance of both the multi-timeline detector module with and without weighting process. For several of network systems, operators need to give some recent traffic more weight or influence on the result than old traffic in the same set of data. The multi-timeline detection system also provides such function to do so. We try employing a weighting process on the multi-timeline detector module and comparing detection performance to those without the weighting process.

The following steps show how we perform experiments with a linear gradual weighting technique as explained in Chapter 4. First, we set the weight length and weight value as 1 for weighting process. Second, we select the best feature from experiment 6.2 for each type of anomaly by using first

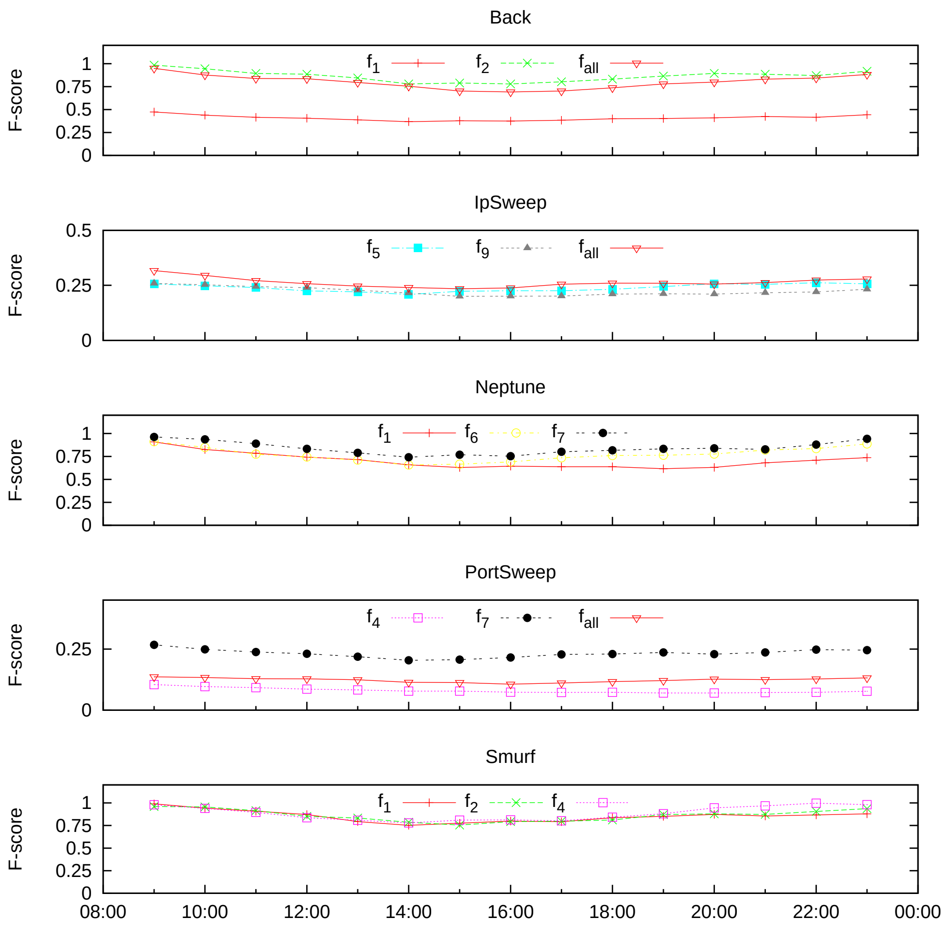

Figure 6.21: Detection performance of selected anomalies over different time occurences by using top 3 features.

learning algorithm, MND. After that we measure F-score (Eq.5.6) for every types of anomalies and compute the average performance of multi-timeline detector module for MND. Third, we alter the weight length $\phi = 1, 5, 10, 15, 20, 25$ days, then average detection performance for all weight length. Next, we change the weight value $\varphi = 1, 3, 5, 7, 9$ then follow the procedure from the first to third step and compute the average detection performance for all value. Finally, we switch the learning algorithm from MND to KNN and OSVM respectively, and plot all computed average values on three-dimensional graphs to compare trends in detection performance between these learning algorithms.

In this experiment, we fixed interval values $\delta = 10$ seconds and the number

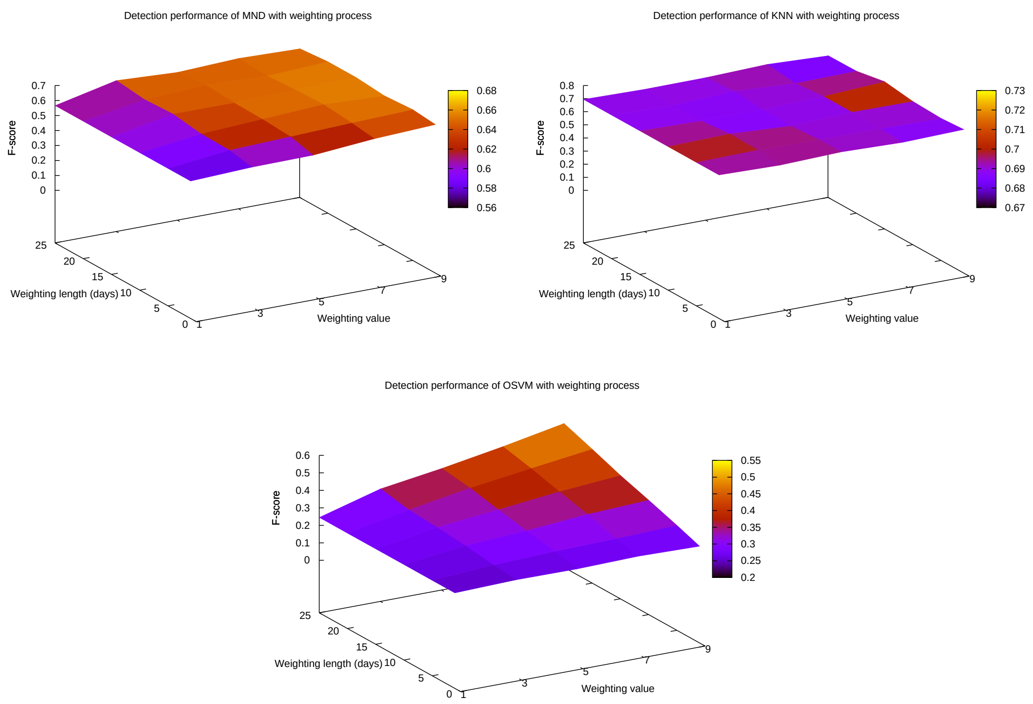

Figure 6.22: Detection performance of multi-timeline module with weighting process by using MND (upper left), KNN (upper right), and OSVM (bottom).

of training days $\beta = 39$ days (exclude weighting). We also used decision function for multi-timeline representation like

$$g(\mathbf{x}_t^p | \mathbf{x}_t^1, \mathbf{x}_t^2, ..., \mathbf{x}_{\varphi-1}^{\phi}, \mathbf{x}_{\varphi}^{\phi}), \tag{6.14}$$

where p is varied upon the number of training days β. We shows the relationship between the weight length ϕ and weight value φ in Figure 4.1.

Performance results from this experiment on weighted timeline are shown in figure 6.22. This figure contains three graphs, the detection performance of multi-timeline module by using MND (upper left), KNN (upper right), and OSVM (bottom). The x-axis represents the weight values from 1 to 9, while the y-axis indicates the weight length during the learning process. The z-axis shows the F-score that contains a value between 0 and 1, where 0 represents the worst and 1 represents the best detection performance.

Experimental results in figure 6.22 show some issues for three learning algorithms. Results from MND (upper left) show that the multi-timeline detector module with the weighting process produces 3.2-16.6% improvement than the module without the weighted process. In case of KNN (upper right), the graph indicates that adding the weighting process randomly produces a small effect between 2.7-6.4% improvement. Employing OSVM (bottom), however, the trend in detection performance is quite different from those by using MND and KNN. The detection performance with OSVM is linearly upward when we increase the weight values and weight length. The improvement of performances by using OSVM are between 3.5-112%. From these three figures, OSVM is a suitable algorithm for the multi-timeline detector module with the linear gradual weighting technique.

Chapter 7

Discussion

In this chapter, we will raise issues from overall our experiments. We then compare different learning algorithms and representation of input data. In the last section, we will point out limitation of the multi-timeline representation in the task of anomaly detection.

7.1 Effects of Different Interval Values on Real-time Detection

We discovered that the interval value has major effects on accuracy of detection performance and time consumption in both learning and detecting processes. In our study, we conducted two experiments which get involved with the interval value; the experiment 1: comparison of different interval values and experiment 5: time consumption. The primary purpose of experiment 1 is to identify a suitable interval value for subsequent experiments, and the main purpose of experiment 5 is to measure time consumption of learning algorithms in the learning and detecting process. In both experiments, we altered the interval value, which is one of the crucial factors for the multi-timeline representation, by using seven different values: 1, 10, 20, 30, 40, 50, and 60 seconds.

Our experimental results strongly suggest that setting the interval value is one of the key factors in the multi-timeline representation. The best time interval value strongly depends on the duration of target anomalies. If target anomalies suddenly change and happen in a short duration, a small value such as 1 or 10 seconds is the feasible interval time for the multi-timeline representation. On the other hand, if target anomalies slowly change and occur over a long time duration, a large value from 30 to 60 seconds is the feasible interval time for detecting such anomalies. We should firstly consider the interval value and learning algorithm together rather than other factors, because both of these factors have a huge impact on time consumption, especially for real-time anomaly detection. One of the following sections will explain the time complexity of each learning algorithm which have been selected to work with the multi-timeline representation. In this study, we found that the interval value at 10 seconds is a feasible interval value for our data, so for the rest of our experiments we always set the interval value at 10 seconds.

We consider the multi-timeline representation as detection on the basis of time interval rather than packet or flow. Therefore, the interval value also have an effect on any process after anomalies been discovered. In accordance to this representation can identify what interval or when an anomaly happen, the interval value have an effect on a fine-grain inspector to specify what type of anomaly occur in network for example. In our study, however, we did not have and conduct experiment on such a fine-grain inspector.

7.2 Feasible Features for Detecting a Particular Anomaly

We discovered that various features have substantial effects on detection performance over different types of anomalies. We performed the experiment 2 to compare detection performance between each individual of nine features on selected types of attacks. In this experiment, we also measured detection performance by using a combination of all nine features, then compare to results by using a single feature. We found that learning algorithms produce a good performance when we apply network features that closely correlate with a specific type of attack.

Table 7.1: Feasible features for experimental attacks.

Attack	Feature
Back	Packet(f_1), Byte(f_2), Combined All(f_{all})
IpSweep	Combined All(f_{all})
Neptune	All except SrcAddr(f_4), DstAddr(f_5), ΔAddr(f_8)
PortSweep	DstPort(f_7), Combined All(f_{all})
Smurf	All except DstAddr(f_5), SrcPort(f_6), DstPort(f_7), ΔPort(f_9)

All results of the three algorithms suggest that feasible features for experimental attacks are the same. However detection performances on the same feature are fairly dissimilar for different algorithms. We can summarize the feasible features for our experimental attacks as shown in Table 7.1. Therefore, we can detect new attacks or anomalies that conform with the experimental attacks by using these feasible features. In summary, we have to carefully select features which highly correlate with types of anomalies. For example, for denial of service attack, we should use the number of packet, flows, or the sum of packet size as a feature. Similary for port scanning activity, we should use the number of destination port as a feature.

7.3 Robustness to Extreme Conditions and Negative Influences

An essential characteristic of the multi-timeline representation is an ability to resist extremely conditions and negative influences from outside. To verify this capability, we conducted two separated parts in the experiment 3, with no packet incident in test data and no packet incident in training data. In the first part, no packet incident in test data, is to examine the capability of multi-timeline representation in an extreme condition, and in the second part, no packet incident in training data, is to examine detection ability when incorrect data have been contaminated into the learning process.

Both parts in the experiment 3 suggest that the multi-timeline representation with a carefully selected algorithm is more likely to tolerate extremely conditions and negative influences. Most of techniques for anomaly detection in network traffic does not have such characteristics. These characteristics play a crucial role in anomaly detection because if the detection system does not have an ability to tolerate extreme condition, attackers or intruders might change the target to the detection system by attacking via this vulnerable point. Moreover, if attackers or intruders realize the detection technique with a lack of robustness, they might develop a technique to compromise system integrity, and then they can penetrate into the network system or easily evade the system as well.

7.4 Increase of Learning Curve with the Number of Training Data

We conducted the experiment 4 to examine the increase of learning curves among three selected algorithms with the multi-timeline representation. We found that two of three algorithms with the multi-timeline representation have a capability to learn in a short period of time before detecting anomalies with satisfied performance. In this examination, we varied the number of data during learning process and measured detection performance in order to show a learning curve of each algorithm.

Learning curves of MND and KNN rise sharply from 3 days to 9 days of training data and then slowly change after on, but learning curves of OSVM start to increase when the number of training data is more than 30 days. Learning curves of OSVM appear to be similar results for both easy and difficult detectable attacks. This summary results show that outstanding learning curves are KNN, MND and OSVM respectively. The MND and KNN mostly learned only from 3 days to 9 days before they can detect most types of attacks with a promising performance, while the OSVM took more than a month to learn data before detection performance begin to increase.

7.5 Time Consumption for Learning and Detecting Process

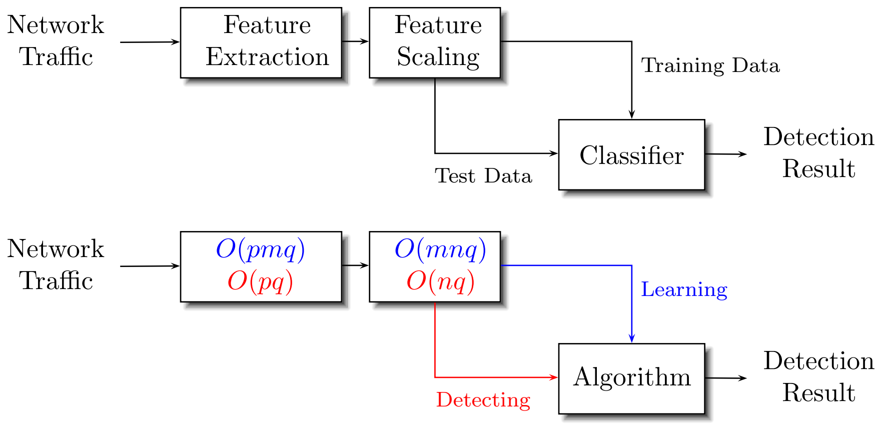

Figure 7.1: Time complexity of preprocessing step.

Time complexity of preprocessing step is one of our concerns for overall system performance. Figure 7.1 indicates time complexity on an interval basis for each step in system procedure. From our analysis, we found that time complexity of the preprocessing step for learning process slightly different from those for detecting process. For the feature extraction, time consumption relies on p the number of packets per interval, m the number of training data (days), and q the number of features for extraction. As a result, time complexity of feature extraction for is $O(pmq)$ for learning process, and $O(pq)$ for detecting process. For the feature scaling, time complexity of learning process is $O(mnq)$, while those for detecting process is $O(nq)$, where n is the number of interval. In our experiment, the feature extraction spent approximately 16 seconds, and the feature scaling spent approximately 12 secords for a one-day long. After preprocessing step, time complexity depends upon the algorithm as shown in Table 7.2. For real-time anomaly detection, we mainly focused on time constraint for detecting process. Therefore, we consider the time complexity of algorithm for detecting process to be the main system performance, because the preprocessing step has the same time complexity for any algorithm. However, we can apply vectorization to preprocessing step, learning process, and deceting process in order to reduce time complexity.

Table 7.2: Computational time complexity for one-day test data.

Algorithm	Learning Process	Detecting Process
MND	$O(mnq)$	$O(nq)$
KNN	$O(1)$	$O(mnq)$
OSVM	$O(m^2nq^2)$	$O(nqs)$

From mathematical analysis point of view, all of the learning algorithms in this experiment have computational time complexity as listed in Table 7.2. This table displays information in three columns: learning algorithm, time complexity of learning process, and time complexity of detecting process. Here m is the number of training data (days), n is the number of intervals in one day long, q is the number of features, and s is the number of support vectors which relies on pattern of training data. This table shows that the MND run in linear time proportionally with m, n, and q for learning process, and proportionally with n and q for detecting process; the KNN run in constant time regardless of the input size for learning process, but run in linear time proportionally with m, n, and q for detecting process; the OSVM run in quadratic time with regard to m and q, and run in linear time proportionally with n for learning process, while run in linear time proportionally with n, q, and s for detecting process [103].

For all algorithems over detecting process, the time between anomaly occur and alert rely on time interval and processing time after that as

$$T(\mathbf{x}) = \delta + d(\mathbf{x}), \tag{7.1}$$

where $T(\mathbf{x})$ is total processing time when an anomaly occur at interval x, δ is an interval value, and $d(\mathbf{x})$ is detecting time processed at interval x.

The most interesting in this experiment is time consumption of OSVM for both learning and detecting process. For learning process, Table 7.2 shows that the OSVM run in quadratic time with regard to m and q, but it seems likely that the OSVM run in linear time proportionally with m and q. The main reason is that the numerical library for OSVM has the potential for what we called vector computation, so that we could reduce computational time of OSVM during learning process from quadratic time to linear time. For detecting process, one of the input factors which affects time consumption is the number of support vectors s. However, the number of support vectors relied on the pattern of input data, which is also in regard to the number of training data. It means that even if the number of training data is the same, it might produce a different number of support vectors. Therefore, the results of OSVM for detecting process appears randomly because of s, the number of support vectors.

Table 7.3: Overall time consumption per time interval for original background traffic (1x).

Process	Extraction	Scaling	Algorithm	Total
Learning	1.97 ms	0.35 ms	2.08 ms (MND)	4.40 ms
			0.00 ms (KNN)	2.32 ms
			1.62 ms (OSVM)	3.94 ms
Process	**Extraction**	**Scaling**	$T(\mathbf{x}) = \delta + d(\mathbf{x})$	**Total**
Detecting	1.96 ms	0.32 ms	10 s + 0.46 ms (MND)	$\approx$ 10 s
			10 s + 0.29 ms (KNN)	$\approx$ 10 s
			10 s + 0.14 ms (OSVM)	$\approx$ 10 s

Table 7.4: Overall time consumption per time interval for 1,000 times of background traffic (1000x).

Process	Extraction	Scaling	Algorithm	Total
Learning	3.13 s	0.37 ms	2.19 ms (MND)	$\approx$ 3.13 s
			0.00 ms (KNN)	$\approx$ 3.13 s
			1.86 ms (OSVM)	$\approx$ 3.13 s
Process	**Extraction**	**Scaling**	$T(\mathbf{x}) = \delta + d(\mathbf{x})$	**Total**
Detecting	2.96 s	0.35 ms	10 s + 0.49 ms (MND)	$\approx$ 12.96 s
			10 s + 0.25 ms (KNN)	$\approx$ 12.96 s
			10 s + 0.19 ms (OSVM)	$\approx$ 12.96 s

Table 7.3 shows overall time consumption per interval for original background traffic (1x). In this experiment, the average and maximum number of background packet are 26.191 and 750 packets/sec, the amount of features is 9, and the number of training day is 39 days. Time consumption during preprocessing steps, namely feature extraction and scaling, are common for all three algorithms. This table shows time consumption on interval basis, but the learning process spends more time for the whole training data. However, we concerns only detecting process in real time, and the results show that the multi-timeline system can perform anomaly detection in a few milliseconds. Even if background traffic is 1,000 times of our experiments, the detecting process still consumes in a few seconds.

Table 7.4 shows overall time consumption per interval for 1,000 times of background traffic (1000x), where the average and maximum number of background packet are 26,191 and 750,000 packets/sec. This table shows that time consumption of scaling step, learning, and detecting algorithm are similar between 1x and 1000x of background traffic. For both learning and detecting steps, most time consumption for 1000x of background traffic have been spent in the extraction process. Therefore, the bottleneck of multi-timeline detection system is feature extraction, because this process relates to the number of packet per interval. This table also shows that the feature extraction step is a bottleneck of the proposed system because this step takes more time when background traffic increases.

From our definition, these two tables suggest that the multi-timeline system can detect anomalies in real time. For learning process, table 7.3 shows that overall time consumption by using MND is 4.40 ms $\times$ 8640 intervals $\times$ 39 days $\approx$ 24.7 minutes, and table 7.4 shows that overall time consumption by using MND is 3.13 s $\times$ 8640 intervals $\times$ 39 days $\approx$ 293 hours. However, learning process can be performed in advance, so we concern more over detecting process. Detecting process for all algorithms, table 7.3 shows that overall time consumption for each interval is approximately 10 s, and table 7.4 shows that overall time consumption for each interval is approximately 12.96 s. These results imply that the time consumption in detecting process of multi-timeline system less than 60 seconds and the system can detect anomalies in real time.

Our experiments show that the proposed system detects anomalies as a real-time system with the following sizes. The proposed system gains detection performance nearly the acceptable F-score 0.5 for 10x of background traffic, where the average and maximum number of background packet are 261.91 and 7,500 packets/sec. Here, the average ratio between anomaly and normal traffic be detected by the system is 0.25. At this size of network, the system approximately takes 10 seconds from extraction step to detect anomalies. All of these imply that the proposed system is suitable for LAN or access networks for real-time detection.

7.6 Detectability over Different Background Traffic and Multiple Anomalies

We concern about applying the detection system in a real network traffic and detecting a real situation with multiple anomalies. Therefore, we also perform two experiments to evaluate the multi-timeline detection system in those environments. We simulate experiments by replicating different background traffic, and altering both single and multiple anomalies along various intervals of the day. These two experiments would demonstrate how the multi-timeline function on different background and multiple anomalies in real network traffic.

Detection results show some interesting issues if we apply the multi-timeline detection system into different network environment. Performance of most features in multi-timeline detection system drop dramatically when background traffic increase, while detection performance with some features, such as the number of destination addresses and ports, slowly decrease in huge background traffic. Altering background traffic from 394k to 394Mbit/sec, detection performance of these two features declined approximately 50%, while those of other features drop more than 80%. As a result, we could employ the number of destination addresses and ports with multi-timeline system into enormous traffic, backbone networks for example.

Detection results for single and multiple anomalies are not much different. Nonetheless, a common result from both types of anomalies is detectability during interval of the day. Detectability of both types are high in early morning and nearly midnight where normal traffic is low, but detectability of those gradually decrease during at noon where normal traffic is peak. However, detectability of multi-timeline system during the day fluctuate around 20-30%. Empirical results suggest that detectability of multi-timeline detection system are very similar between single and multiple anomalies.

7.7 Detection Performance from Weighting Technique

Weighting is an method to bias some data more weight or impact upon a system than other data. Therefore, this method can effectively weight recent timelines to more influence on the decision function than other old timelines. Our study also conducts an experiment on the multi-timeline with a weighting technique using the number of timeline replication as a weighting value. Experiments on a weighting technique employ only three learning algorithm, namely MND, KNN, and OSVM. In machine learning, we could employ boosting algorithms, such as AdaBoost [112], LPBoost [113], or BrownBoost [114], to create a weak data to a strong data that performs function like a weighting technique.

Empirical results reveal that the detection performance of multi-timeline system with weighting relies on two main factors. One factor is the weighting technique applied to the detection module. The weighting technique in this experiment is timeline replication, so both the number of training data and the number of replication as a weighting value mainly affect the F-score. Another factor is the learning algorithm employed to the detection module. Results shown in figure 6.22 indicate that the weighting technique strengthen the role of OSVM. The weighting process with MND has slightly improved when we increased the number of training data and weighting value, and the results from KNN show randomly and tiny change when we added the weighting technique to the multi-timeline detection module.

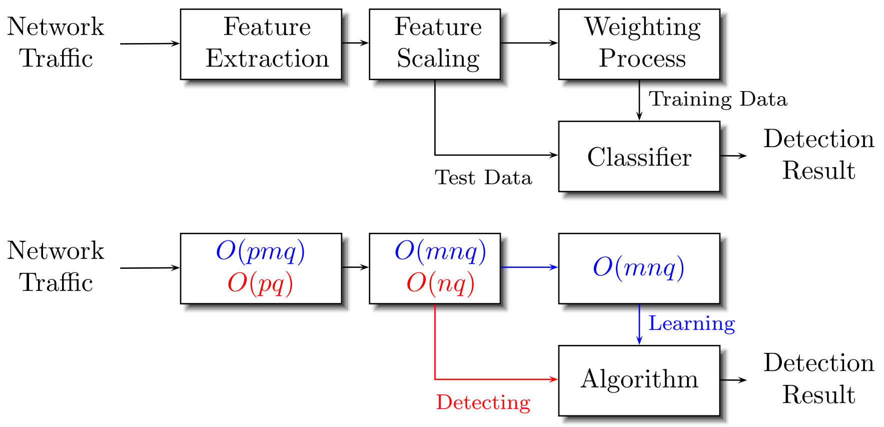

Figure 7.2: Time complexity of multi-timeline detection module with weighting process.

One of our concerns is time complexity after adding a weighting process into the multi-timeline detection module. The weighting technique has an effect on time complexity of the learning process only. In our experiment, we applied timeline replication as weighting, so time complexity relies on both the number of training data and number of replication as weighting value. Analytically, our experiment contains the time complexity of weighting process as $O(mnq)$, where m is the number of training data (days), n is the number of intervals in one day long, q is the number of features. The weighting value, however, is a constant value for multiplying and it does not change the time complexity. Therefore, we could show the time complexity of multi-timeline detection module as shown in figure 7.2.

7.8 Difference of Learning Algorithms

From these all experimental results, we found that several factors have an essential role in detection performance for the multi-timeline representation. Two interconnected factors were the learning algorithm and selected features of network traffic. The results from the experiments 2: comparison of detection performance between individual features and combined all features described in Chapter 6.2 and the first part of experiment 3: no packet incident in test data described in Chapter 6.4 suggest the following issues: (1) we have to select appropriate features, individual or combined features, for a particular type of anomaly when we employ the MND. Additionally, (2) we can use an individual feature for a particular type of anomaly or a combined feature for all types of anomalies when we employ the KNN or OSVM, although detection performance of a combined feature dropped slightly for some types of anomalies.

Table 7.5: Pros and cons of the three learning algorithms.

MND	KNN	OSVM
Pros	Pros	Pros
• Parametric method	• Robust to noisy data	• Kernel-based method
• Simplicity	• Fast training	• Flexibility
Cons	Cons	Cons
• Normal distribution	• Biased by value of k	• Slow training
• Low accuracy	• Memory limitation	• Sensitive to noisy data

We list pros and cons of the three learning algorithms as shown in Table 7.5. Advantages of the MND is a simple algorithm and contains parametric nature so that it does not need to keep old data after learning. However, the MND is suitable for normal distribution and accuracy of this algorithm is still low. Positive characteristics of the KNN are robust to noisy Internet traffic data, and do not need the learning process. Disadvantages of the KNN are the selection of appropriate parameter k and the KNN requires all training data in memory, so that it has a potential memory problem. The OSVM is a kernel-based method that contains flexibility by using various kernels, however, the negative side are that the OSVM needs a long computational time in training process and it is very sensitive to noisy data. From our experiments, it shows that the KNN is the best and the OSVM is the worst of the three learning algorithms.

The next factor is the number of training data in learning process. The results from the experiments 4: learning curves described in Chapter 6.5 show that detection performance fairly increased and converged to a constant value when we increased the number of training data, especially for the MND and KNN. Learning curves of the OSVM, however, increased much slower than those of MND and KNN. Experiment results in this experiment also suggest that both MND and KNN can learn from one or two weeks of training data, whereas the OSVM requires more than one month of training data.

The last factor is the interval value. Results from experiment 1: comparison of different interval values described in Chapter 6.1 show different performances when we assigned different interval values. Short interval values make detecting anomalies more likely in a real-time system, but it needs more computation time for one-day test data, according to Table 7.2. Thus, the benefits of a short time interval value and the increased time consumption are trade-off in the representation.

7.9 Comparison of Representation of Input Data

Experimental results strongly confirm our hypothesis that the multi-timeline representation has a number of capabilities for real-time anomaly detection in computer networks. The multi-timeline representation provided flexibility in using various algorithms, features, and interval values to detect several anomalies . This representation could also detect various anomalies caused by attacks or accidents. Additionally, the multi-timeline representation provides robustness to incorrect training data or data manipulation from attackers. With appropriate learning algorithms and network features, the multi-timeline representation could quickly learn from the training data to detect specific anomalies.

Results in experiment 2 and 3 also provide comparisons between the multi-timeline, real-time, and combination between single and multi-timeline representation. Experimental results indicate that the multi-timeline representation outperform the real-time representation, especially for the Back, Neptune, PortSweep, and Smurf attacks. However, detection performance of the multi-timeline and real-time representations resembles for the IpSweep attack. There are two instances of the IpSweep attack for the multi-timeline and real-time representation. The first instance of IpSweep occurred in a very short period, in 132 seconds, named as Week 3 Wed in Table 5.1. Another instance occurred in a low packet per second, in 1.15 packet per second, named as Week 6 Thu in the same table. These two instances are quite similar to normal traffic so the IpSweep attack are hardly detected by both representations. As a result, detection performances of the multi-timeline and real-time representation are not significantly different.

Results in experiment 3 indicate that the number of t has an effect on detection performance for the real-time and combination representation. Detection performances of real-time quickly drop from those of multi-timeline representation when we added more t in learning process. For the combination representation, however, detection performance slowly drop compared to experimental results by using real-time representation when we added more t in learning process. All these results indicate that the multi-timeline outperforms both real-time and combination representation.

The first part in experiment 4 presents detection performance over no packet incident in test data by using the multi-timeline and real-time representation respectively. The results indicate that detection performances of the multi-timeline representation are better than those of the real-time representation, especially by using MND and KNN. Detection performances of OSVM, however, are not different between both of the representations. The second part in experiment 4 show a difference detection performance between three algorithms by using the multi-timeline and real-time representation. The percentage results of detection performance in these tables are not significantly different between both of the representations. However, trend in percentage different of the multi-timeline and real-time representation resemble when we added more incorrect training data. In summary, these results confirmed that detection performances of the multi-timeline representation are better than those of the real-time representation, but robustness to incorrect training data is not different between both of the representations.

Applying interval-based features instead of packet-based features provides great benefit to the multi-timeline representation. Interval-based features can detect strange behaviors in network traffic even if no packet at all as results in the first part of experiment 4; however, the packet-based features cannot do so. Therefore, throughput of normal traffic has no effect on the multi-timeline representation working with the interval-based representation in term of detectability. For anomaly traffic, however, it is quite difficult to specify detectable throughput because there are many factors of normal and anomaly traffic related to detectability. The key factors are repeating patterns of normal traffic and characteristics of anomaly, the number of source and destination addresses, the number of source and destination ports, packet size, duration of anomaly for example. In our experiments, the multi-timeline representation shows promising results on low throughput of anomaly traffic. For example, characteristics of Neptune attacks as shown in Table 5.1 show the average packet size of anomaly is equal to 60 Bytes and approximates 68 packets per second for three instances, so the throughput of Neptune for our experiments approximates 32 kbit/second. The throughput of Neptune is low and mainly occurs at the peak time, when the average volume of normal traffic at that time is $\approx$ 250 kbit/second. However, detection performance of KNN for the Neptune attack reach nearly the highest point.

Table 7.6: Comparison of F-scores (Avg/Max) between multi-timeline, real-time, and combination representation.

Anomaly	Multi-timeline	Real-time	Combination
Back	0.95/0.98 (KNN)	0.35/0.39 (KNN)	0.69/0.74 (KNN)
IpSweep	0.15/0.32 (MND)	0.09/0.27 (MND)	0.13/0.30 (MND)
Neptune	0.72/0.96 (KNN)	0.49/0.57 (KNN)	0.65/0.81 (KNN)
PortSweep	0.10/0.27 (KNN)	0.10/0.20 (KNN)	0.09/0.25 (KNN)
Smurf	0.94/1.00 (KNN)	0.38/0.53 (KNN)	0.75/0.89 (KNN)
No packet	0.73/0.77 (KNN)	0.69/0.73 (KNN)	0.70/0.75 (KNN)

Table 7.6 summarizes the detection performance between multi-timeline, real-time, combination representation. We indicate average and the best F-score which taken from f_1-f_{all} and the best algorithm (in parentheses) among MND, KNN, and OSVM for Back, IpSweep, Neptune, PortSweep, and Smurf. For no packet incident, however, we compare between MND and KNN rather than three algorithms because both representations with OSVM produce the same highest value of F-score. This table shows that the multi-timeline and combination representation detect most anomalies with F-score higher than the acceptable value 0.5, but most F-scores from real-time representation are lower than the acceptable value.

7.10 Guidance for Applying the Multi-timeline Representation

More concretely, the following steps provide a guidance to apply the multi-timeline representation to a real network environment. First, we have to assign a size for the interval value by considering the trade-offs between notification time after anomalies occur and time consumption of each interval value. However, the interval value might often be assigned by software of various network equipments in many network environments, in this case we cannot freely choose the interval value. Next, designate particular features or combined features for specific network traffic or for an expected anomaly. For example, port scan attacks usually shoot a huge number of attack packets toward a target network, so we can choose the number of destination ports for instance as a network feature for such attacks. In addition, we can select a specific IP address, a range of IP addresses or even port numbers as a network feature. Lastly, we have to select a learning algorithm by considering the amount of training data, detection performance, and time consumption according to Table 7.2. We strongly recommend applying the multi-timeline representation to access networks, place it behind a firewall and virus detector, rather than applying to core networks because traffic data on core networks are more diverse than those at access networks. Although we did not examine performance of the multi-timeline representation in a core network, we firmly believe that the multi-timeline representation could be applied to core networks as well. We intend to conduct experiments by using the multi-timeline representation on core networks as well.

7.11 Limitations of Multi-timeline Representation

Although experimental results show various capabilities of the multi-timeline representation to detect anomalies in real time, they also reveal some limitations. The main limitations of the multi-timeline representation that we discovered consist three main issues: the first issue is that learning algorithms need a period of time to learn prior data, the next issue is that learning process needs attack-free or almost attack-free traffic, and the last issue is that this representation did not provide any details about the attacks or incidents.

The need to learn prior data is an intrinsic nature of machine learning methods, we cannot get out of this limitations. Our experiments, however, show that we can carefully select a proper algorithm that have a capability to quickly learn prior traffic. For example, results in the experiment 4 indicate that learning period of MND and KNN are less than 9 days with satisfied detection performance. Another solution to overcome this limitation is flexibility of the multi-timeline representation to switch from one algorithm to another algorithm. For example, in the first place we have no prior traffic so that we can apply the MND, after operate the system for a while, 15 days for example, we can switch from MND to another algorithm that need more prior data for learning process. Therefore, the need to learn prior data of machine learning is not a big deal for the multi-timeline representation.

One of our limitations is training data over learning process of our detection system. Multi-timeline detection system needs attack-free or mostly attack-free traffic for learning process, but it is not feasible assumption at backbone networks. Therefore, the multi-timeline detection system more suitable for access networks rather than backbone networks.

Providing no detail of attacks or incidents is a real challenge for the multi-timeline representation. One of the key concepts of the multi-timeline representation by using interval-based features is the outstanding ability to detect a wide range of anomalies caused by attacks or accidents. Although learning algorithms with the multi-timeline representation can detect an interval that anomaly occur, but the system provides a few details of attacks or unusual incidents in network traffic. Details of these incidents depend on features. For example, assume that we use the number of packet as a feature, so alarm from the system imply that anomaly relate to the number of packet in network traffic, such as denial of service attacks or outages. A solution for this limitation is that applying more classifiers with different features, so that we can combine details from different sources to identify the incident. Another solution is providing an inspector to examine the interval contained anomalies [115]; it is our future work.

Chapter 8

Conclusion

In this thesis, we have proposed the multi-timeline representation for real-time anomaly detection in network traffic. We start with explaining what anomaly is in the context of network traffic, and then classifying a broad range of network anomalies under two classes: one is caused by human intention and the other is caused by accidents. From our literature review, we found that for many years researchers have proposed a large number of detection techniques for a particular anomaly and for general anomalies. These proposed detection techniques can be classified as signature-based techniques and statistical-based techniques. Owing to limitation of the signature-based, they could hardly detect a novel anomaly, therefore a growing trend towards anomaly detecting in network traffic has been focused on statistical-based techniques. Unfortunately, almost all of statistical-based techniques rely on batch processing, so they take a long time for notification after anomalies occur and not suitable for detection in real time. The literature review also reveals that machine learning techniques have been applied to various and sundry problem domains, including anomaly detection in other domains. However, detecting anomalies in network traffic is much more difficult than those in other domains because many anomalies in network traffic are time and location dependence, known as context anomalies. It means that an incident classified under normal might be an anomaly at another time or another location for example. As a result, representation of input data for anomaly detection in other domains cannot be applied to network traffic. In addition, attackers put effort into evading from detection system if they can discover the technique been used. To solve these issues, we proposed the multi-timeline system for anomaly detection in network traffic which highly suitable for real-time system. This multi-timeline detection system do not require explicit training data known as unsupervised learning. We also firmly believe that the multi-timeline detection system has several properties for real-time anomaly detection, such as flexibility, robustness, quick learning, and short time consumption.

To confirm our hypothesis, we conducted a series of experiments in order to examine several properties of the multi-timeline detection system. There are eight parts in this series as follows. For the first experiment, the purpose is to observe how the interval value have an effect on detection performance, and to identify the best interval value for our experimental data. For the second experiment, the purpose is to investigate which features are highly efficient for particular attacks. For the third experiment, the purpose is to test robustness of the multi-timeline detection system. For the forth experiment, the purpose is to explore how learning algorithms with the multi-timeline representation quickly learn from training data. For the fifth experiment, the purpose is to measure time consumption of learning algorithms with the multi-timeline detection module. For the sixth experiment, the purpose is to compare detection results when test anomalies occur over different volumes of background traffic. For the next experiment, the purpose is to observe effects on anomaly occurrence during a day. For the last experiment, the purpose is to explore performance of the multi-timeline detection module with a weighting technique. We acquired data from two sources, one source from strongly controlled campus network, so we could assume that there is no abnormal traffic in there, the other source from testbed data which have been used in many studies for evaluate their own detection system.

Results from our experiments strongly confirm that the multi-timeline module has many versatile capabilities for real-time anomaly detection in computer networks. We could employ machine learning algorithms with the multi-timeline representation to discover a variety of anomalies caused by attacks or accidents. One of the experiments reveals which features are effective and most likely to detect a particular type of anomalies selected from testbed data. Our result indicates that the multi-timeline technique generally outperform conventional real-time or even a combination between single and multi-timeline technique. Experimental results also show the robustness of the multi-timeline detection system that attackers hardly evade the system or manipulate the system, even if attackers know the methodology used in the detection system. Learning curves in one of the our experiments show that the multi-timeline representation with some machine learning algorithms could quickly learn from our training data, some algorithms could learn only 3 to 9 days of training data. Time consumption results in our experiment suggest that the multi-timeline representation could be more than likely to operate in real time. Our most desirable capability is that the multi-timeline representation provides high flexibility so that we could employ different algorithms or features for different types of anomalies in network traffic, regardless of types of networks, network media or even protocols. The last experiment indicate that we could add a weighting technique as an option into the multi-timeline module to give recent timelines more influence on the result than other old timelines. However, detection performance of the multi-timeline system mainly relies on the weighting technique and learning algorithm.

In summary, we proposed the multi-timeline detection system so that we can apply any machine learning algorithms or use any interval-based feature to detect network traffic anomalies in real time. We conducted a series of experiments to examine several capabilities of the multi-timeline representation, for example, flexibilities by using different algorithms or different features, robustness from incorrect training data, a learning capability, an ability to detect anomalies in real time. Experimental results strongly confirm that the multi-timeline representation have versatile capabilities and flexibilities to discover several network traffic anomalies with promising detection performance, especially for access networks or campus networks. The multi-timeline detection system not only enables network administrators to detect exist or novel types of attacks but can also be used to identify abnormal behavior of their own networks in real-time.

For our future work, we intend to apply the multi-timeline detection system to a real network environment. Although, our experimental results show that the multi-timeline representation with some learning algorithms detected several types of anomalies with promising performance, there are many factors in network traffic that might adversely affect detection performance of the multi-timeline system. Moreover, to fulfill an essential requirement of anomaly detection in real time, we intend to develop an automatic inspector who provide full details of anomalies after it have been detected by the multi-timeline detection system.

For future direction, we have to provide details of occurred anomalies; it is a crucial part to fulfill potential of detection system using the multi-timeline technique. We also plan to implement the multi-timeline representation in computer networks of Bangkok university; however, before that we need to examine other network features for different types of anomalies other than five types of anomalies in this study. Applying the multi-timeline representation in a core network or backbone network is one of our challenges, because traffic of core network richly diverses and very dissimilars from access networks or campus networks. Finally, we have a great desire that the multi-timeline detection module would be applied to a hardware as a piece of network equipment, so that the detection system using our multi-timeline technique would run much faster than software-based system.

Bibliography

[1] CERT Coordination Center. CERT statistics (historical). `http://www.cert.org/stats/cert_stats.html`, 2003.

[2] John Mchugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14–35, 2001.

[3] Howard F. Lipson. Tracking and tracing cyber-attacks: Technical challenges and global policy issues. Special Report CMU/SEI-2002-SR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 2002.

[4] K. G. Coffman and A. M. Odlyzko. Growth of the internet, 2001.

[5] Yufeng Kou and Chang-tien Lu. Spatial weighted outlier detection. In *In Proceedings of SIAM Conference on Data Mining*, 2006.

[6] Shashi Shekhar, Chang-Tien Lu, and Pusheng Zhang. Detecting graph-based spatial outliers: algorithms and applications (a summary of results). In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 371–376, New York, NY, USA, 2001. ACM.

[7] Andreas S. Weigend, Morgan Mangeas, and Ashok N. Srivastava. Nonlinear gated experts for time series: discovering regimes and avoiding overfitting. *International Journal of Neural Systems*, 6(4):373–399, 1995.

[8] Stan Salvador and Philip Chan. Learning states and rules for detecting anomalies in time series. *Applied Intelligence*, 23(3):241–255, December 2005.

[9] Weng-Keen Wong, Andrew Moore, Gregory Cooper, and Michael Wagner. Bayesian network anomaly pattern detection for disease outbreaks. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 808–815. AAAI Press, 2003.

[10] S.E. Guttormsson, II Marks, R.J., M.A. El-Sharkawi, and I. Kerszenbaum. Elliptical novelty grouping for on-line short-turn detection of excited running rotors. *IEEE Transactions on Energy Conversion*, 14(1):16–22, 1999.

[11] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, July 2009.

[13] Kai Hwang, Min Cai, Ying Chen, and Min Qin. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *IEEE Transactions on Dependable and Secure Computing*, 4(1):41–55, 2007.

[14] Chris Sinclair, Lyn Pierce, and Sara Matzner. An application of machine learning to network intrusion detection. In *Proceedings of the 15th Annual Computer Security Applications Conference*, ACSAC '99, pages 371–, Washington, DC, USA, 1999. IEEE Computer Society.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing edition, October 2007.

[16] Pavel Laskov, Patrick Dssel, Christin Schfer, and Konrad Rieck. Learning intrusion detection: Supervised or unsupervised? In Fabio Roli and Sergio Vitulano, editors, *Image Analysis and Processing ICIAP 2005*, volume 3617 of *Lecture Notes in Computer Science*, pages 50–57. Springer Berlin / Heidelberg, 2005.

[17] Daniel Barbará, Ningning Wu, and Sushil Jajodia. Detecting novel network intrusions using bayes estimators. In *Proceedings of the First SIAM Conference on Data Mining*, April 2001.

[18] Liwei (vivian) Kuang. Dnids: A dependable network intrusion detection system using the csi-knn algorithm, 2007.

[19] Latifur Khan, Mamoun Awad, and Bhavani Thuraisingham. A new intrusion detection system using support vector machines and hierarchical clustering. *The VLDB Journal*, 16:507–521, October 2007.

[20] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science - Volume 38*, ACSC '05, pages 333–342, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[21] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*, pages 5–8, 2001.

[22] Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):631–645, May 2007.

[23] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145, 1999.

[24] Pei Sun, Sanjay Chawla, and Bavani Arunasalam. Mining for outliers in sequential databases. In *in ICDM, 2006*, pages 94–106.

[25] Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 631–636, New York, NY, USA, 2003. ACM.

[26] Vir V. Phoha. *Internet security dictionary.* Springer, 2002.

[27] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, February 1987.

[28] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *J. Comput. Secur.*, 6(3):151–180, August 1998.

[29] D. Dasgupta and F. Nino. A comparison of negative and positive selection algorithms in novel pattern detection. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 1, pages 125–130 vol.1, 2000.

[30] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection schemes. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(1):357–373, Feb 2004.

[31] Fabio A. González and Dipankar Dasgupta. Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403, December 2003.

[32] D. Dasgupta and N.S. Majumdar. Anomaly detection in multidimensional data using negative selection algorithm. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1039–1044, 2002.

[33] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. In *Applications of Data Mining in Computer Security*. Kluwer, 2002.

[34] A. K. Gosh, J. Wanken, and F. Charron. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 14th Annual Computer Security Applications Conference*, ACSAC '98, pages 259–, Washington, DC, USA, 1998. IEEE Computer Society.

[35] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust anomaly detection using support vector machines. In *In Proceedings of the International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc.

[36] Katherine A. Heller, Krysta M. Svore, Angelos D. Keromytis, and Salvatore J. Stolfo. One class support vector machines for detecting anomalous windows registry accesses. In *In Proc. of the workshop on Data Mining for Computer Security*, 2003.

[37] Wenke Lee, Salvatore J. Stolfo, and Philip K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *In AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56. AAAI Press, 1997.

[38] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 6–6, Berkeley, CA, USA, 1998. USENIX Association.

[39] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. Adaptive intrusion detection: A data mining approach. *Artif. Intell. Rev.*, 14(6):533–567, December 2000.

[40] Robert Gwadera, Mikhail J. Atallah, and Wojciech Szpankowski. Reliable detection of episodes in event sequences. *Knowledge and Information Systems*, 7:415–437, May 2005.

[41] Mikhail Atallah, Robert Gwadera, and Wojciech Szpankowski. Detection of significant sets of episodes in event sequences. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, ICDM '04, pages 3–10, Washington, DC, USA, 2004. IEEE Computer Society.

[42] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, LISA '99, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[43] E. Albin and N.C. Rowe. A realistic experimental comparison of the suricata and snort intrusion-detection systems. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 122–127, 2012.

[44] J. S. White, T. Fitzsimmons, and J. N. Matthews. Quantitative analysis of intrusion detection systems: Snort and suricata. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 8757 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, May 2013.

[45] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, pages 2435–2463, 1999.

[46] G.B. White, E.A. Fisch, and U.W. Pooch. Cooperating security managers: a peer-based intrusion detection system. *IEEE Network*, 10(1):20–23, 1996.

[47] Wei Fan, Matthew Miller, Salvatore J. Stolfo, Wenke Lee, and Philip K. Chan. Using artificial anomalies to detect unknown and known network intrusions. *Knowledge and Information Systems*, 6(5):507–527, 2004.

[48] Kymie M. C. Tan, Kevin S. Killourhy, and Roy A. Maxion. Undermining an anomaly-based intrusion detection system using common exploits. In *Proceedings of the 5th international conference on Recent advances in intrusion detection*, RAID'02, pages 54–73, Berlin, Heidelberg, 2002. Springer-Verlag.

[49] Ar Lazarevic, Aysel Ozgur, Levent Ertoz, Jaideep Srivastava, and Vipin Kumar. A comparative study of anomaly detection schemes in network

intrusion detection. In *In Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[50] Kenji Yamanishi, Jun-ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

[51] Dit-Yan Yeung and C. Chow. Parzen-window network intrusion detectors. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 4, pages 385–388 vol.4, 2002.

[52] Daniel Barbará, Julia Couto, Sushil Jajodia, and Ningning Wu. Adam: a testbed for exploring the use of data mining in intrusion detection. *ACM SIGMOD Record*, 30:15–24, December 2001.

[53] Karlton Sequeira and Mohammed Zaki. Admit: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 386–395, New York, NY, USA, 2002. ACM.

[54] M. Thottan and Chuanyi Ji. Anomaly detection in ip networks. *IEEE Transactions on Signal Processing*, 51(8):2191–2204, aug. 2003.

[55] Ann Tamaru Alphonso Valdes Debra Anderson, Thane Frivold. *Next Generation Intrusion Detection Expert System Operators Manual.* Space and Naval Warfare Systems Command, 6 1994.

[56] Anderson, Lunt, Javitz, Tamaru, and Valdes. Detecting unusual program behavior using the statistical components of NIDES. may 1995.

[57] Phillip A. Porras and Peter G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997.

[58] Kenji Yamanishi and Jun-ichi Takeuchi. Discovering outlier filtering rules from unlabeled data: Combining a supervised learner with an unsupervised learner. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 389–394, New York, NY, USA, 2001. ACM.

[59] Gaurav Tandon and Philip K. Chan. Weighting versus pruning in rule validation for detecting network and host anomalies. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge*

Discovery and Data Mining, KDD '07, pages 697–706, New York, NY, USA, 2007. ACM.

[60] Christos Siaterlis and Basil Maglaris. Towards multisensor data fusion for dos detection. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 439–446, New York, NY, USA, 2004. ACM.

[61] Abdallah Abbey Sebyala, Temitope Olukemi, Lionel Sacks, and Dr. Lionel Sacks. Active platform security through intrusion detection using naive bayesian network for anomaly detection. In *In: Proceedings of London communications symposium*, 2002.

[62] Zheng Zhang, Jun Li, C. N. Manikopoulos, Jay Jorgenson, and Jose Ucles. Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification. In *Proc. IEEE Workshop on Information Assurance and Security*, pages 85–90, 2001.

[63] Khaled Labib and et al. Nsom: A real-time network-based intrusion detection system using self-organizing maps.

[64] Daniel Barbará, Yi Li, Julia Couto, Jia-Ling Lin, and Sushil Jajodia. Bootstrapping a data mining intrusion detection system. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, pages 421–425, New York, NY, USA, 2003. ACM.

[65] Min Qin and Kai Hwang. Frequent episode rules for internet anomaly detection. In *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 161–168, Aug 2004.

[66] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, and S. Narravula. Towards nic-based intrusion detection. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 723–728. ACM Press, 2003.

[67] Levent Ertz, Eric Eilertson, Aleksandar Lazarevic, Pang ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. Minds – minnesota intrusion detection system.

[68] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. *SIGCOMM Comput. Commun. Rev.*, 35(4):217–228, August 2005.

[69] Jimeng Sun, Yinglian Xie, Hui Zhang, and Christos Faloutsos. Less is more: Compact matrix decomposition for large sparse graphs, 2007.

[70] Wenke Lee and Dong Xiang. Information-theoretic measures for anomaly detection. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*, pages 130–143, 2001.

[71] Kriangkrai Limthong. Performance of interval-based features in anomaly detection by using machine learning approach. *International Journal of Machine Learning and Computing*, 4(3):292–299, June 2014.

[72] Kriangkrai Limthong, Pirawat Watanapongse, and Kensuke Fukuda. A wavelet-based anomaly detection for outbound network traffic. In *8th Asia-Pacific Symposium on Information and Telecommunication Technologies, 2010. APSITT 2010. International Conference on*, Jun 2010.

[73] Anand Narasimhamurthy. Theoretical bounds of majority voting performance for a binary classification problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1988–1995, December 2005.

[74] Giovanni Vigna and Richard A. Kemmerer. Netstat: a network-based intrusion detection system. *Journal of Computer Security*, 7(1):37–71, January 1999.

[75] ShengYi Jiang, Xiaoyu Song, Hui Wang, Jian-Jun Han, and Qing-Hua Li. A clustering-based method for unsupervised intrusion detections. *Pattern Recognition Letters*, 27(7):802–810, May 2006.

[76] A. Kind, M.P. Stoecklin, and X. Dimitropoulos. Histogram-based traffic anomaly detection. *IEEE Transactions on Network and Service Management*, 6(2):110–121, june 2009.

[77] Debra Anderson, Thane Frivold, Ann Tamaru, and Alfonso Valdes. Next generation intrusion detection expert system (NIDES), software users manual beta-update release. Technical Report SRI-CSL-95-0, May 1994.

[78] Ioanna Stamouli, Patroklos G. Argyroudis, and Hitesh Tewari. Real-time intrusion detection for ad hoc networks. In *In WOWMOM 05: Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM05*, pages 374–380. IEEE Computer Society, 2005.

[79] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[80] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 219–230, New York, NY, USA, 2004. ACM.

[81] George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang. An empirical evaluation of entropy-based traffic anomaly detection. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 151–156, New York, NY, USA, 2008. ACM.

[82] Yoshiki Kanda, Romain Fontugne, Kensuke Fukuda, and Toshiharu Sugawara. Admire: Anomaly detection method using entropy-based PCA with three-step sketches. *Computer Communications*, 36(5):575 – 588, 2013.

[83] Selim Aksoy and Robert M. Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern Recognition Letters*, 22(5):563 – 582, 2001. Image/Video Indexing and Retrieval.

[84] J. Grossman, M. Grossman, and R. Katz. *The first systems of weighted differential and integral calculus.* Archimedes Foundation, 1980.

[85] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. Mawilab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the 6th International COnference*, Co-NEXT '10, pages 8:1–8:12, New York, NY, USA, 2010. ACM.

[86] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical network support for ip traceback. *SIGCOMM Comput. Commun. Rev.*, 30(4):295–306, August 2000.

[87] Luis Martin Garcia. Programming with libpcap — sniffing the network from our own application. *Hackin9 Magazine*, 3(2/2008), February 2008.

[88] John W. Eaton, David Bateman, and Soren Hauberg. *GNU Octave version 3.0.1 manual: a high-level interactive language for numerical computations.* CreateSpace Independent Publishing Platform, 2009. ISBN 1441413006.

[89] Sousuke Amasaki and Chris Lokan. The effects of gradual weighting on duration-based moving windows for software effort estimation. In Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Mnnist, Jrgen Mnch, and Mikko Raatikainen, editors, *Product-Focused Software Process Improvement*, volume 8892 of *Lecture Notes in Computer Science*, pages 63–77. Springer International Publishing, 2014.

[90] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, December 2007.

[91] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition.* Academic Press, 4th edition, 2008.

[92] Thomas M. Mitchell. *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[93] Michel M. Deza and Elena Deza. *Encyclopedia of Distances.* Springer, 1 edition, August 2009.

[94] Aharon Bar-Hillel, Tomer Hertz, Noam Shental, and Daphna Weinshall. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6:937–965, December 2005.

[95] T. Lavoie and E. Merlo. An accurate estimation of the levenshtein distance using metric trees and manhattan distance. In *Software Clones (IWSC), 2012 6th International Workshop on*, pages 1–7, 2012.

[96] G.T. Toussaint. On a simple minkowski metric classifier. *IEEE Transactions on Systems Science and Cybernetics*, 6(4):360–362, 1970.

[97] Syed Masum Emran and Nong Ye. Robustness of chi-square and canberra distance metrics for computer intrusion detection. *Quality and Reliability Engineering International*, 18(1):19–28, 2002.

[98] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, July 2001.

[99] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[100] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995.

[101] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge University Press, New York, NY, USA, 2000.

[102] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, mar 2001.

[103] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[104] Gavin C. Cawley and Nicola L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 99:2079–2107, August 2010.

[105] Kristopher Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[106] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings*, volume 2, pages 12 –26 vol.2, 2000.

[107] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, November 2000.

[108] C. Thomas, V. Sharma, and N. Balakrishnan. Usefulness of darpa dataset for intrusion detection system evaluation. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6973 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, March 2008.

[109] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing &amp; Management*, 45(4):427 – 437, 2009.

[110] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[111] C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[112] Ral Rojas. Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting.

[113] Ayhan Demiriz, KristinP. Bennett, and John Shawe-Taylor. Linear programming boosting via column generation. *Machine Learning*, 46(1-3):225–254, 2002.

[114] Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

[115] Johan Mazel, Romain Fontugne, and Kensuke Fukuda. A taxonomy of anomalies in backbone network traffic. In *International Wireless Communications and Mobile Computing Conference, IWCMC 2014, Nicosia, Cyprus, August 4-8, 2014*, pages 30–36, 2014.