Detection of Malicious User Communities in Data Networks

by

Amir Moghaddam
B.Sc., Sharif University of Technology, 2008

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Detection of Malicious User Communities in Data Networks

by

Amir Moghaddam
B.Sc., Sharif University of Technology, 2008

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Kui Wu, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Sudhakar Ganti, Supervisor
(Department of Computer Science)

Dr. Kui Wu, Departmental Member
(Department of Computer Science)

ABSTRACT

Malicious users in data networks may form social interactions to create communities in abnormal fashions that deviate from the communication standards of a network. As a community, these users may perform many illegal tasks such as spamming, denial-of-service attacks, spreading confidential information, or sharing illegal contents. They may use different methods to evade existing security systems such as session splicing, polymorphic shell code, changing port numbers, and basic string manipulation. One way to masquerade the traffic is by changing the data rate patterns or use very low (trickle) data rates for communication purposes, the latter is focus of this research. Network administrators consider these communities of users as a serious threat.

In this research, we propose a framework that not only detects the abnormal data rate patterns in a stream of traffic by using a type of neural network, Self-organizing Map (SOM), but also detects and reveals the community structure of these users for further decisions. Through a set of comprehensive simulations, it is shown in this research that the suggested framework is able to detect these malicious user communities with a low false negative rate and false positive rate.

We further discuss ways of improving the performance of the neural network by studying the size of SOM's.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I wish to acknowledge all people who helped and inspired me during my studies.
I would like to thank my supervisor:

Dr. Sudhakar Ganti whose encouragement, support and patience from the initial to the final level enabled me to develop an understanding of the subject.

I would like to express my gratitude to my thesis committee member:

Dr. Kui Wu for his time, efforts and valuable suggestions.

Last but not least, I wish to thank my family. Without them, I would never have gone this far.

DEDICATION

To my parents, for their never-ending support and the sense of freedom they have given when I wanted it most.

Chapter 1

Introduction

This work describes the design and analysis of a framework to detect malicious user communities in data networks based on communication patterns. This topic was appealing to the researchers due to two main reasons: the first one is that security in networks is one of the today's main concerns for network administrators and researchers. The second reason is that, even though we mainly focused on peer-to-peer networks communications and security, it was particularly interesting to see how data mining, network data management, social networks models and graph algorithms could be generalized for use in many network security aspects. Analyzing network security incidents based on malicious user communities and not just individual malicious users make it a suitable candidate for network administrators, which is the focus of this thesis. In our approach, we exploited Self-Organizing Maps (SOMs) [4] as one of the data mining techniques in data clustering and Girvan-Newman's community detection algorithm in graphs [5] .

Due to a large number of network security incidents, network security has become one of the major concerns in the past two decades for network administrators [6]. The issue has an elevated importance, if the network faces a group of malicious users co-operating with one another. This co-operation can be in the form of spamming, denial-of-service attacks, spreading confidential information, or sharing illegal contents. Existing methods to detect these security incidents can be classified into two different categories: *a)* Signature-based behavior detection [7] and *b)* Anomaly-based behavior detection [8].

Signature based behavior detection techniques rely almost entirely on string matching. These techniques try to find a known malicious pattern, stored in their pattern databases, somewhere inside packets. If they find a malicious pattern, they raise an

alarm, otherwise they do not consider the packet as a threat. While these techniques are powerful in detecting existing incidents or breaking the string match of a poorly written signature, they are unable to detect novel incidents, known as zero attacks. They need to keep their pattern databases updated regularly.

In contrast, the goal of anomaly based detection techniques is to detect patterns within network traffic that deviates from a constituted normal behavior. Each network has its own standards for its users. An anomaly-based behavior detection system is able to potentially detect an abnormality the first time it is used. Anomaly-based detection techniques try to find users or streams of traffic that do not follow these standards and the intrusive activity triggers an alarm because it might deviate from normal activities.

There are different ways to evade existing anomaly detection systems for users who want to establish communications for malicious purposes. Users can send their traffic on different dynamic port numbers, change the pattern of their traffic to new patterns, use an abnormal data rate traffic (such as very low trickle rate), and so on. These users also could construct communities which are distributed throughout the network by sharing and updating their information with one another, like Botnets. Botnets [9] are a collection of infected computers throughout a network that are controlled by an attacker, called botmaster. There are different variation of botnets. Covert botnets are specialized peer-to-peer botnets within a switched sub-network that use encryption to evade from edge-based detection systems [10], [11]. The authors of [12] proposed methods to detect discrete sub-network botnet communication patterns used by covert botnets by exploiting network traffic analysis techniques. Our research assumes that such communication patterns can be established and features selected for the purpose of community detection. Therefore, this research presents a distributed model to detect communities of users who masquerade their traffic by using abnormal data rate patterns as an example.

In the proposed system, which consists of a two-tier architecture model, first a model to detect abnormal data rate traffic passing through a network has been developed as the first tier of the framework. This model exploits self-organizing maps as a type of artificial neural network to classify packets (users) into normal and abnormal clusters. Unlike most of the existing systems which use packet payload for their classification process, only the packet headers and other statistical data about the users are analyzed in this tier. Self-organizing maps are unsupervised neural network tools which map a high dimensional space to a low dimensional space,

while preserving the topological mapping between input data and output data. After classifying the users into different clusters, we create a graph that consists of the users of the network. In this graph, nodes are the users who are classified as abnormal users based on our classification process and are suspicious. The edges are the connectivities of these suspicious users with either other suspicious users or other non-suspicious users. We then pass this graph to the second tier of the framework to detect hidden community structures of these suspicious users.

In the second tier, the graph of suspicious users of the network is filtered to only those who communicate with abnormal data rate patterns and form some community structures in the graph. This tier applies a community detection algorithm to detect communities of users hidden within the graph of suspicious users.

In this thesis, first, we studied various security incident detection techniques including intrusions or network standards deviations, and focused on anomaly-based behavior detection techniques. Then we proposed a two-tier framework to detect the abnormal data rate patterns in a stream of traffic by using Self-Organizing Maps and the hidden community structure of users who are considered as malicious users by Girvan-Newman's community detection algorithm. Finally, we performed comprehensive simulations to study the precision and to verify the ability of our framework in detecting abnormal data rate traffic passing through a network. To extend the research to generic scenarios, we used network simulation approaches with different topologies and structures to study the performance of our framework under various scenarios.

The remainder of this thesis is organized as follows. In Chapter 2, we review the existing methods in detecting network security incidents and summarize related work. In Chapter 3, we describe the architecture of the suggested framework and explain the structure of each tier in detail along with a study of performance improvement of the self-organizing map. In Chapter 4, we present simulation results using various network scenarios and evaluate the performance of our framework. We conclude our work in Chapter 5 with a discussion on ways to further improve our framework.

Chapter 2

Background and Related Work

In this chapter, we provide a brief background about network security incidents. Then we review current security incident detection techniques, specifically anomaly detection systems and also review existing work on community detection algorithms in graph theory.

2.1 Network Security Incidents

The Internet was designed to provide scalability, ease of use, and openness among all people, specifically research and academic communities [13]. However, these goals have led the Internet to a very poor level of security. In 1988, Morris Worm, the first major Internet security incident occurred, thus causing the rise of security incidents on the Internet and also for the creation of methods to defend against them [14]. The number of network security incidents has increased rapidly in the past two decades with the rapid growth of the Internet. The main process in maintaining security of a network has been to detect the security incidents first and then reacting to the incidents appropriately.

Figure 2.1 illustrates this development. It clearly shows how steeply incidents have increased since 1999. These incidents include illegal attempts to unauthorized access to a system, denial of service attacks, spamming and so on. CERT has decided not to publish the number of incidents since 2004, due to a very large number network security incidents that occur every year.

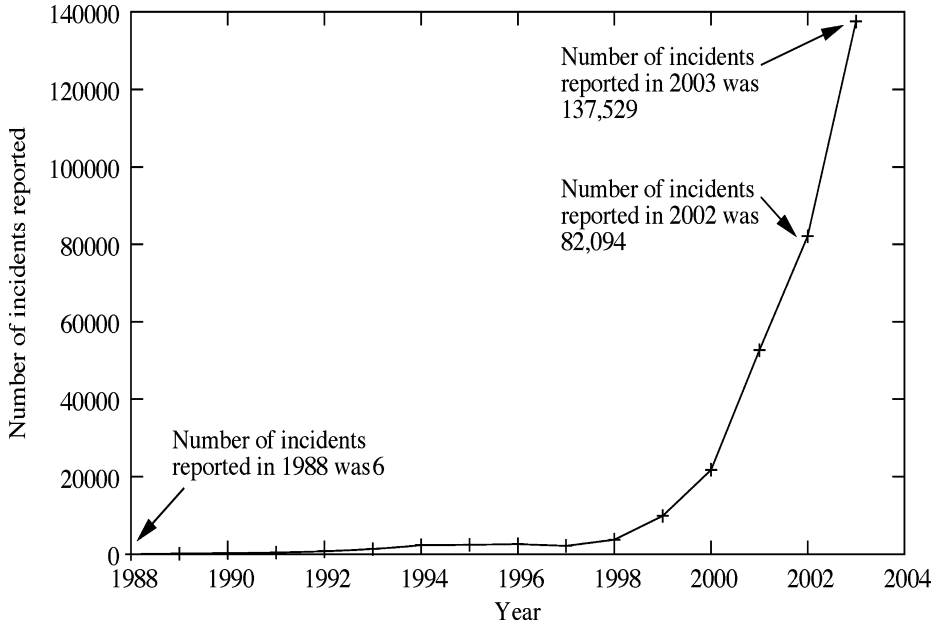

Figure 2.1: The number of network security incidents reported by CERT from 1988 to 2003.

2.2 Security Incident Detection Techniques

One of the hardest tasks in managing and monitoring of large networks is the detection of security incidents hidden in the stream of network traffic. Security incident detection systems are generally classified in two main categories: *Host-based Detection Systems* and *Network-based Detection Systems*. In both categories, they look for specific patterns, signatures, or characteristics that indicate the traces of malicious activity in the network. There is another category called *Hybrid Detection Systems* which is a combination of both methods and these systems provide the protection effectively. The main difference between *host-based* and *network-based* systems is that host-based systems are related to processing specific patterns, signatures, or characteristic in *log files* such as event and kernel logs on computers. On the other hand, network-based systems look for specific patterns, signatures, or characteristic in the network traffic packets. Table 2.1 shows a comparison between host-based and network-based detection systems.

Signature-based detection and *anomaly-based detection* techniques are the major approaches for network-based Intrusion Detection Systems (IDS). Each approach has its strengths and weaknesses. Our research work focuses on *anomaly-based detection*

Benefit	Host-based	Network-based
Attack Anticipation	Good at detecting suspicious patterns.	None.
Damage Assessment	Excellent in determining extent of compromise.	Weak damage assessment capabilities.
Detection	Strong in detecting insider incidents. Weak in detecting outsider incidents.	Strong in detecting outsider incidents. Weak in detecting insider incidents.
Deterrence	Strong deterrence for insider.	Strong deterrence for outsider.
Prosecution Support	Strong prosecution support capabilities.	Very weak because there is no data source integrity.
Response	Weak real-time response. Good for long-term attacks.	Strong response against outsider attacks.

Table 2.1: Comparing Host-based and Network-based Detection Systems [1].

systems. In the following, we present both approaches as well as their advantages and disadvantages.

2.2.1 Signature-based Detection Systems

While signature based detection systems are powerful in detecting existing and known attacks, they are poor in detecting novel attacks. Signature based detection systems identify an attack if the pattern of stream of traffic matches one of the known characteristics of malicious patterns. String matching is the key in signature based detection techniques. Any Intrusion Detection System (IDS) that is based on signature-based detection is easy to develop and update. Most of the current commercial anti-viruses are based on signature detection techniques.

In signature-based detection area, we can divide intrusive malware (malware is a term for any kind of malicious software) into three different groups [15]: *1)* Basic malware, *2)* Polymorphic malware, and *3)* Metamorphic malware.

Basic Malware

In basic malware, the execution entry point is changed so that control is given to the malicious payload, as shown in Figure 2.2. If the detection system can find a pattern in the viral code, it would be able to detect the malware intrusion.

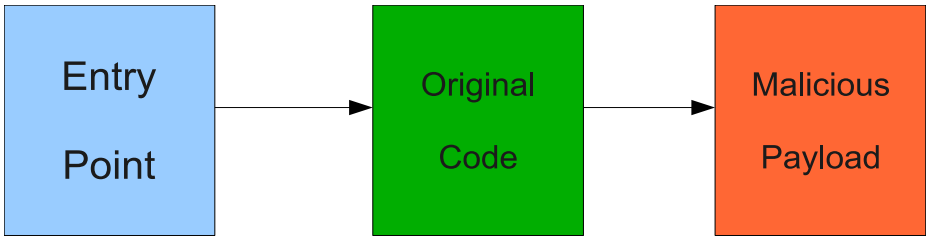

Figure 2.2: Basic malware pattern.

Polymorphic Malware

Polymorphic Malware have an original code which remains intact called polymorphic engine. This engine remains stable and unchanged during intrusions. But it generates new and different mutants at each attack which makes signature-based detection difficult since the signatures vary. They consist of three parts: original code, decryption module, and viral code as shown in Figure 2.3. To detect polymorphic malwares, static analysis based on API sequencing is used [16].

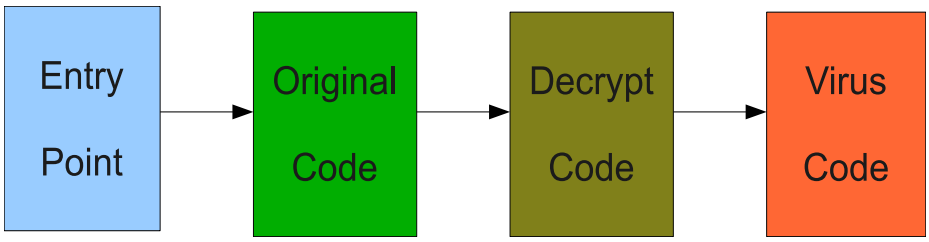

Figure 2.3: Polymorphic malware pattern.

Metamorphic Malware

Metamorphic Malware are the most difficult malware to detect. They have the ability to reprogram their viral code to a new code. This process is done such that its functionality is retained to evade signature-based detection systems. This ability is due to a metamorphic engine that is implemented with a disassembler to parse the viral code in the metamorphic malware. All the new variant forms of the metamorphic malware have different patterns than the original malware, as shown in Figure 2.4. In Figure 2.4, $S_i \in S$ are different signatures of variants of a single metamorphic malware. Storing patterns of multiple variants of the metamorphic malware is not an easy task thus making it difficult for the signature-based detection systems to detect them.

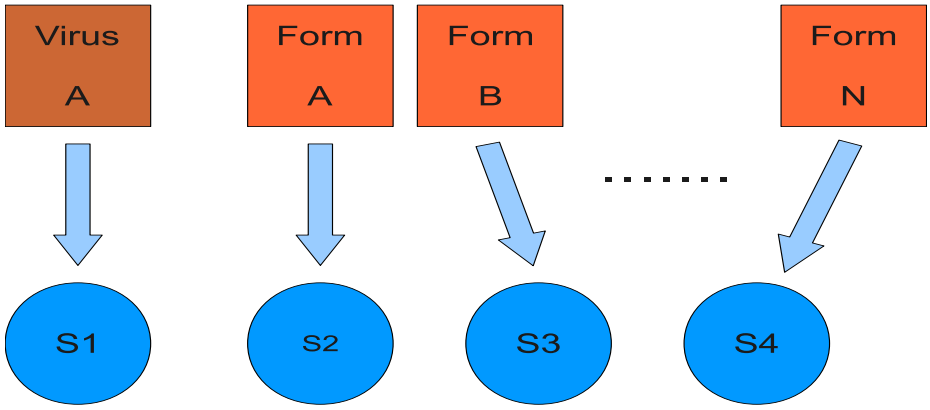

Figure 2.4: Metamorphic malware pattern.

2.2.2 Anomaly-based Detection Systems

Anomaly-based detection systems are powerful in detecting novel attacks. Neural networks are the key in anomaly-based detection techniques.

A lot of research results are available in the literature on the detection of network security incidents based on anomaly behavior by using neural networks. Lippmann and Cunningham [17] developed an intrusion detection system based on keyword counting using neural networks. They looked for user actions in a host-based detection system and used attack-specific keyword counts as their neural network's input. In [18], a self-organizing feature map with a unique neighborhood searching algorithm has been used to detect some types of TCP attacks. In [19], actual system states

of a machine and activities taken on that machine are collected as multidimensional vectors. These vectors are used to classify user anomaly behavior by using self-organizing map as a classification module. A network based anomaly detection scheme is proposed in [20]. They proposed a new two-tier architecture to detect intrusions using unsupervised learning technology. They use self-organizing map clustering to recognize patterns in packet payloads.

There has been some published material on abnormal data-rate intrusions. In [21], authors focused on low-rate patterns in some TCP attacks. They used a dynamic time warping method to detect periodic short burst flows that are considered to be low-rate TCP attacks. [22] studies low-rate TCP attacks at edge routers. Flows are marked as suspicious if their burst length value is larger than the round trip times of other connections in the same router.

To our knowledge, none of these works studied the community structures of malicious users in a network. In this thesis, we focus on community structure of malicious users and how they interact in their social networks. We used the concept of neural networks and community detection in graphs to detect the malicious users who masquerade themselves by using abnormal data-rate patterns and the community structure that they belong to.

Chapter 3

Architecture of the Proposed Framework

This chapter presents the proposed framework to detect malicious user communities that communicate with abnormal data rates. Figure 3.1 shows a local area network (LAN) in which the users communicate through a set of routers. We establish a copy of the framework on each router to detect the distributed abnormal data rate communication among the users. The proposed framework consists of a two-tier architecture model.

- Abnormal Data Rate Traffic Detection Tier
- User Community Detection Tier

In the first tier, the detection of users who masquerade their traffic by using abnormal data rate patterns, i.e., low data rate patterns is carried out first. Such users know that if they send their information with a normal data rate pattern in the network, their traffic can be easily detected by regular anomaly detection systems so they reduce their data rate to a lower rate; therefore, they can hide their activity. Please note that the communication data rate pattern is not the scope of this thesis. As discussed in the introduction, there are many malicious applications such as Botnets that use low data rates for communications purposes. Hence low data rate is used as one of the factors to detect the malicious community. Here is where a self-organizing map (SOM) is used as a tool to classify users into two different groups of normal and abnormal users. Normal users are those who send their packets using a normal data rate pattern according to the constituted standards of the network as the way of their

communication. Abnormal users are those who are suspected of deviating from the standard behavior.

This tier of the framework has three major components that work in this order:

1. Packet pre-processing
2. Packet classification
3. Creating a graph of suspicious users

After accomplishing all these tasks, the graph generated in the last component is passed to the second tier of the framework to detect malicious user communities. In this graph, suspicious users are the nodes and their connections are edges.

In the second tier, a community detection algorithm that is suited to the structure of the network and given graph is invoked to detect various user communities. The graph of suspicious users from the first tier is passed on to the second tier and users that do not belong to any communities in the graph are removed. This purifies the graph and gives us the user communities of the monitored network.

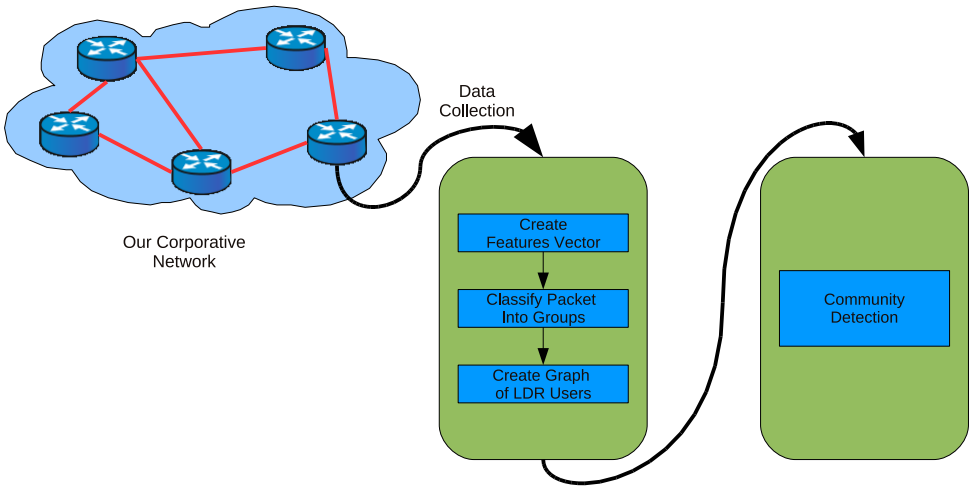

Figure 3.1: Two-tier Framework to Detect Malicious User Communities.

In the following sections, we first describe about the self-organizing maps (SOM) as they are the main tool used in the first tier. Then, we elaborate the details of each tier of our framework.

3.1 Self-Organizing Maps

Self-Organizing Map (SOM) is a type of artificial neural network that is applied in many applications including visualization, data representation, data reduction, density reduction, classification, and so on [4].

SOM is an unsupervised learning algorithm which belongs to the group of competitive learning networks and uses a competitive algorithm to produce the output. It keeps the topological mapping between input and output, which is useful when dealing with a large sets of data, each with multitude of dependencies. SOM reduces the dimensions of data through the use of self-organizing neural networks and maps a high-dimensional space to a low-dimensional space (usually 2-dimensional), while preserving the topological mapping between input data and output data as shown in Figure 3.2.

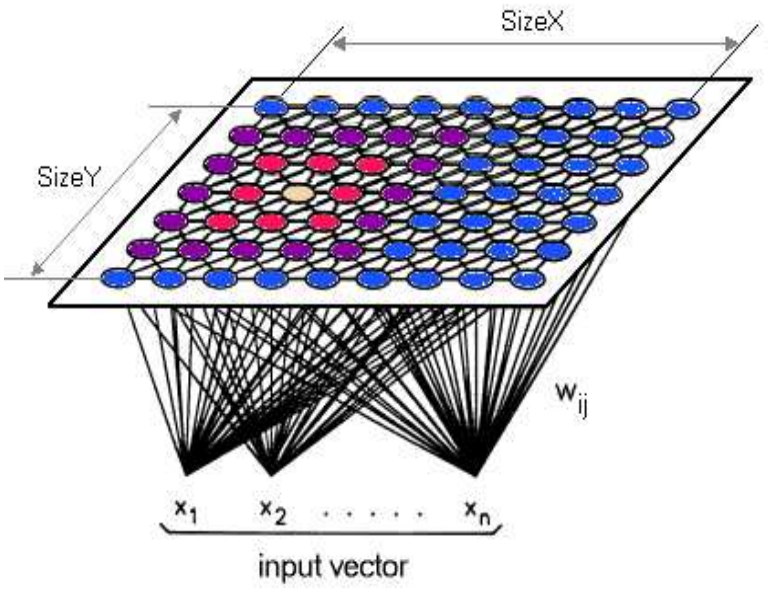

Figure 3.2: Mapping a high dimensional data set to a 2-dimensional data set [2].

One of the intriguing benefits of using SOM is the fact that resulting map can also show the dependencies between input data vectors. The map consists of various parts in its axes which are called *Neurons* or *Units*. Neurons which are neighbor to each other have some dependencies and relationships among themselves.

As an example, suppose we have a set of data about different colors. We know that

one of the methods of representing colors is "RGB Mode" which consists of 3 different numbers from 0 to 255, representing the colors Red, Green and Blue respectively. In this example our input data vectors, which are RGB colors represented as (R,G,B) are 3-dimensional vectors. We can map these 3-dimensional vectors to a plane, called Map, which is a 2-dimensional space using SOM, as shown in Figure 3.3. In Figure 3.3, it can be seen that each input data vector is mapped to one of the points in the map. Points topographically close to each other have similar colors which indicates their relations. Each of the point in this map has its own reference vector. The values of each reference vector are the RGB values of the color which that reference vector represents.

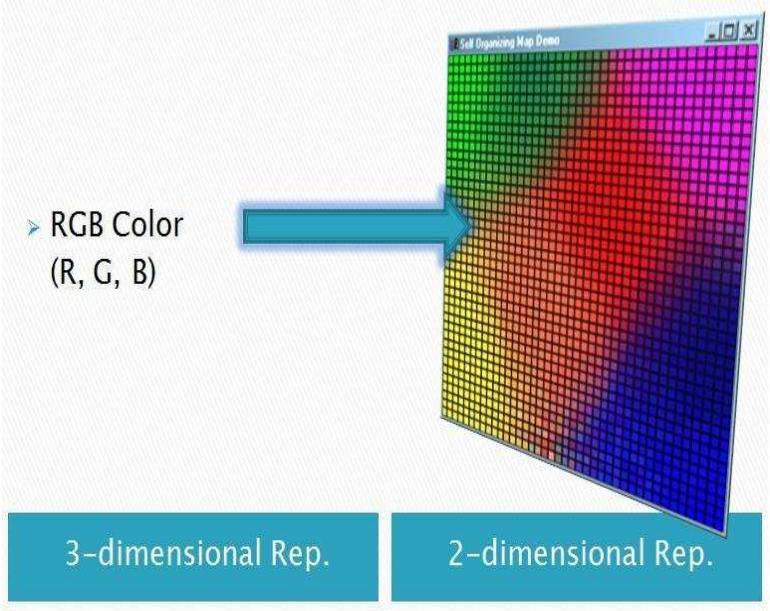

Figure 3.3: Mapping 3-dimensional RGB color data set to a 2-dimensional color data set.

Each SOM consists of three main components: *1)* Input Data Vectors; *2)* Map; and *3)* Reference Vectors. To build a SOM and and to take advantages of its properties, two processes are required: the *Learning Process* and the *Classification Process*. In the following, we first introduce SOM's components and then we describe the processes.

3.1.1 Input Data Vectors, Map, Reference Vectors

There are three main components in each SOM:

- Input Data Vectors
- Map
- Reference Vectors

An *Input Data Vector* $x_i = X_i(t) \in \Re^n$, is a vector consisting of the input data, where n is the number of dimensions of the input and t is the discrete time for inputs that change over time. In the color set data example, the input data vector is an array of different RGB mode colors where $n = 3$.

A *map* is the output of a SOM, usually in 2-dimensions, that consists of some interconnected cells called Neurons (Units).

With every neuron j in the map, there is an associated parametric *Reference Vector* $m_j = M_j(k) \in \Re^n$ which has the same number of dimensions as the input data vector $X_i(t)$. There is no clear definition of which m_j values can be used as our reference vectors. However, there is some research in defining these values as optima of some energy functions [23], [24], [4]. In mapping a high-dimensional input data vectors onto a two-dimensional space, SOM is defined as a "nonlinear projection" of the probablity density function ($p(x)$) of those input data vectors. According to [25]: "an optimal mapping in finding the best matching neurons would be the one that maps the probability density function $p(x)$ of our input data vectors in the most reliable way, trying to preserve at least the local structures of $p(x)$". In our simulations, we use one of the most popular methods, stochastic approximation derivation [26], which is based on the original form of SOM learning process.

The initial values of these Reference Vectors can be selected at random as the SOM stochastic algorithm runs through a regression process. However, these values are more suitable if initialized in some orderly fashion along with the 2-dimensional subspace spanned by the two principal eigenvectors of the Input Data Vectors [4].

To obtain the map, the SOM has to be trained with some input data vectors. We have to know the characteristics of these input data vectors to be able to interpret the map. This interpretation means getting an idea about each unit in the map. Each of these input data vectors will be mapped to one unit in the map and since we know the characteristic of our input, we would know the dependencies between our neurons in the map. In order to train a SOM, we need to go through the Learning process described in the following section.

3.1.2 Learning Process

The learning process is unsupervised, which means we do not need to monitor the training process or correct the output which is called the "map". The learning process uses competitive algorithms and methods to train SOM as to how to map a sample input to a neuron in the map.

In our framework, SOM is trained by collecting known data representing normal behavior and abnormal behavior of the network to produce a traffic model consisting of clusters (neurons) such that each packet passing through the network can be placed in one of these clusters. After training SOM in the learning phase, we have a *traffic model* of the network containing clusters for normal packets and abnormal packets.

In this section, we will show how SOM finds the corresponding map according to the sample input data vectors and how it can be applied in our methodology. Learning is done in two phases [27]:

- Finding the *best matching unit* (BMU)
- Moving the *winner* along with its neighborhood towards the best value

Best Matching Unit (BMU)

In the first phase of the learning process, at each iteration k, we find the *best matching unit* (BMU_j) for the input data vectors $x_i \in \Re^n$ in the map.

We used the *Euclidean Distance* as the best candidate to find the *BMU*. The smallest value of the Euclidean distances $\|x_i - m_j\|$, is defined as the best matching node and denoted by the subscript c [27]:

$$\|x_i - m_c\| = min_j\{\|x_i - m_j\|\} \tag{3.1}$$

$$c = arg\ min_j\{\|x_i - m_j\|\} \tag{3.2}$$

Each input data vector $x_i \in \Re^n$ is compared with each reference vector m_j and the best matching unit (BMU) of that input data vector is obtained. Therefore, the input data vector is mapped onto the neuron c in the map.

Winner Update

In the second phase of learning process, after finding the winner (BMU), we move it along with its neighborhood toward its best value. Neurons topographically close to

a winner neuron, up to a certain distance (radius), are called the winner's neighborhood. The radius of this neighborhood is a parameter itself which changes during the learning phase. Values of $m_j = M_j(k) \in \Re^n$ as neighbors of a BMU are defined as convergence limit of the following formula [4]:

$$M_j(k+1) = M_j(k) + \beta(k) * C_{j,c}(k) * \Big(X_i(t) - M_j(k)\Big) \tag{3.3}$$

where $M_j(k)$ is the reference vector of the j^{th} neuron in the map at iteration k, $\beta(k)$ is a monotonically decreasing function of time called *learning rate* function, $C_{j,c}(k)$ is the *neighborhood function*, and $X_i(t)$ is the i^{th} input data vector at time t.

The *neighborhood function* depends on the lattice distance between two neurons n_1 and n_2 in the map. For the neighborhood function, we have different options such as Gaussian function, Bubble function and so on. The simplest function is the one that has its value as *one* for all the BMU's neighborhood and is *zero* for the rest of neurons in the map. In our framework, we use the *Gaussian* function, as it is one of the most widely applied functions.

The Gaussian function can be written as [4]:

$$C_{j,c}(k) = \exp\Big(-\frac{\|r_c - r_j\|^2}{2\delta^2(k)}\Big) \tag{3.4}$$

where $r_c \in \Re^2$ and $r_j \in \Re^2$ are the radius vectors of neurons with indices c and j respectively, c is the index of the BMU. As $\|r_c - r_j\|$ increases, $C_{j,c}(k) \to 0$. The parameter $\delta(k)$ is a monotonically decreasing function of time which corresponds to the radius of the neighborhood set of neurons around a BMU. The exact form of this function is not important and can be formed as a simple linear function of time.

The *Learning Rate* function is another monotonically decreasing function. According to *Stochastic Approximation Theory* [27], the learning rate function, at least, should satisfy the following formulas:

$$\sum_{k=0}^{\infty} \beta(k) = \infty \tag{3.5}$$

$$\sum_{k=0}^{\infty} \beta^2(k) < \infty \tag{3.6}$$

In our simulations, we use a simple function of the type $\beta(k) = \frac{a}{b+k}$, where a and b are constants. Further discussion about The *Learning Rate* function and the

parameter $\delta(k)$ is presented in *Performance Improvement of the Self-organizing Map* section.

Figure 3.4 illustrates the best matching unit of a particular input in the RGB color example and the neighborhood concept. It also shows the topology of the map, which in this case is *hexagonal*. The other common topology used in SOMs is *rectangular* in shape. It can be seen from this figure that a neuron is found in the map for a particular input data, this neuron is called BMU for that input data. After finding this neuron, the reference vector as well as its neighborhood are updated with better (more precise) values.

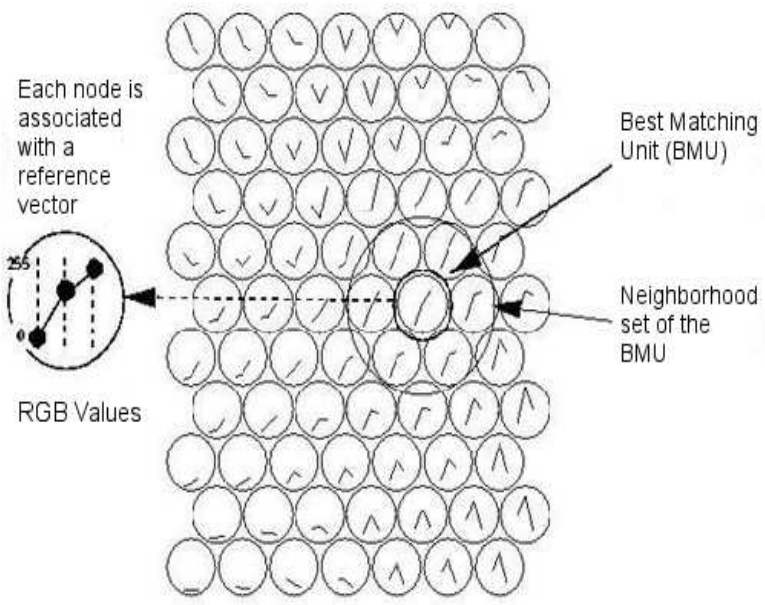

Figure 3.4: Finding the best matching unit for a particular input and its neighborhood units [3].

Figure 3.4 clearly shows the dependencies between neurons of the map. It can be observed that neurons (units) which are close enough to each other have almost the same linear shape, which means their actual values or colors are similar to each other. By exploiting this ability in our framework, we use our trained SOM in the *classification process* described in the following section to classify the packets (users) into clusters of normal and abnormal packets (users). These clusters are nothing but the neurons in the map.

3.1.3 Classification Process

The second process after generating a map is the *classification process* which uses a similar *Euclidean Distance* measure as in the learning process to find best matching units for input data vectors. We compare each input data vector with existing reference vectors in the map, find the BMU for that input data vector, and map it to the neuron to which the BMU belongs. At the end of this process, each input data vector, in our case packets collected from the network, is mapped to a neuron (clusters of normal and abnormal packets (users)).

In the following sections, we describe the tiers of our framework in detail and discuss how the above processes are used to detect malicious user communities.

3.2 Abnormal Data Rate Traffic Detection Tier

A packet passing through any network contains information in both its header and payload. Packet payloads carry information about the communication patterns or the contents. However, use of packet payload can be problematic in some cases. One case for not using the packet payload for analysis is when data encryption is used. We cannot inspect contents of packets to get information about their communication patterns when they are encrypted . Another problem is the size of packet payload. Analysis of packets with large packet payload is slow and time consuming. Therefore, using the packet payload analysis increases the time complexity of the framework dramatically, where as building a fast framework close to a real-time system for inspecting thousands of packets is very important. Therefore, only packet headers are considered in this framework.

To classify packets sent by users of a network, some features to reveal characteristics of the communication methods of these users are needed. The number of these features should be as small as possible to decrease the complexity of the framework but should be sufficient to show the dependencies and similarities of the packets regarding the data rate patterns. In our framework, we create vectors of features as the input data vectors of the SOM for each packet. These features are extracted from packet headers along with other statistical data. We use **5 different features** that are listed below to classify packets into two groups of normal and abnormal packets. Based on our simulation results presented in the performance evaluation, analysis and comparisons section of this thesis, we are able to detect malicious user communities

with very low false negative rates and false positive rates and high accuracy by using these 5 features:

1. Third nibble of source IP address
2. Third nibble of destination IP address
3. Protocol used in sending the packet
4. Average of inter-arrival time between sending packets of a user
5. Variance of inter-arrival time between sending packets of a user

For illustration purposes, it is assumed that the network is constructed based on IPv4 addressing (e.g., `xx.yy.zz.aa`). A further assumption is that the `'zz'` nibble represents the subnet; This nibble is extracted from sender's and receiver's IP addresses to be considered as *the first and second elements* of our input data vectors of features. The most significant two nibbles of an IP address, (e.g `xx.yy`) are usually the same for users within a LAN and also they change infrequently. The last nibble, (e.g. `'aa'`) is usually dynamic for a user and does not give us any useful information about the sender or receiver. So in our work, we do not consider these nibbles for classification purposes. Users belonging to communities are distributed in different LANs. However, they are not randomly distributed. With a high probability, one can say that if a LAN contains a malicious user belonging to a community, there would be definitely more malicious users from the same community in that LAN. The reason is that the user communities concept is based on communication of different users who either know each other or send information to specific users. Therefore, the `'zz'` nibble is selected from an IP address of both sender and receiver to specify the sender and the receiver. However, to construct the graph of suspicious users, we also add `'aa'` nibble of both sender's and receiver's IP addresses to the input data vectors to be able to distinguish users from one another within the same LAN. But these nibbles are not considered as our features for classification purposes.

The third feature that is extracted from the packet header is the protocol field. In data networks, usually applications with different protocols may have different data rates. There could be some packets belonging to specific protocols (such as web updates) which naturally use a different data rate pattern (such as low data rate) and these packets are not sent by malicious users and do not deviate from the standards of the network. To distinguish packets which naturally have been sent in a low data

rate because of the valid applications from those which have been sent by malicious users, we collect the protocol field used in the packets as one of our main features in our data collection.

In order to collect all the information about data rate patterns of a traffic and reveal the dependencies between the users, the average inter-arrival time of packets and the variance of these inter-arrival time are calculated by using moving averages for each host and updated whenever a user's packet passes through the framework. For each user in our network, we start calculating and storing the inter-arrival time of sending packets by the user and also the variance of this inter-arrival time. By using moving averages, we store just the updated value of inter-arrival time and its variance for each host.

In off-line detection of malicious user communities, which is our focus in this research, a reasonable measurement window size (MWS) must also be defined. We use this MWS to collect input data vectors of users in the network within its size. It can be either a Timing Window or Packet-counting Window. In Time Measurement Window Size (TWMS), we collect input data vectors within each pre-defined period of time and in Packet-counting Measurement Window Size (PWMS) we collect input data vectors to the number of a pre-defined value. This MWS can be easily obtained by examining the monitored network and determining its best value according to the constraints of the network. During each MWS, every time a packet passes through the network, we create it's data vector and save it in a file called input data vector file. This file is used as the input for our neural network system.

A "7-tuple Input Data Vector" (`'zz'` and `'aa'` nibbles of sender's and receiver's IP addresses, protocol number, average of inter-arrival time, variance of inter-arrival time) is created and five of them are used for classification purposes as discussed before. Five of these features can be easily obtained from packet headers and the other two can be obtained with some simple and fast mathematical calculations.

In the on-line detection of malicious user communities, the dependencies between these features could be different in various networks as some malicious communities tend to communicate regularly using an abnormal data rate and some may communicate irregularly. Therefore, in order to apply our framework in an on-line detection system, we need to collect a history of the monitored network and its users frequently to be able to get to a correct decision about a user. After collecting a history of the network, for each packet passing through the network, we gather the input data vector of features based on the current packet along with the history of the network

and then decide whether this packet belongs to a suspicious user group or not. The combination of these features, just as far as off-line detection systems, can reveal the dependencies of packets easily and using SOM helps us to classify packets into normal and abnormal packets.

3.2.1 Creating Graph of Suspicious Users

The input data vector file is created by collecting the 7-tuple data vectors within each data collection window as discussed earlier. This input data vector file is then processed by the SOM to classify each input data vector into one of the neurons (clusters) of normal or abnormal users. Users of those input data vectors which have been classified as the abnormal neurons are then identified, based on the pair of `'zz','aa'` nibbles of sender's IP address. We then create the graph of these suspicious users. Nodes in this graph are suspicious users who might belong to some malicious communities with abnormal communication rate patterns. Edges show the connections between the users; in other words, the users to whom these suspicious users have sent their packets to in an abnormal data rate pattern. After creating this graph, it is then passed on to the second tier of our framework to purify the graph and identify the user communities of the monitored network. The way we create this graph is based on the characteristic of the community detection algorithm that we use in the second tier such as creating a directed or undirected graph, weighted or unweighted graph and so on.

3.3 User Community Detection Tier

In this tier, we first obtain the graph of suspicious users from the first tier of the framework. Then communities hidden in the graph of suspicious users are extracted. Edges are undirected in this graph. We have no information about the connections of local users located within the same LAN. This framework has no access to these connections since the framework is established on the routers and generally a LAN uses Layer 2 technology. There are several algorithms [28] proposed to detect community structures in graphs. The most popular algorithm suited to this work is Girvan-Newman algorithm [5]. Unlike most of other community detection algorithms which are based on *Edge Centrality*, the focus in Girvan-Newman algorithm is based on the concept of *Edge Betweenness*. Most other algorithms assign a weight for each edge in

the graph and then start their progress from the strong weights toward the weakest weights. In these algorithms, edges having highest weights are the most central edges within the community.

Although these kind of algorithms work fine on most of the graphs, they have some pathological cases like their inability to classify a node of a graph that belongs to a community that has just one connection though an edge within the network. For example, it is very possible that in a malicious user community users may just communicate with one of the users known as master, such as connections in botnets. Therefore, these users will have only one connection in the malicious user community that they belong to.

On the other hand, the Girvan-Newman algorithm tries to find edges that are least central in the communities. Also in our research, we face several networks that will consist of communities that are densely connected within themselves. Since these communities perform illegal tasks within their own community structures, they rarely connect to other communities, which is another reason to use a community structure detection based on edge betweenness. The detection of communities in a graph in the Girvan-Newman algorithm is based on progressively removing edges that are between communities.

The *edge betweenness* concept is a measure that puts weights on edges in a graph based on scores. The score for each edge is the number of shortest paths between vertices through that edge. The main idea of the community structure detection algorithms based on the edge betweenness concept is that it is very likely for edges connecting different and separate communities to get high edge betweenness weight since all the shortest paths from one community to another community must traverse through them [29]. In order to get a hierarchical map of the graph consisting of just hidden community structures of suspicious users, the edges with the highest weights of edge betweenness are incrementally removed from the original graph. At the end of the refining process, a disconnected graph revealing malicious user communities is generated.

The following is the pseudocode for the Girvan-Newman algorithm [28]:

1. The betweenness of all the edges in the graph is computed.
2. The edge with the largest betweenness is removed. If we have edges with the same and largest amount of betweenness, one of them is picked at random.
3. The betweenness of all the edges on the running graph is recalculated.

4. We repeat the the cycle from step 2 until no edges remain.

The performance evaluation of Girvan-Newman algorithm in undirected graphs is studied in [28] and [5].

3.4 Performance Improvement of the Self-organizing Map

There are many factors that affect the quality of the clustering model produced by the SOM. These factors are usually involved in the process of the learning phase, such as learning rate, neighborhood function, and so on. In the learning phase of the framework, clusters of network traffic are produced by providing training data for SOM which in turn constructs a map. Here we face one more important factor, the number of clusters in the map, which is really essential to the quality of the model. In this section we study how different sizes change the quality of model.

In order to perceive this relation, the quantization error needs to be studied which is a criterion used in this thesis to indicate the fitness of the clustering model. According to [23], the local minimum of the error function can be considered as the convergence state of SOM and this error function is written as:

$$E = \sum_{i=1}^{N} \sum_{j=1}^{M} \big(c - j\big) \big\|x_i - m_j\big\|^2 \tag{3.7}$$

where c is the index of the best matching unit in the map for x_i, j is the index of a unit in the map, x_i is an input data vector, m_j is the j^{th} reference vector in the map, N is the number of samples in the training phase and M is the map size.

From equation 3.7, two formulas can be inferred in order to measure the quality of the model. The first one is called the *average quantization error* and the second one is called the *average distortion measure*. The average quantization error is the mean of

$$\big\|x_i - m_c\big\| \tag{3.8}$$

for all x and the average distortion measure is calculated by taking the average of the following equation for all x:

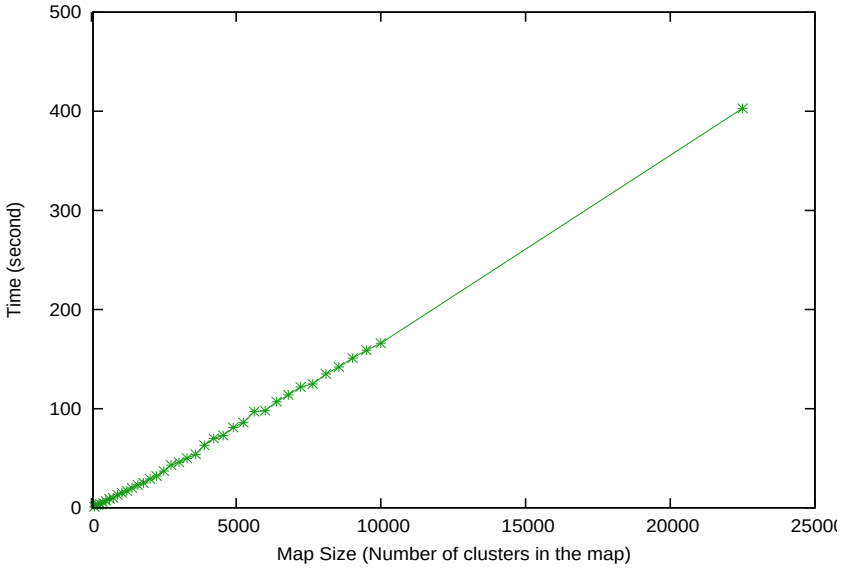

Figure 3.5: Training time versus number of clusters

$$\frac{\sum_{j=1}^{M} F\big(c,j\big) \times \big\|x_i - m_j\big\|^2}{N} \tag{3.9}$$

where c is the index of the best matching unit of x_n in the map, m_i is the i^{th} reference vector in the map, and F is the neighborhood function defined as:

$$F\big(c,j\big) = \frac{\alpha(t) \times \exp\big(-\frac{\big\|r_c - r_j\big\|^2}{2\delta^2(t)}\big)}{N} \tag{3.10}$$

where $\alpha(t)$ is the learning rate, $r_c \in \Re^2$ and $r_j \in \Re^2$ are the radius vectors of units c, j, $\delta(t)$ is the width of the kernel and N is the number of samples in training phase.

Figure 3.5 shows the plot of training time versus map size from which we can observe a linear increase of the training time with respect to the number of clusters. Therefore, number of clusters in a map must satisfy two conditions: *1)* it should be big enough to cover different kinds of data-rate patterns in the network. *2)* it should be as low as possible for a low complexity of the system in the training phase and in the classification phase. By considering these two conditions, we use the concept of quantization error and distortion measure to find the best fitting value for the number of clusters in the map.

The SOM in our framework is trained with different number of clusters and the

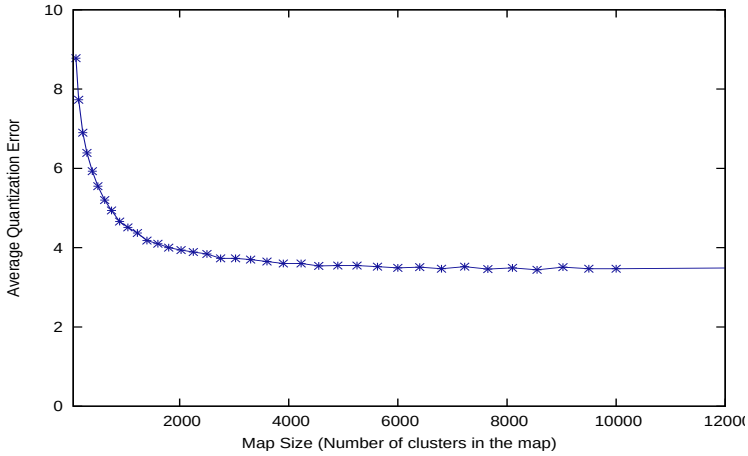

Figure 3.6: Average quantization error versus number of clusters

average quantization error, average distortion measure for each of them are calculated. An input vector file containing 15000 samples of data vectors from a simulated network is used. As shown in Figure 3.6, it can be seen that as the number of clusters increases, average quantization error decreases which means we would have a lower quantization error for traffic model if we have more clusters in map. The average distortion measure (Figure 3.7) shows a similar result. From Figure 3.7, it can be seen that there are some points as local minima. It is obvious that the very first local minimum has the lowest time complexity too. The difference between the distortion measure of the very first local minimum and other local minima is negligible. Therefore, we consider the first local minimum as the best candidate for the number of clusters in the map or map size.

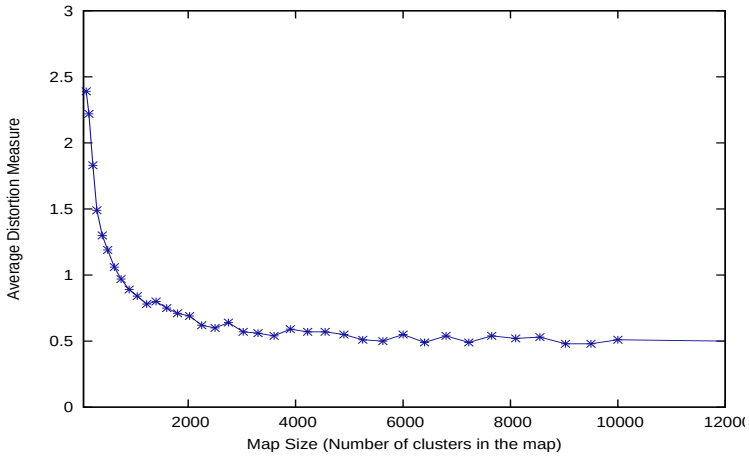

Figure 3.7: Average distortion measure versus number of clusters

Chapter 4

Performance Evaluation

4.1 Simulation Environment

We used OMNET++ 4.0 (http://www.omnetpp.org) discrete event simulation tool to build networks of users with two different topologies: *Ring Type Network* and *Network With Random Topologies*.

Each network consists of several LANs connected to each other based on the two topologies through a set of routers. We set up some *probes* between routers to measure the inter-packet arrival times originating from LANs connected to those routers. Also, some malicious user communities are randomly created and spread throughout different LANs within the network which use abnormal data-rates for their traffic.

These communities are created by constructing different cliques. After constructing these cliques, we remove each edge of these cliques with a probability p (where p is a small value, like 10%). We randomly spread users of these malicious communities in the network to communicate among themselves and also with other users as well.

At each router, a *probe* is used to collect required features for the Input Data Vector for the traffic sent only by the LANs connected to that router. Since these routers are connected to each other according to the topology of the network, it is reasonable to assume that a router stands in the middle way of a packet while it's not connected to the source LAN or destination LAN. Each of these probes send their data to the first tier of the framework at the end of each *Measurement Window Size* to start the classification process.

In the following sections, we present our simulation results. For each network

topology, we used two different data rate configurations for our framework. In the first configuration (*Rate 1*), we assumed that the maximum inter-arrival time for a normal user to send packets is 120 seconds and the minimum and the maximum inter-arrival times for an abnormal user are 90 seconds and 5 minutes. In the second configuration (*Rate 2*), the maximum inter-arrival time for a normal user to send packets is assumed to be 45 seconds and the minimum and the maximum inter-arrival times for an abnormal user are assumed to be 30 seconds and 3 minutes. These times are approximately standard times in daily usage of applications in networks.

4.2 Performance Evaluation

We conducted various simulation experiments using both rate configurations and results concur very well for all. We calculated *false negative rate*, *false positive rate*, *true positive rate*, and *accuracy* for all of our simulation results to test the quality and effectiveness of our framework. We then draw *Receiver Operating Characteristic (ROC) Curves* for each set of results.

Our work is a two-class prediction problem, in which the outcomes are labeled either as normal packets (N) or abnormal packets (AN). Therefore, we have four possible outcomes. A *false positive* for our case occurs when we classify a packet as an abnormal while it is a normal packet. A *false negative*, on the other hand, occurs when we can not classify an abnormal packet. In other words, we classify an abnormal packet as a normal packet. Our focus is to have a high *accuracy* in classification of normal and abnormal packets and a good ROC Curve while reducing both false negatives and false positives. It is crucial to detect malicious users and abnormal packets as much as possible. According to [30], "the *accuracy* of a measurement system is the degree of closeness of measurements of a quantity to its actual value". The accuracy of a system is calculated by the following formula:

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \tag{4.1}$$

where TP is number of true positives, TN is number of true negatives, FP is number of false positives, and FN is number of false negatives of the system.

To draw the ROC curves, we use the true positive rates (TPR) vs. false positive rates (FPR) of the results. The best possible classification method, in ROC space, appears in the upper left corner of a ROC curve. Any point above the diagonal line

($f(x) = x$) represents a good classification method and any point below the diagonal line means a poor classification. Points along the diagonal line from the left bottom to the top right corners illustrate a complete "random guess" classification method.

Figure 4.1 illustrates a typical SOM generated by one of our simulations. The neurons (clusters) are shown in black and white color in this figure. Based on the learning process, neurons in black color can be considered as clusters of either normal packets or abnormal packets. If they are considered as normal packets, then neurons in white color are of abnormal packets and vice versa. Neurons topographically close to each other have almost similar color implying the relationship and dependencies between one another. It can be seen from this figure that neurons in white or black color are not concentrated just in one area of the map. Spreading neurons in different areas of the map indicates the variation of features in input data vectors.

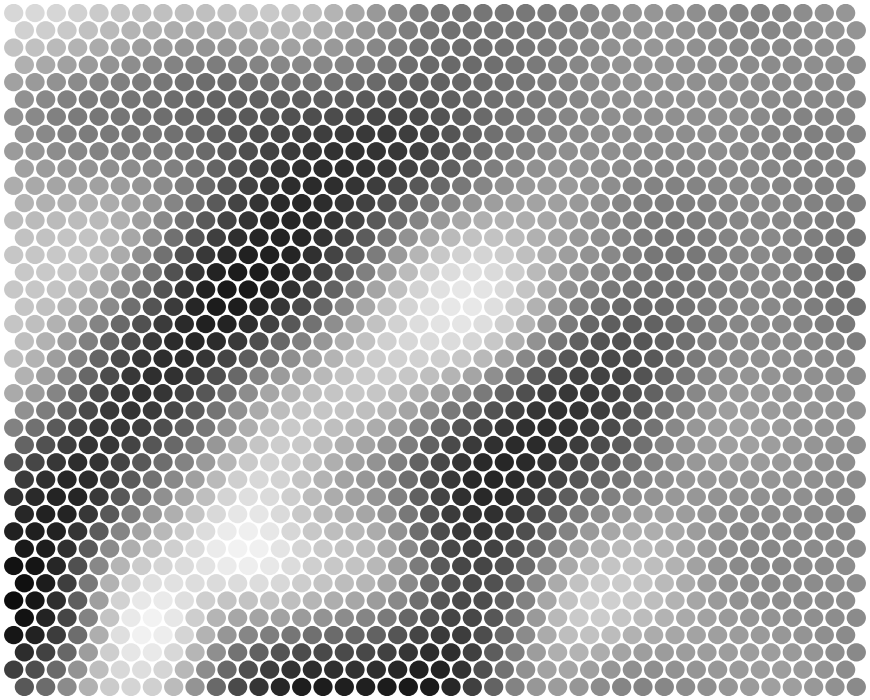

Figure 4.1: A sample generated SOM.

4.2.1 Ring Type Network Scenario

In this scenario, a *Ring Type Network* (Figure 4.2) is simulated that consists of 1000 users with a random communities of malicious users. There are 10 different LANs

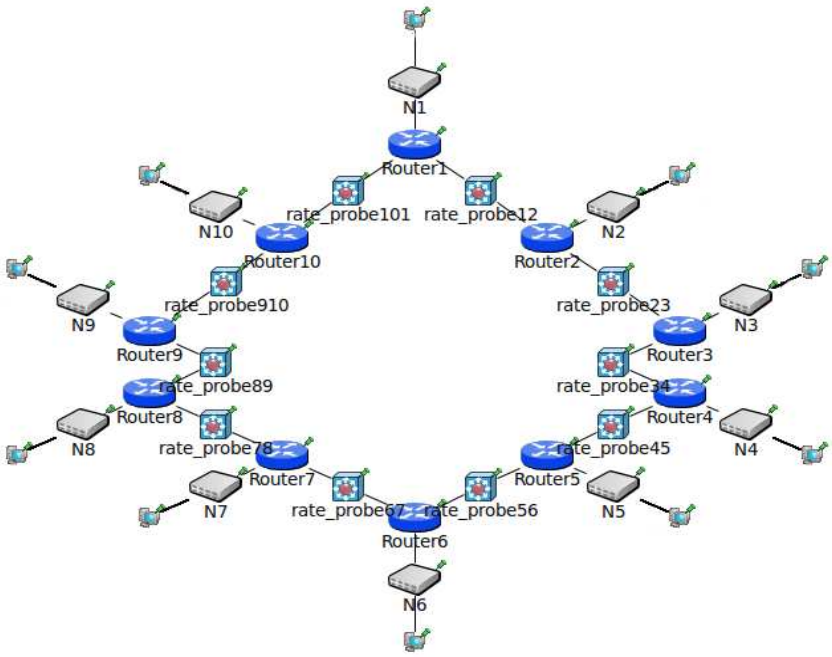

Figure 4.2: A sample network with a ring type topology.

connected to 10 routers and each LAN consists of 100 users. We gather input data vectors in three different methods to ensure the validity of our frame work: collecting a *small number (100,000) of input data vectors*, a *medium number (250,000) of input data vectors*, and finally a *large number (1,000,000) of input data vectors* in total to be processed. The results of each method is presented in what follows. Each value presented in the tables is the average value of the results of running each simulation several times for each scenario.

Small Number of Input Data Vectors

In the first method, we collect *100,000 packets* in total from the network. These packets have been generated by *1000* users in our network. In Table 4.1, the result of our first data rate configuration and in Table 4.2, the result of our second data rate configuration with 5 different scenarios are presented. The number of malicious communities and the average total number of malicious users in these communities in the network is varied between each scenario. In the first scenario, we run the simulation by having just three malicious communities and the average total number of abnormal users inside the network is set to be about *7%* of total number of users

in the network. Then, we increase the number of malicious communities and total number of abnormal users inside the network such that in the last scenario, on average, about *16%* of users are abnormal. For this method, during the training process in our first tier, a unit in the map is considered as a cluster for abnormal packets if the ratio of the number of abnormal packets to the number of normal packets mapped to that unit is larger than $10 - 15\%$. This value is almost the same as the average of ratio of the total number of malicious users to the total number of users in the network.

The numbers of abnormal users have been generated by taking the average of different *random numbers in specific ranges*, each with different seeds. In other words, for each scenario, we considered a range for the total number of abnormal users and then we generated a random number within that range and ran the simulations. In these tables, *F.N. Rate* means False Negative Rate, *F.P. Rate* means False Positive Rate, and *T.P. Rate* means True Positive Rate.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	3	77	96763	3235	23.731	5.281	76.269	94.128 ± 0.309
2	4	83	96373	3625	21.457	6.556	78.543	92.903 ± 0.593
3	5	109	95339	4659	16.511	7.439	83.489	92.139 ± 0.189
4	6	150	93307	6690	11.629	8.065	88.371	91.696 ± 0.270
5	7	167	92768	7231	8.784	8.961	91.216	91.052 ± 0.437

Table 4.1: Ring Type Network Scenario of 1000 users with a collection of *100,000* packets as input data vectors using the first data rate *(Rate 1)* configuration.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	3	65	98339	1660	23.771	2.922	76.229	96.735 ± 0.545
2	4	97	97344	2655	17.309	3.712	82.691	95.925 ± 0.512
3	5	112	96986	3012	15.846	3.501	84.154	96.133 ± 0.233
4	6	141	96072	3927	8.940	4.283	91.060	95.536 ± 0.857
5	7	168	95243	4757	10.920	4.942	89.080	94.775 ± 1.488

Table 4.2: Ring Type Network Scenario of 1000 users with a collection of *100,000* packets as input data vectors using the second data rate *(Rate 2)* configuration.

From Table 4.1 and Table 4.2, we can see that the larger numbers of malicious users in the network, we get better false negative rates and true positive rates. The false positive rates obtained from these simulations are very favorable.

As Figure 4.3 illustrates, the ROC values of our framework are all above the diagonal line for small data collection in ring type topology for both data rate configurations. As the number of malicious users in user communities increases, the ROC values in our ROC space get better values which confirms the quality and effectiveness of our framework.

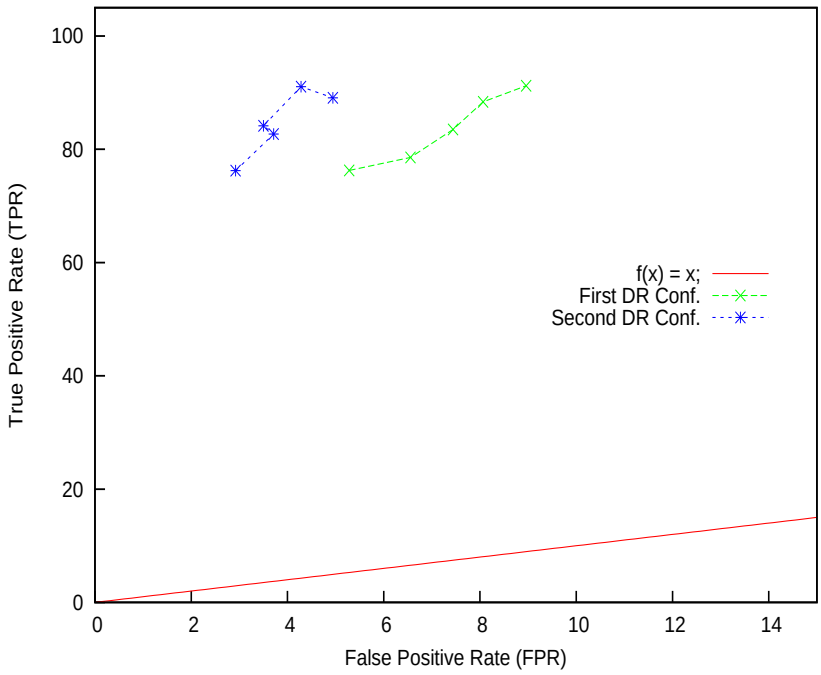

Figure 4.3: Partial ROC curve for small number of data collection in ring type networks.

Medium Number of Input Data Vectors

In the second method, we collect *250,000 packets* in total from the network. These packets have been generated by *1000* users in our network. In Table 4.3, the result of our first data rate configuration and in Table 4.4, the result of our second data rate configuration with 5 different scenarios are presented. In the first scenario, we run the simulation by having just three malicious communities and the average total number of abnormal users inside the network is set to be about *7%* of total number

of users in the network. We then increase the number of malicious communities and the total number of abnormal users inside the network such that in the last scenario, on an average, about *16%* of users are abnormal. The method of generating numbers of abnormal users and the differences between each scenario is the same as the first method. A unit in the map is considered as a cluster for abnormal packets if the ratio of the number of abnormal packets to the number of normal packets mapped to that unit is larger than $10 - 15\%$ like the previous method.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	3	66	243301	6698	22.704	4.840	77.296	94.737 ± 1.932
2	4	94	239912	10087	12.128	6.092	87.872	93.677 ± 1.229
3	5	117	238152	11846	10.048	6.137	89.952	93.693 ±1.354
4	6	142	235026	14972	7.679	7.069	92.321	92.893 ± 0.716
5	7	167	231405	18594	7.172	7.462	92.828	92.563 ± 1.175

Table 4.3: Ring Type Network Scenario of 1000 users with a collection of *250,000* packets as input data vectors using the first data rate *(Rate 1)* configuration.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	3	82	244505	5494	14.740	3.156	85.260	96.590 ± 0.286
2	4	94	243682	6317	13.147	3.432	86.853	96.323 ± 0.999
3	5	116	242115	7885	11.448	3.482	88.552	96.269 ± 0.402
4	6	140	240399	9600	6.233	3.955	93.767	95.959 ± 0.632
5	7	163	237702	12297	6.068	4.817	93.932	95.124 ± 0.603

Table 4.4: Ring Type Networks Scenario of 1000 users with a collection of *250,000* packets as input data vectors using the second data rate *(Rate 2)* configuration.

From Table 4.3 and Table 4.4, like the previous method, we can see that larger the number of malicious users in the network, the better false negative rates and true positive rates we would get. Also our false positive rates are still extremely good. While the values of false positive rates are almost similar to the previous method, which is a good starting point, the values of our false negative rates decreased which implies that by collecting more input data vectors, the precision and the quality of our proposal increases. You also can see that the accuracy of the system and the values of true positive rates are better than the previous method.

As Figure 4.4 illustrates, the partial ROC values of our framework are all above the diagonal line for average data collection in ring type topology for both data rate configurations. As the number of malicious users community increase, the ROC values in our ROC space get better values which definitely confirms the quality and effectiveness of our framework.

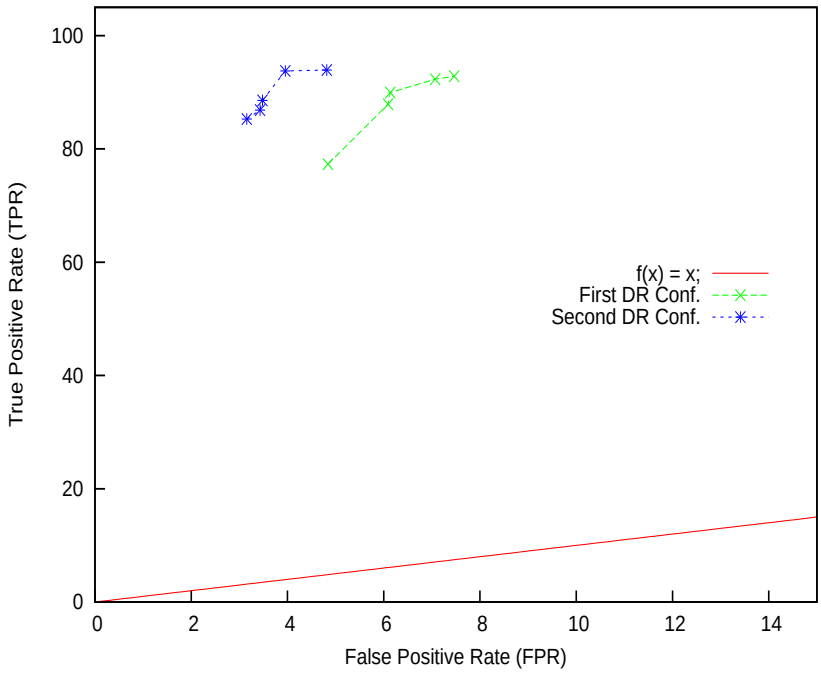

Figure 4.4: Partial ROC curve for medium number of data collection points in ring type networks.

Large Number of Input Data Vectors

In the third and last method of networks with ring type topologies, we collect *1 million packets* in total from the network. These packets, again, have been generated by *1000* users in our network. In Table 4.5, the result of our first data rate configuration and in Table 4.6, the result of our second data rate configuration with 7 different scenarios are presented. In the first scenario, we ran the simulations by having just *One* malicious community. The average total number of abnormal users inside the network is set to be about *1.5%* of total number of users in the network to have a more precise result about the quality of our framework. We then increased the number of malicious communities and total number of abnormal users inside the network such that in the last scenario, on the average, about *14%* of users are abnormal. The method of generating numbers of abnormal users and the difference between each scenario is the same as the first and second methods discussed above. A unit in the map is considered as a cluster for abnormal packets if the ratio of the number of abnormal packets to the number of normal packets mapped to that unit is larger than $10-15\%$ like the previous method.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	1	19	993039	6960	34.184	1.200	65.816	98.589 ± 0.676
2	2	37	986601	13398	22.691	2.135	77.309	97.598 ± 0.730
3	3	60	976925	23074	13.576	3.462	86.424	96.305 ± 0.236
4	4	63	975255	24744	13.774	3.543	86.226	96.211 ± 0.700
5	5	89	964354	35644	8.093	4.811	91.907	95.073 ± 0.440
6	6	99	961868	38130	8.303	4.402	91.697	95.449 ± 0.071
7	7	126	947445	51385	6.397	5.154	93.603	94.782 ± 1.094

Table 4.5: Ring Type Networks Scenario of 1000 users with a collection of *One Million* packets as input data vectors using the first data rate *(Rate 1)* configuration.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	1	16	996244	3755	28.526	0.816	71.474	99.084 ± 0.431
2	2	34	991191	8808	19.850	1.405	80.150	98.433 ± 0.086
3	3	40	989724	10274	17.324	1.879	82.676	97.962 ± 0.639
4	4	67	981945	18054	9.170	2.115	90.830	97.760 ± 0.315
5	5	95	973435	26565	7.659	2.652	92.341	97.216 ± 0.216
6	6	110	969159	30840	5.850	3.138	94.150	96.779 ± 0.536
7	7	145	958446	41553	5.001	3.228	94.999	96.698 ± 0.597

Table 4.6: Ring Type Networks Scenario of 1000 users with a collection of One Million packets as input data vectors using the second data rate *(Rate 2)* configuration.

In Table 4.5 and Table 4.6, like the previous two methods, we can see that larger the number of malicious users in the network, the better false negative rates and true positive rates we would get. Also our false positive rates are still extremely favorable again. The values of false positive rates are much better than the last two methods which indicates that the values of our false negative rates are reduced. This implies that by collecting more input data vectors, the precision and the quality of our proposed framework increases. We can also see that the accuracy of the system and the values of true positive rates are better than the previous methods.

As Figure 4.5 illustrates, the partial ROC values of our framework are all above the diagonal line for average data collection in ring type topology for both data rate configurations. As the number of malicious users community increase, the ROC values in our ROC space get better values which confirms the quality and effectiveness of our framework even more.

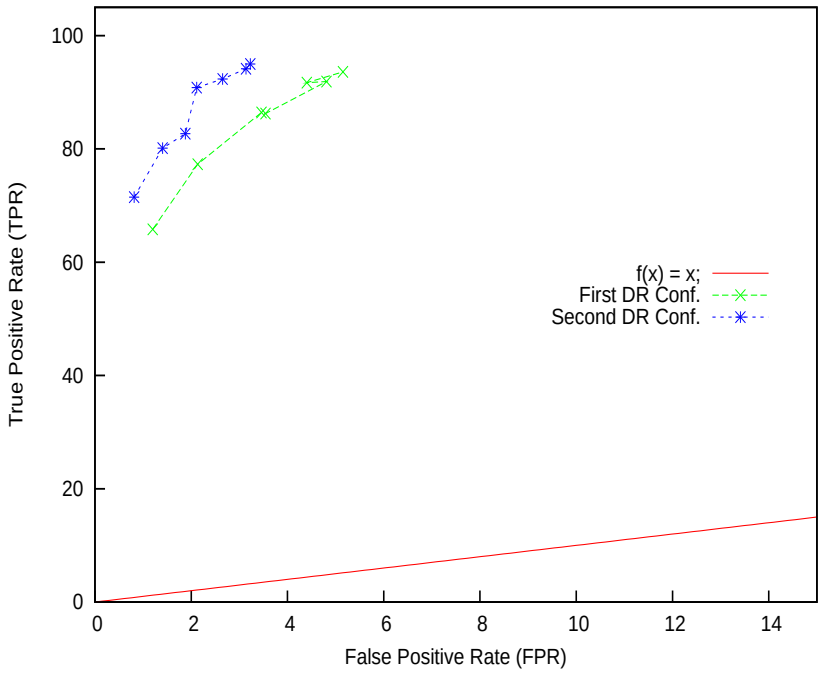

Figure 4.5: Partial ROC curve for Large number of data collection in ring type networks.

Our simulation results show that by collecting larger number of input data vectors we would have better false negatives and false positives for Ring Type Networks, although all of these results are pretty much similar in different scenarios which

ensures the validity of our framework. Our simulation results also show that by having more abnormal users in the network we would have much better false negative rates. The reason behind this conclusion is that increase in the number of abnormal users means collecting more data about characteristics of normal users and abnormal users in the network. This data helps during the training phase of our self-organizing map and in classifying normal and abnormal users to different groups.

The accuracy of our system in detecting these communities is presented in Tables 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 by collecting different samples of input data vectors and taking the average of multiple runs for the first data rate configuration (*Rate 1*) and the second data rate configuration (*Rate 2*). It can be seen from these tables that our framework is able to detect malicious users communities with very low false negative and false positive rates, and these rates are also pretty much similar for different rate configurations.

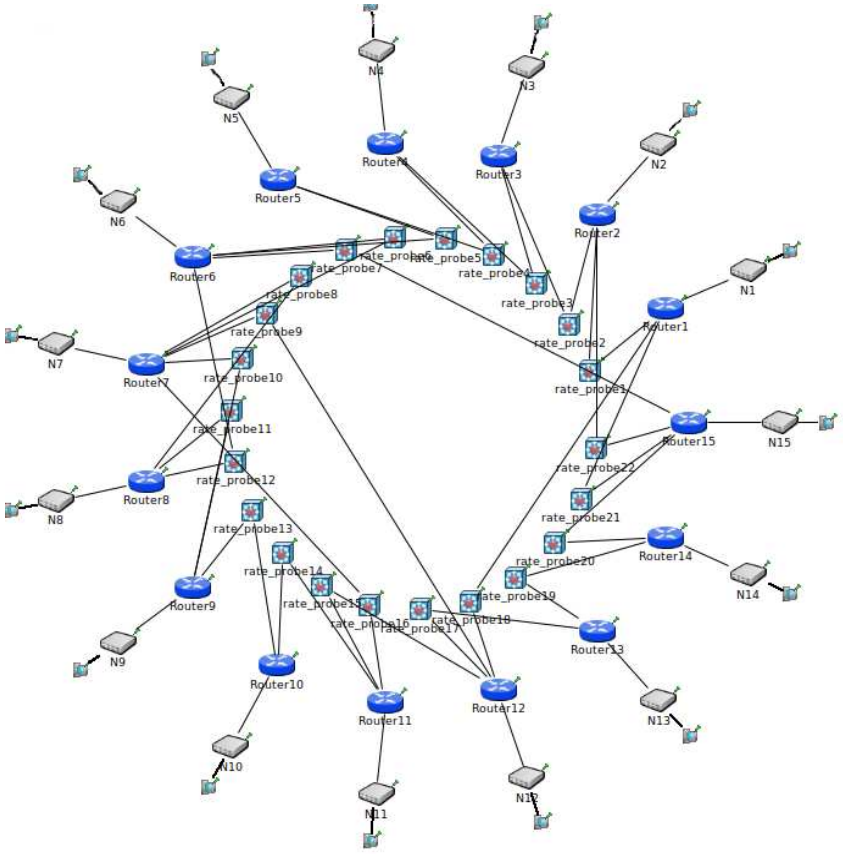

Figure 4.6: A sample network with random topology

4.2.2 Networks With Random Topologies

In *Networks With Random Topologies* (Figure 4.6), we set up routers and LANs and then connect them to each other with random connections. We ensure that all routers are connected to each other.

For this network, we increased the size to about *4000 users* and *15 routers*. The data rate configurations are the same as the ring type network. In this case, running each simulation takes a very long time and we used *1 million packets* in total from the network for input data vectors. In this type of network, a unit in the map is considered as a cluster for abnormal packets if the ratio of the number of abnormal packets to the number of normal packets mapped to that unit is larger than $3 - 5\%$ during training process in our first tier. This value is close to the ratio of the total number of malicious users to the total number of users in the network.

In Table 4.7, the result of our first data rate configuration and in Table 4.8, the

result of our second data rate configuration with 7 different scenarios are presented. In the first scenario, we run the simulation with just *one* malicious community. Then, we increased the number of malicious communities and the total number of abnormal users inside the network such that in the last scenario, on the average there are about *5%* of users as abnormal.

The numbers for abnormal users have been generated by taking the average of different *random numbers in specific ranges* as explained before. The simulations are run multiple times for each scenario and then we took the average of these simulation results.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	1	20	982718	1721	53.910	1.583	46.090	98.327 ± 1.029
2	2	45	983647	4670	38.652	3.729	61.348	96.111 ± 1.494
3	3	62	984104	5881	30.736	4.762	69.263	95.084 ± 1.422
4	4	110	982560	11000	20.067	6.960	79.933	92.896 ± 0.966
5	5	132	1007188	12864	14.452	6.961	85.548	92.947 ± 0.969
6	6	159	984600	15397	12.625	8.294	87.376	91.639 ± 1.557
7	7	183	982006	17988	10.046	8.865	89.954	91.113 ± 1.613

Table 4.7: Random Type Networks Scenario of about 4000 users with a collection of *One Million* packets as input data vectors with the first data rate *(Rate 1)* configuration.

Scenario	*Communities*	*Abnormal Users*	*Normal Packets*	*Abnormal Packets*	*F.N Rate*	*F.P Rate*	*T.P. Rate*	*Accuracy*
1	1	27	998338	1660	40.380	1.486	59.620	98.449 ± 0.991
2	2	62	995949	4048	19.570	2.963	80.430	96.970 ± 0.360
3	3	80	994601	5394	18.656	3.686	81.344	96.234 ± 0.417
4	4	98	993483	6513	17.695	3.734	82.305	96.176 ± 1.239
5	5	126	991858	8139	12.561	3.913	87.439	96.017 ± 0.205
6	6	143	990678	9320	10.774	4.578	89.226	95.367 ± 1.441
7	7	183	987878	12118	11.048	5.354	88.952	94.577 ± 0.066

Table 4.8: Random Type Networks Scenario of about 4000 users with a collection of *One Million* packets as input data vectors with the second data rate *(Rate 2)* configuration.

From Table 4.7 and Table 4.8, it can be seen that the more the number of malicious users in the network, the better false negative rates and true positive rates we would get. Also our false positive rates are extremely favorable.

As Figure 4.7 illustrates, the ROC values of our framework are all above the diagonal line for both data rate configurations. As the number of malicious users community increase, the ROC values in our ROC space get better values which confirms the quality and effectiveness of our framework.

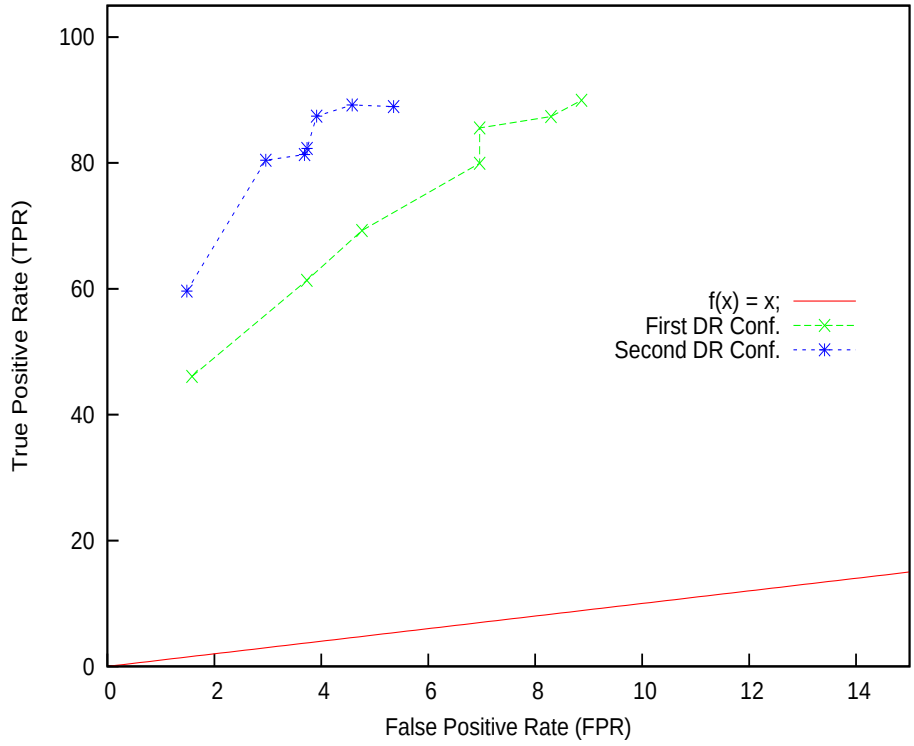

Figure 4.7: Partial ROC curve for networks with random topologies.

Chapter 5

Conclusion and Future Work

Network security has become one of the major concerns in the past two decades for network administrators. Existing methods to detect these security incidents can be classified into two different categories: *a)* Signature-based behavior detection and *b)* Anomaly-based behavior detection.

In this thesis, we focused on Anomaly-based behavior detection systems by designing and analyzing a framework to detect malicious user communities based on data communication rates. By exploiting data mining, network data management, social networks models and graph algorithms, we proposed a novel two-tier architecture to detect malicious user communities in data networks. In the first tier, specific features are extracted from packet headers along with a few statistical measures to build a feature vector for each packet passing through the network. These feature vectors are built based on reasonable measurement window sizes. We then used self-organizing maps as a classification module to classify users into groups of normal data-rate users and abnormal data-rate users. After classifying users into groups, a graph of suspicious users is created from abnormal data-rate groups and passed this graph to the second tier. In the second tier, the Girvan-Newman algorithm based on the concept of edge betweenness is invoked to detect communities hidden in suspicious user graphs. We also presented an improvement for performance of self-organizing map by studying how different number of clusters in the map affect quality of the traffic model.

The results of our simulations show that our framework is able to detect malicious users communities with high accuracy and very low false negative and false positive rates. According to our results, higher the number of communities and malicious users, the better false negative rate, true positive rate, and accuracy we would obtain

in detecting malicious users communities.

The main contributions of this thesis have two main aspects. First, we proposed a novel two-tier architecture to detect malicious user communities in data networks based on communication rate. Studying anomaly behavior of systems based on a communication rate is a topic which is rarely discussed in research before. Second aspect is that we performed simulation studies of the proposed framework for abnormal data rate communication in networks with different topologies.

5.1 Further Research Issues

To determine a complete framework to detect malicious user communities in data networks, there are many related interesting issues that need further investigation. There are three main open issues that need to be addressed:

1. In our proposed framework, feature vectors are built using off-line methods by considering a reasonable measurement window size. In large and fast networks, we need to detect malicious users using on-line methodologies. It will be interesting to study how to incorporate our framework in on-line detection systems to detect malicious user communities. In order to apply our framework in an on-line detection system, we need to collect the history of the monitored network and its users very frequently in order to make a correct decision about a user.

2. Our framework is not designed to detect malicious user communities in all types data networks. It does not consider NAT (Network Address Translation), which is used widely on LANs. All the traffic from hosts of the LAN to the public network have the same source IP address. It will be interesting to see how we can extend our framework to distinguish normal users and malicious users on LANs with NAT.

3. Validation of our framework is carried out via simulation using two different network topologies with two different user data rate configurations, which showed a very good false negative and positive rates. A study not only by simulation but on a real network (with real users etc.) scenarios also should be performed in future.

Based on our analysis, the proposed framework shows as a promising technique to detect malicious user communities based on communication rates. Many opportunities for further improvement and extension of our framework still exist. In this section, a few key issues are highlighted with the hope of inspiring further research interests in this area.

Bibliography

[1] H. Kozushko, "Intrusion detection: Host-based and network-based intrusion detection systems," vol. 1, September 2003. Independent Study.

[2] "Kohonen network - background information." `http://www.lohninger.com/helpcsuite/kohonen_network_-_background_information.htm`, 2008.

[3] T. Honkela, "Von foerster meets kohonen: Approaches to artificial intelligence, cognitive science and information systems developmen," *Kybernetes*, vol. 34, no. 1/2, pp. 40–53, 2005.

[4] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[5] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 7821–6, June 2002.

[6] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Comput. Surv.*, vol. 39, no. 1, p. 3, 2007.

[7] W. Lee, S. J. Stolfo, and K. W. Mok, "A data mining framework for building intrusion detection models," *Security and Privacy, IEEE Symposium on*, pp. 01–20, 1999.

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

[9] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, (New York, NY, USA), pp. 41–52, ACM, 2006.

[10] B. Shirley and C. D. Mano, "A model for covert botnet communication in a private subnet," in *Proceedings of the 7th international IFIP-TC6 networking conference on AdHoc and sensor networks, wireless networks, next generation internet*, NETWORKING'08, (Berlin, Heidelberg), pp. 624–632, Springer-Verlag, 2008.

[11] B. Shirley and C. Mano, "Sub-botnet coordination using tokens in a switched network," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, pp. 1 –5, Dec 2008.

[12] N. V. Sorensen, S. B. Sorensen, K. D. Feuz, G. Kerzhner, and C. D. Mano, "Detecting covert botnets using communication patterns." `http://digital.cs.usu.edu/~xqi/Teaching/REU09/Website/Neal/nsorensen.html`.

[13] H. F. Lipson, "Tracking and tracing cyber-attacks: Technical challenges and global policy issues," tech. rep., CMU/SEI-2002-SR-009. CERT Coordination Center. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA., 2002.

[14] J. A. Rochlis and M. W. Eichin, "With microscope and tweezers: the worm from mit's perspective," *Commun. ACM*, vol. 32, no. 6, pp. 689–698, 1989.

[15] P. Vinod, V. Laxmi, and M. Gaur, "Survey on malware detection methods," in *Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09)*, pp. 74–79, Jun 2009.

[16] C. Eagle, *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. San Francisco, CA, USA: No Starch Press, 2008.

[17] R. P. Lippmann and R. K. Cunningham, "Improving intrusion detection performance using keyword selection and neural networks," *Computer Networks*, vol. 34, no. 4, pp. 597 – 603, 2000. Recent Advances in Intrusion Detection Systems.

[18] J. Zheng, M. Hu, B. Fang, and H. Zhang, "Anomaly detection using fast sofm," in *Grid and Cooperative Computing GCC 2004 Workshops* (H. Jin, Y. Pan, N. Xiao, and J. Sun, eds.), vol. 3252 of *Lecture Notes in Computer Science*, pp. 530–537, Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-30207-0_66.

[19] L. Vokorokos, A. BALÁŽ, and M. Chovanec, "Intrusion detection system using self organizing map," *Acta Electrotechnica et Informatica No*, vol. 6, no. 1, p. 1, 2006.

[20] S. Zanero and G. Serazzi, "Unsupervised learning algorithms for intrusion detection," in *IEEE Network Operations and Management Symposium, 2008. NOMS 2008*, p. 10431048, IEEE, April 2008.

[21] H. Sun, J. Lui, and D. Yau, "Defending against low-rate tcp attacks: dynamic detection and protection," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, p. 196205, 2004.

[22] A. Shevtekar, K. Anantharam, and N. Ansari, "Low rate tcp denial-of-service attack detection at edge routers," *IEEE Communications Letters*, vol. 9, no. 4, p. 363365, 2005.

[23] H. Ritter and K. Schulten, "Kohonen's self-organizing maps: exploring their computational capabilities," in *Neural Networks, 1988., IEEE International Conference on*, pp. 109–116, Jul 1988.

[24] S. Luttrell, "Self-organisation: a derivation from first principles of a class of learning algorithms," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, pp. 495 –498 vol.2, Jun 1989.

[25] T. Kohonen, J. Hynninen, and et al., "Som_pak: The self-organizing map program package," tech. rep., Helsinki University of Technology, 1996.

[26] T. Kohonen, "Self-organizing maps: Optimizaion approaches," *Proceedings of the International Conference on Artificial Neural Networks*, pp. 981 – 990, June 1991.

[27] F. Mulier and V. Cherkassky, "Learning rate schedules for self-organizing maps," in *Pattern Recognition, 1994. Vol. 2 - Conference B: Computer Vision Image Processing., Proceedings of the 12th IAPR International. Conference on*, vol. 2, pp. 224 –228 vol.2, Oct 1994.

[28] S. Fortunato, "Community detection in graphs," *arXiv*, vol. 906, p. 103, 2009.

[29] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004.

[30] J. Taylor, *An introduction to error analysis: the study of uncertainties in physical measurements*. Physics - chemistry - engineering, University Science Books, 1997.