Towson University
Office of Graduate Studies

# TOWARDS EFFICIENT THREAT DETECTION IN MOBILE NETWORKS

by
Linqiang Ge

A Dissertation
Presented to the Faculty of the Graduate School
of Towson University
in Partial Fulfillment of the Requirements for the Degree of
DOCTOR OF SCIENCE
Department of Computer and Information Sciences

TOWSON UNIVERSITY
Towson, Maryland, 21252

May 2016

DISSERTATION APPROVAL PAGE

This is to certify that the dissertation prepared by **Linqiang Ge**, entitled "**Towards Efficient Threat Detection in Mobile Networks**", has been approved by this committee as satisfactorily completing the dissertation requirements for the degree of **Doctor of Science** in Information Technology.

| | |
|---|---|
| Dr. Wei Yu | 1/6/2016 |
| Chair, Dissertation Committee | Date |
| Dr. Chao Lu | 1/6/2016 |
| Member, Dissertation Committee | Date |
| Dr. Alex Wijesinha | 1/6/2016 |
| Member, Dissertation Committee | Date |
| Dr. Michael McGuire | 1/6/2016 |
| Member, Dissertation Committee | Date |
| Dr. Chao Lu | 1/6/2016 |
| Chair, Department of COSC | Date |
| Dr. Ramesh Karne | 1-6-2016 |
| Doctoral Program Director | Date |
| Dr. Janet DeLany | 1-7-2016 |
| Dean of Graduate Studies | Date |

*Dedicated*

*To*

*My Parents*

# Table of Contents


**Page**

**Abstract** . . . . . **vii**

**Acknowledgement** . . . . . **ix**

**List of Tables** . . . . . **x**

**List of Figures** . . . . . **xi**

**Chapter 1 Introduction** . . . . . **1**

1.1 Motivation . . . . . 1
1.2 Significance of Proposed Research . . . . . 4
1.3 Organization of Dissertation Research . . . . . 7

**Chapter 2 Background and Related Work** . . . . . **8**

2.1 Mobile Network Security . . . . . 8
2.2 Scalability Issue in Network Security . . . . . 10
2.3 Cloud Computing and MapReduce Techniques . . . . . 13

**Chapter 3 Behavior-based Malware Detection Approach on Mobile Devices 16**

3.1 Overview . . . . . 16
3.2 Artificial Neural Networks . . . . . 18
3.2.1 Feedforward Neural Networks (FNN) . . . . . 18
3.2.2 Recurrent Neural Networks (RNN) . . . . . 20
3.3 Permissions and System Calls . . . . . 20
3.3.1 Overview . . . . . 21
3.3.2 Permissions . . . . . 21
3.3.3 System Calls . . . . . 24
3.4 An ANN-Based Malware Detection System . . . . . 25
3.4.1 Permission-Based Detection . . . . . 25
3.4.2 System Call-Based Detection . . . . . 31
3.5 Performance Evaluation . . . . . 33
3.5.1 Evaluation Methodology . . . . . 33
3.5.2 Evaluation Results . . . . . 35
3.6 Discussion . . . . . 38
3.7 Summary . . . . . 40

**Chapter 4 Effective Sampling and Data Aggregation Techniques in Host-based Intrusion Detection . . . 41**

4.1 Overview . . . 41
4.2 A Host-based Intrusion Detection Architecture in MANET . . . 44
4.3 Our Approaches . . . 48
4.3.1 Overview . . . 48
4.3.2 Sampling Techniques . . . 52
4.3.2.1 Simple Random Sampling . . . 53
4.3.2.2 Stratified Sampling . . . 53
4.3.3 Data Aggregation Techniques . . . 56
4.3.4 Detection . . . 57
4.4 Analysis . . . 61
4.4.1 Anomaly-based Detection . . . 62
4.4.2 Simple Random Sampling . . . 64
4.4.3 Stratified Sampling . . . 67
4.4.4 Data Aggregation Techniques . . . 71
4.4.5 Impact of Intrusion Detection on MANET . . . 72
4.5 Performance Evaluation . . . 74
4.5.1 Methodology . . . 74
4.5.2 Evaluation Results . . . 80
4.6 Summary . . . 82

**Chapter 5 MapReduce Based Machine Learning Techniques for Processing Massive Network Threat Monitoring Data . . . 84**

5.1 Overview . . . 84
5.2 Our Approach . . . 86
5.2.1 Design Rationale . . . 87
5.2.2 Algorithms Design . . . 89
5.2.3 Implementation . . . 97
5.3 Performance Evaluation . . . 101
5.3.1 Evaluation Methodology . . . 101
5.3.2 Evaluation Results . . . 104
5.4 Summary . . . 106

**Chapter 6 Final Remarks . . . 108**

**References . . . 109**

**Curriculum Vita . . . 120**

# Abstract

**Towards Efficient Threat Detection in Mobile Networks**

**Linqiang Ge**

With the popularity of mobile networks, it has become a burgeoning target for cyber-attacks. For example, malware has proven to be a serious problem for the mobile platform because malicious applications can be distributed to mobile devices through an application market. From the defender's perspective, how to effectively detect threats and enhance the cognitive performance of mobile networks becomes a challenging issue. In addition, mobile networks have limited network resources and mobile devices are characterized by limited storage capacity, constraint battery life time, and limited computation resources so that developing a scalable, reliable and robust cyber threat defense system is challenging.

To address those challenges, in this dissertation we develop effective schemes to efficiently conduct threat detection in mobile networks. First, we develop an Artificial Neural Network (ANN)-based malware detection scheme to detect unknown malware on mobile devices. Second, to enable the effective detection and desirable impact on the performance of mobile networks, we develop both sampling and aggregation techniques to achieve desirable tradeoffs between the detection accuracy

and the use for network resources. Third, we develop MapReduce-based Machine Learning (MML) schemes with the goal of rapidly and accurately detecting and processing of malicious traffic in a cloud environment.

# Acknowledgement

I would like to gratefully and sincerely express my sincere gratitude to Dr. Wei Yu for his guidance, understanding and patience. His mentorship was paramount in providing a well rounded experience consistent my long-term career goals. He encouraged me to not only grow as a researcher but also as an instructor and an independent thinker.

I would also like to thank Dr. Chao Lu for his help and guidance in getting my graduate career started on the right foot and providing me with the foundation for becoming a synthetic doctoral student.

I would also like to acknowledge Dr. Alexander Wijesinha and Dr. Michael McGuire to provide guidance on my dissertation. In addition, I would like to thank my colleagues and our research group members who collaborate with me and give me many academic helps.

Towson, Maryland Linqiang Ge

January, 2016

# List of Tables

**Page**

Table 4.1 Notations . . . . . 64

Table 5.1 Description of Extracted Features. . . . . 99

Table 5.2 Detection Accuracy of the Naïve Bayes Machine Learning Scheme.104

Table 5.3 Detection Results of Logistic Regression Scheme. . . . . 105

# List of Figures

**Page**

Figure 1.1 Technical Approaches . . . . . 4

Figure 2.1 The MapReduce Workflow . . . . . 14

Figure 3.1 A Typical Structure of FNN . . . . . 19

Figure 3.2 An Example of Mapped Permissions . . . . . 22

Figure 3.3 Distribution of Permissions . . . . . 23

Figure 3.4 Workflow . . . . . 26

Figure 3.5 Dumping Permissions . . . . . 27

Figure 3.6 An Example of Permissions . . . . . 29

Figure 3.7 Detection Rate for Permission Based Detection vs. Training Set Ratio (FNN with 10 Nodes) . . . . . 34

Figure 3.8 Detection Rate for Permission Based Detection vs. Training Set Ratio (FNN with 20 Nodes) . . . . . 34

Figure 3.9 Detection Rate for Permission Based Detection vs. Training Set Ratio (RNN with 10 Nodes) . . . . . 36

Figure 3.10 Detection Rate for System Call Based Detection vs. Training Set Ratio (FNN with 10 Nodes) . . . . . 36

Figure 3.11 Detection Rate for System Call Based Detection vs. Training Set Ratio (FNN with 20 Nodes) . . . . . 37
Figure 3.12 Detection Rate for System Call Based Detection vs. Training Set Ratio (RNN with 10 Nodes) . . . . . 37
Figure 3.13 Error Rate for Permission Based Detection vs. Training Set Ratio (1-gram) . . . . . 38
Figure 3.14 Error Rate for System Call Based Detection vs. Training Set Ratio (1-gram) . . . . . 38

Figure 4.1 System Architecture . . . . . 44
Figure 4.2 System Workflow . . . . . 48
Figure 4.3 An Example of System Logs . . . . . 56
Figure 4.4 An Example of Original System-Logs . . . . . 57
Figure 4.5 An Example of Aggregated System Logs . . . . . 58
Figure 4.6 Detection Rate vs. Sampling Rate (Experiment) . . . . . 76
Figure 4.7 Error Rate vs. Sampling Rate (Experiment) . . . . . 77
Figure 4.8 Error Rate vs. Sampling Rate (Theory) . . . . . 78
Figure 4.9 End-to-End Delay of Normal Mission Application without Intrusion Detection Application . . . . . 78
Figure 4.10 Sampling Ratio vs. End-to-End Delay of Normal Mission Application 79

Figure 5.1 Machine Learning Based on MapReduce Framework. . . . . 92

Figure 5.2 MapReduce Based Framework for Parallel Machine Learning. 93
Figure 5.3 Cost of Gradient Descent using MapReduce. . . . . 94
Figure 5.4 Detection Work flow . . . . . 97
Figure 5.5 Flow Extraction. . . . . 98
Figure 5.6 The Information of a Data Packet. . . . . 99
Figure 5.7 An Example of Data Selection Output. . . . . 99
Figure 5.8 Screenshot of Feature Values Computing for Logistic Regression. 100
Figure 5.9 Screenshot of Feature Values Computing for Bayes Machine Learning. . . . . 100
Figure 5.10 A Cloud Computing Testbed. . . . . 102
Figure 5.11 Time Cost versus Number of Nodes for Naïve Bayes . . . . . 106
Figure 5.12 Time Cost versus Number of Nodes for Logistic Regression . . 107

# Chapter 1

# Introduction

## 1.1 Motivation

With the development of modern mobile operating systems and computing and communication technologies, smart mobile devices have been widely used to support applications (voice, video, game, music, GPS navigation, etc.). To improve the productivity for business operation, organizations are having the new wave of transition to mobile enterprise, enabling employees using smart mobile devices for performing mobile working and improving the productivity of the business operation. The modern mobile computing techniques will dramatically improve the accessibility and operational efficiency of enterprise business. Particularly, smart mobile devices have been used to support numerous applications and integrated to enterprise information infrastructure for organizations. As such, the mobile-enabled enterprises focus on improving the flexibility and productivity by enabling their employees and customers to access business applications easily.

With the popularity of smart mobile devices, it has become a burgeoning target for cyber-attacks as well. On one hand, malware, as a malicious application that can be installed on mobile devices, can gain access to these devices and collect user sensitive information. Malware has proven to be a serious problem for the mobile

platform as malicious applications can be distributed to mobile devices through an application market. On the other hand, some types of wireless networks (e.g., Mobile Ad Hoc Networks) demand a robust, diverse, and resilient communication and computation infrastructure, which enables the network-centric operation with a very low rate of downtime. Nonetheless, the nature of these networks also leads to cyber security risks as mobile nodes are deployed in the open field that could be hostile and the wireless communication makes the information accessible by the adversary, who may actively intercept, disrupt, or manipulate the information. The adversary may hack into hosts and network devices inside the network using sophisticated attack techniques to prey on the vulnerabilities of system components and disrupt the effectiveness of mobile devices and networks.

From the defender's perspective, effectively detecting malware and enhancing the cognitive performance of users and system administrators are critical. There are several challenging issues in monitoring and detecting cyber attacks in wireless networks. First, unlike wired networks, resources in wireless networks (i.e., communication bandwidth and host storage and computation capability) are much limited. To enable the monitoring and detection of cyber attacks, we shall transmit a large amount of suspicious information over the wireless network in real time, which has limited network bandwidth resources connected hosts to the operation center. Nonetheless, transmitting a large amount of data associated with threat monitoring

and detection activities over wireless networks will pose a negative impact on other mission related applications supported by networks themselves. Therefore, the monitoring and detection of cyber attacks should be designed such that their impact on the normal operation of the network should be controlled and limited. Second, although a number of HIDS (Host-based Intrusion Detection Systems) [1–3] and agent-based intrusion detection framework [4] have been developed, those systems mainly aim at securing enterprise networks, which have defined structures. Because wireless networks are mostly ad hoc in nature and have limited bandwidth and computing resources, the existing HIDS cannot be directly used for MANET.

Developing a scalable cyber threat defense system is also a challenging task. Cyber-threats are significantly more dangerous than what they have ever been and are growing in number and sophistication. Due to the widespread nature of cyber-threats (malware propagation, etc.), the traffic monitoring across a large-scale network has become an essential part of effectively detecting and defending against contemporary cyber-attacks. Nonetheless, threat monitoring over a large-scale network also leads to massive data collected from monitored hosts and network equipment. Collected massive data poses serious challenges for cyber operations because an ever growing large and complex threat monitoring system collected from a large computer network needs to capture, store, manage, and process big data. As such, there is an urgent need to develop techniques to efficiently process the threat

monitoring data into manageable, useful, and exploitable information.

## 1.2 Significance of Proposed Research

In this dissertation, we consider a threat monitoring and detection framework for enterprise networks, which consists of a number of mobile devices. The system that we consider consists of the three main components: mobile devices, cloud infrastructure, and an operation center listed below.

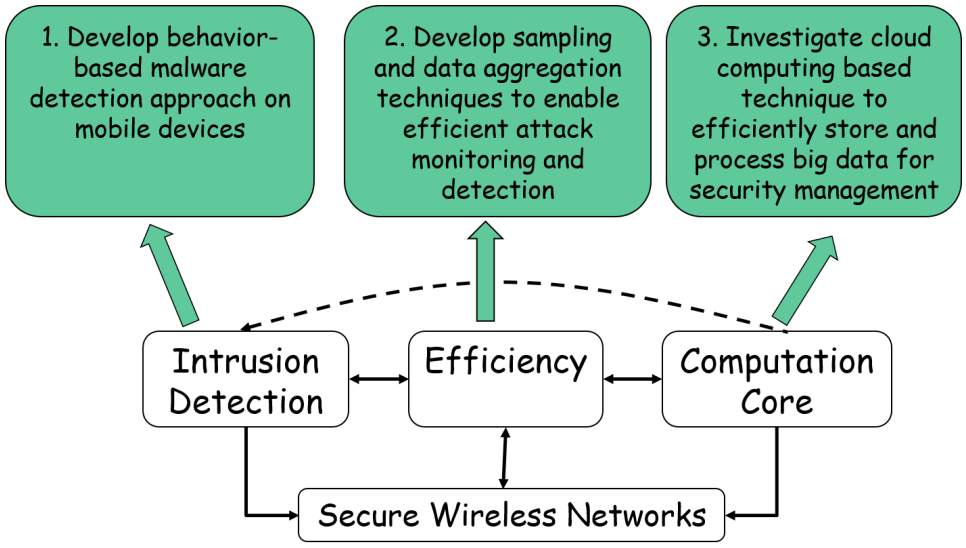


Figure 1.1: Technical Approaches

Due to the exponential increase in the use of smart mobile devices, malware threats on those devices have been growing and posing security risks. As shown in Figure 1.1, based on the framework, we have made several contributions in this dissertation.

- First, we develop an Artificial Neural Network (ANN)-based malware detection scheme to detect unknown malware. In our scheme, we consider both permissions requested by applications and system calls associated with the execution of applications to distinguish between benign applications and mal-ware. We used ANN, a representative machine learning technique, to understand the anomaly behavior of malware by learning the characteristic permissions and system calls used by applications. After that, the trained ANN is used to detect new malware.

- Second, we develop sampling and data aggregation techniques in MANET, which is a typical wireless network, to enable effective attack monitoring and detection. To be specific, we develop both simple random sampling and the stratified sampling techniques to achieve desirable tradeoffs between the detection accuracy and the consumption for network resources. The simple random sampling technique uniformly samples the detection information and the stratified sampling technique stratifies the detection information and samples them with different priorities. We derive closed formulae to analyze the impact of sampling techniques and key parameters on detection accuracy. We also develop two types of data aggregation techniques: lossless and lossy aggregation, in order to reduce the use of resources (e.g., energy consumption and bandwidth) for transmitting threat detection information through MANET,

while preserving the desired detection accuracy for cyber security operation. We conduct both real-world experiments and simulation studies to evaluate the effectiveness of our proposed sampling and data aggregation techniques in terms of energy consumption and detection accuracy.

- Third, we develop an efficient threat monitoring scheme to improve the scalability of defense system. Our developed scheme can process the real-time data streams generated by threat monitoring agents that collect the statuses of hosts or networks and then detect suspicious activities. To ensure that the threat detection methods are efficient, MapReduce-based machine learning (MML) schemes are proposed to efficiently deal with threat monitoring data. The main idea of the MML system is to speed up the machine learning (ML) process using parallel computing techniques. To accurately and rapidly detect traffic anomalies, two MapRduce-based ML schemes are developed to profile the dynamic characteristics of traffic flows and then to detect anomalies based on learned classifiers: Logistic Regression and Naïve Bayes. The experimental results demonstrate that our proposed MML proposed monitoring schemes can accurately and efficiently detect attack traffic flows over massive data collected from networks.

## 1.3 Organization of Dissertation Research

This dissertation is structured as follows. In chapter 2, we introduce the background and related work of mobile networks and security, scalability issue, and cloud computing. In Chapter 3, we present the behavior-based malware detection scheme on mobile devices. In chapter 4, we discuss effective sampling and data aggregation techniques in MANET. We study the MapReduce-based machine learning techniques for efficiently processing massive network threat monitoring data in Chapter 5. Finally, we conclude the dissertation in chapter 6.

# Chapter 2

# Background and Related Work

In this chapter, we introduce the background of our research, including security in mobile networks, scalability issue in network security, and cloud computing and MapReduce techniques.

## 2.1 Mobile Network Security

The security issues have become a primary concern in mobile networks. MANET is a typical mobile network. Generally speaking, MANET is a self-organizing network without fixed infrastructure. Because of its dynamically changing topology, the vulnerability of wireless communication links, the limited physical protection of nodes, MANET is vulnerable to security threats [5–8]. There are a number of research efforts on studying the security issues in MANET [9, 10] and a number of research efforts to study the host-based IDS [1, 2, 11]. Nonetheless, these systems are mainly aimed at securing wired networks with a defined structure and cannot directly be used for the MANET, which is dynamic and has limited host and network resources. The most of the existing IDS systems rely on a central correlation engine to analyze the data produced from the individual IDS. The large amount of data produced by the system can overwhelm the limited resources in MANET. In addition,

a number of the existing IDS systems are signature-based and cannot effectively conduct cyber attack detection in MANET.

The detection of malware on a mobile platform can be categorized into static analysis, dynamic analysis, and permission analysis. These techniques have been investigated by [12–16]. For example, Bose *et al.* [12] proposed a malware behavioral-based detection scheme on mobile handsets. Shamili *et al.* [13] presented a distributed Support Vector Machine (SVM) scheme to conduct malware detection, along with a statistical classification model. Deepak *et al.* [14] proposed a signature-based malware detection scheme. Schmidt *et al.* [16] conducted the static analysis of malware on the Android platform. To measure the effectiveness of different schemes on malware detection, Shabtai *et al.* [15] evaluated several classification and anomaly detection schemes and feature selection methods for mitigating malware on mobile devices.

Through permission analysis, malware detection can be conducted through the analysis of extracted security configurations and policy rules [17–20]. For example, Aung *et al.* [18] developed a machine learning-based detection on the Android platform by monitoring permission related features and events. Huang *et al.* [19] conducted the permission-based detection for Android malware by using machine learning schemes such as AdaBoost, Naive Bayes, Decision Tree (C4.5), and Support Vector Machine. David *et al.* [20] presented a Self-Organizing Map (SOM) scheme

to identify the permission-based security model using 1,100 android applications.

Neural networks can be used to learn and classify anomaly activities based on limited data sources [21]. There have been a number of research efforts on using neural networks to carry out threat detection [21–24]. For example, Mukkamala *et al.* [22] investigated schemes to conduct intrusion detection using neural networks and SVMs. Linda *et al.* [23] proposed a neural network-based approach to conduct intrusion detection for critical infrastructures. Golovko *et al.* [24] discussed the use of neural networks and artificial immune systems for carrying out malware and intrusion detection.

## 2.2 Scalability Issue in Network Security

To defend against cyber-attacks, the development of effective cyber threat monitoring system is critical, and it should be able to characterize, track, and mitigate security threats in networks in a timely manner. Developing a scalable cyber threat defense system is also a challenging task. To detect those attacks, a large amount of threat monitoring and detection tools, including the Advanced Intrusion Detection Environment (AIDE) (http://aide.sourceforge.net/), OSSEC (http:// www.ossec.net/), and others, have been developed with the intention to monitor behavioral changes on hosts and network devices. By leveraging these tools, massive data (e.g., system logs, security logs, application logs, and traffic logs) generated by hosts (e.g., computers, mobile

devices, and others) and network devices (e.g., routers, firewalls, and others) can be collected for carrying out cyber-threat situational awareness. In addition to passively logging the activities associated with attacks, Honeypots can directly and actively interact with attacks and collect more insightful data from these attacks. For a large network, the collected threat monitoring related data will be massive and is featured by a high volume of data size, a high velocity of data transmission, and a high variety of data types [25]. For example, about a gigabyte of data per day needs to be gathered for further analysis [25]. In addition, data collected from different monitoring systems have different data formats, which can be structured or unstructured. Therefore, effectively processing and analyzing massive threat monitoring related data is a challenging issue.

To solve the scalability problem in network security, many efficient techniques have been developed and integrated for security purpose. Particularly, sampling techniques have been mainly used for network traffic measurement and accounting to obtain the vast quantities of traffic data continuously collected for network monitoring and management [26–31]. For example, Mai *et al.* [29] studied the impact of flow sampling techniques, including the random packet sampling, random flow sampling, intelligent sampling, and sample-and-hold on volume-based anomaly detection and port-scan detection. Ficara *et al.* [30] proposed sampling techniques to accelerate pattern matching in those network intrusion detection. Their solution

consists of two matching stages with the Deterministic Finite Automata (DFAs). One match is conducted on the traffic by a "sampled" DFA, and if necessary, a more accurate processing is conducted through another DFA (reverse DFA) to confirm the match.

Data aggregation, as another technique to improve the efficiency, has been extensively studied in sensor networks [32–36]. For example, using a tree structure, Dina *et al.* [33] proposed to maintain additional region leader information at sensor nodes, enabling the determination of aggregated records based on a tree routing model. LEACH [36] proposed a hierarchical protocol based on a cluster structure, in which the network is divided into a number of clusters and some nodes are randomly selected as the cluster header. Based on the recorded signal strength to cluster headers, each node selects a cluster to join. Shrivastava *et al.* [34] proposed a synopsis diffusion approach based on a ring topology for data aggregation. During the subsequent query aggregation period, nodes are divided into a set of rings based on the distance to the ring center. The aggregation starts from the outermost ring to the center. In addition, the data aggregation can be operated in either a dense or sparse network. Gao *et al.* [35] proposed a sparse data aggregation technique. By forming a tree structure, the hot node receives data from other nodes and performs data aggregation.

## 2.3 Cloud Computing and MapReduce Techniques

The acceleration of data generation requires new technologies to analyze massive data. With a large data storage space, high computational capacity, and low infrastructure investment, cloud computing can offer a platform for massive data analysis. Cloud computing [37–51] is a technology that uses the Internet and central remote servers to provide computation, software, data access, and storage services that do not require knowledge of users' physical location and the configuration of servers. Most current clouds are built on the top of a modern data center [52]. Cloud incorporates different service models [53] such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) and so on. Cloud computing is gaining popularity in both academia and industry and both have been active in the research on cloud computing architectures. For example, Vecchiola *et al.* [54] introduced a .NET-based Cloud Computing platform, which provides a set of APIs that allow developers to build .NET applications that leverage their computation using the cloud. Huang *et al.* [55] developed a low-cost, scalable, and secured platform that enables web-delivery of application-based services with a set of common business and operational services.

By leveraging cloud computing techniquee, a large number of distributed servers can be used for data access, computation, and storage. MapReduce is a parallel programming model primarily designed for batch processing big data in a distributed

computing environment [56]. MapReduce is designed using the concept of divide-and-conquer and follows the master/slave computing paradigm, consisting of the map function and the reduce function [57]. The purpose of the map function is to split and distribute data sets to different servers for processing whereas the purpose of the reduce function is to collect the results for data sets and generate the final result.

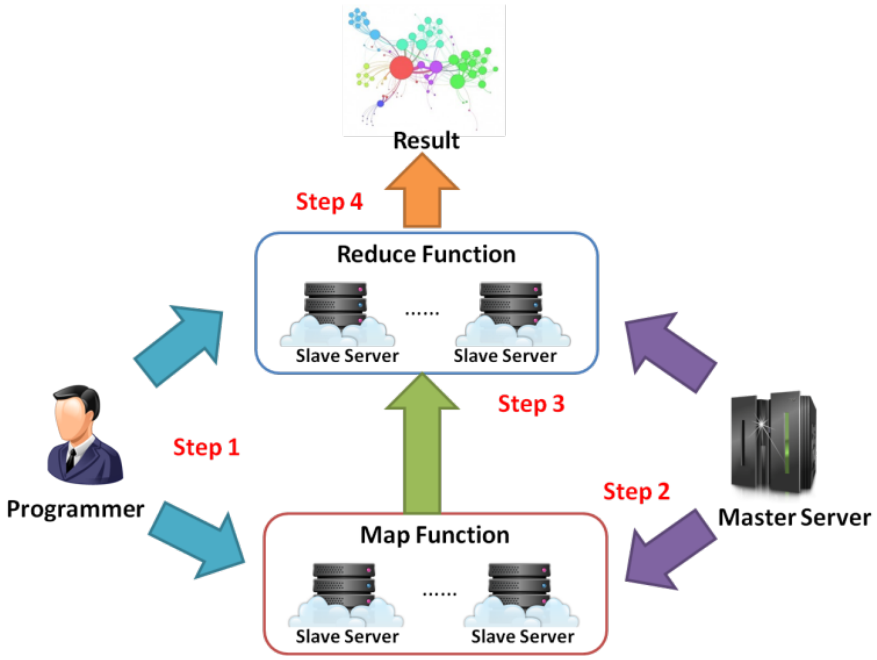


Figure 2.1: The MapReduce Workflow

As shown in Figure 2.1, the workflow of MapReduce is detailed as follows: *Step 1. Functions definition:* In this step, programmers need to define both the map and the reduce functions, which will be used by MapReduce to implement data analysis; *Step 2. Data split and distribution:* For a large data set, the master server will the data them into several relatively small subsets and distribute the subsets to

map slave servers for computation; *Step 3. Data computation:* With the defined map function, the map slave servers can concurrently process small subsets and generate intermediate results; and *Step 4. Data resolving:* After collecting intermediate results, based on the defined reduce function, the reduce slave servers will resolve and aggregate the intermediate results to produce the final result. It is worth noting that MapReduce has a built-in fault-tolerant feature, through which data can be duplicated and assigned to different servers for processing. The working status of slave nodes shall be periodically reported to the master node. If a slave node does not reply to the request in a given time, it will be considered as a failure node. Then, the task that the failure node was initially assigned to process will be reassigned to other nodes.

# Chapter 3

# Behavior-based Malware Detection Approach on Mobile Devices

## 3.1 Overview

The rapid growth of smart mobile devices has led to a renaissance for mobile services. These devices can augment cognitive abilities with multi-function applications related to web, education, travel, game, financial, and many others. For example, face recognition applications can help identify or verify a person to enhance human cognitive abilities. The Android platform is an open source operating system for smart mobiles and provides services, including security configuration, process management, and others [58]. With 48% of smartphone subscribers using Android mobiles, Android leads the smartphone market in the U.S. [59].

Nonetheless, the popularity of Android mobile devices has led to enormous security challenges. Malware, as a malicious application that can be installed on mobile devices, can gain access to these devices and collect user sensitive information. Malware has proven to be a serious problem for the Android platform because malicious applications can be distributed to mobile devices through an application market. From the defender's perspective, how to effectively detect malware and enhance the cognitive performance of users and system administrators becomes a

challenging issue. Traditional static analysis techniques heavily rely on capturing malicious characteristics and bad code segments embedded in software. This makes it infeasible to deal with a large population of unknown malware. Therefore, it is critical to develop a machine learning based system that can dynamically learn the behavior of malware and augment the human cognition process of defending against malware attacks in the battle of mobile security.

In this chapter, we propose an Artificial Neural Network (ANN) based malware detection system that uses both permissions and system calls to detect unknown malware. In our system, we consider two types of ANNs: Feedforward Neural Networks (FNN) to learn the patterns of permissions and Recurrent Neural Networks (RNN) to understand the structure of system calls. Permission requests are collected from applications to distinguish between benign applications and malware. We also collected system calls associated with application execution to capture the runtime behaviors of benign applications and malware. Through the training process, the ANN can learn the anomaly behaviors of malware in terms of permission requests and system calls. The resulting model can be further used to detect unknown malware. To evaluate the effectiveness of our malware detection system, we used real-world malware and benign applications to conduct experiments on Android mobile devices. The resulting data shows that our system can effectively detect malware.

Note that the materials in this chapter are adapted from my previous publication [60, 61]

## 3.2 Artificial Neural Networks

We consider ANN to conduct malware detection. Generally speaking, a neural network refers to a network or circuit that mimics the structure and behavior of biological neurons [62]. The parameters of a neural network are set through a training process that uses known data sets as inputs. After that, the trained neural network can be used as a classifier to conduct detection.

### 3.2.1 Feedforward Neural Networks (FNN)

FNNs are a well-known and widely used type of neural network [63–67]. An FNN consists of a certain number of layers and a number of units called artificial neurons or nodes that are organized in layers. In a typical setting, an FNN has an input layer, an output layer, and one or more hidden layers between the input and the output layer. In an FNN, all data and computation flows are in one direction: from input to output data. Except for input units, each unit in a layer is connected to all the units in the previous layer and receives inputs directly from the nodes in the previous layer. Each connection may have a different strength or weight. During the training process, the weight can be adjusted through learning algorithms such as

BackPropagation (BP). The typical structure of an FNN is illustrated in Figure 3.1.

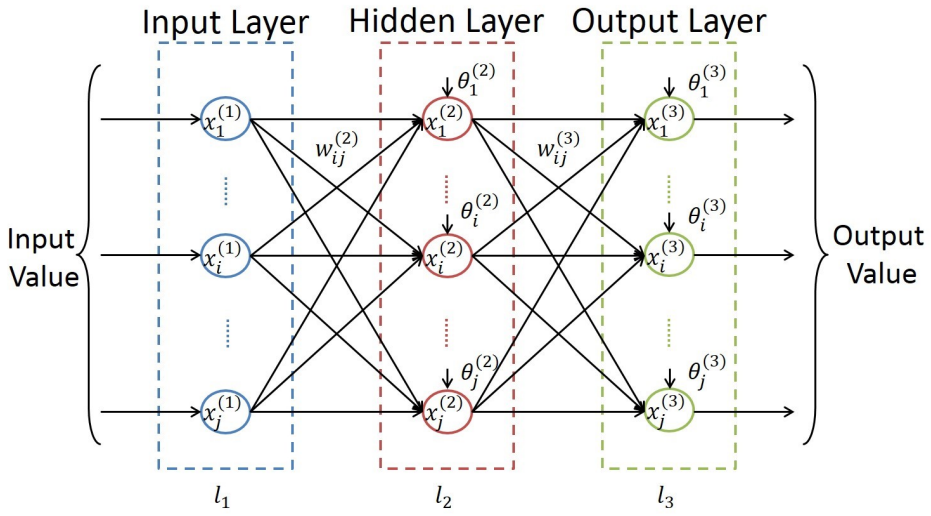


Figure 3.1: A Typical Structure of FNN

Here, l represents the layer of the FNN, where l=1 is for the input layer, l=2 is for the hidden layer, and l=3 is for the output layer. In principle, the output values are compared with the correct answer to compute the value of a predefined error-function that is then sent back through the network. With the backward propagation errors between real and estimated values from the output layer to the hidden layer and from the hidden layer to the input layer, errors in each layer can be estimated and the assigned weights $\omega_i j^{(}(l))$ can be updated correspondingly. After repeating this procedure many times, the neural network eventually reaches a state where the computed error is small. At this moment, the training process is complete.

### 3.2.2 Recurrent Neural Networks (RNN)

Unlike the FNN, the fundamental feature of an RNN is that the network contains at least one feedback connection. This makes an RNN useful for handling temporal classification problems or learning sequences. Similar to an FNN, an RNN consists of a number of units and multiple layers: input layer, output layer, and one or more hidden layers. When the data is fed to an RNN, a state activation is generated in the hidden layers. In the next time slot, the previous state activation is fed back to the hidden layer, combining with new input data. During the training process, the weight of unit connections and feedback connections can be adjusted through learning algorithms such as Back Propagation Through Time (BPTT). The BP algorithm used in an FNN cannot be directly applied to an RNN because of the inherent cycles present. Hence, BPTT unfolds the network over time, eliminating cycles and allowing the neural network to be trained as if it consists of several connected FNNs where the BP algorithm can be used.

## 3.3 Permissions and System Calls

In this section, we first review the typical malware detection techniques. Then we examine in detail how permissions and system calls can be used as the fundamental detection data source.

### 3.3.1 Overview

There are several types of detection techniques. Static analysis [13] has been used to carry out malware detection through the process of decompiling executable software, generating source code, and then using code analysis tools to inspect the recovered source code. Static analysis is limited by the capability of code analyzers and can only deal with applications that involve a small number of permissions and system calls.

Permission and dynamic analysis schemes are promising techniques to defend against a large class of unknown malware. To be specific, permission-based detection sets security policy rules. When an application is installed, the permission-based detection extracts security configurations and checks them against security policy rules. Conversely, dynamic analysis-based detection [68] executes the mobile application and monitors the applications dynamic behavior. Based on the runtime behavior, the malware can be detected. As malicious behavior is always difficult to hide and can be used as a feature to identify malware, we can use ANN techniques to accurately characterize the behavior of applications.

### 3.3.2 Permissions

Android provides third-party applications that have the capability of accessing re-sources such as phone hardware, settings, user data, and others through permissions. For

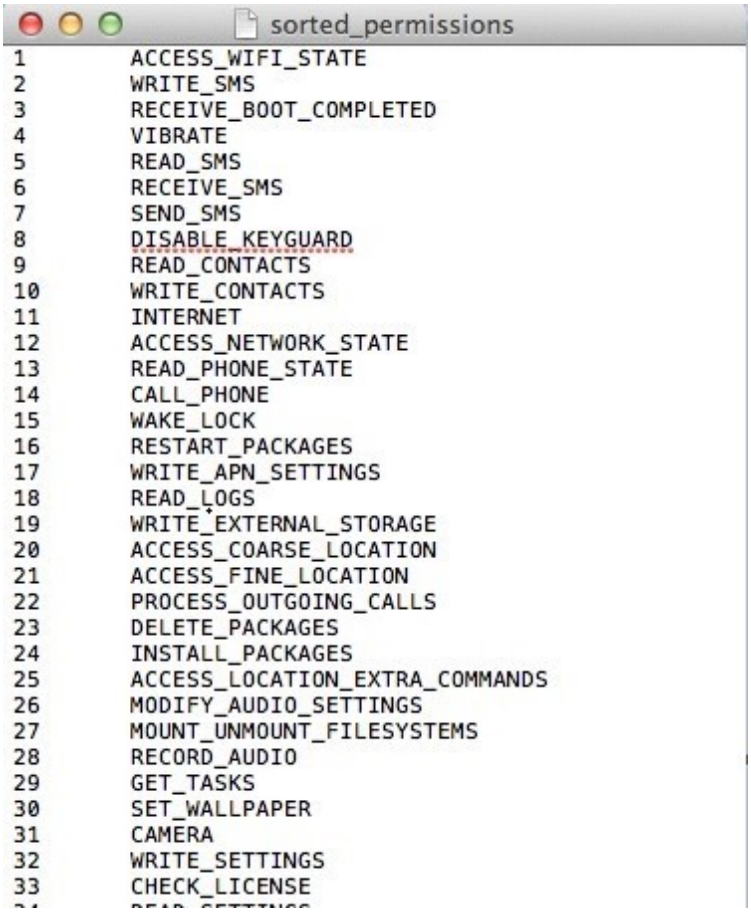


Figure 3.2: An Example of Mapped Permissions

example, the INTERNET permission allows applications to open network connections. Each application must declare in advance what permissions it requires, and users are notified during the installation about the permissions that it will obtain. Users can cancel the installation process if they do not want to grant a permission to the application, but they might not have the knowledge to determine which permissions should be requested by and granted by a particular application. Usually, different types of applications request reasonable permissions. Nonetheless, even an application

requesting a reasonable permission might conduct malicious behavior. For example, a social network application that requests to only access the contact may additionally copy contacts personal information to a remote server.

To show the potential of using permissions to detect malware, we investigated the distribution of permissions requested by electronic books. We installed 96 benign applications from Google Play and used 92 digital book malware samples from the Android Malware Genome Project [69]. For each Android application, we extracted permissions from the corresponding application package (APK) file. The details of the retrieving process will be presented in Section 3.4.1. We define each captured permission as one feature and map it to an integer. Figure 3.2 shows an example of mapped permissions.

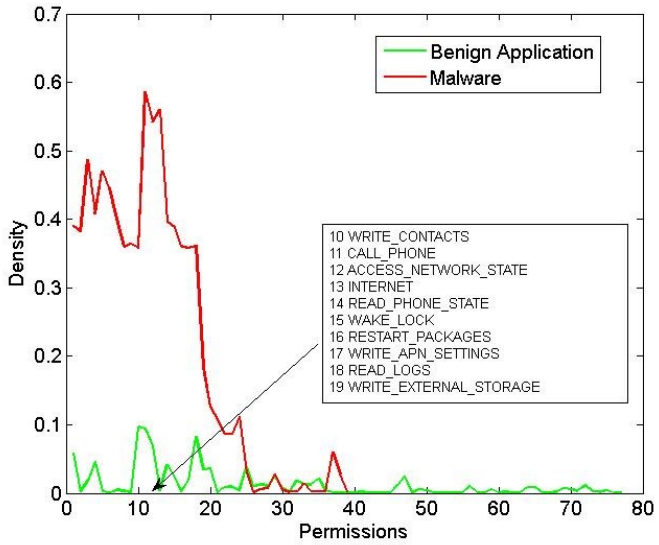


Figure 3.3: Distribution of Permissions

After retrieving the permissions from all applications, the distribution of permissions can be computed. One such example is shown in Figure 3.3. As we can see, most malware samples heavily request permissions 1-20, which are WRITE SMS, SEND SMS, READ CONTACT, etc. We can conclude that electronic book applications that request permissions 1-20 are probably malware. Hence, the permissions requested by an application can be used to recognize whether the application contain malware.

### 3.3.3 System Calls

A system call is the mechanism used by applications to request a service from the operating system kernel. System calls provide the interface between the process and operating systems. The operating system provides services, including the creation and execution of new processes and access control of resources. The sequence of system calls occurs consecutively over time and can capture actions performed by applications during execution. In Android, there are 56 system calls in the library and the main types of system calls consist of: Process Control for controlling processes, File Management for managing files, Device Management for managing devices, Information Maintenance for setting system data and obtaining process information, and Communications for establishing connections. As system calls provide an essential interface between the application and operating system, we shall examine system calls to capture the runtime behavior of the interactions

between applications and the operating system.

## 3.4 An ANN-Based Malware Detection System

We now present the workflow of our proposed ANN-based malware detection system as shown in Figure 3.4. We would like to emphasize that the workflow is general and can be used for both permission-based detection and system call-based detection. In the offline training phase, we first collected real-world benign and malicious applications. Next, we executed the collected applications and dumped the data sources. In order for machine learning algorithms to learn the feature pat-terns of malware and benign applications, all data sources needed to be parsed and mapped to the format required by the FNN and RNN algorithms described in Section 2. Using the mapped data as input, we then trained the neural network. In the online detection phase, we dumped the data sources from new applications and the trained neural network would be used to determine whether the new application is malware or benign. As permissions and system calls contain different features and have different formats, we first introduce permission-based detection and then system call-based detection in the following subsections.

### 3.4.1 Permission-Based Detection

***Offline Training*** We now discuss the steps used for the offline training process.

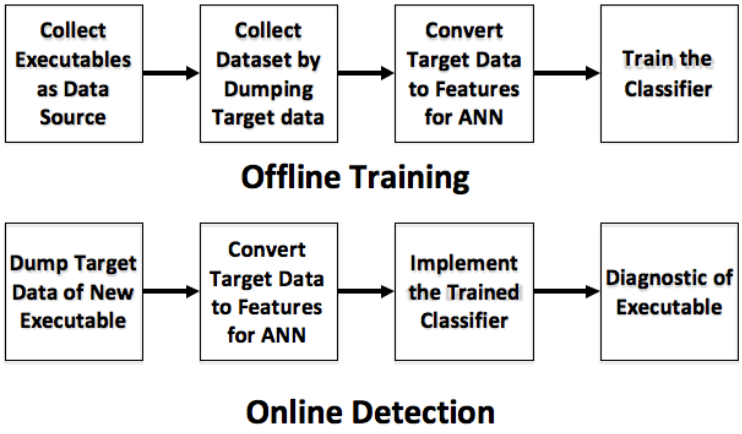


Figure 3.4: Workflow

**Step 1:** Data source collection and classification. The first step in the offline training phase is to collect the data source from the executing applications. With real-world benign applications and malware samples, we consider that applications in the same category should exhibit similar activities and we use such activities to learn the anomaly profile. Based on these learned profiles, we can categorize applications as benign or malicious.

**Step 2:** Dumping Permissions of Data source. Using the benign application and malware samples, we dump the permissions requested by each application. In the Android system, all permissions are included in the Android-Manifest.xml file. After collecting application apk files, we use a known reverse engineering tool Android Asset Packaging Tool (aapt) to reconstruct the source code and obtain the AndroidManifest.xml file for each application. An example is shown below:

<manifest xmlns: android="http://schemas.android.com/apk/res/android"

package="com.android.app.QQ_for_Pad_v_1.9.3" > A:

android:versionCode(0x0101021b)=(type 0x10)0x7 A:

android:versionName(0x0101021c)="2.1-update1" A:

package="com.android.spare_parts" <uses-permission

an-droid:name="android.permission.READ_PHONE_STATE"/> <us-es-permission

an-droid:name="android.permission.CAMERA"/> ... </manifest>

```
Linqiangs-MacBook-Pro:tools linqiangge$
Linqiangs-MacBook-Pro:tools linqiangge$ ./aapt dump permissions QQ_for_Pad_v_1.9.3.a
pk
package: com.tencent.android.pad
uses-permission: android.permission.READ_PHONE_STATE
uses-permission: android.permission.CAMERA
uses-permission: android.permission.WRITE_EXTERNAL_STORAGE
uses-permission: android.permission.MOUNT_UNMOUNT_FILESYSTEMS
uses-permission: android.permission.VIBRATE
uses-permission: android.permission.INTERNET
uses-permission: android.permission.WAKE_LOCK
uses-permission: android.permission.ACCESS_NETWORK_STATE
uses-permission: android.permission.SYSTEM_ALERT_WINDOW
uses-permission: com.android.launcher.permission.INSTALL_SHORTCUT
uses-permission: android.permission.MODIFY_AUDIO_SETTINGS
uses-permission: android.permission.RECORD_AUDIO
uses-permission: com.tencent.android.pad.permission.IM_SERVICE
uses-permission: com.tencent.android.pad.permission.EXT_IM_SERVICE
permission: com.tencent.android.pad.permission.IM_SERVICE
permission: com.tencent.android.pad.permission.EXT_IM_SERVICE
permission: com.tencent.android.pad.permission.WRITE_SETTINGS
permission: com.tencent.android.pad.permission.READ_SETTINGS
Linqiangs-MacBook-Pro:tools linqiangge$
```

Figure 3.5: Dumping Permissions

We then use the command aapt dump permission to collect all permissions requested by each application. Figure 3.5 shows an example of the dumping process and the corresponding result.

**Step 3:** Feature extraction. Next, we collect a set of files where each file consists of permissions requested by one application. For training, we process the data and map them to the format required by the ANN. To this end, we developed

a mapping algorithm to convert the original permissions into usable input. As described previously, we use Algorithm 1 to define each permission as one feature and assign an integer to each feature.

Using the example shown in Figure 3.5, we now explain Algorithm 1. In this algorithm, we care about the feature (i.e., permission name) and the feature value, defined as whether it was requested by the application. Note that one permission can be requested only once by an application. If a particular permission is requested, its feature value is 1; otherwise its feature value is 0. After the first for loop of Algorithm 1, we obtain the output shown in Figure 3.6.

**Algorithm 1:** Permission Mapping Algorithm

```
   input     : Original Android system calls stored in raw data folder
   output    : Permission Feature Vector A
 1 n = gram number;
 2 foreach file in raw data folder do
 3 |   foreach line in file do
 4 |   |   remove all information except the system call name;
 5 |   end
 6 |   store file in system call name data folder;
 7 end
 8 foreach file in system call name data folder do
 9 |   foreach line in file do
10 |   |   map system call name to integers as feature names;
11 |   end
12 |   store file in mapped-integer data folder;
13 end
14 if n > 1 then
15 |   map to n-gram format;
16 end
17 foreach file in feature-value pair data folder do
18 |   add target to files;
19 |   combine files together;
20 end
```

Because the ANN only accepts integers as input, we map each permission name to an integer number after processing the name sequence of permissions. After the second for loop in Algorithm 1, the mapping produces output similar to “01,02,03,06,09,15,20”. As examples, INTERNET is mapped to 11, READ PHONE STATE is mapped to 13, and SEND SMS is mapped to 7. We can extend this idea to use 2-grams as a detection feature by applying two contiguous permissions instead of one. As an example, we combine every two contiguous integers

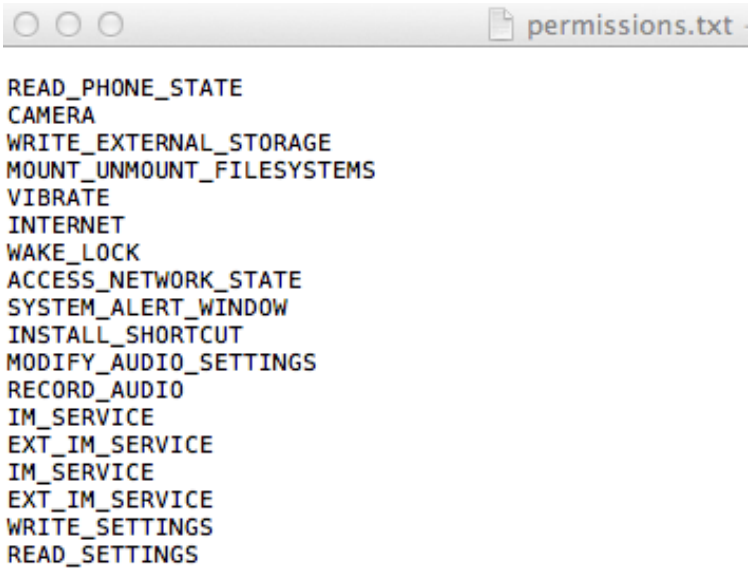


Figure 3.6: An Example of Permissions

and the mapping produces output similar to “0102,0203,0304,0405” where “0102” represents the permissions ACCESS WIFI STATE and WRITE SMS requested sequentially. After we have the input to the ANN mapped as an integer sequence,

the next step is to obtain the value for the each feature. Recall that we use the appearance of a permission as the feature value. For each feature that appears, its value is assigned as 1. For the features that do not appear, we assign their values as 0. After the last two for loops in Algorithm 1, we obtain a feature vector for the input of ANN as follows:

1,0,0,0,0,0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,

0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0

**Step 4:** Classifier learning. In this step, we use the learning module established in the neural network to learn the application behavior from training data. We input the feature vectors to the Matlab Neural Network Toolbox built-in Matlab R2013a (8.1.0.604) to implement permission-based detection. We set the number of nodes in the hidden layer to 10 and then 20.

***Online Detection*** The workflow of the online detection phase is similar to the one described in the offline training phase. Similarly, to classify an application, the first step is to dump permissions and map the permission sequence to the format required by the ANN. We can then use the trained ANN to determine whether a new application is either malware or benign. We use the established ANN and test data as input from new applications. The test file has the same format as the training file, which consists of the feature vector associated with each application. The online

detection process outputs the result file which contains the classification result. In our implementation, the result is either +1 or -1. Here, when the number is positive, the ANN classifies it as a benign application; when the number is negative, the ANN classifies it as malware.

### 3.4.2 System Call-Based Detection

The workflow of the detection system based on system calls is similar to the detection system based on permissions. The major difference is to use a different data source. In the following, we briefly introduce the workflow of system call-based detection. Offline Training As before, we now discuss the steps for offline training.

**Step 1:** Data set collection and classification. The first step is to collect the da-ta set. After we collect real-world benign applications and malware samples, we categorize them into different groups.

**Step 2:** System calls recording. We record the system calls used by our benign applications and malware samples by applying a known tool Strace. In order to install Strace, we use the Nexus Root Tookit v1.6.2 to obtain root permission on Android devices. Next, we run Strace and capture the system calls used by the benign applications and malware. To install malware on an Android device from a remote computer, we use the Android Debug Bridge (ADB).

**Step 3:** Feature extraction. We then record a set of files where each file contains

the system calls generated by each executed application. To use the ANN, we need to process the data and map them to the required format described previously. Using Algorithm 1, we map each system call named to an integer. As an example, clock-gettime is mapped to 1, recvfrom is mapped to 5, and ioctl is mapped to 7. Again, we can extend this idea to use 2-grams as a detection feature by applying two contiguous system calls as a detection feature instead of one. To construct the mapping for 2-grams, we combine each pair of contiguous integers and generate output similar to "0101 0101 0105 0507 0701 0117 1717 1717 1717 1706" where "0105" represents system calls clock-gettime and recvfrom executed sequentially. We then capture the density of system calls by computing the ratio of the number of instances of each system call to the total number of system calls generated by the application. We can then express a feature and its value as feature:value such as "1:0.2283 2:0.0369 3:0.0387 4:0.0267 5:0.0848" where feature 1 has a density of 0.2283, feature 2 has a density of 0.0369, etc.

**Step 4:** Classifier learning. This step is the same as Step 4 for permission- based detection. Afterwards, we have completed the training process of the ANN and are ready to use it to conduct online detection.

***Online Detection*** The workflow of the online detection phase is similar to the one in the offline training phase. Similarly, to classify an application, we execute it, dump the system calls, and map the sequence of system calls to the format required

by the ANN. Using the ANN established through the offline training phase, we can determine whether a new application is malware or benign.

## 3.5 Performance Evaluation

In the following, we present the performance evaluation.

### 3.5.1 Evaluation Methodology

Using real-world malware and benign applications collected on the Android platform, we show the effectiveness of our developed detection system. We installed 96 benign software applications from Google Play and evaluated 92 digital book malware samples from the Android Malware Genome Project (http: //www.malgenomeproject.org/).

We installed and executed applications on the Sumsang Galaxy Nexus and Google Nexus 7 smartphones in our experiments. First, we collected and transmitted each application's permission requests and system calls to a remote computer which conducted both the offline and online detection processes described in Section 4. A Samsung Notebook NP700G equipped with Intel Core i7 2.40GHZ processor, 16GB RAM, and 320GB hard drive served as our detection computer. Again, we used the Matlab Neural Network Toolbox built-in Matlab R2013a (8.1.0.604) that contains both of the FNN and RNN implementations used in our experiments. The number of hidden nodes in the FNN and the RNN are set to 10 and then 20.

With a larger training set, more information can be used to train the ANN classifier, leading to higher detection accuracy. To validate this hypothesis, we let $p \in [0,1]$ which define the training set ratio as the ratio of the number of training samples to the total number of samples. If n is the number of total applications then np is the number of applications used for training and $n(1-p)$ is the number of applications used to validate the accuracy of the trained ANN. To measure the effectiveness of our detection system, we define the detection rate as the probability of correctly classifying the malware. That is, the ratio of the number of malware correctly detected to the total number of malware samples. We also define the error rate as the probability of falsely classifying applications. That is, the ratio of the number of applications falsely classified to the total number of applications.

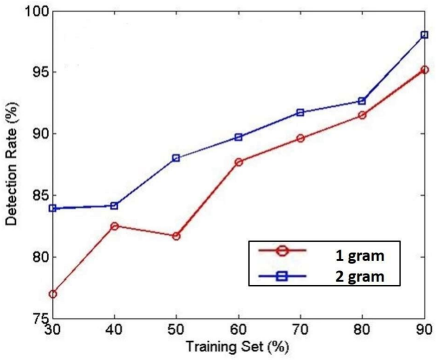


Figure 3.7: Detection Rate for Permission Based Detection vs. Training Set Ratio (FNN with 10 Nodes)

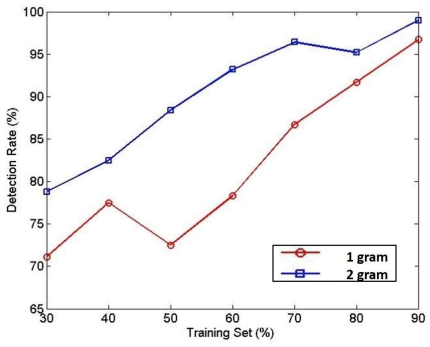


Figure 3.8: Detection Rate for Permission Based Detection vs. Training Set Ratio (FNN with 20 Nodes)

### 3.5.2 Evaluation Results

***Permission-Based Detection:*** Figure 3.7 illustrates the relationship between the detection rate and the training set ratio in terms of the length of grams when an FNN with 10 hidden nodes is used. As we can see, in general, the detection rate rises as the training set ratio increases. The permission-based detection with 2-gram data as input can achieve a better detection rate than the permission-based detec-tion with 1-gram data as input. For example, when the training set ratio is 60 %, the detection rate reaches almost 90 % when 2-grams are used while the detection rate is 85 % when 1-grams are used. As we expected, when using more training data, more knowledge of malware can be obtained, leading to increased detection accuracy.

Figure 3.8 shows the detection rate versus training set ratio when the number of hidden nodes of the FNN is set to 20. Similar to Figure 3.7, as we increase the size of the training set, the detection rate increases. Like before, detection using 2-gram data as input achieves better performance than detection using 1-gram data as input. In the case of 2-gram data as input, when the training set ratio is higher than 50%, the FNN with 20 hidden nodes performs better than the one with 10 hidden nodes. We also observed that, in the case of 1-gram data as input, the FNN with 10 hidden nodes performs better than the FNN with 20 hidden nodes. One reason may be caused by limited malware samples.

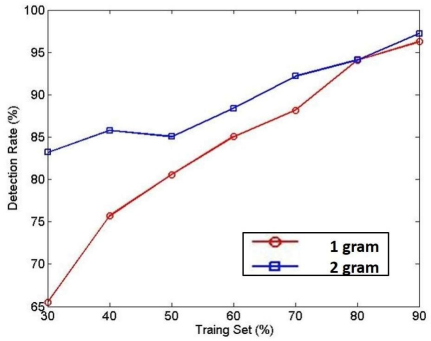


Figure 3.9: Detection Rate for Permission Based Detection vs. Training Set Ratio (RNN with 10 Nodes)

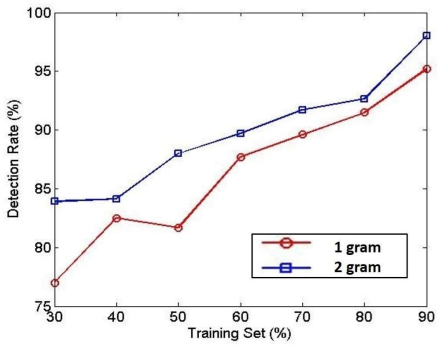


Figure 3.10: Detection Rate for System Call Based Detection vs. Training Set Ratio (FNN with 10 Nodes)

Figure 3.9 illustrates the result of an RNN with 10 hidden nodes. In comparison with Figure 3.7, we can see that the FNN achieves better performance in both the 1-gram and 2-gram cases than when using the RNN. Hence, we conclude that the FNN is more effective for permissions-based detection.

***System Call-Based Detection:*** Figures 3.10, 3.11 and 3.12 illustrate the relationship between the detection rate and training set ratio in terms of the length of data grams when we take system calls as input. Similar to the permission-based detection shown in Figures 3.7, 3.8 and 3.9, when more samples are used in the training process, a higher detection rate can be achieved. For example, when we use a training set of 90%, both the FNN and the RNN achieved detection rates of more than 93%. When the hidden nodes are set to 10, the RNN obtains better detection accuracy than the FNN for both permission-based and system call-based detection.

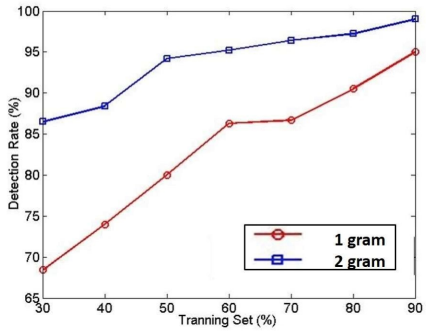


Figure 3.11: Detection Rate for System Call Based Detection vs. Training Set Ratio (FNN with 20 Nodes)

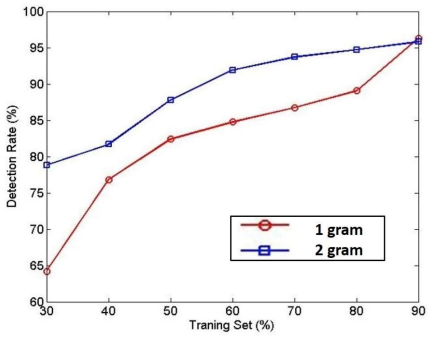


Figure 3.12: Detection Rate for System Call Based Detection vs. Training Set Ratio (RNN with 10 Nodes)

We also study the accuracy of our detection system using another metric: error rate. We expect that with a larger training set, our detection will produce a lower error rate. Figures 3.13 and 3.14 illustrate the relationship between error rate and the training set ratios when we take permissions and system calls as inputs to an FNN and an RNN. In our evaluation, we selected two scenarios to validate that our de-tection system obtains low error rates; other scenarios are essentially similar.

We used 1-grams for data input and set the hidden layer of the FNN and RNN to contain 10 nodes. We have several observations from Figures 3.13 and 3.14. First, for both permissions-based and system call-based detection, the error rates of both the FNN and RNN decrease as the training set ratio increases. This can be explained by observing that as we use more data in the training process, the FNN and RNN have a better chance to learn input data. This leads to the generation of a more

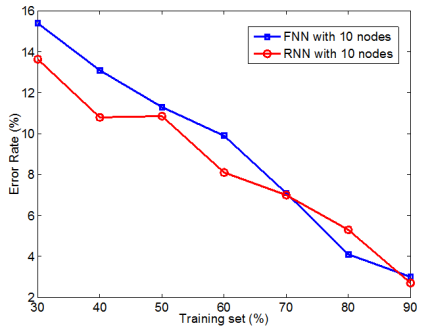


Figure 3.13: Error Rate for Permission Based Detection vs. Training Set Ratio (1-gram)

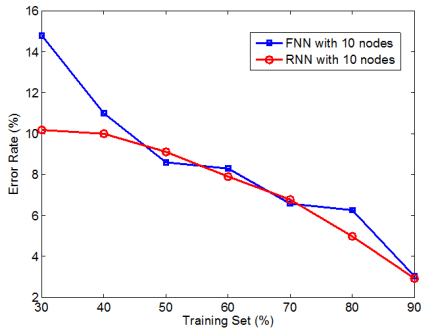


Figure 3.14: Error Rate for System Call Based Detection vs. Training Set Ratio (1-gram)

accurate network for classification and a lower error rate. Second, the error rates are low for both the FNN and RNN in our detection system. For example, using a training set of 60% with permissions-based detection, the error rate is 10 % using the FNN and 8% using the RNN. Similar results have been obtained using system call based detection. Thus, we have confirmed that our detection system obtains high detection rates as well as low error rates, ensuring detection accuracy.

## 3.6 Discussion

We now discuss some issues related to our malware detection system.

The major overhead of our ANN-based detection system comes from the training process. It is worth noting that the training process consists of procedures for collecting data sources, mapping data sources, and training the neural network.

After the network is well trained, the online detection procedure can be fast. Overhead for the training process can be presented by $T = np(T_d + T_m) + T_l$, where $n$ is the number of total applications, $p$ is the training set ratio, and $T_d$, $T_m$, $T_l$ are the average overhead for: dumping permissions and system calls from one application, mapping process and training the neural network, respectively.

As an example, consider training using 1-grams. In our experiment, we implemented the permission-based detection and measured the execution time of each step. With p = 90% and n = 188, the average time consists of 0.000343 second to dump permissions, 0.00012 second to map permissions, and 0.41 second to train neural networks. Hence, the total overhead of the training process is 0.613 second. Similarly, we investigated the overhead of system call-based detection. We note that in order to dump system calls associated with the execution of applications, we need to manually execute applications on real-world mobile devices and the execution times can be random, depending on the application. In our experiments, the overhead of mapping process is 0.00026 second and the total time for the training process is 0.194 second. It is worth noting that the computation overhead linearly increases with the number of applications. To make our system scale, one possible solution is to take advantage of powerful hardware for neuromorphic approaches to conduct threat analysis and detection.

## 3.7 Summary

Malware attacks on smart mobile devices have been growing and posing security risks to mobile users. In this chapter, we developed an ANN-based malware detection system to automatically learn the behavior of applications and to detect unknown malware. In our developed system, we systematically compared the per-mission requests from application requests and system calls to capture the behavior of applications. Using real-world malware and benign applications, we conducted experiments on Android mobile devices. Our data shows the effectiveness of our developed detection system.

# Chapter 4

# Effective Sampling and Data Aggregation Techniques in Host-based Intrusion Detection

## 4.1 Overview

In this chapter, we address the issue of monitoring and detecting cyber attacks in MANET. A MANET demand a robust, diverse, and resilient communication and computation infrastructure, which enables the network-centric operation with a very low rate of downtime. Nonetheless, the nature of MANET leads cyber security risks, because mobile nodes are deployed in the open field, which could be hostile and the wireless communication makes the information accessible by the adversary, who may actively intercept, disrupt, or manipulate the information. The adversary may hack into hosts and network devices inside the network using sophisticated attack techniques to prey on the vulnerabilities of system components and disrupt the mission of MANET.

There are several challenging issues in monitoring and detecting cyber attacks in MANET. First, unlike wired or infrastructure mode wireless networks, resources in MANET (i.e., communication bandwidth and host storage and computation capability) are very limited. To enable cyber attack monitoring and detection to secure MANET

as the homeland in battle fields, we shall transmit a large amount of suspicious information over MANET in real time, which has limited bandwidth resources connected hosts to the operation center. Nonetheless, transmitting a large amount of attack monitoring and detection data over MANET has a negative impact on other mission related applications supported by MANET itself. Therefore, the monitoring and detection of attacks should be designed such that its impact on mission related applications should be controlled and limited. Second, although a number of HIDS [1–3] have been developed in the past, those systems mainly aim at securing enterprise networks that have defined structures. Because MANET is mostly ad hoc in nature and has limited bandwidth and computing resources, the existing HIDS cannot be directly used for MANET.

To address these issues, in this chapter we first study the host-based detection architecture to monitor and detect cyber attacks and secure MANET. To enable the effective detection and desirable impact on the performance of MANET, we develop two sampling techniques and investigate proper settings for those sampling techniques to achieve desirable tradeoffs between the detection accuracy and the consumption for network resources. In particular, we develop both simple random sampling and the stratified sampling techniques. The simple random sampling technique uniformly samples the detection information and the stratified sampling technique stratifies the detection information and sample them with different priorities.

We derive closed formulae to analyze the impact of sampling techniques and key parameters on detection accuracy. We investigate the impact of attack detection on the performance of MANET and formalize an optimization problem of allocating network resources for the normal mission related application supported by MANET and the application for conducting attack monitoring and detection in MANET. We discuss various issues, including the system architecture options, dynamic sampling, and data aggregation. We also implemented our proposed sampling techniques and conduct experiments on a real-world testbed. Our experimental data show that the stratified sampling technique achieves much better performance than the simple random sampling technique in terms of detection accuracy and the consumption of network resources. We also evaluate the performance impact of sampling techniques on the performance of MANET using the ns-3 based simulation.

While sampling techniques have been widely used for traffic measurement and accounting to deal with the vast amount of traffic data continuously collected for network monitoring and management [30], little research has been paid to investigate sampling techniques on the host-based detection in MANET, which have limited network and computing resources. We would like to point out that our developed sampling techniques mainly deal with the host detection information in MANET, which are different from the information in the traditional wired and sensor networks. Our sampling techniques consider tradeoffs between bandwidth reduction and detection

accuracy. We derive closed formulae to study the relationship between detection accuracy and sampling techniques and sampling rate. Our experimental results validate our theoretical findings well. In addition, we also briefly discuss various system architecture option, dynamic sampling, and aggregation techniques.

Notice that the materials in this chapter are adapted from my previous publication [70, 71].

## 4.2 A Host-based Intrusion Detection Architecture in MANET

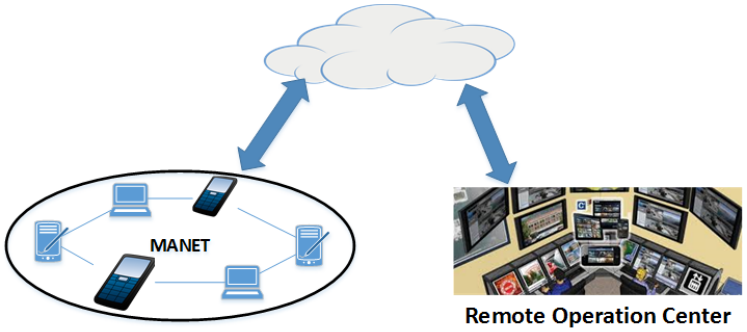


Figure 4.1: System Architecture

MANET may operate in hostile environments. The adversary may hack into the entities of such systems by preying on vulnerabilities in host and network components to disrupt supported missions and inflict significant damage. To monitor and detect cyber attacks in MANET, we study the host-based detection architecture shown in Figure 4.1. In this architecture, there are two main components: (i) the host-based threat monitoring agent, which is installed and executed on hosts in

MANET, and (ii) an operation center, which enables the human analyst to monitor and detect attacks in MANET.

To monitor and detect cyber attacks, hosts in MANET are deployed with the threat monitoring software denoted as the threat monitoring agent. Generally speaking, the agent collects the suspicious information in real time from system logs, security logs, application logs, and others, and forwards detection reports to the operation center, which further conducts threat analysis and detection. Generally speaking, the monitoring agent shall monitor suspicious activities on the host, including the integrity of system files, dynamic behavior, suspicious processes, illegal resource accesses and suspicious system function calls, changes in user privileges, login attempts, and many others. Considering that hosts are mobile and have limited storage and computing resources, the agent on hosts shall have a small memory and CPU footprint by default and shall not affect system usage. In addition to monitoring and detection, agents on some high performance hosts could detect intrusions by parsing host events directly to extract the meaningful information using system level semantics and compare activities with patterns that are deemed anomalous on hosts. The agent updates the monitoring and detection information to the operation center in real time and receives commands from the operation center to dynamically update the monitoring and detection policies enforced locally. In this chapter, we use the OSSEC, a well-known open-source, host-based intrusion

detection system [3] as an example, to conduct experiments to demonstrate our proposed sampling techniques. Nevertheless, this architecture is generic and other host-based intrusion detection and network-based intrusion detection can be generally applied as well.

The operation center is responsible for managing agents on hosts in MANET and conducting monitoring and detecting attacks. The operation center will receive the detection information and alerts from agents and manage a large number of agents in the system to detect, track and classify attacks in a time and resource efficient way. As shown in Figure 4.1, the detection information from agents will be transmitted to the operation center through dynamic routes in MANET, which consist of wireless links with limited bandwidths [72]. The operation center will provision threat analysis and detection tools to conduct cyber attack monitoring, detection, and visualization, and to aid in the mitigation of cyber attacks. The interaction between agents and operation center could be conducted in an on-demand way through control and management protocols. For example, the security analyst located at the operation center could dynamically select a area (denoted as monitored region that consists of a number of hosts), which may have a high security risk and send commands to agents associated with the monitored region and update monitoring policies to collect relevant detection information dynamically.

There are several challenging issues in monitoring and detecting attacks in

MANET. First, unlike wired or infrastructure based wireless network, the resources of MANET (i.e., communication bandwidth and host storage and computing capability) are very limited. To enable attack monitoring and detection, we need to transmit a large amount of attack monitoring and detection data over MANET that has limited network and computing resources to the operation center in real time. Nonetheless, transmitting a large amount of data over MANET clearly poses a negative impact on the normal mission related applications supported by MANET. Hence, the monitoring and detection of attacks should be designed in such a way that it has a limited and controllable impact on the normal mission related applications in MANET. Second, although many HIDS (Host-based Intrusion Detection Systems) have been developed [1–3] in the past, those systems mainly aim at securing enterprise networks with fixed infrastructures, which do not need to worry about dynamic packet transmission routes, energy consumption, low bandwidth, and other constraints. Therefore, the existing HIDS cannot be directly used in MANET without addressing these fundamental challenges.

To enable effective attack monitoring and detection in MANET, we shall address the following fundamental problem: *How can we develop techniques to transmit the attack detection information with desirable and controllable impact on the performance of MANET while achieving the desired detection accuracy?* To address this issue, in our research project we consider the following two orthogonal dimensions for

developing our techniques: (i) sampling, and (ii) aggregation. Due to the limited space, in this chapter, we focus on the development of effective sampling techniques for monitoring and detecting cyber attacks in MANET with desirable and controllable impact on the performance of MANET.

## 4.3 Our Approaches

In this section, we introduce our proposed approaches in detail.

### 4.3.1 Overview

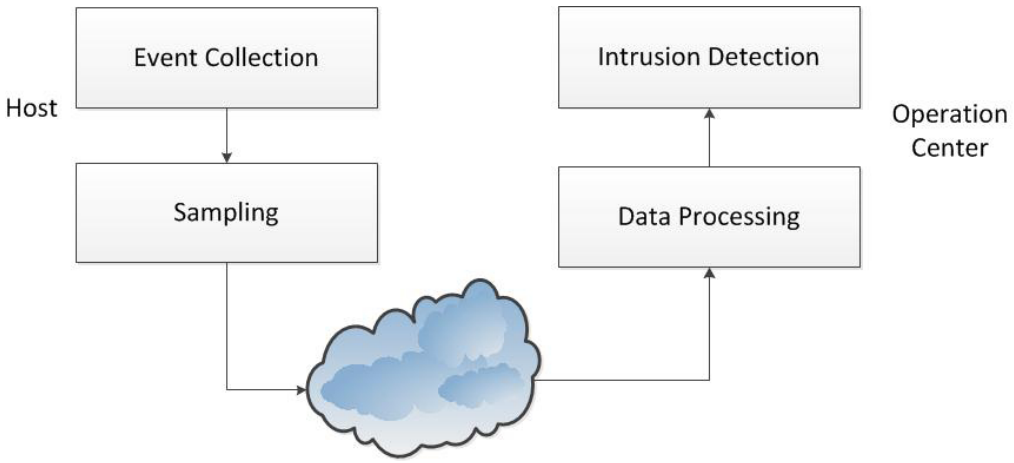


Figure 4.2: System Workflow

Recall that in our system, there are two major components: (i) monitoring agents, and (ii) the operation center. From Figure 4.1, we can see that monitoring agents are deployed on mobile hosts in MANET and the operation center is located

remotely through bandwidth-limited network links. The basic workflow is listed as follows: The monitoring agent located in MANET traps suspicious activities on hosts and store such information temporarily in local. Because the volume of detection information can be high given a large number of hosts in the network, transmitting all detection information through MANET will disrupt its normal mission related application in MANET. To address this issue, we develop sampling techniques. The system workflow is illustrated in Figure 4.2. As we can see, the original detection information will be the input to our developed sampling algorithms and the sampled detection information will be transmitted through MANET to the operation center. The operation center processes the received detection information that is sampled and maps the information to the format used by detection algorithms, which make detection decision.

**Step 1: Data collection.** In this step, we install the agent on mobile hosts to collect suspicious activities on hosts (e.g., system logs, security logs, application logs, and others). Given a large number of hosts in MANET, a large volume of real time data from agents will be generated and transmitted to the operation center for processing and detect malicious threats. Because MANET has limited bandwidth and computing resources, we cannot afford to transmit all logged data at hosts to the operation center. To reduce the impact on MANET, we deploy the sampling process, which will be described in the next step.

**Step 2: Sampling.** In this step, collected data will be the input to the sampling process. We develop two sampling techniques: (i) simple random sampling, and (ii) stratified sampling. Both techniques use a sampling rate to control the amount of detection information to be transmitted through MANET, which is defined as the fraction of data to be selected from the total data generated at the host. Obviously, the sampling process could effectively reduce the traffic load overhead to MANET. Nevertheless, there is a tradeoff between sampling rate and detection accuracy. A higher sampling rate will result in a higher detection accuracy and requires a larger amount of bandwidth, which might exceed the capacity of MANET, posing a negative impact on the normal mission related applications. The detailed description of sampling techniques and their impact on the performance of MANET will be presented in Section 4.3.2 and Section 4.4, respectively.

**Step 3: Data processing.** Typically, a HIDS collects and analyzes system logs to identify and detect malicious activities on hosts. Nevertheless, in MANET, data will be transmitted to the operation center. To facilitate the detection process, we develop a feature-based data process to describe incidents. We define a series of basic features and each event could be described as a combination of features. Using the break-in attempt as an example, the attack could be defined by the following basic features: *pam*, *syslog*, and *authentication_success*. It is initiated by "pam" (the authentication facility in UNIX), then recorded by "syslog", and followed by

"authentication_success". As another example, a critical system file change may reflect *syscheck* and *sysfile_integrity* because it is detected by "syscheck" program in OSSEC and the integrity of system files. With the feature extraction, each incident contains the following fields: *time*, *source host*, *source program*, *severity level*, and *features*. The details of processing data can be found in Algorithm 2.

**Algorithm 2:** Data Processing Algorithm

**input** : An array of collected binary data $Array(A)$; Empty array $Array(B)$
**output** : $\{X_i\}$, hashable array of events with feature data

```
1  Array[1 : n] = X_1, X_2, ... X_n;
2  foreach Element A_i in Array(A) do
3  |  k = extract_bin(A_i);
4  |  B_i['features'] = k['features'];
5  end
6  Sort Array(B) by features;
7  Remove duplicate entry in Array(B);
8  foreach element A_i in Array(A) do
9  |  k = extract_bin(A_i);
10 |  for i = 1 to n do
11 |  |  if B_i['features'] == k['features'] then
12 |  |  |  B_i['events'] += abstract(k);
13 |  |  else
14 |  |  |  pass;
15 |  |  end
16 |  end
17 end
18 foreach Element B_i in Array(B) do
19 |  Aggregate B_i['events'];
20 end
21 X ⟵ B;
```

Therefore, the sampled data will be converted into a stream of numerical data, which represents threat magnitude of events. A larger value represents a higher

probability of threats. This value will be the input to the intrusion detection algorithm, which will make detection decision. The process of detection is described next.

**Step 4: Intrusion detection.** The operation center is responsible for coordinating agents on hosts in MANET and conducting detection. With the sampled data in Step 3, the operation center provisions threat analysis and detection algorithms, enabling the cyber analyst to monitor, query, detect, and visualize the detection information with the aim of mitigating cyber attacks by planning network defense resources. As a preliminary result, we have implemented a statistical anomaly-based detection technique, which analyzes collected data in comparison with the normal profile to make the detection decision. To be specific, we obtain the statistics of processing data through the offline training, and then determine whether the received detection data contains attack or not. The detailed description of detection algorithms will be discussed in Section 4.3.4. Note that other advanced detection algorithms could be provisioned in our prototypical system and we leave the full investigation in our future study.

### 4.3.2 Sampling Techniques

We now introduce the sampling techniques in detail. Recall that the main objective of sampling techniques is to balance the tradeoff between reduction of bandwidth usage and detection accuracy.

#### 4.3.2.1 Simple Random Sampling

The simple random sampling is a baseline sampling technique. With this sampling technique, each detection event will be selected randomly with an equal probability. On a monitoring agent, we select $n$ events (denoted as sampling data) out of $N$ original events (denoted as the full set of detection data) such that each of $C_N^n$ distinct samples has an equal probability of being drawn. In principle, the sample, as an unbiased random selection of detection data, is used to represent the behavior of the original full set of data. Nevertheless, because the sampled data is only a subset of the full set of data, it will incur some error to detection decision.

To conduct detection in the operation center, we use the sampled data to detect attacks. However, in real-world practice, the detection information should not be treated equally. To improve detection accuracy, we shall categorize the detection information into different groups based on various priority levels. For this purpose, we introduce the stratified sampling technique, which will be described next.

#### 4.3.2.2 Stratified Sampling

We now introduce an enhanced sampling technique denoted as stratified sampling. The original detection data from a host contains different events that tell information from the system security aspect. As an example, an event where a malware copies itself to a system directory is more risky than a normal system call from the security

aspect. To quantify this, we introduce the stratified sampling technique that defines the priority levels of individual events and put various weights during the sampling process. In particular, the original detection information of $N$ units will be first divided into sub-groups of $N_1, N_2, \ldots, N_L$ units and each group contains events that have the same priority level. These subgroups are disjoint and together comprise the full set of detection data, i.e., $N_1 + N_2 + \ldots + N_L = N$. To obtain the benefits of stratification, the true values of $N_k$ could be estimated. When the strata are determined, a sample is drawn from each group and the drawings are made independently in different strata. The sample sizes within strata are denoted by $n_1, n_2, \ldots, n_L$, respectively.

Nonetheless, in both the random and stratified sampling, a sampling error will exist because sampling process only uses a subset of detection data to estimate the characteristics of the full set of detection data. In principle, a sample selected from the full set of detection data is the one of all possible samples. Any value computed from the sample is based on the sampled data and is denoted as the sample statistics. The sample statistics may not be close to the statistics of the full set of detection data. If one statistics measure is $\theta$ and the true value of the statistical measure for the full set of detection data is $\hat{\theta}$, the difference between $\hat{\theta}$ and $\theta$ is defined the sampling error. We will further investigate the impact of detection accuracy caused by sampling process in Section 4.4.

**Algorithm 3:** Sampling Algorithms

**input** : An array of converted data $Array[X]$; Sampling rate $P_s \in [0\%, 100\%]$;
**output** : $[Y]$, array of sampled data

```
Array[X] = X_1, X_2, ..., X_N;
if Simple Random Sampling then
    for i = 1 : N do
        for j = 1 : P_s × N do
            a = random(min = 0, max = 100);
            if a < P_s then
                X_i = Y_j;
            else
                pass;
            end
        end
    end
    [X] ⟶ [Y].
else
    if Stratified Sampling then
        Array[X'];
        ArrayA[layer][n];
        ArrayA = stratify(X, layers=L);
        //A is a two dimension array. First dimension is layer, second
          dimension is data in this layer
        c = element_count(A);
        P'_s = P_s × element_count(X)/c;
        foreach layer in A do
            foreach element in layer do
                a = random(min = 0, max = 100);
                if a < P'_s then
                    X' += element;
                else
                    pass;
                end
            end
        end
        [X'] ⟶ [Y].
    else
        pass;
    end
end
```

```
Oct 16 00:19:01 web CRON[32132]: pam_unix(cron:session): session opened for user bbs by uid=0}
Oct 16 00:19:01 web CRON[32133]: pam_unix(cron:session): session opened for user bbs by uid=0}
```

Figure 4.3: An Example of System Logs

### 4.3.3 Data Aggregation Techniques

To enable the effective detection and reduce the impact on network performance, we also consider the data aggregation techniques. Generally speaking, data aggregation is a process, which reduces the volume of data while preserving the meaning information of data [73–77]. The aggregation techniques can be categorized into two groups: lossless and lossy aggregation. To be specific, for the lossless aggregation technique, we could adopt the compression strategy to consolidate the detection information under the constraint, in which the decompressed data contains the exactly same amount of information as the original one. Different from the existing compression mechanisms, we consider the lossless aggregation technique that use the syntax of detection information to effectively remove the duplicated information and compress the pivotal data with a high aggregation ratio.

We use the OSSEC [78] to investigate the feasibility of lossless data aggregation, which generates the system logs shown as in Figure 4.3. As we can see, the content of messages have lots of redundant information. In our preliminary study, we found that for a sample system-log with 10000 records, there are only 2367 unique message bodies, 3 daemons and 3993 unique timestamps. To remove the redundant

Sample Input:

*Oct 16 00:19:01 web CRON[32132]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:19:01 web CRON[32133]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:19:01 web CRON[32133]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:19:02 web CRON[32132]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:20:01 web CRON[1709]: pam_unix(cron:session): session opened for user www-data by (uid=0)*
*Oct 16 00:20:01 web CRON[1711]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:20:01 web CRON[1709]: pam_unix(cron:session): session closed for user www-data*
*Oct 16 00:20:01 web CRON[1711]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:21:01 web CRON[5292]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:21:02 web CRON[5292]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:22:01 web CRON[7528]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:22:01 web CRON[7528]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:23:01 web CRON[13862]: pam_unix(cron:session): session opened for user bbs by (uid=0)*
*Oct 16 00:23:02 web CRON[13862]: pam_unix(cron:session): session closed for user bbs*
*Oct 16 00:23:50 web su[15748]: Successful su for alex by root*
*Oct 16 00:23:50 web su[15748]: + pts/1 root:alex*
*Oct 16 00:23:51 web su[15748]: pam_env(su:session): Unable to open env file: /etc/default/locale: No such file or directory*
*Oct 16 00:23:51 web su[15748]: pam_unix(su:session): session opened for user alex by root(uid=0)*
*Oct 16 00:24:01 web CRON[15842]: pam_unix(cron:session): session opened for user bbs by (uid=0)*

Figure 4.4: An Example of Original System-Logs

information and reduce the bandwidth consumption on transmitting data through MANET. As a proof-of-concept, the detailed steps on data aggregation are described below. First, we split the log file into several parts: DATE (e.g., Oct 16 00:19:01), Domain (e.g., web CRON [32133]), Message Body (e.g., pam_unix(cron:session): session closed for user bbs), and Message Source (e.g., uid=0). We then scan the system log files and find unique strings and generate the index for those unique strings, and output strings and index for each system log entry into a binary file and then apply generic compression methods such as *bzip2* and *Lempel–Ziv–Markov chain (LZMA)* algorithm [79] to compress the file. The examples of original system logs and aggregated system logs can be found in Figure 4.4 and Figure 4.5.

### 4.3.4 Detection

Generally speaking, an intrusion detection system can be classified as either signature-based or anomaly-based detection. In a signature-based detection, a large repository

Sample Output:

*Daemons:*
*CRON*
*su*

*Index:*
*1 1 1*
*0 0 0*
*0 0 2*
*2 0 1*
*3 0 3*
*2 0 0*
*2 0 4*
*2 0 1*
*4 0 0*
*5 0 1*
*6 0 0*
*5 0 1*
*7 0 0*
*8 0 1*
*9 2 5*
*8 1 6*
*10 1 7*
*9 1 8*
*11 0 0*

*Messages:*
*pam_unix(cron:session): session opened for user bbs by (uid=0)*
*pam_unix(cron:session): session closed for user bbs*
*pam_unix(cron:session): session opened for user www-data by (uid=0)*
*pam_unix(cron:session): session closed for user www-data*
*Successful su for alex by root*
*+ pts/1 root:alex*
*pam_env(su:session): Unable to open env file: /etc/default/locale: No such file or directory*
*pam_unix(su:session): session opened for user alex by root(uid=0)*

*Date_Time:*
*Oct 16 00:19:01*
*Oct 16 00:19:02*
*Oct 16 00:20:01*
*Oct 16 00:21:01*
*Oct 16 00:21:02*
*Oct 16 00:22:01*
*Oct 16 00:23:01*
*Oct 16 00:23:02*
*Oct 16 00:23:50*
*Oct 16 00:23:51*
*Oct 16 00:24:01*

Original Size: 1698 Bytes
Compressed Total Size: 360 Bytes (compression ratio: 21%)
Compressed Original Size: 420 Bytes (compression ratio: 25%)

Figure 4.5: An Example of Aggregated System Logs

of known attack signatures [80] will be maintained and used to detect attacks. The disadvantage of this approach is that it cannot deal with new attacks. In a anomaly detection, the system administrator commonly defines the baseline or normal measures of system behavior (e.g., activities on hosts, network traffic rate, and others). In the anomaly-based detection, system activities will be monitored and compared with the normal baselines to determine whether an attack was occurred. One common technique is to estimate the normal behavior of the protected system and generate a detection alert whenever the deviation between a given observation and the normal behavior exceeds a predefined threshold [81, 82].

In our developed system, we adopt the system log as as the basic detection sources to formalize detection information on hosts [3]. Recall that the anomaly detection works through establishing a "normal" operation profile in a system and detect suspicious activities by comparing the system run-time behavior with the normal operation profile. With the development of modern operating systems, the user and administrator need to know what are happening on hosts and network components. It is desirable to provide human readable and reliable data to define system status. The benefit of system logs provide a standard data format, which could be collected and processed to detect threats.

In this chapter, we use the OSSEC, which is a well-known open source HIDS, to demonstrate the effectiveness of our proposed sampling algorithms. In particular,

we install the OSSEC client on the end host that collect system logs in real time. Then the daata processing components introduced in Section 4.3.1 will process the collected data and transmit the information to the OSSEC server, which is denoted as the operation center. Then the server take the responsibility of intrusion detection based on analyzing system logs. In its release package, the OSSEC server executes simply “compare-match” mechanism to identify the abnormal behavior. In this paper, we develop the statistical threshold-based detection algorithm to detect intrusions, which will be detailed below.

We consider the statistical threshold-based detection, which is a generic and fundamental technique to conduct anomaly-based detection. The insight of this technique works in the following way: We first compute the mean and the standard deviation of collected detection data. The mean ($m$) and the standard deviation ($v$) are the statistical measures of normal behavior of protected system. We then establish a threshold as $T_a = k \cdot v$, where $k$ determines the sensitivity degree of deviation from the system normal behavior. Recall that in our system, we use the attack feature and map it to the number and assume that the larger the number, the higher severity the attack is. A data $X_i$ is treated to be anomalous if it deviates from the mean by more than a threshold, i.e. $X_i > T_a$. The statistical anomaly based detection is shown in Algorithm 4. As we can see, in this anomaly-based detection we obtain the statistical characteristics of detection information through

offline training and compare it with the run-time data and make decision.

## 4.4 Analysis

In this section, we first show the analytical results for investigating the accuracy of anomaly-based detection. We then study the impact of sampling techniques on detection accuracy and bandwidth reduction in MANET. Our experimental results in Section 4.5 match our theoretical results. All notations for sampling techniques can be found in Table 4.1. Note that here, we use *Gaussian* white noise as an example in our theoretical analysis to provide insights into the effectiveness of our developed sampling techniques. Nevertheless, we would like to clarify that the Gaussian distribution has been widely used as a model of quantitative phenomena in the natural and behavioral sciences. The use of the Gaussian distribution can be theoretically justified by assuming that many small, independent effects are additively contributing to each observation. Indeed, the distribution of detection data is still open question, which is largely depend on the attack behaviors. As shown in Section 4.5, the real-world experimental results and theoretical analysis results are consistent and follow the same trend, which demonstrate the impact of sampling algorithms and sampling rate on detection accuracy. The difference between the experimental results and analytical results indicates that the real-world attack information may not follow the Gaussian white noise distribution and further

study can be one direction of our future work.

### 4.4.1 Anomaly-based Detection

**Algorithm 4:** Statistical Anomaly-Based Detection

**input** : An array of converted data
**parameter:** $a$, measure of degree of deviation
**output** : $T_a$, Threshold

```
Array[1 : n] = X_1, X_2, ..., X_n;
sum += Array[i];
m = sum/n;
v = std(Array[i]);
Set T_a = m + a.v ;
foreach element X_i in Array[i] do
    if X_i > T_a then
        X_i is abnormal data;
    else
        X_i is normal data;
    end
end
```

We assume that the anomaly-based detection uses the statistical-based detection scheme described in Section 4.3 because it is a generic and representative one. To evaluate detection accuracy, we consider two metrics. One is the detection rate $P_D$, which is defined as the probability of correctly determining attack. The other is the false positive rate $P_F$, which is defined as the probability that the attack is mistakenly detected while no attack exists. To make detection decision, we determine the anomaly detection threshold $T$, where $T = m + k \cdot v$, $k$ is the parameter to determine the degree of deviation from normal behavior. Without loss

of generality and simplifying our analysis, we assume that the background activities follows the *Gaussian* random distribution with the mean $m$ and standard deviation $v$. With the attack in place, we assume that the mean and standard deviation of monitored activities are $m_a$ and $v_a$, respectively. Then we have Theorem 1 for detection accuracy.

**Theorem 4.4.1.** *For the statistical-based detection described in Section 4.3.1, the detection rate $P_D$ can be derived by*

$$P_D = 1 - \Phi(\frac{T_a - m_a}{v_a}), \tag{4.4.1}$$

*and the false positive rate $P_F$ can be derived by*

$$P_F = 1 - \Phi(\frac{T_a - m}{v}), \tag{4.4.2}$$

Here $\Phi(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$ is the standard error function, $T_a$ is, $v_a$ is ..., $m$ is .... Theorem 1 can be proved based on the standard definition of probability density function. In particular, $P_D = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{T_a - m_a}{v_a}} e^{-\frac{1}{2}y^2} dy$ and $P_F = 1 - \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\frac{T_a - m}{v}} e^{-\frac{1}{2}y^2} dy$. We have some observations from Theorem 1. First, the detection rate grows when the attack is stronger, which means the statistical based anomaly detection could effectively detect attacks, which show a relatively strong activities. Second, there are some tradeoffs between detection rate and false positive rate. If the value of threshold declines, a higher detection rate could be achieved. However, a smaller value of threshold always incurs a higher false positive rate.

| | |
|---|---|
| $Y$ | Total Population |
| $\sum^n y_i$ | Total Samples |
| $\overline{Y}$ | Population Mean |
| $\bar{y}$ | Sample Mean |
| $S^2$ | Population Variance |
| $s^2$ | Sample Variance |
| $P_s$ | Sampling Rate |
| $N_h$ | Total number of units in stratum $h$ |
| $n_h$ | Number of units in stratum $h$ sample |
| $y_{hi}$ | Value obtained for $i^{th}$unit |
| $W_h = \frac{N_h}{N}$ | Stratum proportion |
| $s_h^2$ | Variance in stratum $h$ |
| $P_h = \frac{n_h}{N_h}$ | Sampling fraction in the stratum |
| $\bar{y_{st}}$ | Sample mean in the stratified sampling |

Table 4.1: Notations

### 4.4.2 Simple Random Sampling

Recall that we sample the detection information randomly based on a sampling rate $P_s \in [0,1]$. The sampling process will have a negative impact on detection accuracy. To measure such an impact, we define the error rate as the ratio of detection rate, i.e., sampled detection data vs. detection rate with the full set of detection information.

After the sampling and data processing described in Section 4.3, we obtain the sampled detection data as the input to the detection algorithm. The variance of the sample mean $\bar{y}$ for a simple random sample is

$$V(\bar{y}) = E(\bar{y} - \bar{Y})^2, \tag{4.4.3}$$

$$= \frac{S^2}{n}\frac{(N-n)}{N} = \frac{S^2}{n}(1-P_s), \tag{4.4.4}$$

where $P_s = n/N$ is the sampling rate.

We then obtain the standard error of $\bar{y}$ from the following,

$$\sigma_{\bar{y}} = \frac{S}{\sqrt{n}}\sqrt{(N-n)/N} = \frac{S}{\sqrt{n}}\sqrt{1-P_s}. \tag{4.4.5}$$

To derive the closed formulae, we assume that the estimation of $\bar{y}$ follows a Gaussian distribution. Then, the interval estimate of mean value of full detection data is $\bar{Y} \in [\bar{y} - \frac{ts}{\sqrt{n}}\sqrt{1-P_s}, \bar{y} + \frac{ts}{\sqrt{n}}\sqrt{1-P_s}]$, where $t$ is the value of normal deviation corresponding to desired confidence probability.

We now apply our sampling process to the anomaly-based detection and derive error limits for attack detection. The results are shown in Theorem 4.4.2 that is listed below.

**Theorem 4.4.2.** *By using the simple random sampling technique, the error limit of detection rate for the anomaly-based detection is*

$$\Delta P_d \leq max(p_l, p_u). \tag{4.4.6}$$

*Here, we have*

$$p_l = \frac{1}{\sqrt{2\pi}} \int_{\frac{T_a-\bar{y}}{\sigma}}^{\frac{T_a+e_{rd}-\bar{y}}{\sigma}} e^{-\frac{y^2}{2}} dy, \tag{4.4.7}$$

$$p_u = \frac{1}{\sqrt{2\pi}} \int_{\frac{T_a-\bar{y}-e_{rd}}{\sigma}}^{\frac{T_a-\bar{y}}{\sigma}} e^{-\frac{y^2}{2}} dy, \tag{4.4.8}$$

*where $\bar{y}$ is the sample mean, $e_{rd}$ is the sampling error and can be denoted as,*

$$e_{rd} = \frac{t\sigma_{\bar{y}}}{\sqrt{n}}\sqrt{1-P_s}, \tag{4.4.9}$$

*and $\sigma_{\bar{y}}$ is the variance of the sample as the estimate of variance of detection data.*

*Proof.* In the random sampling, we consider the sample as a representative from the full set of detection data. Nevertheless, because of sampling error $e$, there is an error between the sample mean and the mean of full set of detection data. We have

$$\bar{Y} = \bar{y} \pm e = \bar{y} \pm \frac{ts}{\sqrt{n}}\sqrt{1-P_s}. \tag{4.4.10}$$

According to Theorem 4.4.1, the detection rate is

$$P_d = 1 - \Phi(\frac{T_a - m}{\sigma}). \tag{4.4.11}$$

Then we derive the error limits of detection rate as following,

$$\begin{aligned} \Delta P_d &= (1-\Phi(\frac{T_a\breve{}(\mu+e)}{\sigma})) - (1-\Phi(\frac{T_a-\mu}{\sigma})), && (4.4.12)\\ &= \Phi(\frac{T_a-\mu}{\sigma}) - \Phi(\frac{T_a-\mu-e}{\sigma}), && (4.4.13)\\ &= \frac{1}{\sqrt{2\pi}}\int_{\frac{T_a-\bar{y}-e}{\sigma}}^{\frac{T_a-\bar{y}}{\sigma}} e^{-\frac{y^2}{2}}dy = p_u. && (4.4.14) \end{aligned}$$

Similarly, we have

$$p_l = \frac{1}{\sqrt{2\pi}}\int_{\frac{T_a-\bar{y}}{\sigma}}^{\frac{T_a+e-\bar{y}}{\sigma}} e^{-\frac{y^2}{2}}dy. \tag{4.4.15}$$

Given the threshold for anomaly detection, to balance the tradeoffs between detection rate and false positive rate, we have

$$\Delta P_d \leq \begin{cases} p_u, \text{when } T_a > \bar{y}, \\ p_l, \text{when } T_a < \bar{y}, \end{cases} \tag{4.4.16}$$

Then, we have

$$\Delta P_d \leq max(p_l, p_u). \tag{4.4.17}$$

Hence, Theorem 4.4.2 is proved. □

### 4.4.3 Stratified Sampling

Recall that in the stratified sampling, the full set of detection information is first divided into the mutually exclusive stratums (or groups) and each group is assigned by a priority to determine the weight of selection during the sampling process. The estimation used in the stratified sampling is $\bar{y_{st}}$ where $st$ stands for stratified, and we have

$$\bar{y_{st}} = \frac{\sum_{h=1}^{L} N_h \bar{y_h}}{N} = \sum_{h=1}^{L} W_y \bar{y_h}, \tag{4.4.18}$$

and the sample mean is

$$\bar{y} = \frac{\sum_{h=1}^{L} n_h \bar{y_h}}{n}. \tag{4.4.19}$$

The difference between $\bar{y}$ and $\overline{y_{st}}$ is that in $\overline{y_{st}}$, estimates from individual strata receive ratio $N_h/N$. It is evident that $\bar{y}$ coincides with $\bar{y_{st}}$ when in every stratum we have $\frac{n_h}{n} = \frac{N_h}{N}$.

If the simple random sample is taken in each stratum, an unbiased estimate of $s_h^2$ becomes

$$s_h^2 = \frac{1}{n_h - 1} \sum_{i=1}^{n_h} (y_{hi} - \bar{y_h})^2. \tag{4.4.20}$$

Then with the stratified sampling, an unbiased estimate of variance $\bar{y_{st}}$ is

$$v(\bar{y_{st}}) = s^2(\bar{y_{st}}) = \frac{1}{N^2} \sum_{h=1}^{L} N_h (N_h - n_h) \frac{S^2}{n_h}. \tag{4.4.21}$$

and

$$\sigma_{\bar{y_{st}}} = \sqrt{s^2(\bar{y_{st}})} = \sqrt{\sum_{h=1}^{L} \frac{W_h^2 s_h^2}{n_h} - \sum_{h=1}^{L} \frac{W_h s_h^2}{N}}. \tag{4.4.22}$$

Notice that the derivation of interval estimate of mean value is similar to the one in the simple random sampling, that is, $\bar{Y} \in [\bar{y_{st}} - ts(\bar{y_{st}}), \bar{y_{st}} + ts(\bar{y_{st}})]$. Based on this, we can derive the sampling error in the stratified sampling. Using the same metrics defined above, that is, the error limit of detection rate, we have Theorem 4.4.3 listed below.

**Theorem 4.4.3.** *By using the stratified sampling technique to estimate the characteristics of the full set of detection information, the error limit of detection rate is*

$$\Delta P_{dst} \leq max(p_{lst}, p_{ust}). \tag{4.4.23}$$

*Here, we have*

$$p_{lst} = \frac{1}{\sqrt{2\pi}} \int_{\frac{T_a - \bar{y_{st}}}{\sigma st}}^{\frac{T_a + e_{st} - \bar{y_{st}}}{\sigma_{st}}} e^{-\frac{y^2}{2}} dy, \tag{4.4.24}$$

$$p_{ust} = \frac{1}{\sqrt{2\pi}} \int_{\frac{T_a - \bar{y_{st}} - e_{st}}{\sigma st}}^{\frac{T_a - \bar{y_{st}}}{\sigma_{st}}} e^{-\frac{y^2}{2}} dy, \tag{4.4.25}$$

*where $\bar{y_{st}}$ is the estimate of the mean of full data, $e_{st}$ is the sampling error and can be denoted as,*

$$e_{st} = t \sqrt{\frac{1}{N^2} \sum_{h=1}^{L} N_h (N_h - n_h) \frac{\sigma_{\bar{y_{st}}}^2}{n_h}}, \tag{4.4.26}$$

*where $\sigma_{\bar{y_{st}}}$ is the variance of sample as the estimate of variance of detection data.*

*Proof.* The proof is similar to the simple random sampling and the basic idea is listed below. We first divide the full data set into stratums. We then conduct

the simple random sampling in each stratum and derive the corresponding sample mean $\bar{y_{st}}$ and variance $v(\bar{y_{st}})$. Based on them, we could then derive the sampling error $e_{st}$. After substituting same variables in Theorem 4.4.2, we derive the error limits of detection on the stratified sampling technique. □

We now show an example, in which the detection data is prioritized based on the severity level of security risks. The stratums of detection data will be numbered from 1 to $L$. Without loss of generality, we introduce a stratum weight $w_i = i^k$, where $i = 1, 2, \ldots, L$ and $k$ is the parameter to map the security severity level to sampling weight. Hence, the sampling weight for stratum $h$ is

$$P_h = \frac{n_h}{N_h} = \frac{w_i}{\sum_1^L w_i}. \tag{4.4.27}$$

Then, we have Theorem 4.4.4 listed below.

**Theorem 4.4.4.** *If the stratified sampling is the weight-based sampling, the sampling error can be denoted as,*

$$e_{st} = \frac{t\sigma_{\bar{y_{st}}}}{N}\sqrt{\sum_1^L N_h^2(\frac{1}{P_s \times P_h} - 1)}, \tag{4.4.28}$$

*where $f$ is the total sampling rate for extracting information from the full set of detection information.*

*Proof.* We have

$$P_h = \frac{n_h}{N_h} = \frac{w_i}{\sum_1^L w_i}, \tag{4.4.29}$$

and $\sigma_{\bar{y_{st}}}$ is the variance of sample as an estimate of variance of detection data. We have

$$n_h = P_h \times P_s \times N_h. \tag{4.4.30}$$

Then, the sampling error can be denoted as,

$$e_{st} = t\sqrt{\frac{1}{N^2}\sum_{h=1}^{L} N_h(N_h - n_h)\frac{\sigma^2_{\bar{y_{st}}}}{n_h}}, \tag{4.4.31}$$

$$= t\sqrt{\frac{1}{N^2}\sum_{h=1}^{L} N_h(N_h - P_h \times P_s \times N_h)\frac{\sigma^2_{\bar{y_{st}}}}{P_s N_h}}, \tag{4.4.32}$$

$$= t\sqrt{\sum_{h=1}^{L} N_h^2 \sigma^2_{\bar{y_{st}}}\frac{1 - P_h \times P_s}{P_h \times P_s \times N^2}}, \tag{4.4.33}$$

$$= \frac{t\sigma_{\bar{y_{st}}}}{N}\sqrt{\sum_{h=1}^{L} N_h^2(\frac{1}{P_h \times P_s} - 1)}. \tag{4.4.34}$$

□

From Theorem 4.4.2, Theorem 4.4.3 and Theorem 4.4.4, we observe that with the increase of sampling rate $f$, the sampling error declines and the detection error decreases as well. We derive the numerical data based on one simple example described below. We set the detection threshold $T = m + 2 \cdot v$ and let the confidence probability be $80\%(t = 1.28)$. According to Theorem 4.4.2, we derive detection errors for different sampling rates. To analyze the effectiveness of stratified sampling technique, we assume that there are five stratums and the stratum weight is $w_i = i^2$, where $i = 1, 2, \cdots, 5$. We substitute the above corresponding parameters in Theorem 4.4.3 and derive numerical results for error rates. The numerical data is shown in Figure 4.8. As we can see, the error rate declines when the sampling rate grows and the stratified sampling technique achieves a better detection accuracy than the simple random sampling technique.

### 4.4.4 Data Aggregation Techniques

We now define metric and conduct the theoretical analysis on the bandwidth reduction of our proposed data aggregation techniques. Using a simplified case, we show the aggregation performance in terms of the effectiveness of our data aggregation techniques to conduct the bandwidth reduction. The metric that we use is the data rate $T$, which is defined as the average rate of data transmission rate over the network. We assume that there is $N$ services nodes in MANET, which executes our proposed aggregation techniques. We introduce the aggregation ratio, $R_a$, to measure the efficiency of data aggregation and it is defined as the reduced data size vs. the size of input data. Our analysis is based on the simple scenario, in which the service nodes collect the detection data and use the aggregation techniques to consolidate the data. After that, the aggregated data is transmitted to the operation center.

Consider the following scenario: a batch of $K_t$ packets needs to be transmitted from the service node $t$ ($t = 1, \ldots, N$) to the operation center. Packets are transmitted through communication channels. We assume there are $L$ channels in total and each channel has a data rate of $R_i$ bits per second and the propagation delay of $d_i$ seconds ($i = 1, \ldots, L$). We assume that all packets have the same size with $S$ bits of payload and $H$ bits of protocol header. We assume that the bandwidth is large enough so that we ignore the query delay in the data rate computation and all packets from one

service node are transmitted through one communication channel. For the system without using data aggregation, the overall network throughput $T$ can be computed by $T = \sum_{t=1}^{N} \frac{K_t(S+H)}{\frac{K_t(S+H)}{R_i}+d_i}$, where $i$ is the index of communication channel selected for node $t$.

Then, we consider the use of data aggregation. The payload and size of protocol header are $(1-R_a)S$ and $(1-R_a)H$, respectively. The overall network throughput $T^{'}$ after using data aggregation become $T^{'} = \sum_{t=1}^{N} \frac{K_t(1-R_a)(S+H)}{\frac{K_t(1-R_a)(S+H)}{R_i}+d_i}$.

With the same network condition, it is easy to prove that $T^{'}$ is smaller than $T$. This means that in the system using data aggregation, the use of bandwidth is smaller than the system without the use of data aggregation. In addition, we can see from the above analysis, the aggregation ratio is the key factor for the bandwidth reduction. After multiplying $1/(1-R_a)$, we have $T^{'} = \sum_{t=1}^{N} \frac{K_t(S+H)}{\frac{K_t(S+H)}{R_i}+\frac{d_i}{1-R_a}}$.

In comparison with data rate for the system without data aggregation, because $R_a \in [0,1]$, we know that $T^{'}$ is smaller than $T$. With the increase of aggregation ratio, $T^{'}$ declines. From the above analysis, we can see that the data aggregation techniques can reduce the network bandwidth usage.

### 4.4.5 Impact of Intrusion Detection on MANET

We now show the impact of intrusion detection on MANET. Without loss of generality, we assume that MANET has limited bandwidth capacity $C$ and needs to support two

types of applications: the normal mission related application $S_n$ and the intrusion detection application $S_i$. We adopt the average end-to-end delay as a metric for normal mission-related application, and the detection accuracy and end-to-end delay as metrics for intrusion detection application to meet the design goals for our developed system. To optimally allocate the network resources, we assign a parameter $W$, which is defined as the ratio that the allocated bandwidth vs. requested bandwidth. In order to optimize the overall performance of MANET, we shall consider the following constraints: (i) the required total bandwidth should not be larger than the maximum capacity of network, (ii) the quality of service on intrusion detection application $S_i$ should meet requirements, and (iii) when the network is heavily loaded, the quality of normal mission related application $S_n$ should not be severely impacted by the intrusion detection application $S_i$.

We then formalize the impact of intrusion detection on MANET as follows:

$$\textbf{\textit{Objective.}} \quad Min \left\{ \frac{\sum_{i=1}^{N} \frac{P}{W_i B_i} + \sum_{s=1}^{K} \frac{P}{W_S B_S}}{N+K} \right\} \tag{4.4.35}$$

$$\textbf{\textit{S.t.}}$$

$$\begin{cases} \sum_{i=1}^{N} W_n B_n + \sum_{s=1}^{K} W_i B_i \leq C \\ \sum_{i=1}^{N} \frac{P}{W_i B_i N} \leq Q_{normal}^{max} \\ \sum_{S=1}^{K} \frac{P}{W_S B_S K} \leq Q_{IDS}^{max} \\ W_n, W_i \in [0,1] \end{cases}$$

where $P$ is packet size, $B_n, B_i$ denote requested bandwidth from the normal mission related application and intrusion detection application to monitor and detect attacks

in MANET, $N$ is the number of traffic flows associated with normal mission related application, $K$ is the number of traffic flows associated with monitoring and detecting attacks in MANET, $Q_{max}$ is the required maximum delay for each application (e.g., $Q^{max}_{normal}$ is the required maximum delay for the normal mission related application and $Q^{max}_{normal}$ is the required maximum delay for the application that conduct monitoring and detecting attacks). With Equation (4.4.35), we could derive the weight for each application and achieve the optimal bandwidth setting for MANET that shall support both the intrusion detection and normal mission related applications.

## 4.5 Performance Evaluation

In this section, we investigate the effectiveness of our proposed sampling techniques in a real-world testbed and evaluate their performance impact on MANET using ns-3 based simulation [83].

### 4.5.1 Methodology

We implemented the host-based intrusion detection system using virtual machines and deployed virtual machines acting as host monitoring agents and the operation center. We use the OSSEC as an example to validate our proposed schemes. Note that our developed sampling strategies and theoretical framework can be generally applied to other host-based and network-based intrusion detection systems as well.

We deployed the OSSEC agent on the host and the OSSEC server on the operation center. The detection data is collected by the OSSEC agent on the host and transmitted to the OSSEC server. During the data collection process, we simulate known attacks, including the port scanning, brute force password cracking attacks, and others.

We would like to point out that we implemented our sampling techniques based on simulated attacks, which can emulate real-world attack behaviors. To validate the effectiveness of our developed techniques, we simulated several attacks against hosts and generated events related to those attacks. In this way, we can generate a large amount of detection related data by adjusting the attack parameters such as scan rate and further study the effectiveness of sampling techniques and obtain the insightful relationship between detection accuracy and sampling rate. Note that there are many attacks are various and emerging endlessly. Defending the new attacks is always open research topic. In this chapter, we simulate and generate generic attacks, which contain the common malicious behaviors, to analyze the the effectiveness of sampling techniques. We also simulate the normal system operations, including the system patching and others, which may pose false positives for attack detection. After the detection data is collected by the OSSEC agent, we use such data as input to validate our proposed sampling and detection techniques.

The OSSEC agent records activities such as accessing and modifying system critical areas. However, some normal applications will exhibit similar behavior

(e.g., installing the new software). To obtain the false positive rate, we conducted experiments on the same testbed and collected the detection information as background data, in which no attack is in place. Based on the collected background data, we obtained the detection threshold for the statistical-based anomaly detection discussed in Section 4.3. To measure detection rate, we collected hundreds of data entries and each entry represents one attack event. Through the feature mapping algorithm described in Section 4.3, we converted each feature into a number that is used to present the attack severity. We then used the converted data as the input to validate the effectiveness of our developed simple random sampling and stratified sampling techniques.

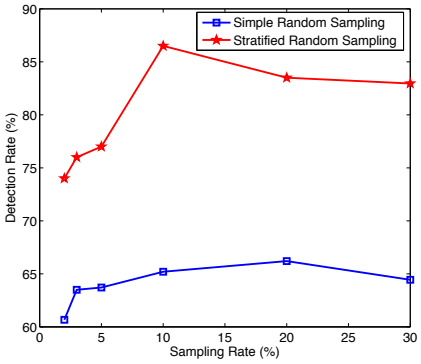


Figure 4.6: Detection Rate vs. Sampling Rate (Experiment)

To evaluate the impact of transmitting detection data on MANET, we implemented a simulation environment based on NS-3 [83], which is a well-known network simulation tool in the networking community. The evaluation environment is also

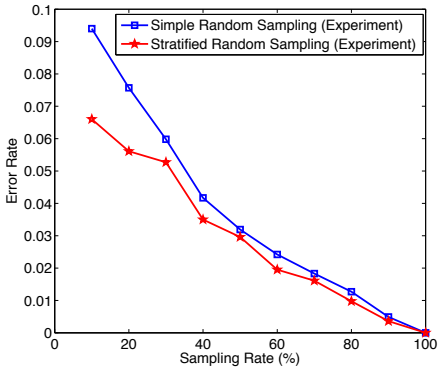


Figure 4.7: Error Rate vs. Sampling Rate (Experiment)

based on virtual machine that runs the version of Ubuntu Linux 11.10 with 2GB memory. In our simulation, we consider an outdoor environment with 50 mobile nodes and one operation center. Nodes in the network move according to a random Waypoint mobility model in an $800m \times 1000m$ rectangular field and their initial positions are randomly assigned. Each node moves from a random location to a random destination with a randomly assigned speed, which is uniformly distributed in the range of $0-20m/s$. The operation center is statically located at the lower left corner of the rectangular field and each simulation lasts for 20 seconds.

To simulate the traffic associated with the normal mission related applications on MANET, we randomly select $N$ source/destination pairs and the data rate is 56 Kb/s in NS-3. We monitor each traffic flow and measure the throughput and end-to-end delay. To validate the impact of intrusion detection on MANET, we consider a nearly saturated MANET. We introduce a parameter to emulate the sampling ratio, defined

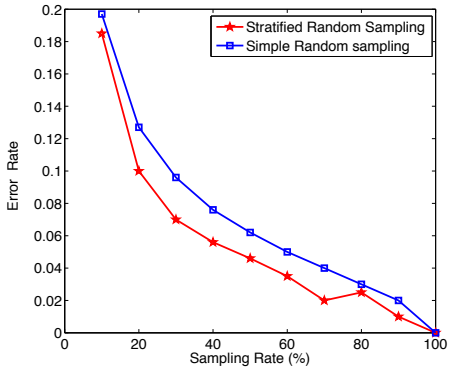


Figure 4.8: Error Rate vs. Sampling Rate (Theory)

as the ratio between the number of nodes running the normal mission related application over the total number of nodes. To simulate traffic for monitoring and detecting attacks, we randomly choose a number of nodes to send CBR (Constant Bit Rate) traffic to the operation center. We monitor the normal traffic flows and measure the performance when the amount of traffic associated with monitoring and detecting attacks increases.

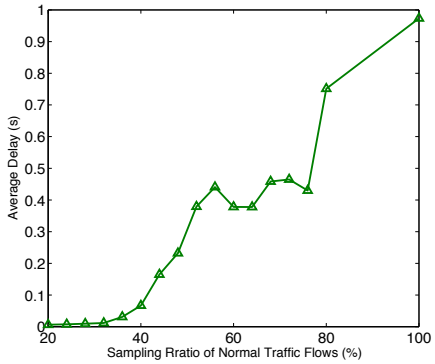


Figure 4.9: End-to-End Delay of Normal Mission Application without Intrusion Detection Application

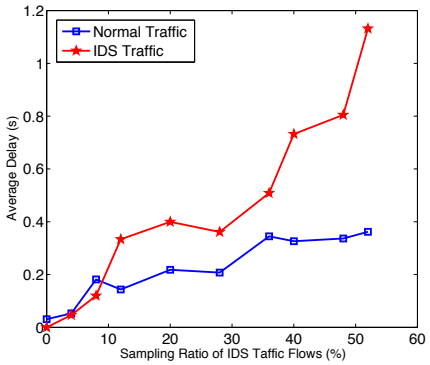


Figure 4.10: Sampling Ratio vs. End-to-End Delay of Normal Mission Application

We also implement the aggregation techniques and evaluate their impact on the MANET using ns-3. In our evaluation, we consider the following topology: an outdoor 802.11b (1Mbps) MANET with 60 mobile nodes and an operation center. These 60 mobile nodes are clustered into six groups. In each cluster, there is one data collection agent. Nodes configured with IDS application send IDS data to the data collection agent in their cluster and data collection agent send aggregated data to the operation center. Nodes in the network move according to a random Waypoint mobility model in a $600m \times 900m$ rectangular field and their initial positions are randomly assigned. Each node moves from a random location to another randomly selected destination with a randomly selected speed, uniformly distributed in $0-10m/s$. The remote operation center is installed in a static location at $(0,0)$ and our simulation lasts for 20 seconds.

We use two metrics for quantifying the detection accuracy: (i) *detection rate*

defined as the probability of correctly determining the attack, and (ii) *error rate* defined as the error limits of detection rate after the sampling technique is applied. In the stratified sampling, we select the sampling weight as $w_i = \frac{i^2}{\sum_{i=1}^{n} i^2}$. To compute the error rate, we apply the full set of detection dtat to the detection algorithm and obtain the detection rate. We then apply the sampled data with different sampling rates (e.g., $10\%, 20\%, \cdots, 90\%$) to the detection technique and obtain the corresponding detection rate. Assuming that the detection rate is $R$ when the sampling rate is 100% and the detection rate is $R'_k$ when the sampling rate is $k\%$. Then, the error rate is defined as,

$$E = |\frac{R'_k - R}{R}|. \tag{4.5.1}$$

### 4.5.2 Evaluation Results

Figure 4.6 illustrates the relationship between sampling rate and detection rate in terms of the two sampling techniques investigated in Sections III and IV. As we can see from the figure, the detection rate increases when the sampling rate grows. Obviously, the operation center obtains a better detection result with more detection information. In addition, the stratified sampling technique achieves a higher detection rate. This is because the stratified sampling technique gives a higher priority to the detection information with a higher security severity. Note that there is a fluctuate trend when the sampling rate is low, this is because the

sampling error is significant and unstable in a low sampling rate. As the sampling rate increases, the sampling error declines and the sample data becomes to stabilize.

Figure 4.7 illustrates the relationship between sampling rate and error rate for the two sampling techniques. As we can see from the figure, both sampling techniques work as expected with a relatively low error rate, the error rate declines as the sampling rate increases, and the stratified sampling achieves a better detection accuracy. In addition, the analytical results from Section 4.4 are shown in Figure 4.8 and match with our experimental results well. We evaluate the performance of MANET without traffic associated with monitoring and detecting attacks. From the results shown in Figure 4.9, we could see that the end-to-end delay increases as the sampling ratio grows. In other words, as more nodes are selected to transmit data associated with the normal mission related application, the end-to-end delay increases. In our experimental setting, we note that when the sampling ratio approaches above 55%, the end-to-end delay increases rapidly.

We also evaluate the performance of the fifteen normal traffic flows associated with the normal mission related application and different numbers of flows associated with monitoring and detecting attacks transmitted to the operation center. We randomly select nodes that conduct monitoring and detection. Figure 4.10 illustrates the relationship between the sampling ratio for traffic flows associated with the monitoring and detection and the end-to-end delay. As we can see, when the

sampling ratio of traffic flows associated with the monitoring and detection increases, the end-to-end delay of the traffic associated with normal mission related applications grows as well. We conclude that the traffic associated with monitoring and detecting attacks has a negative impact on the traffic flow associated with the normal mission related application. For example, when the sampling ratio increases over 32% of total traffic, the end-to-end delay of traffic associated with the normal mission related application increases 168.5% in comparison with the scenario, in which there is no traffic associated with monitoring and detection. When the amount of traffic flows associated with monitoring and detecting attacks approaches 40%, the end-to-end delay of traffic associated with the normal mission related application increases 440.9%. From this example, to reduce the bandwidth and limit the impact on MANET, the sampling is necessary when we transmit detection information over MANET.

## 4.6 Summary

To secure MANET, in this chapter we studied a host-based detection architecture and investigated the simple random sampling and stratified sampling techniques. We derived the closed formulae to study the impact of detection accuracy vs. sampling techniques along with parameters used in the sampling process. We investigated the impact of attack detection on the performance of MANET and formalized it

as an optimization problem for allocating network resources. We also discussed related issues, including the system architecture options, dynamic sampling, and data aggregation. We conducted experiments on our real-world test bed and our data showed that the stratified sampling technique achieves better performance than the simple random sampling technique in terms of providing tradeoff between detection accuracy and bandwidth cost reduction. The performance impact of transmitting the detection information on MANET was also simulated using ns-3.

# Chapter 5

# MapReduce Based Machine Learning Techniques for Processing Massive Network Threat Monitoring Data

## 5.1 Overview

Networking technology has greatly changed the way that our society functions as a whole, leading to a new era of e-business, social interaction, and virtual organizations. There is an omnipresent need for security and robust detection schemes to protect critical network infrastructures. Cyber-threats are significantly more dangerous than they have ever been and are growing in number and sophistication. Due to the widespread nature of cyber-threats (malware propagation, etc.), large-scale traffic monitoring across networks has become an essential part of effectively detecting and defending against contemporary cyber-attacks. Nonetheless, large-scale threat monitoring over distributed networks leads to extremely big data from monitored end-hosts and network devices [84].

Effectively processing of threat monitoring data from both end-hosts and network devices will better facilitate the detection of cyber-threats as well as help security administrators respond to cyber-threats in a timely manner. In our previous work, the development of effective threat monitoring systems to defend against cyber-attacks

was established [61, 71, 85, 86]. Nonetheless, big data poses serious challenges for cyber operations because an ever growing large and complex threat monitoring system from a large computer network needs to capture, store, manage, and process big data. With continuous, unbounded, rapid, and time-varying data streams generated by end-hosts and network devices, the complexity of storing and processing big network data will significantly increase. As such, there is an urgent need to develop efficient techniques to process and transform these complex, often vast unstructured, amounts of network threat monitoring data into manageable, useful, and exploitable information.

To address big cyber data, we consider a threat monitoring system with an objective of monitoring and processing the real-time data streams generated by threat monitoring agents, which monitors the statuses of end-hosts or networks and then detects suspicious activities. To ensure that the threat detection methods are efficient, MapReduce based machine learning (MML) schemes can efficiently deal with threat monitoring over big data. The main idea of the MML system is to speed up the machine learning (ML) process using cloud computing. The first step is to collect the characteristics of traffic flows (flow duration and average bytes per packet of the flow, average bytes per seconds of the flow, etc.). To accurately and rapidly detect traffic anomalies, two MapRduce based ML schemes are developed to profile the dynamic characteristics of traffic flows and then to detect anomalies

based on learned classifiers: Logistic Regression and Naïve Bayes. In the proposed MML schemes, the computational burden of the learning process is spread across multiple machines. The learned computational results from multiple machines are then integrated into one single learned classifier. Lastly, the learned classifier will then be used to recognize whether a new traffic flow is either normal or abnormal (benign or malicious).

Using real-world datasets consisting of both botnet and normal traffic, we develop a cloud computing test bed and conduct experiments to evaluate the effectiveness of the developed MML schemes in terms of learning accuracy, training set size, and training and detection processes overhead. The experimental data shows that the proposed MML schemes rapidly detect anomalous traffic flows with the same accuracy as standard machine learning schemes without using MapReduce.

Notice that the materials in this chapter are adapted from my previous publication [87].

## 5.2 Our Approach

In this section, we first introduce the design rationale of our approach and then present our MapReduce-based machine learning (MML) schemes in detail.

### 5.2.1 Design Rationale

To defend against cyber-attacks, anomaly-based intrusion detection systems have been widely developed. Anomaly-based detection refers to the issue of finding patterns in data that do not conform to an expected behavior [88]. In anomaly detection, the system administrator commonly defines the baseline (i.e., normal) measures to qualify normal system behavior (e.g., network traffic volumes). The threat monitoring and detection system monitors various system segments and compares their states to defined profiles. If the observed states are beyond the defined profiles, anomaly alerts will be issued.

Due to the widespread nature of threats such as malware propagation, a large-scale traffic monitoring system across networks has become essential. Such threat monitoring systems can lead to extremely large amounts of data collected from monitored end-hosts and network devices. In the realm of cyber security, big data refers to the management and analysis of large-scale information, which exceeds the capabilities of traditional data processing technology. With the continuous, unbounded, rapid, and time-varying data streams generated by end-hosts and network devices; the complexity to store and process big network data will significantly increase. Hence, the effective processing of threat monitoring data from both end-hosts and network devices will facilitate the detection of cyber-threats and help security administrators respond to cyber-threats in a timely manner. In this chapter, the key focus is on the

network based intrusion detection system (IDS), which analyzes network traffic in order to identify the presence of malicious traffic flows. Using traffic flows as an example to demonstrate our idea, we begin with the following collection of traffic flow characteristics: flow duration, average bytes per packet of the flow, average bytes per second of the flow, etc. To accurately detect traffic anomalies, we then implement the MML schemes to profile the characteristics of traffic flows and to detect traffic anomalies based on a learned classifier.

To make the threat detection capability efficient, how to make machine learning schemes to efficiently deal with big network data for threat monitoring is critical. The main idea is to speed up the machine learning process by using the cloud computing system. Our developed MML schemes will distribute the computational task of the learning process across multiple physical machines. The detection system consists of both the offline training and the online detection phases. In the offline training phase, we use a training set, where collected network traffic flows consist of both normal flows and attack flows. The subsets of the training set are then assigned to different computers to conduct the training process independently. The computational results of the learning phase from different computers are then integrated into one single learned classifier. In the online detection phase, the use of the learned classifier is then set in place to determine whether a traffic flow in question is either normal or malicious. The detail of the algorithm design and the

detection workflow will be introduced in the following sections.

### 5.2.2 Algorithms Design

Being one of the most dangerous network-based attacks, botnets can be massive. Coordinated groups of compromised hosts have the ability of conducting malicious activities such as spamming, DDoS (Distributed Denial-of-Service) attacks, etc. To accurately and rapidly learn the anomalous behavior of malicious traffic associated with botnets, we develop two MML schemes: Logistic Regression and Naive Bayes in a cloud computing environment.

***MapReduce Based Logistic Regression Machine Learning Scheme***

Logistic regression is a type of probabilistic statistical classification model, which is commonly used for binary classification problems [89, 90]. We implement the traditional logistic regression algorithm in the MapReduce framework to carry out the learning process in parallel. In our detection system, we first collect the characteristics of traffic flows. Each characteristic is considered as one feature of observed data and each feature has its own numerical value. The detailed feature definition and extraction process will be introduced in Section 5.2.3. Because each flow is either normal or malicious, we consider each flow as one observation which can be represented as $(X,Y)$, where $X = (x_1, x_2, \ldots, x_n)$ is the feature value vector and $Y \in (0,1)$ is the class value. For logistic regression machine learning, we take our

input features $x_i$, multiply each one by the regression coefficient $\theta = (\theta_0, \theta_1, \ldots, \theta_n)$, and then add them up as $z = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$. The result z will be put into the logistic (sigmoid) function [91]. In this way, we will obtain a number between 0 and 1. We then consider input observations $> 0.5$ as class 1 and input observations $\leq$ 0.5 as class 0. In this way, the logistic regression classifier is a probability estimate process. The detailed principle of the logistic regression algorithm can be found in [89].

During the training process, it is critical to determine the best regression coefficients. Suppose we train a dataset with m observations, $(X_1, y_1)$,$((X_2, y_2))$,$\ldots$,$(X_m, y_m)$ and each observation has $n$ features $X = (x_1, x_2, \ldots, x_n)$. We implement the gradient descent as an optimization method to find the best regression coefficients. Gradient descent can be used for most machine learning schemes to update parameters iteratively in order to minimize the cost function [92]. In logistic regression, the gradient descent rule is

$$\theta_i = \theta_i - \alpha \frac{\partial L(\theta)}{\partial \theta} = \frac{\alpha}{m} \Sigma_{i=1}^{m} (h(\theta^T X_i) - y_i) X_i, \tag{5.2.1}$$

where $\alpha$ is the arbitrary learning rate that determines the step size, $L(\theta)$ is the cost function, and h is the sigmoid function. With a training set input, the parameter $\theta$ will be iteratively updated based on 5.2.1.

It is worth noting that in the parameter updating process of gradient descent, all iterations need to visit all of the training samples for a given parameter. It is

challenging to develop an efficient mechanism to process and transform complex, large amounts of data into useful detection information. In an attempt to remediate this, we apply the MapReduce framework to distribute the computational task to multiple nodes in the cloud. Recall that MapReduce is a parallel programming model primarily designed for batch processing big data in a distributed computing environment. MapReduce is designed using the concept of divide-and-conquer and follows the master/slave computing paradigm. The master node receives the computational training task and sends the subset to the slave nodes to separately conduct different training processes. Eventually the computational results will be combined together as learned classifier. Figure 5.1 illustrates the main idea of this process.

Using the MapReduce framework, we implement gradient descent for the logistic regression learning scheme and update the parameters iteratively. Suppose that we have large-scale data, which contains one million observations (one million traffic flows in our performance evaluation), then we need to conduct the gradient descent as

$$\theta_i = \theta_i - \alpha \frac{\partial L(\theta)}{\partial \theta} = \theta_i - \frac{\alpha}{1000000} \Sigma_{i=1}^{1000000} (h(\theta^T X_i) - y_i) X_i, \tag{5.2.2}$$

Then, we use MapReduce to distribute the dataset to four different computers. Computer one uses subset $((X_1, y_1), (X_2, y_2), \ldots, (X_250000, y_250000))$ to carry out gradient

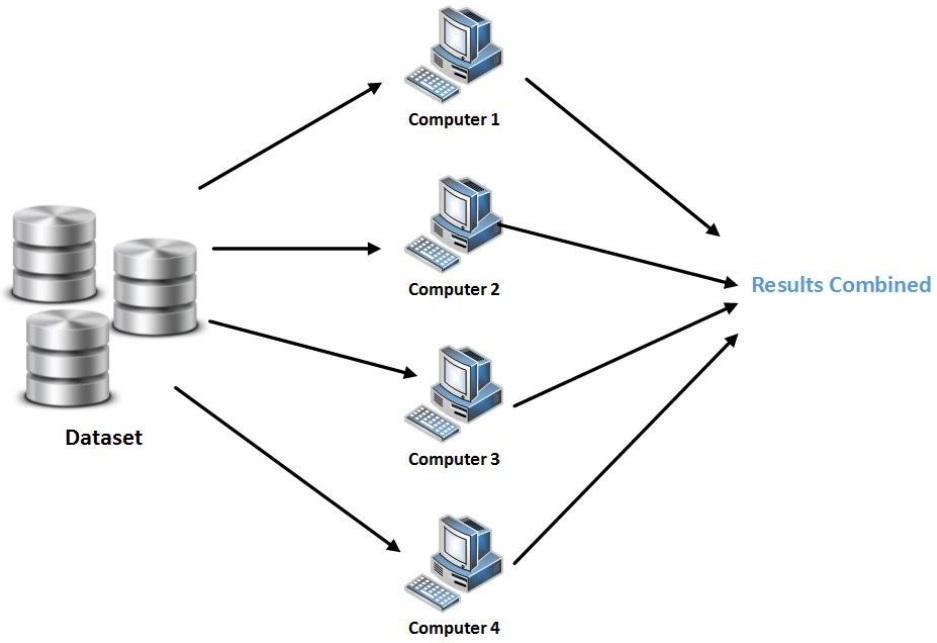


Figure 5.1: Machine Learning Based on MapReduce Framework.

descent. Similarly, computers 2, 3, and 4 also get their respective subsets. In doing so, we can derive the computational process as:

$$\begin{cases} ((X_1,y_1),(X_2,y_2),..,(X_{250000},y_{250000})) & tmp1 = \frac{\alpha}{250000}\Sigma_{i=1}^{250000}(h(\theta^T X_i)-y_i)X_i \\ ((X_1,y_1),(X_2,y_2),..,(X_{500000,y_500000})) & tmp1 = \frac{\alpha}{250000}\Sigma_{i=250000}^{500000}(h(\theta^T X_i)-y_i)X_i \\ ((X_1,y_1),(X_2,y_2),..,(X_{750000,y_750000})) & tmp1 = \frac{\alpha}{250000}\Sigma_{i=500001}^{750000}(h(\theta^T X_i)-y_i)X_i \\ ((X_1,y_1),(X_2,y_2),..,(X_{1000000},y_{1000000})) & tmp1 = \frac{\alpha}{250000}\Sigma_{i=750001}^{1000000}(h(\theta^T X_i)-y_i)X_i \end{cases}$$

$$\theta_i = \theta_i - \frac{\alpha}{100000}(tmp1+tmp2+tmp3+tmp4) \tag{5.2.3}$$

In our MapReduce based logistic regression scheme, each iteration has a map phase and a reduce phase. Figure 5.2 illustrates the MapReduce based framework for conducting this parallel machine learning process. As we can see, the MapReduce

framework is based on key/value tuples and relies on two built-in functions: a map function and a reduce function. Suppose that we have data samples $(A_1, A_2, \ldots, A_m)$ collected from threat monitors on end-hosts and network devices. Each data sample contains values of pre-defined features (e.g. system logs, source and destination addresses, and others). In the Map function, the map workers visit the training samples in parallel and perform key matching to list associated key/value pairs. Visiting one training sample generates n key/value pairs. The keys are 1 to n and the values in the gradient descent algorithm are $(h_\theta(x^{(}(i))) - y^{(}(i)))x_j^{(}(i))$. Then, we can obtain intermediate results such as $K_1 \rightarrow a_{11}^{(}(1)), a_{12}^{(}(1)), \ldots, a_{1m}^{(}(1))$ , where $a_{1m}^{(}(1))$ represents the value of key $K_1$ from sample m computed by map worker 1. In the reduce phase, the values of the same key are added up to yield $\Sigma_{(}i = 1)^m((h_\theta(x^{(}(i))) - y^{(}(i)))x_j^{(}(i))$ and the parameters are updated by $\theta_j := \theta_j - \frac{\alpha}{m}((h_\theta(x^{(}(i))) - y^{(}(i)))x_j^{(}(i))$. Then, the aforementioned process will be further validated in our developed cloud computing test bed until convergence of the subtasks, where the learned model is ready for use.

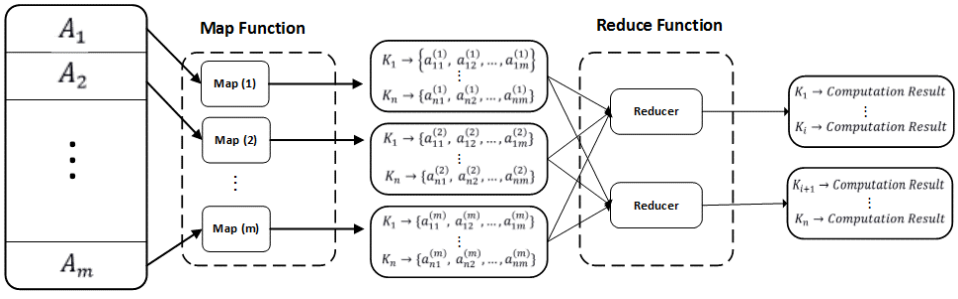


Figure 5.2: MapReduce Based Framework for Parallel Machine Learning.

We use real world data to evaluate the performance of our parallel machine learning scheme. Based on the cost function of logistic regression (LR) machine learning, we collected the cost of gradient descent in the MapReduce process shown in Figure 5.3. As we can see from Figure 5.3, the computational cost of gradient descent declines as the number of iterations increases. This demonstrates that our LR MML scheme can speed up the learning process. In addition, we compared the cost of different arbitrary learning rates $\alpha$. We can see that the higher the arbitrary learning rate, the cost declines at a higher speed. Hence, we conclude that with a big arbitrary learning rate, the gradient descent algorithm can quickly locate the optimal point in a function.

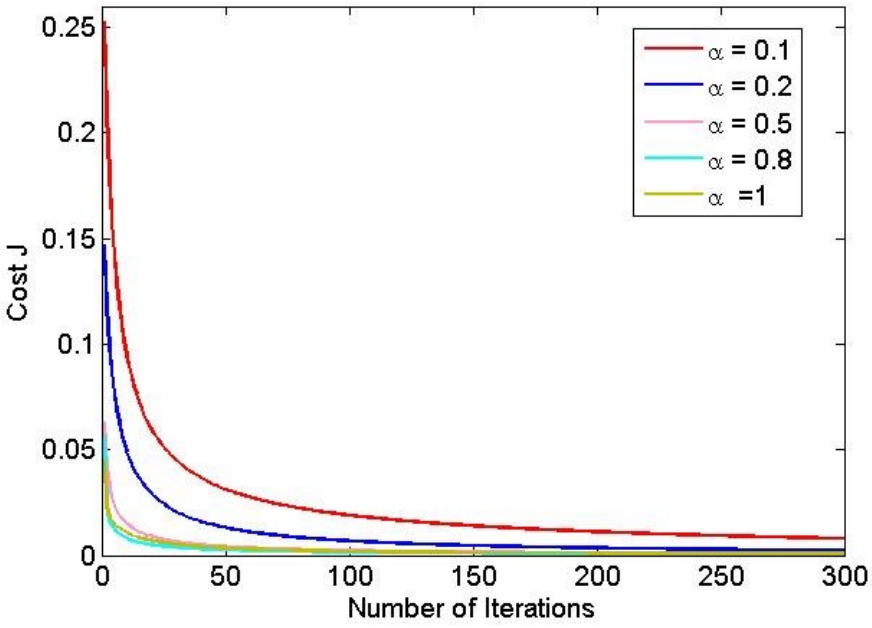


Figure 5.3: Cost of Gradient Descent using MapReduce.

Next, we explore another common MML technique of naïve Bayes.

***MapReduce Based Naïve Bayes Machine Learning Scheme***

The other machine learning algorithm we implement is the Naïve Bayes classifier. Generally speaking, the Naïve Bayes classifier is a probabilistic classifier based on applying Bayes' theorem [93]. The Naïve Bayes can construct a classifier given a set of training data with class labels. In our system, the training data is the observation of features determined by features associated with traffic flows and the class falls into two categories: 1 (a positive one when the monitored data is normal) and 0 (a negative one when the monitored data is malicious). Denote $X = (x_1, x_2, \ldots, x_n)$ to be one observation of E, where $x_n$ is the value of feature n. Then, the probability of observation E in this category c is $P(c|x_1, x_2, \ldots, x_n)$. Using Bayes theorem, we have

$$P(c|x_1, x_2, \ldots, x_n) = \frac{P(c|x_1, x_2, \ldots, x_n)}{P(x_1, x_2, \ldots, x_n)} \tag{5.2.4}$$

Then, we can define two types of parameters for the Naïve Bayes model: $q(c)$ for $c \in 1, 0$, where $P(c) = q(c)$ and $q_i(x_i|c)$ with $P(x_i|c) = q_i(x_i|c)$. Then, we have the Naïve Bayes model as

$$P(c|x_1, x_2, \ldots, x_n) = q(c)\Pi_{i=1}^{n} q_i(x_i|c) \tag{5.2.5}$$

The Naïve Bayes model learns the probability of classes when each feature is given. The Naïve Bayes model analyzes all features and combines them to form

the probability of each class. After that, whenever a new observation appears, the classifier then distinguishes the class according to the value of each feature. The detailed principle of the Naïve Bayes model can be found in [93].

Similar to logistic regression, we can use the maximize likelihood estimate (MLE) method to learn the optimal parameters. Suppose we are given a training set that contains m observations $(X_1, y_1), (X_2, y_2), \ldots, (X_m, y_m)$ and each observation has n features of $X = (x_1, x_2, \ldots, x_n)$. Based on the log-likelihood function [36] of the Naïve Bayes model, we then apply gradient descent to maximize the likelihood. The rule is similar to logistic regression as:

$$\theta_i = \theta_i - \frac{\alpha}{m} = \frac{\alpha}{m} \Sigma_{i=1} 6m(q(c) + q_i(X_i|c_i)) \tag{5.2.6}$$

where $q(c) = \frac{\Sigma_{i=1}^{m}[c_i=c]}{n} = \frac{count(c)}{n}$, $q_i(X_i|c) = \frac{\Sigma_{i=1}^{m}[c_i=c \& X=X_i]}{\Sigma_{i=1}^{m}} = \frac{count(X_i|c_i)}{count(c)}$. Hence, in the Naïve Bayes algorithm, we need to conduct only one iteration to optimize the parameter $\theta_i$. We also apply the MapReduce framework to improve the learning efficiency of Naïve Bayes to train the data set in parallel. The mechanism is similar to the logistic regression learning scheme. Each computer receives a subset of the training data and then computes $q(c)$ and $q_i(X_i|c)$. In the following subsection, we introduce the procedures to implement the parallel machine learning scheme in the MapReduce framework.

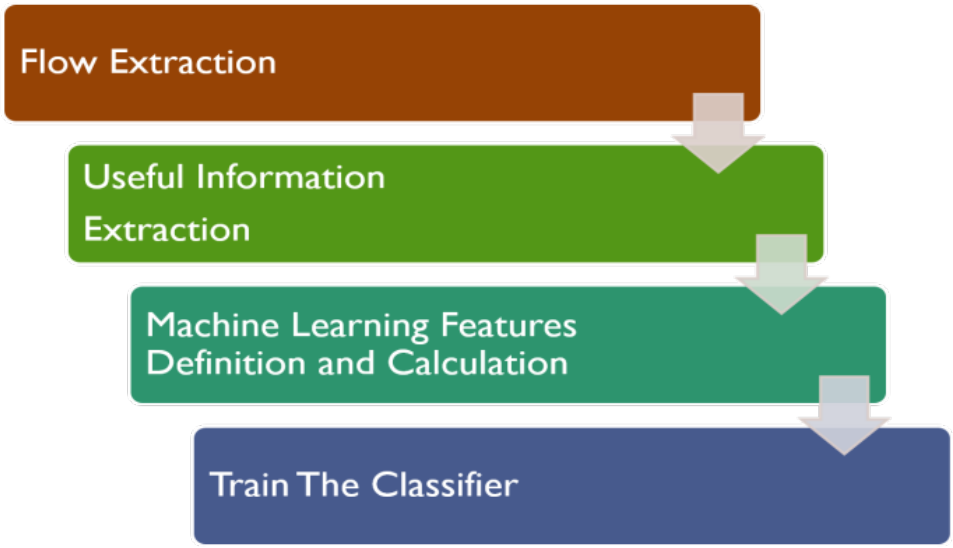


Figure 5.4: Detection Work flow

### 5.2.3 Implementation

Figure 5.4 illustrates the workflow of the offline training process. In this process, we first extract and select the useful information from traffic flows as samples. Then, we define the features for machine learning and give the features a value for each sample. Lastly, we conduct the training process for the classifier.

The detailed steps are illustrated as follows:

*Step 1: Flow Extraction:* To conduct network traffic monitoring, we collect a large volume of traffic data. Recall that in common practice, data will be stored in PCAP (packet capture) format and a large number of flows will be stored in one PCAP file. Therefore, we need to extract traffic flows from each PCAP file and separate each traffic flow into a single file. Figure 5.5 illustrates what was extracted

from 1,045,225 traffic flows to form a large dataset.

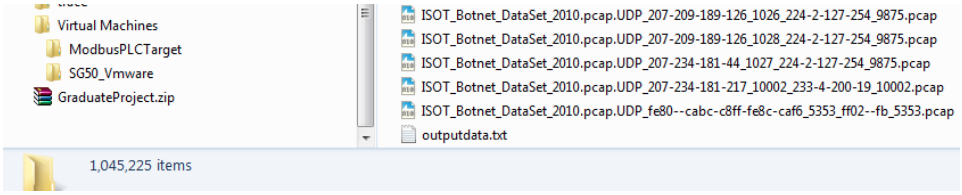


Figure 5.5: Flow Extraction.

*Step 2: Useful Information Selection:* The trace file of each traffic flow consists of massive packets. As shown in Figure 5.6, in each packet, there is plenty of data recorded and most of this data is useless and redundant. In this step, we propose to select the most useful information, which will be used for our machine learning process. The tool TSHARK (https://www.wireshark.org/docs/man-pages/tshark.html) is used to extract and select the useful information from each flow in the sample. Figure 5.7 shows an example flow output file through this step, where each row represents one packet in this flow. Note that each column contains one characteristic of packets, where the selected characteristics include: the time, the size of the packet (bytes), the type of network, and the destination MAC (media access control) address. For a destination MAC address, when the MAC address is of the format aa:aa:aa:aa:aa:aa, it can be regarded as an instance of malicious traffic. We can see that only small amounts of data in packets are selected so that the size of the input data is then placed into the machine learning algorithms, which are used to largely reduce the data size and to help speed up the data processing.

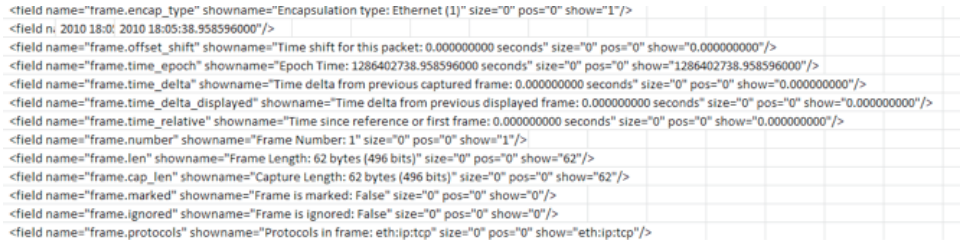


Figure 5.6: The Information of a Data Packet.

```
Oct  6, 2010 18:23:57.992603000 62      65535   Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:23:57.992834000 62      5840    Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:23:57.993173000 60      65535   Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.038100000 103     5840    Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.052309000 105     65486   Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.052466000 54      5840    Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.052566000 76      5840    Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.254036000 60      65464   Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.556363000 68      65464   Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.556633000 68      5840    Ethernet II, Src: aa:aa:aa:aa:aa:aa
Oct  6, 2010 18:24:08.757903000 60      65450   Ethernet II, Src: aa:aa:aa:aa:aa:aa
```

Figure 5.7: An Example of Data Selection Output.

*Step 3: Machine Learning Features Definition and Computation:* Based on the output data of Step 2, we define and compute the features for the machine learning process for each sample. In our study, we consider eight features during the training stage, which are shown in Table 5.1:

Figure 5.8 shows a screenshot of the computed input data for the logistic regression

| | |
|---|---|
| duration | defined as the total time duration of the flow |
| bpp | defined as average bytes per packet of the flow |
| bps | defined as the average bytes per second of the flow |
| pps | defined as the average packets per second of the flow |
| VarIAT | defined as the variance of packet inter-arrival time of the flow |
| VarBpp | defined as the variance of bytes-per-packet of the flow |
| tws | defined as the average TCP window size of the flow |
| VarTws | defined as the variance of TCP window size of the flow |

Table 5.1: Description of Extracted Features.

classifier. In this screenshot, each row represents one observation and each column stands for each computed feature value. For example, in the first observation, the value of Average Bytes per Packet is 90.1429 and the value of Average Bytes per Second is 194.15. In addition, the last column represents the class of data, either 0 or 1, meaning that the observation is either non-malicious or malicious traffic, respectively. Similarly, we can compute feature values as input for Naïve Bayes, which is shown in Figure 5.9.

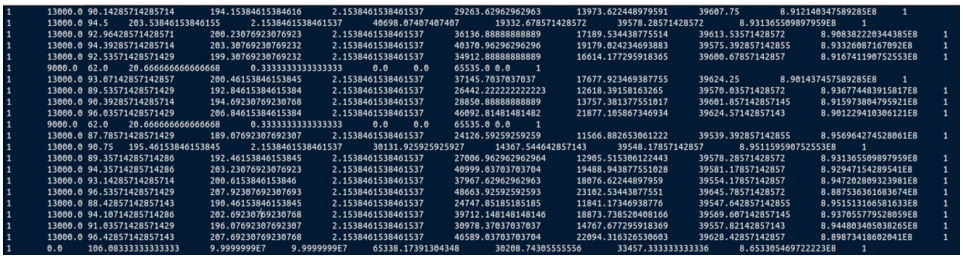

Figure 5.8: Screenshot of Feature Values Computing for Logistic Regression.

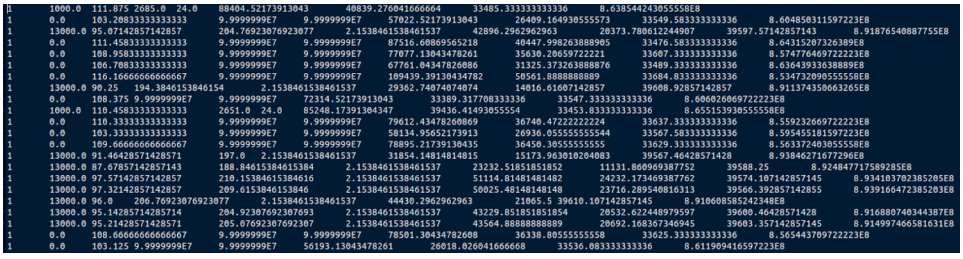

Figure 5.9: Screenshot of Feature Values Computing for Bayes Machine Learning.

*Step 4: Train the Classifier:* After Step 3, we have the input data ready for the machine learning schemes. We input the data to the HDFS (Hadoop File System) using the command: “sudo -u hdfs hadoop fs -put train.txt /inputs/”, which means

that the input data is train.txt and we store it in the "/inputs" folder on the HDFS (Hadoop Distributed File System). We train the input data with our machine learning schemes with the command: "sudo -u hdfs hadoop jar BigData-1.0-SNAPSHOT.jar bigdata.towson.edu.BigDataCC/inputs/train.txt/output", where our executable java program is called BigData-1.0-SNAPSHOT.jar. Then, the trained model will be generated under the "/output" folder on the HDFS, which can be used for the offline detection process. For the online detection process, similar to the offline training process, we use our data preprocessing program to obtain the input file and then use the classification program to classify the input test data based on the trained model.

## 5.3 Performance Evaluation

To validate the effectiveness of our proposed approach, we developed a cloud test bed and used it to conduct experiments. In the following, we first present the evaluation methodology and then show the experimental results.

### 5.3.1 Evaluation Methodology

As shown in Figure 5.10, we built a cloud computing testbed with one master node and three slave nodes. Each slave node can act as both a map node and a reduce node. Each node is a DELL Optiplex 9010 computer with Intel Core i7 3.40GHZ 8

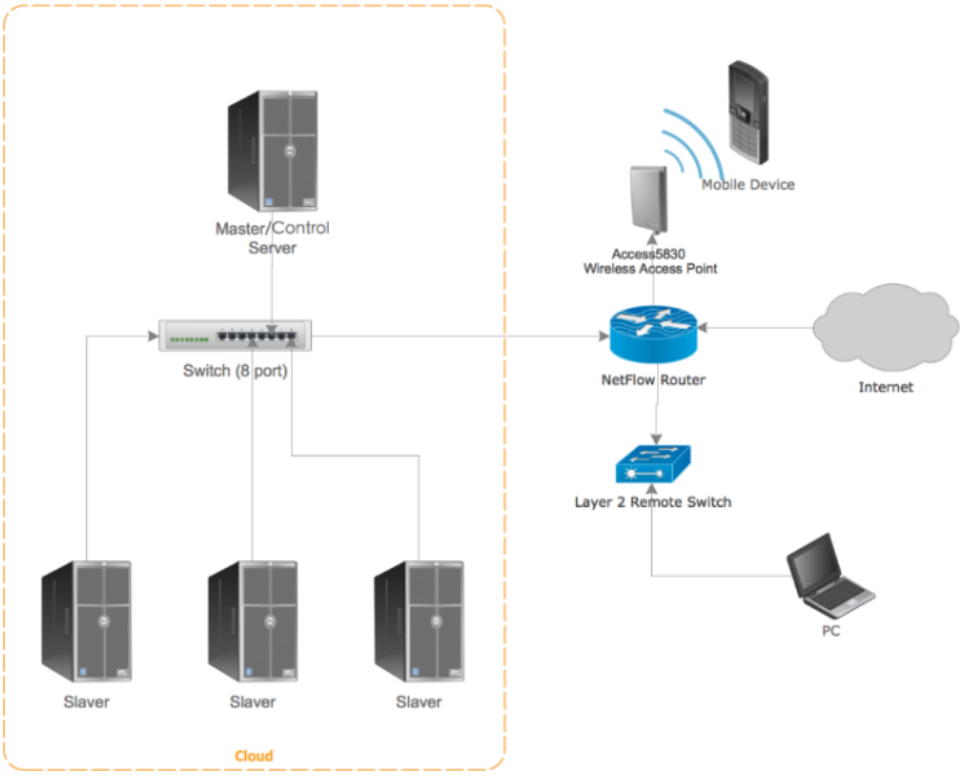


Figure 5.10: A Cloud Computing Testbed.

processors with 16GB RAM and 2TB hard drive. We use the Cloudera Manager as the central interface to perform management tasks such as configurations, management, and monitoring of the designed system. We downloaded cloudera-manager-installer.bin from the Cloudera website. The executable permission was configured by using the command “chmod u+x cloudera-manager-installer.bin”. Then, the installer can be executed with the command “sudo ./cloudera-manager- installer.bin” to install the Cloudera Manager. To install the Cloudera Manager on hosts in the cloud, the Cloudera Manager Admin Console is used to install and configure CDH (Cloudera

Distribution including Apache Hadoop).

To evaluate the effectiveness of our developed system, we obtained the ISOT dataset [94] from http://www.uvic.ca/engineering/ece/isot/datasets to conduct our experiments. The ISOT dataset is the combination of several publicly available malicious and non-malicious datasets. Specifically, the ISOT dataset contains 1,675,424 total traffic flows, which consists of 55,904 (3.33%) malicious traffic flows and 1,619,520 (96.66%) non-malicious traffic flows.

For the accuracy of detection, we expect that with a number of training samples, the machine learning schemes will have more information for the training process, leading to higher detection accuracy. To validate this hypothesis, we define the following evaluation metrics: *(i) Detection Rate:* It is defined as the probability of correctly classifying the malicious traffic flows and is the ratio of the number of malicious traffic flows correctly classified versus the total number of malicious traffic flows. *(ii) False Positive Rate:* It is defined as the probability of falsely classifying non-malicious traffic flows and is the ratio of the number of non-malicious traffic flows falsely classified as malicious traffic flows versus the total number of non-malicious traffic flows. In our experiments, we show the correlation between these metrics and the number of traffic flow samples used for the training process.

| Training Samples Numbers | Detection Rate | False Positive Rate |
|---|---|---|
| 20 | 78.19% | 28.68% |
| 100 | 99.31% | 0.18% |
| 140 | 99.30% | 0.69% |
| 200 | 99.19% | 0.81% |
| 1000 | 99.92% | 0 |

Table 5.2: Detection Accuracy of the Naïve Bayes Machine Learning Scheme.

### 5.3.2 Evaluation Results

We select 20,000 test samples (10,000 malicious traffic flows and 10,000 non-malicious traffic flows) from the dataset, which are not used in the training process. In terms of the training samples, in each group of samples, the number of malicious flows and the number of non-malicious flows are equal. For example, when we use 100 samples for training, 50 are malicious traffic flows and 50 are non-malicious traffic flows.

Table 5.2 illustrates the relationship between the detection accuracy of the Naïve Bayes machine learning scheme and the number of samples used for the training process. As we can see, the Naïve Bayes machine learning scheme can achieve a high detection accuracy. The detection rate from using 20 samples is 78.91%. Starting from 100 samples, the detection rate of each scenario is close to 100%, meaning that all malicious traffic flows can be accurately identified. In addition, the false positive rate is near 0, meaning that very few numbers of non-malicious flows

| Training Samples Numbers | Detection Rate | False Positive Rate |
|---|---|---|
| 20 | 99.48% | 0.14% |
| 100 | 99.55% | 0.12% |
| 140 | 99.99% | 0.02% |
| 200 | 99.48% | 0.14% |
| 1000 | 1% | 0 |

Table 5.3: Detection Results of Logistic Regression Scheme.

are falsely classified as malicious flows. Table 5.3 shows the detection accuracy of the logistic regression machine learning scheme. In comparison with the Naïve Bayes machine learning scheme, when there are 20 training samples, the logistic regression machine learning scheme can achieve a high detection accuracy near 99% and a low false positive rate near 0%.

In addition to the detection accuracy, the time efficiency of the MML schemes are also measured. The metric we used is the training time, which is defined as the time taken for the training process. Figure 5.11 illustrates the relationship between the training time of the Naïve Bayes machine learning scheme and the number of slave nodes used in the training process. As we can see, the training duration declines when the number of nodes increases. When we use 3 nodes to carry out the training process, it takes only 10.35 minutes to complete the training process. If we only use 1 node, it will take 30.58 minutes. Similarly, in the logistic regression based machine learning scheme, the training time can be significantly reduced when more slave nodes are used (Figure 5.12).

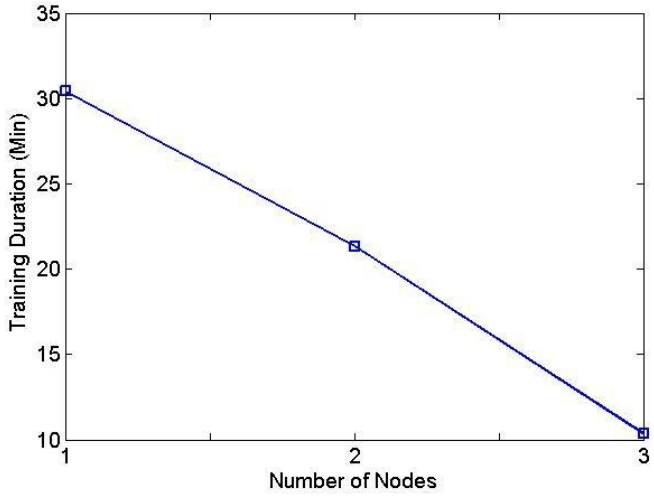


Figure 5.11: Time Cost versus Number of Nodes for Naïve Bayes

## 5.4 Summary

In this chapter, we addressed the issue of detecting malicious traffic flows from a large scale of monitored traffic data. To make threat detection efficient, we developed Logistic Regression and Naive Bayes MapReduce based machine learning schemes to deal with a large amount of threat monitoring data. The main contribution is to speed up the machine learning process using the MapReduce framework in a cloud computing environment. We demonstrated our MapReduce based machine learning schemes in cloud computing environment with a variety of existing structures of data flows, detection techniques, and implementation protocols. Using a real-world traffic dataset consisting of both botnet traffic and normal traffic, we conducted experiments and evaluated the effectiveness of our developed MapReduce based

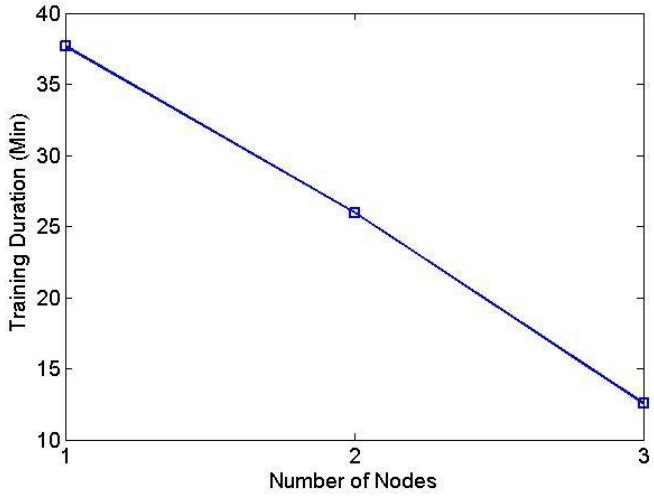


Figure 5.12: Time Cost versus Number of Nodes for Logistic Regression

machine learning schemes. Our results verify the accuracy and efficiency of our proposed MML detection schemes to detect malicious traffic.

# Chapter 6

# Final Remarks

In this dissertation, we developed schemes to enable efficient threat detection in mobile networks. To be specific, we first developed a machine learning-based scheme that can dynamically learn the behavior of malware on mobile devices and augment the human cognition process of defending against malware attacks. We then developed sampling and aggregation techniques with proper settings to reduce the bandwidth use. We also developed MapReduce-based Machine Learning (MML) schemes to rapidly and accurately process and detect malicious traffic in a cloud environment. In the near future and beyond, research on other types of wireless networks can be carried out, including intelligent transportation systems and wireless networks in a three dimensional environment.

# Bibliography


[1] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of ACM Conference on Computer and Communications Security*, 2002, pp. 255–264.

[2] P. de Boer and M. Pels, "Host-based intrusion detection systems," 2010.

[3] Ossec. [Online]. Available: http://www.ossec.net/

[4] B. Uphoff and J. S. Wong, "An agent-based framework for intrusion detection alert verification and event correlation," *International Journal of Security and Networks*, vol. 3, pp. 193–200, 2008.

[5] D. Djenouri, L. Khelladi, and N. Badache, "A survey of security issues in mobile ad hoc and sensor networks," *IEEE Communications Surveys and Tutorials*, vol. 7, pp. 2–28, 2005.

[6] P. Joshi, "Security issues in routing protocols in MANETs at network layer," *Procedia Computer Science*, vol. 3, pp. 954–960, 2011.

[7] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, "Security in mobile ad hoc networks: challenges and solutions," *IEEE Wireless Communications*, vol. 11, pp. 38–47, 2004.

[8] M. Taghiloo, J. Taghiloo, and M. Dehghan, "A SURVEY OF SECURE ADDRESS AUTO-CONFIGURATION IN MANET," 2008.

[9] R. Sheikh, M. S. Chande, and D. K. Mishra, "Security issues in MANET: A review," in *Proceedings of IFIP International Conference on Wireless and Optical Communications Networks*, 2010.

[10] M. Taghiloo, M. Tajamolian, M. Dehghan, and R. Mousavi, "Virtual address space mapping for IP auto-configuration in MANET with security capability," pp. 1–7, 2008.

[11] M. M. Yasin and A. A. Awan, "A study of host-based IDS using system calls," *Procedia Engineering*, 2004.

[12] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '08, 2008.

[13] A. S. Shamili, C. Bauckhage, and T. Alpcan, "Malware detection on mobile devices using distributed machine learning," in *Proceedings of the 2010 20th International Conference on Pattern Recognition*, ser. ICPR '10, 2010.

[14] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," *Mob. Inf. Syst.*, vol. 4, no. 1, pp. 33–49, Jan. 2008.

[15] A. Shabtai, "Malware detection on mobile devices," in *Proceedings of 2010 Eleventh International Conference on Mobile Data Management (MDM)*, May 2010.

[16] A.-D. Schmidt, R. Bye, H.-G. Schmidt, J. Clausen, O. Kiraz, K. Yuksel, S. Camtepe, and S. Albayrak, "Static analysis of executables for collaborative malware detection on android," in *Proceedings of IEEE International Conference on Communications (ICC)*, June 2009.

[17] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008.

[18] M.-Y. Su and W.-C. Chang, "Permission-based malware detection mechanisms for smart phones," in *Proceedings of 2014 International Conference on Information Networking (ICOIN)*, Feb 2014.

[19] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for android malware," in *Advances in Intelligent Systems and Applications - Volume 2*, ser. Smart Innovation, Systems and Technologies, J.-S. Pan, C.-N. Yang, and C.-C. Lin, Eds. Springer Berlin Heidelberg, 2013, vol. 21, pp. 111–120.

[20] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10, 2010, pp. 73–84.

[21] J. Cannady, "Artificial neural networks for misuse detection," in *Proceedings of NATIONAL INFORMATION SYSTEMS SECURITY CONFERENCE*, 1998, pp. 443–456.

[22] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of the 2002 International Joint Conference on Neural Networks*, 2002.

[23] O. Linda, T. Vollmer, and M. Manic, "Neural network based intrusion detection system for critical infrastructures," in *Proceedings of International Joint Conference on Neural Networks*, June 2009.

[24] V. Golovko, S. Bezobrazov, P. Kachurka, and L. Vaitsekhovich, "Neural network and artificial immune systems for malware and network intrusion detection," in *Advances in Machine Learning II*, ser. Studies in Computational Intelligence, J. Koronacki, Z. Raś, S. Wierzchoń, and J. Kacprzyk, Eds. Springer Berlin Heidelberg, 2010, vol. 263, pp. 485–513.

[25] S. Marchal, X. Jiang, R. State, and T. Engel, "A big data architecture for large scale security monitoring," in *Proceedings of 2014 IEEE International Congress on Big Data (BigData Congress)*, June 2014.

[26] K. Yoshida, *Sampling-Based Stream Mining for Network Risk Management*, 2006.

[27] S. Goldberg and J. Rexford, "Security vulnerabilities and solutions for packet sampling," in *Proceedings of IEEE Sarnoff Symposium*, 2007.

[28] K. Bartos, M. Rehak, and V. Krmicek, "Optimizing flow sampling for network anomaly detection," in *Proceedings of International Conference on Wireless Communications and Mobile Computing*, 2011.

[29] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06, 2006, pp. 165–176.

[30] D. Ficara, G. Antichi, A. D. Pietro, S. Giordano, G. Procissi, and F. Vitucci, "Sampling techniques to accelerate pattern matching in network intrusion detection systems." pp. 1–5, 2010.

[31] W. J. Scheirer and M. C. Chuah, "Syntax vs. semantics: competing approaches to dynamic network intrusion detection," *International Journal of Security and Networks*, vol. 3, pp. 24–35, 2008.

[32] S. Lai and B. Ravindran, "Achieving Max-Min lifetime and fairness with rate allocation for data aggregation in sensor networks," *Ad Hoc Networks*, vol. 9, pp. 821–834, 2011.

[33] D. Q. Goldin, "Faster In-Network Evaluation of Spatial Aggregationin Sensor Networks," in *Proceedings of International Conference on Data Engineering*, 2006.

[34] N. Shrivastava, C. Buragohain, and D. Agrawal, "Medians and beyond: New aggregation techniques for sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 2004.

[35] J. Gao, L. J. Guibas, N. Milosavljevic, and J. Hershberger, "Sparse data aggregation in sensor networks," in *Proceedings of Information Processing in Sensor Networks*, 2007.

[36] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Mobile Computing and Networking*, 1999, pp. 174–185.

[37] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004.

[38] J. Lin and C. Dyer, *Data-intensive Text Processing with MapReduce*, ser. G - Reference, Information and Interdisciplinary Subjects Series. Morgan & Claypool, 2010.

[39] J. T. Morken, "Distributed netflow processing using the map-reduce model," *PHD Thesis, Norwegian University of Science and Technology*, 2010.

[40] M. Ebrahimi, "Solving linear programs in mapreduce," *Master Thesis, Universität des Saarlandes*, 2011.

[41] D. Alves, P. Bizarro, and P. Marques, "Flood: Elastic streaming mapreduce," in *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, 2010.

[42] C. Doulkeridis and K. Norvag, "On saying enough already! in mapreduce," in *Proceedings of the ACM 1st International Workshop on Cloud Intelligence*, 2012.

[43] F. Halim, R. H. Yap, and Y. Wu, "A mapreduce-based maximum-flow algorithm for large small-world network graphs," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011.

[44] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," *Cloud Computing*, pp. 115–131, 2010.

[45] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *Proceedings of 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, 2010.

[46] E. E. Marinelli, "Hyrax: cloud computing on mobile devices using mapreduce," DTIC Document, Tech. Rep., 2009.

[47] H. Shivhare, N. Mishra, and S. Sharma, "Cloud computing and big data," in *Proceedings of 2013 International Conference on Cloud, Big Data and Trust*, 2013.

[48] Z. Chen, F. Han, J. Cao, X. Jiang, and S. Chen, "Cloud computing-based forensic analysis for collaborative network security management system," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 40–50, 2013.

[49] H. Liu and D. Orban, "Cloud mapreduce: a mapreduce implementation on top of a cloud operating system," in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2011.

[50] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan, "Kahuna: Problem diagnosis for mapreduce-based cloud computing environments," in *Proceedings of 2010 IEEE Network Operations and Management Symposium (NOMS)*, 2010.

[51] J. Zhang, D. Xiang, T. Li, and Y. Pan, "M2m: A simple matlab-to-mapreduce translator for cloud computing," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 1–9, 2013.

[52] W.-T. Tsai, X. Sun, and J. Balasooriya, "Service-oriented cloud computing architecture," in *Proceedings of 2010 Seventh International Conference on Information Technology: New Generations (ITNG)*. IEEE, 2010, pp. 684–689.

[53] D. Leaf, "Overview: Nist cloud computing efforts, nist senior executive for cloud computing," 2010.

[54] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: a software platform for .net-based cloud computing," *High Speed and Large Scale Scientific Computing*, vol. 18, pp. 267–295, 2009.

[55] Y. Huang, H. Su, W. Sun, J. M. Zhang, C. J. Guo, J. M. Xu, Z. B. Jiang, S. X. Yang, and J. Zhu, "Framework for building a low-cost, scalable, and secured platform for web-delivered business services," *IBM Journal of Research and Development*, vol. 54, no. 6, pp. 4–1, 2010.

[56] N. Nurain, H. Sarwar, M. Sajjad, and M. Mostakim, "An in-depth study of map reduce in cloud environment," in *Proceedings of 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, Nov 2012.

[57] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, Jan. 2008.

[58] H. Pieterse and M. Olivier, "Android botnets on the rise: Trends and characteristics," in *Information Security for South Africa (ISSA), 2012*, Aug 2012, pp. 1–5.

[59] K. Mallinson, "Smartphone revolution: Technology patenting and licensing fosters innovation, market entry, and exceptional growth." *IEEE tranactions on Consumer Electronics Magazine*, vol. 4, no. 2, pp. 60–66, April 2015.

[60] G. X. Wei Yu, Linqiang Ge and X. Fu, "Towards neural network based malware detection on android mobile devices," *Springer Book Series: Cybersecurity Systems for Human Cognition Augmentation*, 2014.

[61] Y. Wei, H. Zhang, L. Ge, and R. Hardy, "On behavior-based detection of malware on android platform," in *Proceedings of 2013 IEEE Global Communications Conference (GLOBECOM)*, Dec 2013.

[62] A. Nere, A. Hashmi, M. Lipasti, and G. Tononi, "Bridging the semantic gap: Emulating biological neuronal behaviors with simple digital neurons," in *Preceedings of 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA2013)*, Feb 2013.

[63] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'89, 1989.

[64] X. Yu, M. Efe, and O. Kaynak, "A general backpropagation algorithm for feedforward neural networks learning," *IEEE Transactions on Neural Network*, vol. 13, no. 1, pp. 251–254, Jan 2002.

[65] G. Arulampalam and A. Bouzerdoum, "A generalized feedforward neural network architecture for classification and regression," *Neural Networks*, vol. 16, no. 5–6, pp. 561 – 568, 2003, advances in Neural Networks Research: {IJCNN} '03.

[66] Y. Kim and J. Ra, "Weight value initialization for improving training speed in the backpropagation network," in *Proceedings of International Joint Conference on Neural Networks*, Nov 1991, pp. 2396–2401 vol.3.

[67] M. Hagan and M. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, Nov 1994.

[68] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '11, 2011.

[69] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, ser. SP '12, 2012.

[70] D. Zhang, L. Ge, R. Hardy, W. Yu, H. Zhang, and R. Reschly, "On effective data aggregation techniques in host-based intrusion detection in manet," in *Proceedings of IEEE International Consumer Communications and Networking Conference (CCNC)*, Jan 2013.

[71] W. Yu, L. Ge, D. Zhang, R. L. Hardy, and R. J. Reschly, "On effective sampling techniques in host-based intrusion detection in tactical manet," *Int. J. Secur. Netw.*, vol. 8, no. 3, pp. 154–168, Nov. 2013.

[72] H. Deng, W. Li, and D. P. Agrawal, "Routing security in wireless ad hoc networks," *IEEE Communications Magazine*, vol. 40, pp. 70–75, 2002.

[73] X. T. Dang, N. Bulusu, and W. chi Feng, *RIDA: A Robust Information-Driven Data Compression Architecture for Irregular Wireless Sensor Networks*, 2007.

[74] D. Tsitsipis, S. Dima, A. Kritikakou, C. Panagiotou, and S. Koubias, "Data merge: A data aggregation technique for wireless sensor networks," in *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, sept. 2011, pp. 1 –4.

[75] C. Yang, Z. Yang, K. Ren, and C. Liu, "Transmission reduction based on order compression of compound aggregate data over wireless sensor networks," in *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, oct. 2011.

[76] S. Ozdemir and H. Cam, "Integration of false data detection with data aggregation and confidential transmission in wireless sensor networks," *Networking, IEEE/ACM Transactions on*, vol. 18, no. 3, pp. 736 –749, june 2010.

[77] Y.-C. Fan and A. Chen, "Efficient and robust schemes for sensor data aggregation based on linear counting," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 11, pp. 1675 –1691, nov. 2010.

[78] *OSSSE*, http://www.ossec.net/, 2010.

[79] A. Kattan, "Universal intelligent data compression systems: A review," 2010.

[80] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," 2000.

[81] W. Stallings, *Network security essentials*. Prentice Hall, 2007, vol. 2.

[82] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[83] Ns3. [Online]. Available: http://www.nsnam.org/

[84] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2032–2033, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367572

[85] D. Zhang, W. Yu, and R. Hardy, "A distributed network-sensor based intrusion detection framework in enterprise networks," in *Proceedings of IEEE Military Communication Conference (MILCOM)*, Nov 2011.

[86] W. Yu, Z. Chen, G. Xu, S. Wei, and N. Ekedebe, "A threat monitoring system for smart mobiles in enterprise networks," in *Proceedings of the 2013 Research in Adaptive and Convergent Systems*, 2013.

[87] G. X. Linqiang Ge, Hanling Zhang, C. C. Wei Yu, and E. P. Blasch, "Towards mapreduce based machine learning techniques for processing massive network threat monitoring data," *Networking for Big Data, published by CRC Press & Francis Group*, 2014.

[88] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.

[89] J. Liu, J. Chen, and J. Ye, "Large-scale sparse logistic regression," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[90] R. Xi, N. Lin, and Y. Chen, "Compression and aggregation for logistic regression analysis in data cubes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, pp. 479–492, April 2009.

[91] R. Garg, A. Varna, and M. Wu, "A gradient descent based approach to secure localization in mobile sensor networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, March 2012.

[92] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and Y. Xiang, "Internet traffic classification by aggregating correlated naive bayes predictions," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 5–15, Jan 2013.

[93] C. Chelba and A. Acero, "Conditional maximum likelihood estimation of naive bayes probability models using rational function growth transform," Microsoft Research, Tech. Rep. MSR-TR-2004-33, April 2004. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=70051

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

**Curriculum Vita**

**Name:** Linqiang Ge

**Research Interests:** Cyber Security, Computer Networks, Cyber-Physical Systems, and Information Assurance, including security and privacy issues in wireless networks, Cyber-physical systems and security, Mobile/celluler networks security, and large scale data publishing/processing.

**Education:**

- D.Sc. in Information Technology — September 2011 - May 2016
  Towson University, Towson, MD, U.S.A.
- M.S. in Applied Information Technology — June 2011
  Towson University, Towson, MD, U.S.A.
- B.S. in Mathematics — June 2009
  Qingdao University, Qingdao, P.R.China.
- B.S. in Economics — June 2009
  Qingdao University, Qingdao, P.R.China.

**Teaching Experience:**

- Lecturer — 09/2013 - 12/2015
  Department of Computer and Information Sciences, Towson University.
- Teaching Assistant — 09/2011 - 05/2013
  Department of Computer and Information Sciences, Towson University.

**Research Experience:**

- Researching Assistant — 01/2011 - Present
  Department of Computer and Information Sciences, Towson University.

**Working Experience:**

- Assistant Professor 02/2016 - present
  Department of Computer Science, Georgia Southwestern Sate University.
- Research Assistant 08/2011-12/2015
  Department of Computer Science, Towson University.
- Math Lab Tutor 09/2010 - 06/2011
  Department of Mathematics, Towson University.
- Website Developer 02/2011-05/2011
  Department of Psychology, Towson University.
- Software Engineer 01/2009-12/2009
  Qingdao Chinsoftware Co.,Ltd.

**Research Projects**

- Modeling and Defense of Malware Attacks on Tactical Mobile Ad Hoc Networks (PI: Wei Yu). Source: Army Research Laboratory (ARL). Grant Number: W911NF-11-2-0092. Duration: 02/2014-05/2015.
- A Network Sensor Based Defense Framework for Active Network Security Situation Awareness and Impact Mitigation – Phase II (PI: Wei Yu, Co-PI: Chao Lu). Source: Air Force SBIR contract. Duration: 12/2013-02/2016.
- Membership Inference in a Differentially Private World and Beyond (PI: Wei Yu). Source: National Science Foundation (NSF). Grant Number: CNS-1117175. Duration: 09/2011-08/2015
- A Network Sensor Based Defense Framework for Active Network Security Situation Awareness and Impact Mitigation (PI: Wei Yu, Co-PI: Chao Lu). Source: Air Force SBIR contract. Duration: 01/2012-01/2013 and 12/2013-02/2016.

- A Distributed Host-Based Intrusion Detection Framework for Military Network Operation (PI: Wei Yu). Source: Army Research Laboratory (ARL). Grant Number: W911NF-11-1-0193. Duration: 05/2011-05/2014.

**Refereed Book Chapters:**

- **Linqiang Ge**, Hanling Zhang, Guobin Xu, and Wei Yu, “Towards MapReduce Based Machine Learning Techniques for Processing Massive Network Threat Monitoring Data", accepted to appear in ***Networking for Big Data***, published by CRC Press & Francis Group, USA, 2015.

- Wei Yu, **Linqiang Ge**, Guobin Xu, and Xinwen Fu, “Towards Neural Network Based Malware Detection On Android Mobile Devices", accepted to appear in ***Springer Book Series: Cybersecurity Systems for Human Cognition Augmentation***, 2014.

**Refereed Journal Publications:**

- Zhijiang Chen, **Linqiang Ge**, Guobin Xu, Wei Yu, Robert F. Erbacher, Hasan Cam, and Nnanna Ekedebe, “A Threat Monitoring System in Enterprise Networks with Smart Mobile Devices", ***International Journal of Security and Networks (IJSN) – Inderscience Publisher***, January 2015.

- Wei Yu, David Griffith, **Linqiang Ge**, Sulabh Bhattarai, and Nada Golmie “An Integrated Detection System against False Data Injection Attacks in the Smart Grid", in the ***International Journal of Security and Communication Networks (SCN) – John Wiley & Sons***, 2014.

- Difan Zhang, **Linqiang Ge**, Wei Yu, Rommie Hardy, Robert J. Reschly, and Hanlin Zhang, “Effective Aggregation Techniques for Host-based Intrusion Detection in MANET", in the ***International Journal of Security and Networks (IJSN) – Inderscience Publisher***, 2013.

- Wei Yu, **Linqiang Ge**, Difan Zhang, Rommie Hardy, and Robert J. Reschly,, "Effective Sampling Techniques for Host-based Intrusion Detection in MANET", in the ***International Journal of Security and Networks (IJSN) – Inderscience Publisher***, 2013.

**Refereed Conference Publications:**

- Xiaofei He, Xinyu Yang, Jie Lin, **Linqiang Ge** and Wei Yu, "Defending against Energy Dispatching Data Integrity Attacks in Smart Grid", in *Proc. of 34rd IEEE International Performance Computing and Communications Conference.* 2015.

- Sulabh Bhattarai, Stephen Rook, **Linqiang Ge**, Sixiao Wei, Wei Yu, and Xinwen Fu, "On Simulation Studies of Cyber Attacks against LTE Networks", in ***Proc. of IEEE International Conference on Computer Communication and Networks (ICCCN)***, August 2014, Shanghai, P. R. China (Acceptance Ratio: 28%).

- Sixiao Wei, Wei Yu, **Linqiang Ge**, Khanh D. Pham, Erik P. Blasch, Dan Shen, and Genshe Chen, "Simulation Study of Unmanned Aerial Vehicle Communication Networks Addressing Bandwidth Disruptions", in *Proc. of SPIE Defense, Security, and Sensing (DSS)*, May 2014, Baltimore, MD, USA.

- **Linqiang Ge**, Wei Yu, Khanh D. Pham, Erik P. Blasch, Genshe Chen, and Dan Shen, "Toward Effectiveness and Agility of Network Security Situation Awareness using Moving Target Defense (MTD)", in *Proc. of SPIE Defense, Security, and Sensing (DSS)*, May 2014, Baltimore, MD, USA.

- Wei Yu, Hanlin Zhang, **Linqiang Ge**, and Rommie Hardy, "On Behavior-based Detection of Malware on Android Platform", in *Proc. of IEEE Globe Communication (GLOBECOM) – Communication and Information System Security (CISS) Symposium*, December 2013, Atlanta, GA, USA.

- **Linqiang Ge**, Wei Yu, and Mohammad Ali Sistani, "On Localization Attacks Against Cloud Infrastructure", in *Proc. of SPIE Defense, Security, and Sensing*

*2013*, April/May 2013, Baltimore, MD, USA.

- Difan Zhang, **Linqiang Ge**, Rommie Hardy, Hanlin Zhang, Wei Yu, and Robert J. Reschly, “On Effective Data Aggregation Techniques in Host-based Intrusion Detection in MANET", in *Proc. of the 10th Annual IEEE Consumer Communications and Networking (CCNC) – Green Communications and Computations Track*, January 2013.

- **Linqiang Ge**, Difan Zhang, Rommie Hardy, Hui Liu, Wei Yu, and Robert J. Reschly, “On Effective Sampling Techniques for Host-based Intrusion Detection in MANET", in *Proc. of IEEE Military Communication (MILCOM) – Track 3: Cyber Security and Trusted Computing*, October 2012, Orlando, FL, USA.

- Sulabh Bhattarai, **Linqiang Ge**, and Wei Yu “A Novel Architecture against False Data Injection Attacks in Smart Grid", in *Proc. of IEEE International Conference on Communication (ICC) – Communication and Information Systems Security Symposium (CISS)*, June 2012, Ottawa, Canada.

- Difan Zhang, Hanlin Zhang, **Linqiang Ge**, Wei Yu, Chao Lu, Genshe Chen and Khanh Pham, “On Effectiveness of Network Sensor-based Defense Framework", in *Proc. of SPIE Defense, Security, and Sensing 2012,*, April/May 2012, Baltimore, MD, USA.

**Technical Skills**

- **Language:** Java, C++,R, HTML, CSS, JavaScript, PHP, and SQL.

- **Operating System:** Windows, Linux, and MAC OS.

- **Software & Tools:** Matlab, NS-2, Gridlab-D, Wireshark, Eclipse, Netbeans, Android SDK, Xcode, Photoshop, Dreamweaver, MS office suits, etc.

**Honors and Awards:**

- **Travel Grant Awards:** *GEC 21, 2014*.
- **University Scholarship Awards:** Qingdao University, 2005-2009.

**Conference Technical Program Committee (TPC):**

- Technical program committee member of *Asia Pacific Conference on Wireless and Mobile 2014*.
- Technical program committee member of *International Conference on Connected Vehicles & Expo (ICCVE) 2013*.
- Technical program committee member of *IEEE Consumer Communications and Networking Conference (CCNC) 2013*.

**Research Paper Review:**

- IEEE International Conference on Computer Communications (INFOCOM), 2015
- IEEE International Conference on Communications (ICC), Track: Ad-hoc and Sensor Networking, 2015.
- The 34th Annual IEEE International Conference on Computer Communications, Track:Information security and privacy, 2015.
- IEEE Wireless Communications and Networking Conference (WCNC), Track: Services, Applications, and Business, 2015.
- IEEE Global Communications Conference (GLOBECOM), Communication and Information System Security Symposium, 2014.
- Second International Symposium on Security in Computing and Communications (SSCC'14), Track: Security and Privacy in Networked Systems, 2014.

- IEEE International Conference on Communications (ICC), Communication and Information Systems Security Symposium, 2014.
- IEEE Wireless Communications and Networking Conference (WCNC), Track: Services, Applications, and Business, 2014.
- IEEE SmartGridComm Symposium, Track: Smart Grid Cyber Security and Privacy, 2013.
- The Premier International Military Communications Conference (MILCOM), Track: Cyber Security and Trusted Computing, 2012.