Deep Learning based Intrusion Detection System for Internet of Things

Shiven Chawla

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Cyber Security Engineering

University of Washington

2017

Committee:

Geethapriya Thamilarasu, Chair

Yang Peng

Dong Si

Program Authorized to Offer Degree:
Computing & Software Systems (CSS)

University of Washington

Abstract

Deep Learning based Intrusion Detection System for Internet of Things

Shiven Chawla

Chair of the Supervisory Committee:
Dr. Geethapriya Thamilarasu
Computing & Software Systems (CSS)

With the increase in number of Internet connected devices, security and privacy concerns are the major obstacles impeding the widespread adoption of Internet of Things (IoT). Securing IoT has become a huge area of concern for all, including the consumers, organizations as well as the government. While attacks on any system cannot be fully prevented forever, real-time detection of the attacks are critical to defend the systems in an effective manner. Limited research exists on efficient intrusion detection systems suitable for IoT environment. In this thesis, we propose a novel intrusion detection system that uses machine learning algorithms to detect security anomalies in IoT networks. This detection platform provides security as a service and facilitates interoperability between various network communication protocols used in IoT. We provide a framework of the proposed system and discuss the intrusion detection process in detail. The proposed intrusion detection system is evaluated using both, real network traces for providing a proof-of-concept, and on simulation for providing evidence of its scalability. Our results confirm that the proposed intrusion detection system is capable of detecting real-world intrusions effectively.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisor Dr. Geetha Thamilarasu for her continuous and unlimited support, feedback and adjustment of the research. We would also like thank my committee members, Dr. Yang Peng and Dr. Dong Si for their constant guidance and support, and feedback for my research. Without your supervision, this research may have not been completed.

Secondly, I would like to thank Dr. Nancy Kool for her consistant guidance and valuable suggestions on my thesis report and defense presentation.

Thirdly, I would like to thank all teachers and staff at UWB for help and support during my study. The years I spent at UWB were fantastic, full of knowledge, challenges and joy. It will never be forgotten.

Finally, I would like to thank all my friends for their best wishes for me.

DEDICATION

I dedicate my master's thesis to my family; my mother, my father, my sister and my girlfriend for the unlimited encouragement and support during my masters study at UWB.

Chapter 1

INTRODUCTION

1.1 Background

The Internet of Things (IoT) is a network of everyday physical objects that can connect to Internet to communicate and synthesize data using existing network resources. These objects are the interconnected digital devices or sensors that are capable of harvesting data, and exchanging this information over the global Internet. These interactions between sensors, connectivity, and people and processes creates new applications and services. These digital devices or sensors are referred to as the "Things" in the Internet of "Things".

Essentially, IoT networks are created by the interconnection of IoT devices lying within the range of individual users, typically within a range of 10 meters and have an inconsistent topology that can change dynamically with time. IoT technology is increasingly used in various applications such as health care, national security, business augmented services, and at smaller scale in smart-home environments. With the increased popularity, IoT networks are also becoming more susceptible to security attacks. Cyber security attacks are becoming one of the most serious threats to IoT security. These attacks occur in various forms, targeting different resources on a variety of IoT devices. These attacks tend to compromise one or more device(s) in an IoT network which can be further utilized as a "resource" or "platform" for attacks such as distributed denial-of-service, and fraudulent activities such as ransomware, opportunistic-service stealing, and information ex-filtration. Hence, securing the IoT devices and creating intrusion-resilient IoT networks becomes increasingly important for safeguarding such data.

1.2 Research Problem

Security issues pose the greatest challenge against the realization of the IoT paradigm in full bloom. Unlike the traditional networks, IoT networks suffer from a lack of well-established and standardized design concepts such as the client-server model. This deficiency prevents a

large variety of traditional security solutions from being implemented in IoT networks. IoT devices are also resource-constrained, capable of providing only a few megahertz of compute capability, 5-10 megabytes of volatile storage and almost negligible persistent storage. Their power capacity also ranges between 6-24 hours, as they are powered by batteries. Such restrictions also prohibit IoT devices from using resource-intensive security solutions such as Intrusion Detection Systems, Intrusion Prevention Systems, or Security information and event management. Another significant challenge is posed by an increasing number of IoT devices. Researchers predict that total number of devices connected to internet would almost triple to 20.8 Billion units by 2020 [35]. With the increasing number of IoT devices, IoT is becoming a lucrative platform for a variety of Internet attacks which can occur in various forms, targeting different resources on a variety of IoT devices. Keeping a track of such vast number of devices would become difficult day-by-day and securing or patching such devices would become infeasible. Most of the IoT services are provided by big vendors who develop IoT communication protocols as proprietary systems. Therefore, many commercially available IoT devices are not accessible for scrutiny and development by security researchers working in the open-source domain. Also, each pair of connected devices within the IoT network may not communicate using the same communication protocol, therefore, it becomes exceedingly difficult for a security solution to investigate for the presence of a C&C in this heterogeneous network setting. Thus, a centralized intrusion detection system (IDS) becomes incompetent for peer-to-peer heterogeneous IoT networks.

In rules-based intrusion detection, defender tries to define a set of rules to decide that a particular behavior is that of an intruder and in signature-based intrusion detection, defender tries to match the signatures of network traffic with a particular set of signatures. Various rules/signatures based intrusion detection approaches [39, 18, 28, 29, 32, 8, 10, 7, 19, 21, 44] have been proposed by researchers to detect intrusions using signatures and patterns associated with network traffic. Nonetheless, cyber crime is ever-evolving and can be quite flexible. We have witnessed that the attacker can use well-known communication protocols such as Bluetooth, Wi-Fi, etc., to vantage a command and control (C&C) point from a remote location [4, 5, 27], and exploited network structures ranging from a more centralized wireless sensor networks to the peer-to-peer IoT networks [23, 27, 33, 11, 8, 22, 31, 30, 20, 3].

Therefore, there is a need to develop a next generation intrusion detection system that can accurately detect behavioral anomalies in IoT network. Such an intrusion detection system is independent of the type of IoT devices, communication protocols, structure of network as well as be resilient to network topology changes.

1.3 Research Objectives

Our goal is to develop a secure, portable and ready to deploy security system which provides a practical and effective solution for securing future large-scale IoT networks. In this thesis, we present an Integrated Intrusion Detection (IID) system that works independent of IoT protocols and network structure, and requires no prior knowledge of security threats.The goal of this research is to develop an artificially-intelligent intrusion detection system to provide security as a service to IoT networks. The proposed intrusion detection system monitors network traffic in the IoT network silently and classifies it as malicious or benign. Accordingly, it can perform a mitigation action if an intrusion is detected. It is designed to work for a wide variety of IoT networks, ranging from Personal Area Networks (PANs) to Metropolitan Area Networks (MANs), to enterprise class Local Area Networks (LAN) and Wide Area Networks (WANs).

The proposed intrusion detection system uses the concept of seamless deep-learning to adapt with changing threat landscape and network topology. It requires no prior knowledge of IoT-specific compromises such as captured network payload-binaries, traffic signatures, or compromised node's ID/ IP address. The proposed intrusion detection system is based on the essential properties of compromised IoT network communication which differs distinctively from the properties of benign IoT network communication. It categorizes the network traffic into sessions and investigates anomalous characteristics of network-activity. The objectives of this research are:

- To deeply understand IoT, and it components.
- To examine anomaly detection techniques; and develop a deep-learning based intrusion detection model for IoT networks.
- To evaluate the model and offer recommendations on improving the model.

1.4 Scope and Limitation

This research is aimed to develop a portable and smart intrusion detection system for providing secure data communication between IoT devices located within a geographically close neighborhood. The data that this system is designed to investigate is real numerical or string data which is comprised in a network packet. Currently, this system is not capable of dealing with raw/un-formatted, or encrypted data. Furthermore, the capabilities of this intrusion detection system are limited to anomalies at transport layer. Therefore, more sophisticated/physical attacks that could alter/tamper with the hardware of IoT devices will not be detected by this system. Although, developed as a portable system, the proposed intrusion detection system is designed to be a logical control (i.e. software), and not a physical control for IoT networks.

1.5 Document Outline

In this thesis, we present our intrusion detection system and evaluate it using real network traces, collected from real-time IoT network simulation. The rest of the thesis is organized as follows: we discuss the literature review and the related work in Chapter 2. In Chapter 3, we describe the proposed methodology and implementation of our intrusion detection system, including network monitoring algorithm, anomaly detection algorithm and describe how the IID dispatches a mitigation response. In Chapter 4, we discuss the implementation details of the IDS developed to demonstrate a proof-of-concept. Chapter 5 shows the results for evaluation of this system on real-time network traffic. We summarize the research, discuss the current limitations and possible solutions, and conclude the discussion in Chapter 6.

Chapter 2

BACKGROUND AND RELATED WORK

Researchers have demonstrated several security attacks against IoT devices, such as the ransomware attack against thermostats, the tea-kettle attack against car-locks and blackberry smart-phones, beside many others. In this section, we review some of these attacks and various security solutions which have been proposed by researchers to detect intrusions across different applications of IoT networks.

2.1 IoT Definition

Today, every device has the capability to generate data and share it over the Internet, and this is what broadly forms the Internet of Things (IoT).Recently, researchers predict that total number of devices connected to internet would exceed 6.4 Billion by 2016 and this number is expected to reach 20.8 billion by 2020 [35]. The list of such devices includes iPhone, iPad, iWatch, Smart TVs and many more devices. The European Research Cluster on IoT (IERC) defines IoT as:

"a dynamic global network infrastructure with self-configuring capabilities based on standard and inter-operable communication protocols where physical and virtual things have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information networ" [37].

2.2 Security Challenges in IoT and Current Intrusion Detection Techniques

Broadhurst *et al.* discusses the nature of hacker groups engaged in cyber-crime[4]. They outline the meaning and the possibility of cyber-crime, the theoretic and experiential challenges encountered towards addressing these cyber offenders, and conclude that profit-oriented cyber-crime activities and the cyber-crime committed by the state actors is more structures and specialized than the typical form of protest activity. They claim that cyber-criminals operate in loose networks, but they are still located in close geographic proximity to each other, even when the attacks are cross-national. Krimmling *et al.* focuses on mitigating security

risks for IoT network protocols that are a promising candidate for the futuristic concept of Smart City Environment such as the Constrained Application Protocol (CoAP)[21]. They propose a rules-based modular intrusion detection framework but their results favored a hybrid approach towards intrusion detection, involving both the rule-based and the anomaly based intrusion detection. Moreover, their IDS framework could only mitigate the routing attacks. Arrignton *et al.* proposes to detect intrusions in smart-homes by using behavioral modeling[2], by detecting anomalies related to non-playing characters (NPCs), i.e. humans performing different roles around/in a smart-home. Although the proposed work does not disclose which IA or hardware implementation has been used, it claims to achieve cost-efficient, and easily verifiable autonomous monitoring for intrusion detection.

Le at al. conferences the implementing an intrusion detection system to gain more insight and visibility on their 6LoWPAN due to its inherent security vulnerabilities [22]. They propose to use cryptography/encryption, and Intrusion Detection Systems for securing IoT devices. An issue that surfaces with this approach is that the IoT devices cannot support heavy encryption due to the lack of power. da Silva *et al.* emphasize that the wireless sensor networks have a wide variety of prospective applications which makes them as a lucrative target for attackers[8]. da Silva *et al.* argues that it becomes necessary to use an intrusion detection systems that can alert the end-user. They propose a rules-based IDS to fit the limitations of the WSNs. Although, they achieved accuracy but their IDS can only detect the following attacks: Message delay, repetition and wormhole attacks; Jamming attacks; Data alteration attacks; and Message negligence, black-hole and selective-forwarding/sinkhole attack.

Kansra *et al.* proposed a hybrid intrusion detection system for generic networks, by combining the signature-based, Snort IDS with the anomaly detection techniques based on Nave Bayes data-mining algorithm[17]. They choose Nave Bayes data-mining for anomaly detection after evaluating different types of data-mining techniques such as clustering, categorization, and the associativity rule. Kansra *et al.* uses the Knowledge Discovery Data Mining (KDD) CUP 20 dataset and Waikato Environment for Knowledge Analysis (WEKA) framework for testing their proposed hybrid IDS. In this paper, they evaluate the 5 data-mining techniques: neural-networks, fuzzy-methods, Bayesian classifiers, k-nearest neighbor

clustering, and decision trees. Kansra *et al.* simulates these data-mining techniques and evaluates them on the metric of precision, recall and true-positives rate.

Zhao *et al.* focuses on intrusion detection in IoT networks that are used in coal mine disaster warning systems. They propose to use the immunity algorithm to process the network information ahead of time to select the features to classify the traffic[45]. Once the features have been selected, the Black Propagation Neural Network (BPNN) is used to determine which traffic is malicious or not. They demonstrate classification success rate of over 97% but it requires 9 weeks of training over network traffic. BPNN is a very heavy machine learning algorithm and it cannot be implemented over a resource-constrained device such as a raspberry-pi.

Wang *et al.* proposed an intrusion detection system using artificial neural networks and fuzzy clustering technique, to increase detection precision and detection stability, especially for less frequent attacks[41]. Although, Wang *et al.* did not focus on IoT network security, but the research used the well-established KDD CUP 1999 dataset, and outperformed other intrusion detection techniques such as decision trees, and nave Bayes, regarding detection precision and detection stability.

Various intrusion detection solutions have been proposed for IoT based networks, most of them are not apprehensive and pre-emptive in nature[39]. Surendar *et al.* demonstrates that formerly proposed SVELTE[29] and INTI[6] suffer from packet-dropping and partial device-participation in intrusion detection process. Surendar *et al.* proposes InDReS, Intrusion Detection and Response System for 6LoWPAN based IoT networks, uses the evidence theory and the rate of packet-dropping to perform malevolent node detection. It shows an increase in throughput and a significant decrease in packet drop ratio in comparison to both INTI and SVELTE, but an increased packet delivery rate only in contrast to INTI. Although, Surendar *et al.* showed an improved performance, it can only detect sinkhole attacks.

Raza *et al.* proposed an intrusion detection system, known as SVELTE, for IoT networks based upon 6LoWPAN protocol[29]. SVELTE used a hybrid of signature detection and anomaly detection based approach to perform intrusion detection for IoT networks. It is implemented with an integrated mini-firewall and targets primarily upon RPL protocol based implementations of IoT network. However, it was only able to detect routing attacks such

as sinkhole attack, and the selective-forwarding attacks. SVELTE detection results were obtained using the signature detection based technique primarily.

2.3 Security Attacks in IoT

In this section we discuss the typical threat-landscape for IoT networks and enumerate distinct attacks that can be launched against an IoT network. We consider a variety of network based attacks that can be launched against a smart-home network. The malicious network traffic in a smart home setting can be used to hamper/alter the unusual behavior of IoT devices, or cause data-exfiltration/data-loss, or even impose physical harm to the end-users [45, 42, 38]. IoT networks are highly penetrable for various attacks. More recently, researchers have been able to cause a car crash in the autopilot mode [36], which can prove to be fatal in case of a real attack. Moreover, smart-pacemakers medical network protocols have been demonstrated to be compromised [13], which can prove fatal for a patient, if the attack was conducted by an attacker. In this section, we enumerate similar IoT specific attacks focused in our research.

2.3.1 Remote Control Attack

Attackers use botnets, man-in-the-middle, and denial-of-service attacks to target remote, connected IoT devices. Once, the attacker has intercepted the authentic communication, he can hijack the behavior of the remote device.

2.3.2 Distributed Denial of Service Attack

A very small cluster of external devices can be easily used to disrupt the services and minimum service-level functioning of small IoT networks such as a smart home. Flooding a smart home IoT network with large number of network packets, and each network packet containing a large payload can bring down the quality-of-service drastically.

2.3.3 Botnet Participation

IoT devices are attractive to the botnet masters because they are usually shipped with insecure defaults, they allow unauthorized access to the management systems, and run insecure exploitable code.

2.3.4 Ransomware

Attackers design "jack-ware" or the ransomware to specifically target the connected devices. Attackers may use jack-ware by threatening to DoS IoT device(s) or lock down critical services until the user pays up.

2.3.5 Bricking Attack

IoT connectivity protocols are an easy target for the bricking attack. The attack is mostly spread by a payload, usually a worm, which is delivered from a remote device accessible to victim network. The worm spreads by jumping directly from one device to its neighboring devices, using only their physical proximity.

2.3.6 Device Weaponization

Many IoT devices can keep surveillance and vigilance over their hosts, such as the baby monitors, TV cameras, and Smart hub microphones. Device weaponization attack involves attacker to be able to gain control of such devices and which enables them to keep vigilance over the victim.

2.3.7 Wormhole Tunnel Attack

Wormhole tunnel attack targets IoT networks and involve a pair of attacker nodes that are linked through a virtual private connection. In this attack, network packets received by the victim node is first forwarded through the wormhole, in order to be replayed later. As a result, RPL network starts creating non-optimized routes.

2.3.8 Increased RPL Rank Attack

RPL is a IPv6 routing protocol for low-powered and lossy networks. Each device in RPL network is labeled with a rank value. RPL rank attack is an indirect attack on IoT networks, aimed at creating inconsistency throughout the network by increasing the rank value to generate large path loops in the network.

2.3.9 Information Leak and Ex-filtration

A large variety of IoT devices, such as smart TVs, smart hubs and refrigerators, are capable of collecting our personally identifiable information and financial information. Implementation flaws, poor configuration, and weak encryption can open back-doors for attackers.

2.3.10 Sinkhole attacks

In a sinkhole attack, a malicious node is required to attract significant amount of network traffic by advertising falsifying routing information and then degrade the overall network performance by dropping the packets.

2.3.11 Ballot-Stuffing Attack

In this attack, a malicious node can ruin the trust of a well-behaved node by providing bad recommendations against it. This will decrease its chances of being selected for the service [24].

2.3.12 Opportunistic-Service Attack

The self-promoting or the Opportunistic service attack is based on self-interest wherein a malicious node can initially provide good service to gain high reputation among peer nodes [24]. Then it can provide selective bad service and ruin the quality-of-service (QoS) for the entire network.

2.4 Summary

The responsibility of developing an effective intrusion detection is typically approached using data clustering and classification algorithms. Although, the related work shows that a hybrid approach comes closer to what we implement in this thesis work, but most of these security solutions are not dynamic and preemptive. For example, Meinel *et al.* tries to prevent battery exhaustion attack by using a proxy system to filter CoAP messages, authenticate and authorize users using the lightweight versions of IPsec and DTLS used in IoT networks but this approach can only be used as a rule-based firewall type security solution [20]. Some IoT based IDS designs propose a lightweight security solution which can only detect two or three types of attacks [33, 11, 3, 26, 32, 7]. Hosseinpour *et al.* proposes a lightweight IDS which uses network traffic in the edge layer to train its detectors, clusters the traffic in cloud, and analyzes the intrusion alerts in the fog layer [16]. Since, it depends on cloud infrastructure, the unavailability of cloud services could render it incapable of providing security. Some of the proposed IDS solutions are highly resource consuming and performance intrinsic, making them unsuitable for deployment over low-powered and resource constrains IoT devices [9]. On the other hand, some IDS solutions are not adaptable and dynamic, needing to be frequently

updated [32, 30]. Yu *et al.* explains properties of resource-constraint devices in 6LoWPAN based IoT networks and illustrates a variety of attacks on typical 6LoWPAN networks, for example, ranking attack, repair attack, and resource depletion attack [43]. It also submits a few techniques of performing routing based attacks on RPL communication.

In this chapter, we reviewed the work related to proposed security solutions in general and intrusion detection in IoT. Typically, a mix of clustering and classification techniques is used by the researchers to find anomalies. However, it becomes exceedingly important to consider the unique challenges that IoT offers. For instance, intrusive detection techniques that demand significantly high processing capability or resource consumption do not suit IoT devices. Moreover, the IoT networks are heterogeneous and transmit data in large amounts. This research also attempts to rectify the shortcomings of these implementations and proposes a peer-to-peer, portable, and artificially intelligent intrusion detection system for IoT networks, designed to operate over low-powered and resource-constrained IoT devices.

Chapter 3

PROPOSED INTRUSION DETECTION SYSTEM

In this chapter, we discuss the network architecture, system architecture, components of proposed IDS, and the design challenges faced during development of proposed intrusion detection system.

3.1 Network Architecture

IoT devices (smart things) communicate with each other in order to exchange sensor outputs, triggers, status messages, etc[12, 14]. an IoT network consists of a IoT coordinator and IoT nodes. The IoT coordinator is used to synchronize and maintain the IoT network.

Figure 3.1 presents the network architecture for the proposed intrusion detection system in an IoT network where the edge-router acts as the IoT hub. In an IoT network, the IoT coordinator is also known as a IoT hub which manages each IoT device in the network. All communications between IoT devices pass through the IoT hub to reach the destination. The maximum data transfer rate is about 250kbps and communication range can vary from 100m, depending on the device's power capacity, in a IoT network. As 40kbps can meet the requirements of low-powered control systems, it is sufficient for controlling most home automation devices [12]. These IoT devices are portable and can be easily spread across a well-defined geographical area, such as within an IoT network. They can be accessed using a wireless communication such as Wi-Fi, Bluetooth BLE, ZigBee, or a proprietary communication protocol such as CoAP or Thread. Usually, the IoT devices are paired with user's mobile device using BLE connection. The proposed intrusion detection system can be easily deployed within the network by placing it within coverage area of IoT network and all the neighboring IoT devices within that network. The proposed intrusion detection system primarily works at the transport layer level, to secure both, the ingress and egress traffic, i.e. network traffic coming into the IoT network and the traffic leaving the IoT network.

As shown in *Figure 3.1*, the intrusion detection system is created as a portable compu-

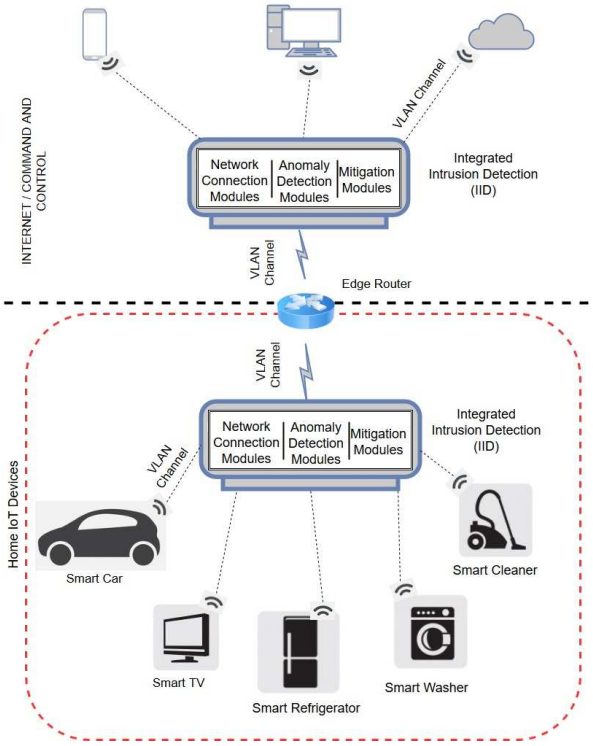

Figure 3.1: Overview of Network Topology

tational unit, with an independent processing and wireless-transmission potentiality. The system is build out of numerous soft-modules which are deployed on this portable computational unit. Typically, a large range of conventional network intrusion detection systems are developed as a hierarchical distributed middleware system whereas the proposed intrusion detection system is developed as a Peer-to-Peer (P2P) consolidated system. IoT networks do not obey a strict hierarchy for control and communication of devices within the network and the IoT devices are usually spread randomly across the network. Additionally, these devices are powered by varying compute-capabilities, and most of the communication among IoT devices occur in a peer-to-peer fashion. Hence, a P2P and Consolidated architecture plays an important role towards increasing system's deployability in IoT networks.

3.2 System Architecture

The main goal of our proposed intrusion detection system for IoT networks is to facilitate on-demand security service, to prevent attacks. The primary responsibility of the pro-

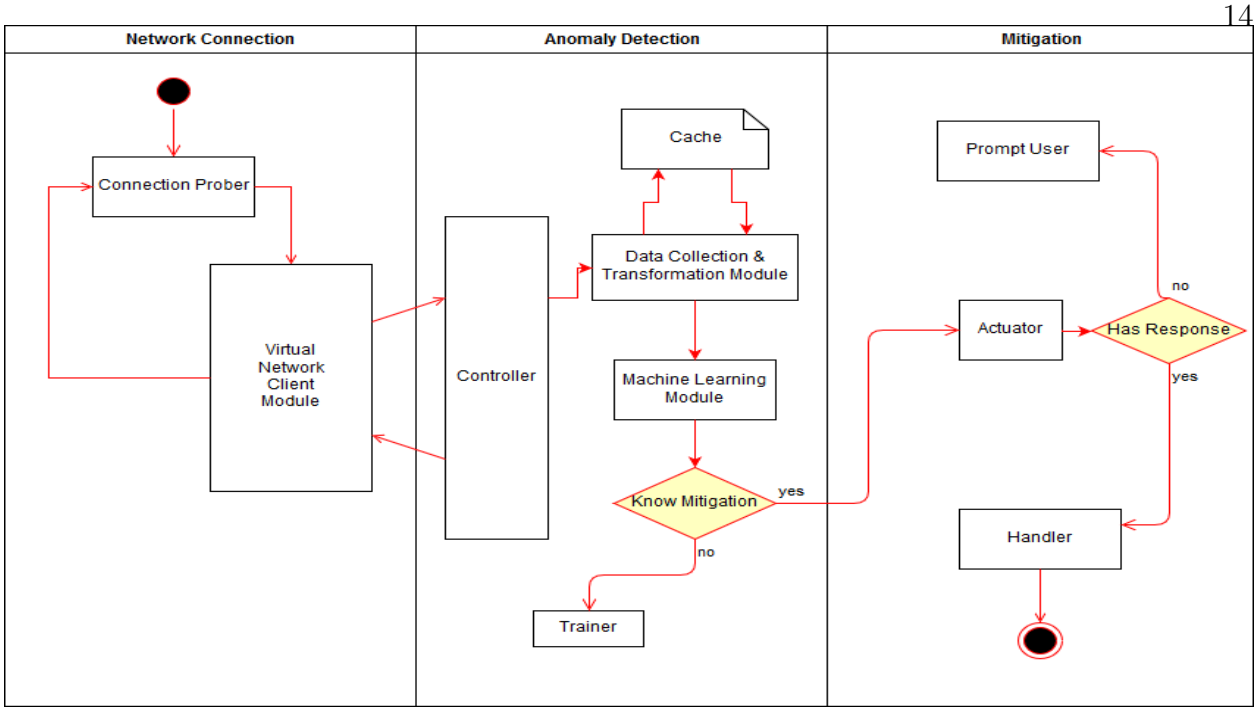

Figure 3.2: IDS Architecture Overview

posed intrusion detection system includes the ability to realize its surroundings, capability to promiscuously listen the wireless communication traffic in its surrounding, and detect intrusions at an early stage.

As shown in *Figure 3.2*, the proposed intrusion detection system functions primarily in three phases - Network Connection phase, Anomaly Detection phase, and Mitigation phase. During the Network Connection phase, the proposed intrusion detection system attempts to determine and deploy appropriate network adapter for facilitating network traffic translation so that it can understand the sniffed network packets from its surrounding. During the Anomaly Detection Phase, the proposed intrusion detection system employs a feed-forward deep-learning algorithm for detecting anomalies in the network traffic within the IoT network. Consequently, if it comes across an intrusion, it reacts to the discovered intrusion in a predefined manner in the Mitigation Phase.

3.2.1 Network Connection Phase

Typically, IoT networks comprise of heterogeneous devices that are capable of communicating with each other in a variety of network communication protocols. During the *Network*

Connection phase, the proposed intrusion detection system uses a Virtual Network Connections (VNC) to intercept the wireless network traffic while monitoring IoT network. Every virtual network connection can use a different network traffic translation adapter. VNC is capable of using many translation adapters simultaneously to listen to a variety of wireless communication at once. As a proof-of-concept, the prototype developed for this research deploys adapters for only 3 IoT communication protocols namely, 6LoWPAN, ZigBee and Bluetooth. *Table 3.1* shows technical specifications for these protocols. During the traffic

Table 3.1: Specifications of IoT protocols targeted

Standard	**Frequency**	**Range**	**Data Rates**
Bluetooth	2.4 GHz	50 m -150 m	1 Mbps
ZigBee	2.4 GHz	10 m -100 m	250 Mbps
6LoWPAN	900 MHz	30 m	100 Kbps

monitoring stage, the captured network traffic is merely a raw dump of wireless transmissions crossing the surroundings of the intrusion detection system. This system uses Scapy, an open-source penetration testing framework for mobile devices, to transliterate cached raw traffic dump into arranged meaningful network transactions. These transliterated network transactions are fed into two destinations, i.e. the Data Collection and Transformation module, and the secondary storage, Cache, in the form of pcap file format. Storing the network transactions in the form of compressed pcap file format helps retrain the machine learning module if the training fails.

The proposed intrusion detection system works on the transport layer and accomplishes traffic monitoring by using two modules in the *Network Connection* phase, i.e. the **Connection Prober** module, and the **Virtual Network Client Module (VNC)** module. In essence, both the modules are designed as cohesive components although due to the overlapping nature of their activities, these modules are implemented as tightly coupled algorithms. For this system, both the modules are implemented as a single algorithm. In this section, we discuss the implementation details for both the components in terms of pseudo code.

(a) Connection Prober

The Connection Prober module is executed periodically and on-line, as the communi-

cation links are constructed dynamically within an IoT network. It is activated when the IID enters in *Network Connection* phase and is responsible for intercepting the probe-signals and broadcast beacons from its neighboring IoT devices. Connection Prober module maintains a list of all the active communications protocols that are being used in the IoT network. It uses this list to maintain active network interfaces which can be used to intercept different wireless communication signals, transmitted in the surrounding environment. Consequently, it attempts to intercept broadcast beacons, handshake messages, or session requests to learn the communication protocols that are being used by IoT devices. If the Connection Prober fails at intercepting the broadcast beacons, or handshake messages, it attempts to deduce the connectionist protocol by apprehending the data-packets transmitted during regular communication of surrounding IoT devices. Thereafter, the connection prober module uses this information for translating the raw-signals to the comprehensible IPv6 network packet format.

At the lower-level, the connection prober module fetches the environment-settings for all the physical network interface cards (NICs) which are capable of catching the wireless-signals. Next, it creates a pipeline from the NIC to obtain the raw traffic feed. Thereafter, it polls between the active interfaces, each of which pipelines a raw input bit-stream to the memory location held by the respective *interface-handles*. In this way, the connection prober establishes a pipe from the physical network interface and reads a bit-stream of network packets.

(b) VNC Module

The Connection Prober module is also responsible for feeding the raw bit stream to a cache, shared by the VNC module. Typically, the connection prober module is designed to use a secondary storage location as a default cache location that is simultaneously replicated in the physical memory location, using a *cache-handle*. After fetching the bit-stream, it uses the known communication protocols to bisect the bit stream into different networks packets. Thereafter, it again feeds these network packets into cache and the Data Collection and Transformation module in the proposed IDS.

The VNC module essentially administers the activation of appropriate network interface and network adapter driver in the IID. Once, the IID learns the surrounding communication protocols from the connection-prober module, it employs a client based network emulator to switch to a compatible virtual network adapter that can be used to translate captured network traffic to understandable IPv6 header format.This client based network emulator is known as the Virtual Network Client (VNC) module in the proposed intrusion detection system. In other words, VNC module transliterates the intercepted data packets from native wireless protocols to comprehensible information for the IID.

We use a very basic form of the VNC module embedded in the form of a sniffer-function. For simplicity, it records network transactions from the IoT simulations, and store them in the form of pcap file format in secondary storage. Consequently, the VNC module passes these packets to the Data Collection and Transformation (DTC) module and intrusion detection system enters the anomaly detection phase.

3.2.2 Anomaly Detection

The proposed intrusion detection system uses perceptual learning as the machine learning algorithm for anomaly detection. After the Data Collection and Transformation module receives the network traffic from the Virtual Network Client module, it extracts the primary features (header-tags) from the network packets and feeds them to the Machine Learning module for binary classification. In the following section, we discuss the implementation aspect of Data Collection and Transformation module and Machine Learning based Anomaly Detection module, as shown in *Figure 3.2*.

(a) Data Collection and Transformation module

The Data Collection and Transformation module is used to strip the captured network packets, extract their header tags and generate features, which are populated into the cache storage. Data Collection and Transformation module is also responsible for feeding these features into Machine-Learning based Anomaly Detection module.

The Data Collection and Transformation module reads each input network packet as a group of strings. As illustrated in *Algorithm 1*, each input network packet is sliced

into distinct layers of the TCP/IP stack, and thereafter, respective header-tags are extracted for each layer in string format. Next, each non-empty layer is designated a label for future reference. Consequently, the extracted header-tags are added to a list under the label of their respective layer. This way, the Data Collection and Transformation removes any repetition of header-tags. The Data Collection and Transformation module is also pipelines these lists back to the cache, as illustrated in *Algorithm 1.* Thus, the DTC module is able to collect the features tabulated in *Table 3.2.*

Algorithm 1 Extracting tags from the sniffed network packets.

Require: T - List of all header tags from all packets in network interface queue.

```
1: function PacketHandler(pkt)        /*where pkt - captured network packet*/
2:     Extract TagString              /*Get tag-string from the packet*/
3:     Divide the TagString for each Layer
4:     for every Layer in Packet do
5:         Get LayerName
6:         if Layer ≠ Empty then
7:             Split the Tags
8:         end if
9:         for every Tag do
10:            Add the Tag to a list
11:        end for
12:        Assign the List to Layer   /*assign list to layer
13:    end for
14: end function
```

Table 3.2: Extracted Feature Set

transmission rate	*reception rate*
transmission to reception ratio	*activity duration*
transmission mode	*source IP*
destination IP	*datavalue information*

(b) Machine Learning based Anomaly Detection module

This module is the principal machine-learning engine of the proposed intrusion detection system, responsible for classifying benign network traffic from malicious network traffic. It employs a perceptual learning model for performing anomaly detection. This module is activated when the intrusion detection system enters the *Anomaly Detection* phase, which consists of two phases, i.e., the training phase and the detection phase. The training phase is performed across long intervals of time, and performed off-line as the training is also a time-consuming activity.

The perceptual model is trained using supervised learning over the tuples of features generated during the data-preprocessing. Before feeding the tuple into the perceptual learning model, each tuple is manually augmented with a binary-classification label representing malicious or benign nature of network packet. The perceptual learning model uses information gain at each perceptual layer to filter out the preferred features, before feeding to the next perceptual layer. For this research, we use a feed-forward deep neural network model as the perceptual learning model. Essentially, the deep learning neural nodes at each layer filter the input tuples, maps them to a definitive ratio, and normalizes these ratios to a binary value. The binary value of 1 signifies a anomalous tuple whereas 0 denotes a benign tuple. We discuss the Machine-Learning based Anomaly Detection module in *Chapter 4* in detail.

3.2.3 Mitigation Phase

The Mitigation phase is responsible for taking a preventive action or a mitigation response when an intrusion is detected. In case an intrusion is detected, the proposed intrusion detection system uses two modules for facilitating mitigation response, i.e. the Actuator module and the Handler module. For this research, we focus primarily towards creating an intrusion detection mechanism, in preference over providing a mitigation mechanism. Therefore, the proposed intrusion detection system follows a primitive approach towards incidence response, wherein the Actuator module is responsible for identifying the intrusion that has been flagged to it by the Machine-Learning module and actuating a mitigation procedure in response to the intrusion. Actuator does not initiate a response by itself but invokes the Handler module with the correct procedure, based upon the signatures of the

discovered malicious network traffic. On the other hand, Handler module is only responsible for executing correct subroutine for mitigation response raised by the Actuator module.

(a) Actuator Module

The Actuator module is responsible for identifying the most suitable mitigation response in the event of an attack within the IoT network. The mitigation response can comprise of simply putting up an alarm to the user, pointing out the compromised IoT device that might be behaving anomalously; or shutting down the communication in the IoT network. When the Actuator module is aware of an appropriate mitigation response, it would activate the Handler module to execute the response or generate an alarm for the end-user.

(b) Handler Module

The Handler module is primarily a set of mitigation procedures hard-coded within IID program to execute a mitigation procedure as a proof-of-concept. A mitigation procedure is invoked by the Actuator module in response to an intrusion, and is further executed by the Handler module. Once, the mitigation response is executed successfully by the Handler module, it logs the type of attack and the mitigation response provided, into a log file and closes the case successfully. In case, if the Handler module is required to raise an alarm for the user, it flags the discovered intrusion for 'requiring user attention' and logs this information in the log file.

3.3 Design Challenges

Traditionally there are two primary approaches towards realizing security for IoT networks, that is, by laying down the rules to find already known malevolent activities which is also known as a signature/rules based intrusion detection system; otherwise, by quarantining the IoT network from Internet and constraining its communication with the Internet, using a firewall. Both the conventional security mechanisms defeat the purpose of smart and ubiquitous connectivity of IoT paradigm. In this research, we propose an artificially-intelligent and portable wireless intrusion detection system to secure a IoT network without compromising its connectivity. Wireless connectivity facilitates movability and adaptability to the proposed intrusion detection device. Contrary to the conventional security approach towards securing

IoT networks, the proposed intrusion detection system uses machine-learning algorithms to observe both, the known and unknown threats, which allows it to adapt to well-established as well as unknown hostile situations. Thus, the proposed intrusion detection system overcomes the drawbacks of a rule-based security approach by using machine-learning algorithm for detecting anomalies within IoT networks.

Over the last decade, security researchers have established that artificially intelligent security systems learn faster as the time progresses, being able to gather more feedback from the surroundings. Sinclair *et al.* shows that the average rate of learning improved by 3.41 times with the progressing time[34]. Hence, it would not be wrong to assume that the proposed intrusion detection system can increase accuracy with the growing number of IoT devices. The proposed intrusion detection system adapts to new network protocols by using *network virtualization* to monitor its surrounding IoT network, and to switch between different network protocols swiftly. Thus, it can simultaneously monitor heterogeneous communication links in IoT networks.

Chapter 4

DETECTION USING DEEP LEARNING

In this chapter, we discuss the deep learning algorithm used to detect intrusions. We also present the details of the various features used for intrusion detection.

4.1 Feature Set

The proposed intrusion detection system uses a set of features, derived from the network traffic. We have choose these features considering the computational capability, and the processing to performance ratio of portable low-powered, resource-constrained IoT devices. Though, the network packet features are obtained directly from the IoT network traffic bit stream representing the header section in IPv6 protocol. Since the length of the headers is static in IPv6 protocol, hence the feature extraction is performed as an operation with constant time complexity. Feature extraction from the bit stream implies that the decoding is not required during extraction.

Typically, the IoT communication protocols broadcast network packets with the data-fields occupying 128 bit positions, i.e. 16 bytes of size, in the IPv6 format. Consequently, the proposed intrusion detection system caches these features and generates meta-features from them, as data pre-processing procedure. Meta-feature are represented by set of 8 bit positions, i.e. 1 Byte each, in the proposed intrusion detection system. During the data pre-processing, IID calculates the probability distribution of the extracted meta-features with respect to a unit interval of time. Observing the frequency of data-packets transmitted in the network simulation, we chose to set one unit interval of time as 3 minutes for this implementation. Next, the IID mathematically generate a vector, $f_p \in R^8$, to represent this probability distribution as illustrated in *Equation 4.1.*

$$f_p = \{P(B_0), P(B_1), ..., P(B_7)\} \tag{4.1}$$

where $P(B_i)$ is the probability of each Byte "1" observed in the i^{th} Byte position, and

$$f = L(f_0), \tag{4.2}$$

where the function $L : R^8 \mapsto R^8$ in *Equation 4.2* is the logical mapping, i.e. if $P(B_i)$ is greater than a half, the probability is mapped to 1, or else, 0.

A set of features represents a tuple of input data for the machine-learning algorithm, as described in later sections. The 16 bytes representation of a tuple is analogous to the set of the meta-features generated for a single data packet in the proposed intrusion detection system. However, the dimensions of the tuple can be reduced by considering the protocol specific format, for ease of reproducibility. It is for the ease of implementation of IPv6 protocol, that we chose to use a 16 Byte representation. In other words, the representation of a tuple can vary in size for different target protocols.

The proposed technique extracts transmission-rate, reception-rate, transmission-to-reception ratio, duration, transmission mode, source-IP, destination-IP, and the data-value information as the primary 8 bytes of meta-features from the IPv6 header of 128 bits (16 bytes). These meta-features are calculated over intervals of unit time, i.e. 3 minutes. The transmission-rate represents the number of network packets sent within one time unit whereas the reception-rate represents the number of network packets received within one time unit. The transmission-to-reception ratio is defined by the average of transmission-rate and the average of reception-rate over one time-unit, i.e. dividing the average transmission-rate by the average reception-rate for each tuple. Duration is defined as the number of time units for which a particular node has been active in the IoT network, i.e. either transmitting or receiving network traffic. Transmission-mode information represents the command state of the node, i.e. whether a node is performing a broadcast or uni cast or multi cast. The source-IP, and the destination-IP represent the source and the destination IPv6 addresses in the Classless Inter-Domain Routing (CIDR) notation, respectively. The data-value of a network packet is signified by a 2-Byte representation of the data payload within the network packet under inspection.

Each tuple is represented as a data-vector d_v which is reduced from f, before feeding into the neural network. Consequently, each meta-feature set can be represented as a feature vector f_v at a time instance n, which is generated as,

$$f_v(n) = d_v(n) \oplus d_v(n-1), \tag{4.3}$$

where $\oplus$ is an exclusive-or operator applied to each position of bits in the vector, as shown

in *Equation 4.3*.

In the following sub-sections, we briefly discuss the feature extraction, training and the traffic classification.

4.1.1 Feature Extraction

The proposed intrusion detection system deploys perceptual learning model for both purposes, i.e. data collection and feature extraction, and machine learning based anomaly detection. Once, the intrusion detection system intercepts the network traffic using during the *Network Connection* phase, it extracts the primary features from the header-tags of the intercepted network packets. These features are extracted by stripping down each network packet and they can be referred as raw features or the primary features, as they constitute of the header-tags, directly extracted from the network packets. The primary features include information such as time-stamp, source and destination IP addresses, etc. which can be directly read from the bit stream of the network packets under investigation. Next, the Data Collection and Transformation module produces secondary features, also known as meta-features, by pre-processing the cached primary features. Meta-features comprise of transmission rate, reception rate and the transmission-rate vs. reception-rate ratio, etc., as shown in *Table 3.2* in *Chapter 3, section 3.2*. Consequently, the Data Collection and Transformation module concatenates the set of primary features with the set of secondary features to create a tuple. Essentially, each tuple is a set of raw features and the meta-features of a data-packet. Thereafter, the Data Collection and Transformation module feeds the tuple into the perceptual learning model for training. In the following section, we discuss the traffic classification before exploring the Machine Learning module in detail, in the later sections.

4.2 Deep Learning based Anomaly Detection

We use a Deep Belief Network (DBN) to fabricate the feed-forward Deep Neural Network (DNN) as the perceptual learning model. A Deep Belief Network is a model of undirected connections between different layers, where each layer comprises n-number of neural nodes. Essentially, a Deep Neural Network is a type of feed-forward Artificial Neural Network constructed by augmenting a Deep Belief Network with additional information. Although, a DNN can be fabricated in different ways, but an advantage of developing a DNN model

from a DBN model is that DBN layers can be initially trained using unsupervised learning algorithm. In this way, DNN can be created from a model pre-trained using unsupervised learning which is very fast in comparison to supervised learning. We use the pre-trained layers of DBN model to create a DNN model, as shown in *Figure 4.2*.

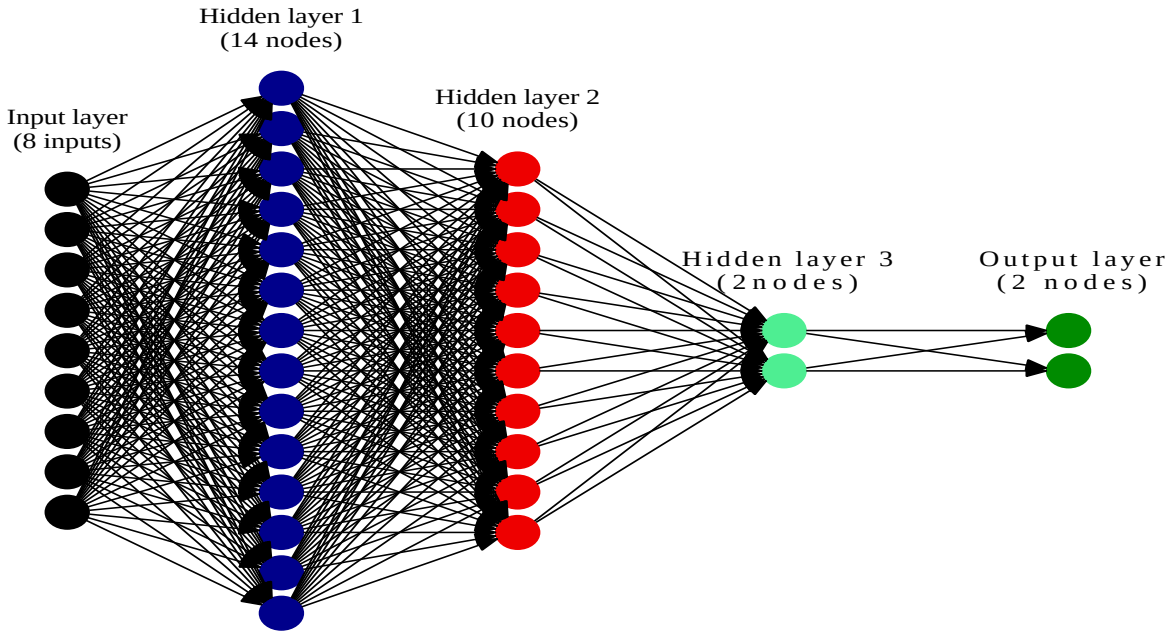

Figure 4.1: Deep Learning model for proposed Intrusion Detection System

As shown in *Figure 4.2*, the weights for all the hidden layers of this DBN model, denoted by w_i , are obtained by performing unsupervised training. However, the parameters generated from this unsupervised training are only used for assigning the initial set of weights. Once, the DBN is successfully constructed, it is transformed to create a Deep Neural Network, i.e. a kind of feed-forward Artificial Neural Network (ANN). For this purpose, a binary-classification layer and label information (a) for each network transaction is added at the top layer of this DBN model to construct a discriminate deep learning model. *Figure 4.3* shows that the DBN is augmented with binary-classification layer and label information to transform into a Deep Neural Network. Now, this DNN model is trained with a bottom-up supervised learning approach using the label information a. During the supervised learning process, each node in a DNN layer is assigned with a weight parameter which are manipulated by using the gradient descent methodology.

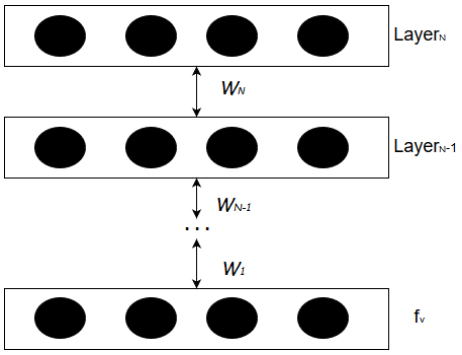

Figure 4.2: Deep Belief Network Structure

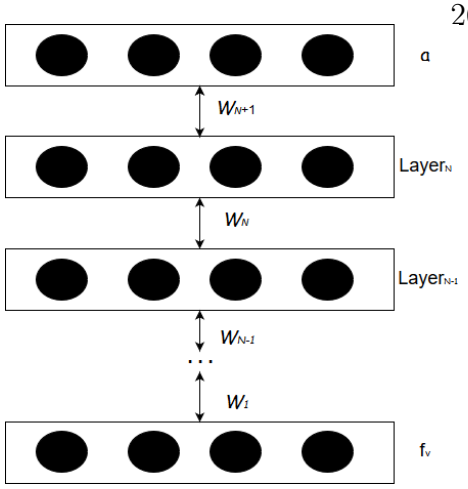

Figure 4.3: Deep Neural Network Structure

The proposed deep learning model uses supervised training and binary classification for identifying malicious activities. If the deep neural network detects an unknown anomaly or a zero-day attack, it stores the corresponding tuple of the filtered features to the 'Cache' as a feedback. This feedback mechanism is used during retraining of the deep neural network, which enriches the feature extraction and labeling functionality of the proposed intrusion detection system. Although, if the features extracted are still not sufficient to classify the network traffic, the feedback is sent to the data collection and transmission module for retrain itself. As shown in the *Figure 4.1*, we developed a 5-layer deep learning model for this research, containing 1 input layer, 3 hidden perceptual layers, and 1 output layer which is a binary-classifier layer. The input layer comprises of 56 nodes which represents an exhaustive list of *maximum* number of network features that can be fed into the deep neural network. As mentioned in the *Feature Extraction* section, these input features are represented as a tuple, formed from a combination of both, the primary and the secondary features. During the supervised training process, each tuple and its label information a is fed to the deep neural network where it pass through the first hidden encode layer and gets filtered out as the x most significant features. Then these x features are passed into the second encode hidden layer where they get filtered into y features and the second encode layer feeds them into the third encode hidden layer. The third hidden encode layer takes the y features as input from the previous layer and filters out two outputs. It also acts as a soft-max layer which fine tunes the results to classify the attack into categories. This

output is passed on to the output layer which represents the two network traffic classes - the malicious and the benign traffic. Output layer does not perform any filtration but ingests the output from the third hidden layer and yields the classification result. Thus, the rest of the hidden layers i.e. the second and the third encode layers also use the labeled traffic to train themselves in the same way as the first encode layer. In this way, each layer of the deep neural network feeds onto this data, and maps the each it to a numerical value. Then, these mapped values are normalized to 0 and 1, where benign network traffic is represented by the value 0 and malign network traffic is represented by the value 1. Thus, the deep neural network develops a binary-classifier for anomaly detection.

We chose to use the deep-learning algorithm because deep learning algorithm outperforms other solutions in multiple domains that are highly unstructured and form heterogeneous patterns. Few examples for such domains would include speech, language, vision, and playing games. Deep learning also gains advantage over other preceding machine learning algorithms because of its ability to extrapolate new features from a limited set of training data. Additionally, the thin and layered structure of sequential deep neural network models makes them the best fit for being deployed over a low-powered and resource-constrained portable IoT device, still facilitating real-time anomaly detection.

As shown in the *Algorithm 2*, the objective function of the proposed deep neural network model, a *binary_crossentropy* loss function, tries to minimize the total cost in the model (also refer to *Equation 4.5*). The deep neural network model is retrofitted for training, and testing the predictions. The machine learning module stores this classifier into a secondary storage known as the classificationStore. For training the proposed intrusion detection system, we divide the dataset into two parts - the *training dataset* and the *testing dataset*. Once trained against the *training dataset*, we test the proposed intrusion detection system against the *testing-dataset*. Although, if the predictions from during testing do not match the results from the *testing dataset*, the system mixes the *training dataset* with the *testing dataset* and re-trains itself with cross-validation. Training process is discussed in the later section in this thesis.

Algorithm 2 Intrusion Detection using Deep-Learning model.

Require: N - List of all header tags from all packets in network interface queue.

function PREDICT(*Cache*) */*where cachePipe - is the pipe established with cache*/*
    matrix ← *Cache* */*translate packets to matrices*/*
    Extract *features* from matrix
    Define $dataset_{train}$ & $dataset_{test}$
    Initialize *Sequential deep-learning model*
    if *initialized* **then**
        Compile *binary-crossentropy* classifier
        m ← *Sequential deep-learning model*
    end if
    Training: $m \leftarrow dataset_{train}$
    if *Training* is complete **then**
        Prediction: $m \leftarrow dataset_{test}$
        if *Predictions* are correct **then**
            Re-Train the *model*
        else
            Invoke *Mitigation Phase*
        end if
    end if
    Store: *classificationStore* ← *Predictions* */*store the classifier model*/*
end function

This simplistic approach of proposed Machine-Learning based Anomaly Detection algorithm for anomaly detection and its ability to detect intrusions within a recourse-constrained environment in real-time makes the proposed intrusion detection system out stand among traditional security approaches that have been discussed in the *Literature Review and Related Work* chapter. Its ability to monitor the surrounding IoT network promiscuously allows it to provide security with 0% overhead to the host-network. The lightweight and movable form-factor, battery-powered hardware, the along with the capability of switching network protocols swiftly, allows it to be adaptable, portable, and scalable for any size of IoT network.

4.2.1 Training Deep Neural Network

This section discusses the proposed training mechanism for the proposed DNN model, illustrated in *Figure 4.4*. At the lowest level, when the feature vector f_v is inputted into the deep neural network, it passes through each layer of the DNN and all the neural nodes in each DNN layer calculates an output using a activation function and generates a filtered result. We have used the rectified linear unit (ReLU) activation function for developing this system. A ReLU function is also known as a ramp function and is analogous to half-wave rectification in electrical engineering. ReLU function is defined as:

$$f(x) = \max(0, x), \tag{4.4}$$

with the input x e.g. a matrix from a convolved image.ReLU sets all negative values in the matrix x to zero and all other values are kept constant.

Each hidden layer links to the next hidden layer by using linear-combinations of outputs and feeds the filtered output generated by the ReLU activation function to the next layer.

To facilitate supervised learning for this research, we fabricate the training set as a set of real-number, K, defined as $\{(f_v^1, a^1), (f_v^2, a^2), ..., (f_v^K, a^K)\}$ samples where each tuple represents a feature-vector, f_v^i and the corresponding binary classification, a^i. Each feature vector f_v represents the probability in the Byte-representation of of meta-features generated from a single data-packet, and a is the binary label information attached to each data-packet. In the training phase, the input feature f_v pass enter the DNN through the external nodes that are present at the bottom of the deep neural network. Initially, the weights attached with each neural node in the DNN are initialized by the DBN model. Consequently, these weight vectors are manipulated as more data passes through DNN layers with each cycle in supervised training.

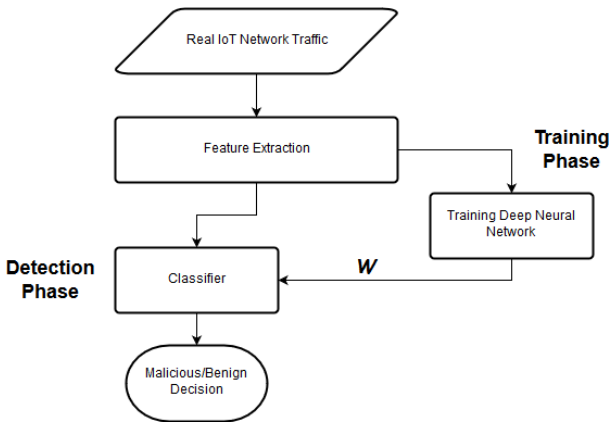

Figure 4.4: Overview of proposed DNN Training

The machine-learning algorithm assigns a cost-function, cumulative cost-function, and an optimization function[25] to manipulate the DNN in this research. We assign a cost function for each layer of the proposed deep neural network as formulated in *Equation 4.5*, defined as the mean square error function between the prediction value and the output, as,

$$C(w, f_v, a) = 1/2 \parallel h_w(f_v) - a \parallel^2, \tag{4.5}$$

where w is the set of weights designated for each connection between simultaneous layers in the proposed DNN, a is the binary label information, and $h_w(f_v)$ is the hypothesis function for every meta-feature vector. The hypothesis function $h_w(f_v)$ is responsible for manipulating weights w on every node in each DNN layer. as illustrated in *Equation 4.6*, the cumulative cost function for a single set of training data k, is defined as,

$$C(w) = 1/K \sum_{k} C(w, f_v^k, a^k) + \lambda/2 \sum_{n}^{5} \sum_{i}^{M_l} \sum_{j}^{M_{l+1}} (w_{ji}^n)^2, \tag{4.6}$$

where the depth of the DNN model is 5 layers, M_l is the number of nodes in the l^{th} layer, and $(w_{ji}^n) \in w$ are the weights attached to the connection between the i^th node in the layer $n-1$ and the j^th node in the layer n. As mentioned earlier, the ReLU function transforms the weights in the set w to generate minimum value for the cost function, $C(w, f_v^k, a^k)$ and the output of this minimized cost function is assigned to w^* as,

$$w^* = \left| \underset{w}{\text{minimize}}\, C(w) \right|, \tag{4.7}$$

where w denotes the minimum absolute value of the cumulative cost function.

4.2.2 Prediction

The network packets captured in real-time are classified as malicious or benign during the detection phase. During the detection phase, deep neural network predicts using the binary-classifier, representing the benign and malicious network activity. The proposed intrusion detection system calculates minimum cost parameter, w and meta-feature vector, f_v for the captured packets and compares them with the deep neural network classifier obtained during the training phase. Based on this comparison, the DNN assigns a binary-classification to the network packet which is under inspection. The classifier designates a logical value to the network packet, telling if the sample packet is benign or malicious, respectively.

The IDS is designed to handle a range of attack scenarios for a smart-home setting. Based on the attack scenario under consideration, the initial set of weights, w can be retrofitted for specific scenario. A supervised-learning templates consists of what is known as the transmission-mode information. The transmission-mode information can be applied to predict specific scenarios. It helps in identifying the scenario, for example, if a compromised device is broadcasting a message other than its availability or a connection-request then it can be an indication of a rank-based attack. Therefore, as a result, we use different supervised-learning templates, based on different meta-features and combined them with the rest of the training samples to predict specific attack scenarios.

Chapter 5

IMPLEMENTATION AND RESULTS

In this chapter, we discuss the implementation of the proposed intrusion detection system for IoT networks. The proposed intrusion system is based upon deep neural network which was implemented using the Keras, an open source neural network library written in Python and tested using the open Cooja network simulator developed in Contiki operating system [40]. The implementation of the system comprised of three stages, i.e., input data collection and preprocessing, creation and training of deep neural network classifier, and testing. Input data collection and pre-processing is stage where Cooja network simulator was used to generate IoT network-traffic dataset as an input for the anomaly detection process (ADP). Creation, and training of the deep neural network classifier are the core sub-processes in the ADP and the intrusion detection process in general. Training assigns weights to each classifier node to filter a certain type of input and matures the binary classifier. This chapter also provides the results of running the implementation of the proposed intrusion detection system for IoT networks.

5.1 Implementation

The proposes intrusion detection system is implemented in 4 files which are described in the *Table 5.1*, and attached to the appendix. Python is used to implement and test the deep neural network developed for the proposed intrusion detection system. It is relatively simple to use and easy to comprehend when compared to other higher level programming languages, such as, Java or C++. Moreover, matrix manipulation in “numpy” and “scipy” (i.e. arithmetic and matrix manipulations libraries in Python) is a fundamental component of Python. It considers a simple integer value as a matrix of one row and one column. This makes it easier to implement anomaly based intrusion detection, as the data in anomaly detection is usually represented in the form of a matrix.

Table 5.1: The files that make the proposed intrusion detection system

File Name	**Description**
generate_TR_RR_TRvRR.py	This file preprocesses the input data and generates the meta-features from the network transactions, such as, transmission-rate, reception-rate, duration, etc.
reader.py	This file preprocesses the input network traffic in real-time over real-network test-bed
TP_FP_TN_FN.py	This file represents the code which processes the raw results from DL_Classification.py and generates performance metrics, such as, true positives, false positives, true negatives, false negatives, f1 score, recall and precision
DL_Classification.py	This file comprises the code for the core machine engine, i.e. the deep neural network, implemented using the Keras library in Python.

We use Keras library because of its light-weight, modularity, and easy extensibility, and create a *Sequential* Deep-Learning model, constructed as a linear stack of DNN layers. We implement a *Sequential* Deep-Learning model because of its simplicity and adaptability to a resource-constrained environment, like IoT networks. In addition, Keras library is fast and can process large amounts of data easily. It automatically distributes the work over different processing threads with the machine, without the need of providing optimization or distributed processing parameters as in the case of other machine-learning libraries. This feature of the Keras library made it possible to implement an anomaly based intrusion detection system to be implemented on a low-powered resource constrained raspberry-pi, with a raw processing speed of approximately 700MHz and a volatile memory of 512 megabytes.

5.1.1 Data pre-processing

The IoT simulation dataset described in the previous chapter, consists of 5 million network transactions from the 6 sensors distributed in several locations in a smart home network simulation in Contiki's Cooja simulation. These network transactions are represented as

features(refer to [reference] chapter). For this research, we use Scapy, an open-source network penetration testing framework, to extract these features by stripping down each network packet.

Once downloaded, the dataset can be represented in a comma separated values (CSV) format. It can be opened using Microsoft Excel but for efficiency purpose, we used SweetScape 010 Editor. SweetScape 010 Editor is fast, efficient, and reliable in handling large datasets, of the order of a few GBs in size. The data would look like the data shown in the *Table 5.1.1.* The first 5 columns in this data represent packet sequence number, source IP, destination IP, and data-value, protocol type. These columns or attributes are only used towards calculating the transmission and reception rates for nodes. After removing the unwanted columns

Table 5.2: Raw Dataset

92	1	-	97	15.4	D 00:12:...:01:01	0xFFFF	IPHC	IPv6	ICMPv6 RPL DIO	AA00 ... AA00
96	1	-	97	15.4	D 00:12:...:01:01	0xFFFF	IPHC	IPv6	ICMPv6 RPL DIO	AA00 ... AA00
98	1	-	97	15.4	D 00:12:...:01:01	0xFFFF	IPHC	IPv6	ICMPv6 RPL DIO	AA00 ... AA00

from the datasets obtained from Cooja simulations, the new dataset would look like the data shown in the *Table 5.1.1*, i.e. a dataset with 4 columns or attributes, representing packet sequence number, source IP, destination IP, and data-value. The 5 million network transactions were pruned out by the input data-preprocessing program in *Appendix A* to make the input dataset of 59529 readings. It is important to note that these network simulations were gathered from two separate simulations, i.e. first simulation with all benign network transactions, and second simulation with a mix of malicious network transactions. Each network transaction in the second network simulation was marked as malicious as the entire network was affected by the malicious activities occuring within the network. Thus, each network

transaction was affected in some way. Therefore, out of these 59529 transactions, a total of 31046 network transactions were malicious while the rest of 28483 network transactions were benign. To test this research, we collected data from Cooja simulations to show that the

Table 5.3: Preprocessed Dataset

11581	1	6	02127401 ... 0C0A0700
11613	1	3	02127401 ... 0C0A0700
13206	5	6	02127401 ... 0C0A0700
13246	5	4	02127401 ... 0C0A0700

proposed intrusion detection system is scalable over six to seven and 16 - 17 IoT devices. We also tested the intrusion detection system over test-bed setup with real IoT sensors (using Texas Instruments 26xx), in which the proposed intrusion detection system was tested over wormhole attacks. For creating the test-bed over real IoT sensors, the data preprocessing was done in real-time using the program mentioned in *Appendix B*. In *reader.py*, we use the Scapy penetration testing library to extract features from the captured network packets. A sample output of the primary features, i.e. the raw features, has been enlisted in the end of the *reader.py*.

5.1.2 Deep Learning Algorithm Implementation

Deep learning is a type of machine learning algorithms, specifically a kind of feed-forward artificial neural network, which uses a succession of many layers of nonlinear processing units for data transformation. Entire intrusion detection was developed as a proof-of-concept over Raspberry-Pi 2, with 700 MHz clock speed and 512 megabytes of RAM.

In our deep learning model, each hidden-layer acts as an encode-filter that filters the convolved input and supply the filtrate to the next layer. In the proposed DNN model, each hidden-layer is developed with a set of neural nodes, uniformly distributed within them. Since, we use a deep neural network, each layer is required to use an activation function which can fire its neural nodes upon receiving the input. For the development of this system, we have used sequential deep-learning model in which each layer is configured to be activated by the Rectified Linear Unit (ReLU), i.e. linear-rectifier function. Functionality of the ReLU activation function has been explained in detail, later in the *Training Section* in this thesis.

Appendix D contains the Keras based implementation of the machine-learning algorithm, which involves the implementation of the deep-learning algorithm. During classification, *DL_Classification.py* reads the training dataset stored in a CSV format using the Pandas library, and stores it in a dataframe, and then converts the dataframe into a matrix, as:

```
df = pd.read_csv('dataset.csv', sep = ',', header = None)
dataset = df.as_matrix()
```

Further, these datasets were bifurcated into the training and testing datasets, by dividing this dataset in a ratio of two:three. Therefore, the training dataset comprised of 18989 benign network transactions and 20697 malicious network transactions, while the testing dataset comprised of 9494 benign network transactions and 10349 malicious network transactions. *DL_Classification.py* separates the dataset into training and testing datasets in the following code:

```
X = dataset[start:instances  ,1:]
X_Test = dataset[instances:  , 1:]
Y = dataset[start:instances  ,0]
```

Then, *DL_Classification.py* creates and instantiates a "Sequential" deep neural network with 3 hidden layers, and equips the processing units within each layer with ReLU activation function, as shown in the code:

```
model = Sequential()
model.add(Dense(14, input_dim = features, init='uniform', activation='relu'))
model.add(Dense(10, init='uniform', activation='relu'))
model.add(Dense(2, init='uniform', activation='sigmoid'))
```

Thereafter, the deep learning model is compiled and fitted with 150 runs, i.e. epochs, and the number of features. Finally, the deep-learning model is compiled and the classifier is assigned and saved in the variable "predictions". Consequently, in every test, the results of the classifier are normalized to a binary value, as described in *Chapter 4, section 4.1.*

5.1.3 Result post-processing

Once, the predictions are generated by *DL_Classification.py*, the predictions are transformed into concrete results by the program in *Appendix C*, which calculates the s, false positives, true negatives, and false negatives for each run. Finally, it writes these metrics into a CSV format file with the respective headers as the column headings and respective epochs as the row numbers.

5.2 Results

This section presents the results obtained from testing the proposed intrusion detection on Cooja Simulations for the sinkhole attack, distributed denial of service attack, blackhole attack, opportunistic service attack and wormhole attack.

In this section, we show graphical representation of the five metrics: recall, precision, F1 score, and the Training Time required by the proposed intrusion detection system over 150 separate runs. Each run comprises of two parts - training cycle and testing cycle.

5.2.1 Blackhole Attack

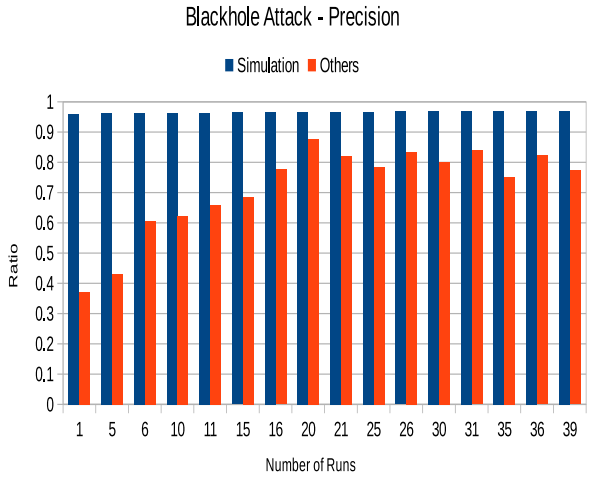

Figure 5.1: Precision for Blackhole Attack

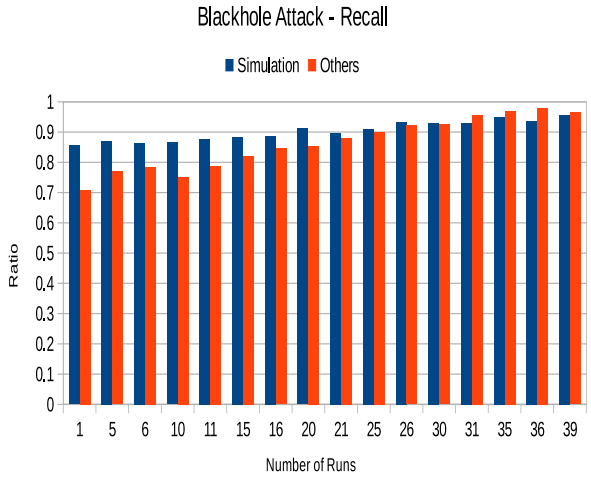

Figure 5.2: Recall for Blackhole Attack

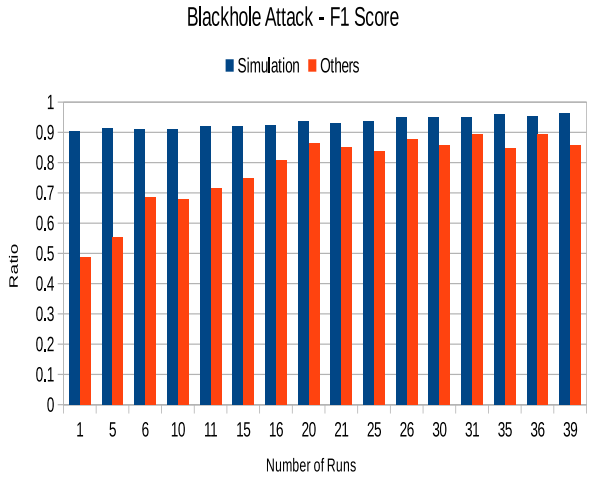

Figure 5.3: F1 Score for Blackhole Attack

Figure 5.1 and *Figure 5.2* compare the precision and recall between the proposed intrusion detection system and the other works for detection of blackhole attack. We can observe that other intrusion etection systems have an average precision or 0.7 whereas the proposed IDS shows a consistent precision of 0.9. But, the recall of our IDS is comparable to other works. *Figure 5.3* shows that the proposed IDS demonstrates a higher F1 score than other works consistently.

5.2.2 Opportunistic Service Attack

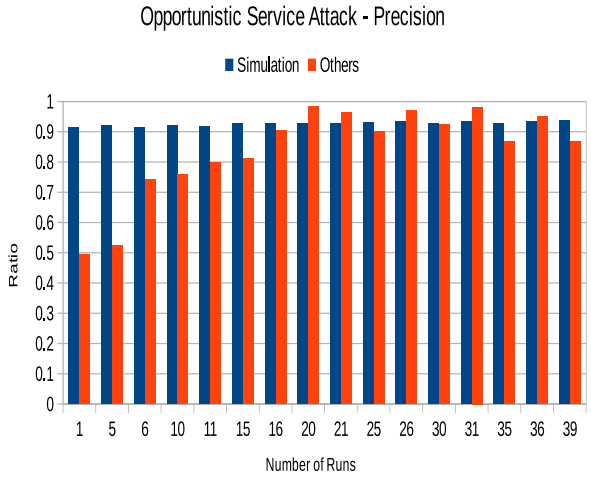

Figure 5.4: Precision for Opportunistic Service Attack

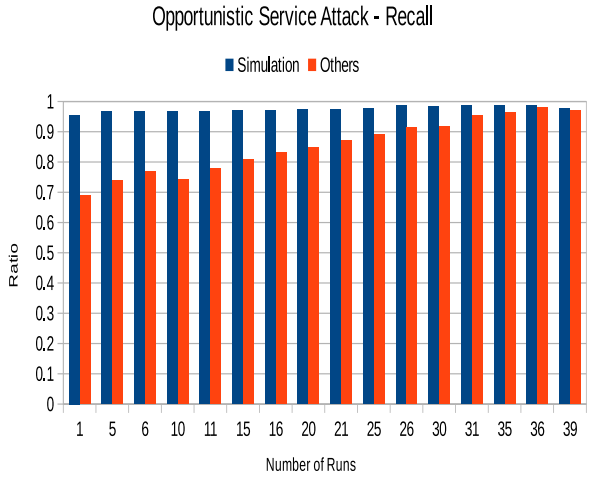

Figure 5.5: Recall for Opportunistic Service Attack

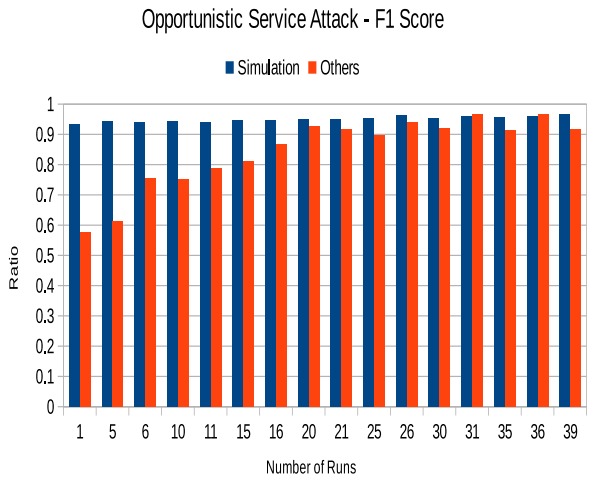

Figure 5.6: F1 Score for Opportunistic Service Attack

From *Figure 5.4* and *Figure 5.5*, we can observe that the precision for opportunistic service attack normalizes to 0.9 from the beginning but it is significantly higher than the precision shown by other IDSs. On the other hand, the proposed IDS shows a recall rate of 0.95 which is significantly higher than the recall rate of other intrusion detection systems. Due to a comparable precision, the proposed IDS and other IDSs demonstrate a comparable F1 score.

5.2.3 DDoS Attack

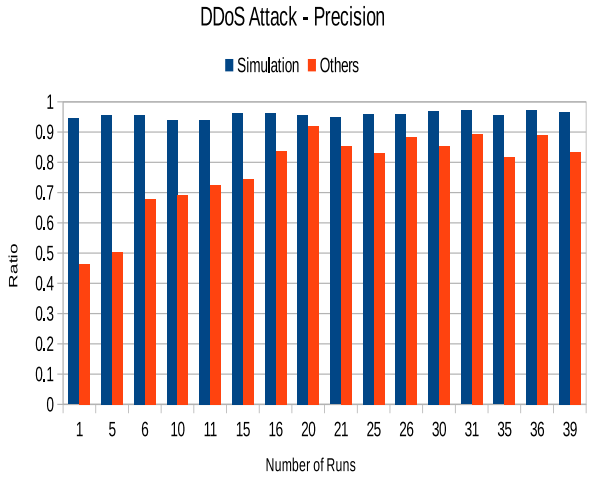

Figure 5.7: Precision for DDoS Attack

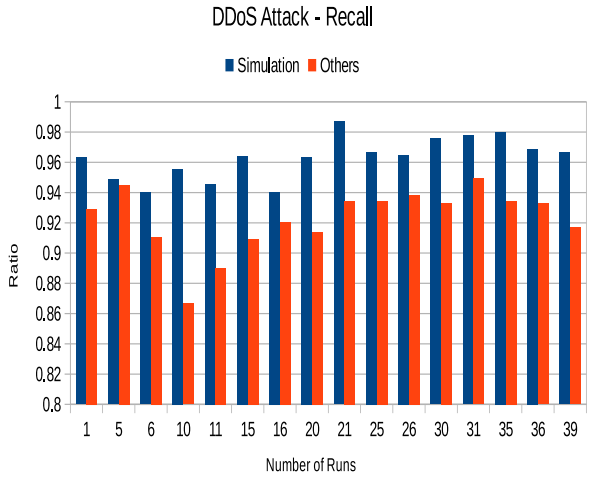

Figure 5.8: Recall for DDoS Attack

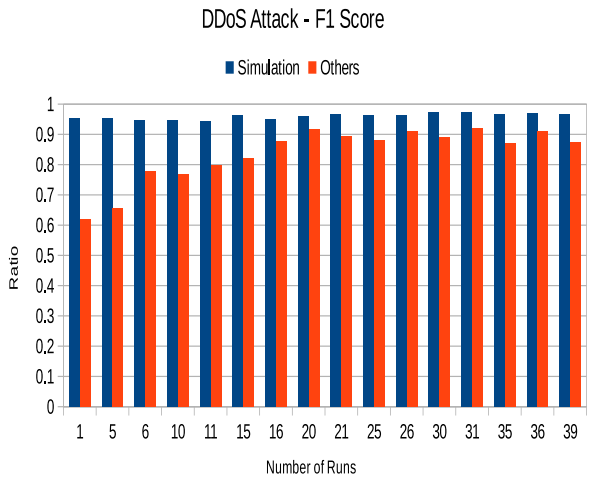

Figure 5.9: F1 Score for DDoS Attack

As shown in *Figure 5.7* and *Figure 5.8*, the proposed IDS demonstrates significantly higher precision (0.9) and recall (0.94) rate in comparison to other IDSs which showed 0.8 precision and 0.9 recall. It can be observed that all the intrusion detection techniques perform good against DDoS attack. But the F1 score for both, the proposed IDS and other IDSs lies in the range of 0.8 - 0.9. To establish the reliability of the proposed intrusion detection system, we tested it against sinkmhole attack and wormhole in test-bed created using real IoT sensors.

5.2.4 Sinkhole Attack

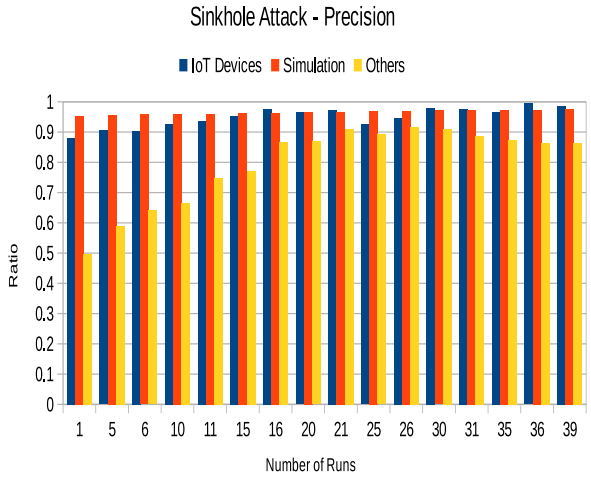

Figure 5.10: Precision for Sinkhole Attack

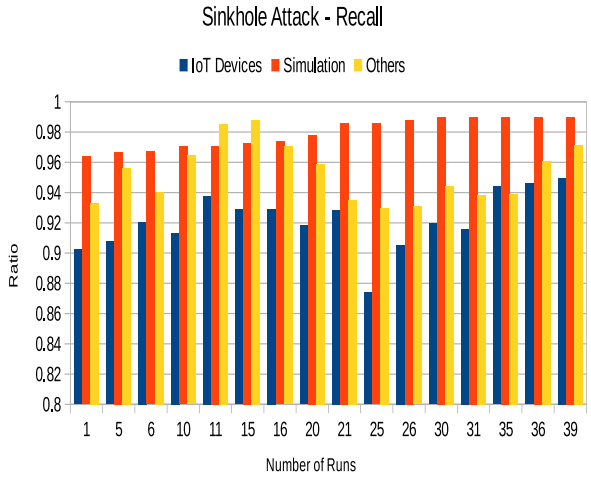

Figure 5.11: Recall for Sinkhole Attack

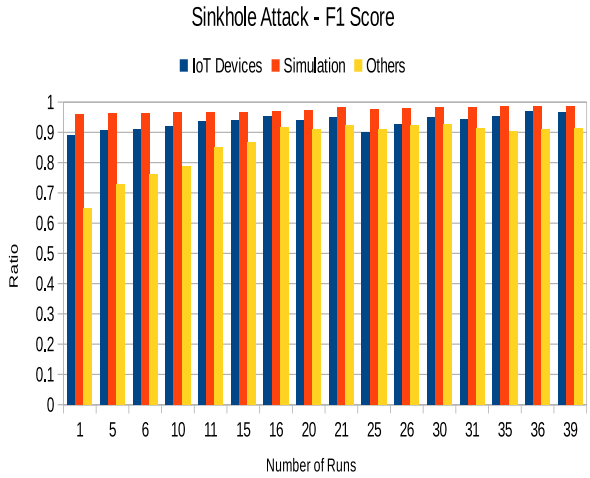

Figure 5.12: F1 Score for Sinkhole Attack

Figure 5.10 shows that precision for proposed IDS normalizes to 0.95 for sinkhole attack in both, simulation test-bed as well as IoT sensor test-bed. On the other hand, *Figure 5.10* shows that recall for IoT sensor test-bed dropped significantly, i.e. from 0.96 to 0.9. This drop can be attributed to lossy wireless transmission medium in IoT sensor test-bed. Despite the drop in recall rate, the F1 score was comparable in both the test-beds.

5.2.5 Wormhole Attack

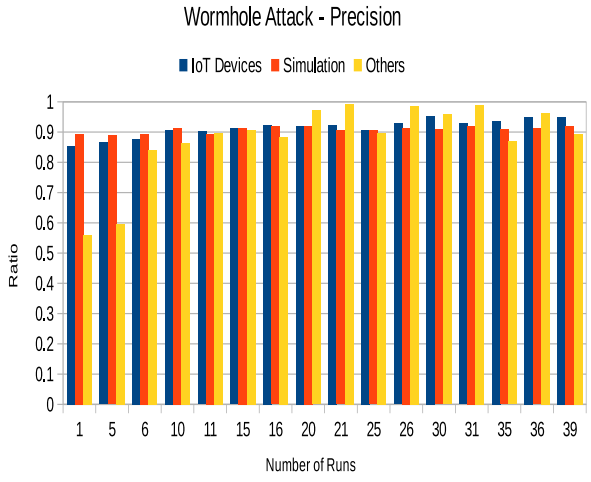

Figure 5.13: Precision for Wormhole Attack

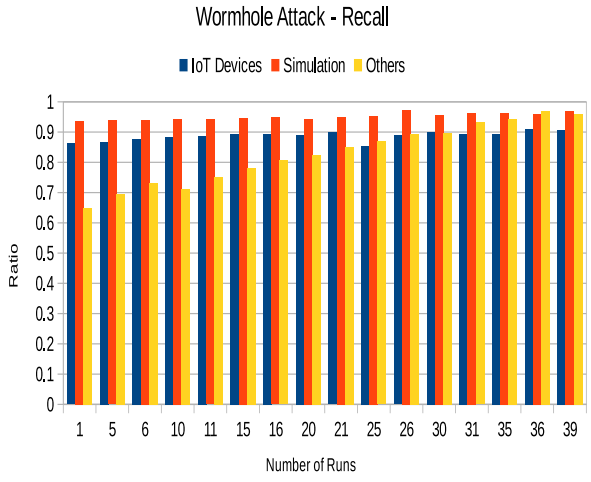

Figure 5.14: Recall for Wormhole Attack

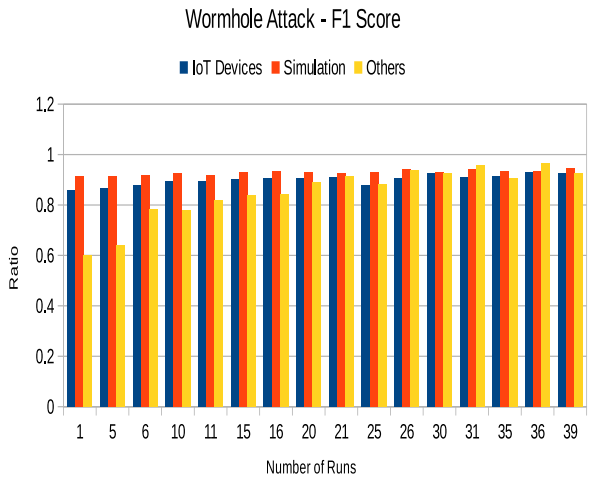

Figure 5.15: F1 Score for Wormhole Attack

We observe from the *Figure 5.13*, *Figure 5.14*, and *Figure 5.15*, that the precision, recall and F1 score values for our IDS and other works, normalizes to an average rate 0.9.

It is important to note that the IID was trained once before these tests and hence, the training time remains the same. It can be seen that the performance of the proposed intrusion detection system in real-network setup is not very different than over simulated network traffic. To draw a comparison with the literature, we also implemented other intrusion detection systems, based upon Inverse Weight Clustering, signature based, and rules-based techniques. We tested them for the above mentioned five attacks, in a simulated network environment.

Chapter 6

EVALUATION

This Chapter discusses the evaluation of the proposed intrusion detection system for IoT networks. It provides details about the performance metrics calculated for the proposed intrusion detection system in the light of the results presented in the previous chapter. The performance metrics can be calculate from the observed correct and incorrect predictions made by the system, and include overall accuracy, loss, sensitivity, precision, and F1 score.

6.1 Analysis

The proposed intrusion detection system, using deep-learning for anomaly detection in IoT networks was tested using the labeled testing dataset. The testing dataset comprises of 19843 (i.e. 33.333% of input dataset), wherein each record constitutes of the 6 values; transmission-rate, reception-rate, transmission-to-reception ratio, duration, transmission mode, source-IP, destination-IP, the data-value information, and the binary label information. The readings for duration, data-value information, source-IP, and destination-IP were originally obtained from the IoT sensors from the Cooja simulator (see *Chapter 5, section 5.1.1*), where as the values for transmission-rate, reception-rate, transmission-to-reception ratio, and transmission mode were calculated from other features obtained from the simulations (see *Chapter 4, section 4.1*). The binary label information represents if the network transaction is benign or malicious.

The Deep Neural Network was trained using labeled training dataset comprising of 39686 (i.e. 66.667% of input dataset). Thereafter, it was run against the testing dataset, but without giving it the binary label information that shows to which binary-classification each record in the testing dataset belongs. In other words, the Deep Neural Network was run against testing dataset without binary labels. Since, the testing dataset had 39686 of unlabeled records, the deep-learning model produces 39686 predictions in the form of “0” or “1”. Subsequently, the results obtained from the testing were compared with the original labels

of the testing dataset for each record, and the results show:

- The total number of predicted intrusions that were actually anomalies was 9836. Therefore, the true positives (TP) value of the intrusion detection system is:

$$TP = 9836 \tag{6.1}$$

- The total number of predicted intrusions that was actually normal traffic was 475. Therefore, the true positives (FP) value of the intrusion detection system is:

$$FP = 4.586964306\% \tag{6.2}$$

- The total number of predicted normal traffic that were actually anomalies was 225. Therefore, the true positives (FN) value of the intrusion detection system is:

$$FN = 2.373475973\% \tag{6.3}$$

To calculate the performance metrics of the proposed model such as sensitivity, precision, and F1 score, the following actual values are needed:

Considering D_t refers to the total number of records in the testing dataset, then

$$D_t = 19843$$

The actual number of normalities in the testing dataset is 9494. Let D_{normal} refers to actual number of normalities in the testing dataset, then

$$D_{normal} = 9494$$

The actual number of anomalies in the testing dataset is 10349. Let $D_{anomaly}$ refers to actual number of anomalies in the testing dataset, then

$$D_{anomaly} = 10349$$

The sensitivity or recall of the proposed intrusion detection system in detecting anomalies is 0.950465.

$$\frac{TP}{D_{anomaly}} = \frac{9836}{10349} = 0.950465 \tag{6.4}$$

The precision of the proposed intrusion detection system in raising alerts upon findings anomalies is 0.953932.

$$\frac{TP}{TP+FP} = \frac{9836}{} = 0.953932 \tag{6.5}$$

The F1 score of the proposed intrusion detection system is is 0.952195.

$$\frac{2 * (precision * recall)}{(precision + recall)} = \frac{1.81335795676}{1.904397} = 0.952195 \tag{6.6}$$

6.2 Discussion

Typically anomaly detection is evaluated by their accuracy and false positive rate. Nonetheless, other metrics of performance, such as the sensitivity, precision, and F1 score are also significant. However, the proposed deep-learning based intrusion detection system has shown 95.04% accuracy towards rightly flagging the abnormal network traffic in IoT network. The high percentage of accuracy indicates that utilizing deep-learning for anomaly based intrusion detection is capable of producing highly accurate results. Furthermore, using deep neural networks for classification has proven to be more accurate than using a clustering algorithm.

The centroids that IWC produces from running K-means algorithm, requires to find out an optimal number, k, of centroids using the hit-and-trial method. On the other hand, deep-learning based artificial intelligence produces better classifier with each run. The fine-tuned deep belief network also plays an important role in building the classifier at the beginning. A more accurate classifier gives less false alarms. Though, an intrusion detection system with a higher false alarm rate might not be used as it is costly to dealing with the alarms that indicate anomalies while they are no anomalies. Though, the proposed intrusion detection system showed a false positive rate of 4.58%. This value equals to the overall loss rate of the intrusion detection system which comprises a false negative rate.

Reflection on system's sensitivity or recall is also important. The sensitivity of the proposed intrusion detection system is 4.58% which means that the false negative rate of the deep-learning model is 2.37%. The false negative rate of the deep-learning model represents the chance that the anomalies will pass through the intrusion detection system undetected. Although, this percentage is not used as a representation of performance usually, but it is critical and depicts the level of the risk the device can take. While 4.58% is negligible and no intrusion detection system can bring this down to a zero percent, the criticality of the

IoT data determines whether this percentage can be accepted or not. For example, this percentage of miscalculations might be tolerable if these intrusions do not pose risk of a human life threatening situation.

In comparison with existing work [15, 1, 29, 39], overall accuracy (95.04%) of the proposed intrusion detection system is higher. Results show that the number of classifiers is irrelevant to IoT environment as binary classifiers seem sufficient to obtain accuracy in detection. Also, the proposed intrusion detection system performs better than traditional intrusion detection systems.

Chapter 7

CONCLUSION

A variety of security solutions are used to reduce security risks for IoT networks but the chance of having an IoT device being compromised, and playing in the hands of an attacker stands very high. There has been a shortage of research that attempts to find anomalies in the behavior of IoT networks which can be attributed to heterogeneous and dynamic nature of IoT networks. We attempt to fill this gap by developing an intrusion detection technique to realize its surrounding network, connect to its peer devices promiscuously and provide intrusion detection as a service for IoT networks.

7.1 Conclusion

The proposed intrusion detection system tactfully utilizes a combination of network virtualization and deep-learning algorithm to successfully adapt to the requirements of IoT networks. It employs different network adapters to intercept network traffic and translate their communication protocol to understandable format. This information is pre-processed to extract different features of network-traffic. These features are fed into a light-weight deep neural network which trains itself to create a binary-classifier which is used to classify malicious network traffic from benign network traffic during the testing/deployment phase. We test the proposed IDS over five different attacks: sinkhole, distributed denial of service, blackhole, opportunistic service, and wormhole attack and 150 runs. obtained an average of 85% accuracy for the detection rate. The detection rate shows a steady increase upto first 30 runs, before the accuracy normalizes at 95%. This indicates that the proposed intrusion detection system was completely trained within first 30 runs. It was observed that the proposed intrusion detection system demonstrated an average false-positives rate of 4% before it got trained completely.

This trend shows that the proposed intrusion detection system can provide effective an reliable security for IoT networks without consuming a lot of computational and power

resources. Based on the test results obtained from sensor-based real IoT network, we can conclude that it is both, practical, and feasible, to implement the proposed intrusion detection system to distinguish between genuine attacks and seemingly-genuine attacks in an IoT network.

7.2 Recommendation and Future Work

We recommend using the proposed intrusion detection system in IoT networks which can tolerate miscalculations of 4.58%, where having few aberrations will not lead to loss in human lives, critical infrastructure or data and assets.

Although the proposed IDS system is at proof-of-concept stage, it can be easily extended to a variety of potential attacks against the Internet of Things and it is highly likely that new attacks will be discovered in future. The IID system can be enhanced to detect location dependent attacks such as cloning of device ID, spoofing and sybil attacks. RPL specific misappropriation attacks, isolation attacks, neighbor attacks and direct attacks can also be detected by tracking device IDs and validating journal entries such as in DODAG (Direction-Oriented Directed Acyclic Graph) table. A more distributed and computationally optimized version of IID may be also be utilized towards identifying zero-day attacks.

BIBLIOGRAPHY

[1] A. Alghuried. *A Model for Anomalies Detection in Internet of Things (IoT) Using Inverse Weight Clustering and Decision Tree*. PhD thesis, February 2017.

[2] B. Arrington, L. Barnett, R. Rufus, and A. Esterline. Behavioral Modeling Intrusion Detection System (BMIDS) using Internet of Things (IoT) behavior-based anomaly detection via immunity-inspired algorithms. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, August 2016.

[3] Azamuddin. *Survey on IoT Security*. PhD thesis, December 2015.

[4] R. Broadhurst, P. Grabosky, M. Alazab, B. Bouhours, and S. Chon. An analysis of the nature of groups engaged in cyber crime. *International Journal of Cyber Criminology, Volume 8, Issue 1*, June 2014.

[5] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Usenix security symposium*. Usenix, August 2011.

[6] C. Cervantes, D. Poplade, M. Nogueira, and A. Santos. Detection of sinkhole attacks for supporting secure routing on 6lowpan for Internet of Things. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, May 2015.

[7] T. Y. Chung, I. Mashal, O. Alsaryrah, C. H. Chang, T. H. Hsu, P. S. Li, and W. H. Kuo. MUL-SWoT: A social web of things platform for Internet of Things application development. In *International Conference on Internet of Things (iThings)*. IEEE, September 2014.

[8] A. P. R. da Silva, M. H. T. Martins, B. P. S. Rocha, A. A. F. Loureiro, L. B. Ruiz, and H. C. Wong. Decentralized Intrusion Detection in Wireless Sensor Networks. In *Proceedings of the 1st ACM International Workshop on Quality of Service & Security in Wireless and Mobile Networks*. ACM, October 2005.

[9] F. M. de Almeida, A. R. L. Ribeiro, and E. D. Moreno. An Architecture for Self-healing in Internet of Things. In *The Ninth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*. UBICOMM, July 2015.

[10] K. P. Freeman and S. Klenner. *Progress in Mine Safety Science and Engineering II*. CRC Press, June 2015.

[11] A. A. Gendreau and M. Moorman. Survey of Intrusion Detection Systems towards an End to End Secure Internet of Things. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, August 2016.

[12] K. Gill. A zigbee-based home automation system. *IEEE Trans. on Consumer Electronics*, May 2009.

[13] W. B. Glisson, T. Andel, T. McDonald, M. Jacobs, M. Campbell, and J. Mayr. Compromising a Medical Mannequin. *arXiv:1509.00065*, August 2015.

[14] J. Han, H. Lee, and K. R. Park. Remote-controllable and energy-saving room architecture based on zigbee communication. *IEEE Trans. on Consumer Electronics, Volume 55, Issue 1*, May 2009.

[15] E. Hodo, X. Bellekens, A. Hamilton, P. L. Dubouilh, E. Iorkyase, C. Tachtatzis, and R. Atkinson. Threat analysis of IoT networks using artificial neural network intrusion detection system. In *Networks, Computers and Communications (ISNCC), 2016 International Symposium on*. IEEE, May 2016.

[16] F. Hosseinpour, P. V. Amoli, J. Plosila, T. Hmlinen, and H. Tenhunen. An Intrusion Detection System for Fog Computing and IoT based Logistic Systems using a Smart Data Approach. *International Journal of Digital Content Technology and its Applications(JDCTA), Volume 10, Issue 5*, December 2016.

[17] M. Kansra and P. D. Chadha. Cluster Based detection of Attack: IDS using Data Mining. *International Journal of Engineering Development and Research*, February 2016.

[18] Karthikeyan, A. Puri, R. Mathur, and A. Mishra. Internet of things (iot) based attendance and intrusion detection system. *International Journal of Innovative Research in Computer and Communication Engineering*, March 2016.

[19] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, and M. A. Spirito. Demo: An IDS framework for Internet of Things empowered by 6LoWPAN. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM, November 2013.

[20] K. F. Krentz. *Intrusion Detection and Prevention for the Internet of Things*. PhD thesis, June 2015.

[21] J. Krimmling and S. Peter. Integration and Evaluation of Intrusion Detection for CoAP in smart city applications. In *IEEE Conference on Communications and Network Security (CNS)*. IEEE, October 2014.

[22] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo. 6LoWPAN: a study on qos security threats and countermeasures using intrusion detection system approach: IoT, 6LoWPAN, RPL, QoS Security Threats, IDS. In *International booktitle of Communication Systems*. IEEE, September 2012.

[23] R. Martin. The Internet of Things (IoT)Removing the Human Element, December 2015.

[24] A. Mayzaud, R. Badonnel, and I. Chrisment. A Taxonomy of Attacks in RPL-based Internet of Things. *International Journal of Network Security, Volume 4, Issue 3*, May 2016.

[25] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, January 2015.

[26] P. Pongle and G. Chavan. Real time intrusion and wormhole attack detection in internet of things, volume 121, issue 9. *International Journal of Computer Applications*, July 2015.

[27] N. Provos, M. A. Rajab, and P. Mavrommatis. Cybercrime 2.0: When the Cloud turns Dark. In *Communications of the ACM*. ACM, March 2009.

[28] S. Raza and M. Hgskola. *Lightweight security solutions for the internet of things*. PhD thesis, June 2013.

[29] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time Intrusion Detection in the Internet of Things. In *Ad Hoc Networks*. Elsevier, November 2013.

[30] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah. A systemic approach for IoT security. In *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*. IEEE, May 2013.

[31] R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed Internet of Things. In *Computer Networks*. Elsevier, July 2013.

[32] A. Sforzin, M. Conti, F. G. Mrmol, and J. M. Bohli. RPiDS: Raspberry Pi IDS A Fruitful Intrusion Detection System for IoT. In *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*. IEEE, July 2016.

[33] T. Sherasiya, H. Upadhyay, and H. B. Patel. A Survey: Intrusion Detection System for Internet of Things. *International Journal of Computer Science and Engineering, Volume 5, Issue 2*, March 2015.

[34] C. Sinclair, L. Pierce, and S. Matzner. An application of machine learning to network intrusion detection. In *Computer Security Applications Conference, 1999. (ACSAC '99) Proceedings. 15th Annual*. IEEE, December 1999.

[35] S. Smith. IoT connected devices to triple to over 38bn units. In *Juniper Research*, 2015.

[36] O. Solon. Team of hackers take remote control of Tesla Model S from 12 miles away. *The Guardian*, September 2016.

[37] SRIA. Internet of things: From research and innovation to market deployment, 2014.

[38] H. Suo, J. Wan, C. Zou, and J. Liu. Security in the Internet of Things: A review. In *2012 International Conference on Computer Science and Electronics Engineering (ICCSEE)*. IEEE, March 2012.

[39] M. Surendar and A. Umamakeswari. InDReS: An Intrusion Detection and response system for Internet of Things with 6lowpan. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, March 2016.

[40] C. Thomson, I. Romdhani, A. A. Dubai, M. Qasem, B. Ghaleb, and I. Wadhaj. Cooja simulator, 2016.

[41] G. Wang, J. Hao, J. Ma, and L. Huang. A new approach to Intrusion Detection using Artificial Neural Networks and fuzzy clustering. In *Expert Systems with Applications*. Elsevier, September 2010.

[42] T. Xu, J. B. Wendt, and M. Potkonjak. Security of IoT systems: Design challenges and opportunities. In *Proceedings of International Conference on Computer-Aided Design*. IEEE, 2014.

[43] Y. Yu. Internet of ThingsAn Inconspicuous Naked Giant. December 2014.

[44] J. W. Zhao, Y. Hu, L. M. Sun, S. C. Yu, J. L. Huang, X. J. Wang, and H. Guo. *Method of choosing optimal features used to Intrusion Detection System in coal mine disaster warning Internet of Things based on Immunity Algorithm*. CRC Press, February 2013.

[45] K. Zhao and L. Ge. A survey on the Internet of Things security. In *2013 9th International Conference on Computational Intelligence and Security (CIS)*. IEEE, December 2013.

Appendix A

GENERATE_TR_RR_TRVRR.PY

```
from __future__ import division
import pandas as pd
import numpy as np
import math

rateCalcWindow = 2000          #milliseconds
filename = 'MalineObservation1_half.csv'
df = pd.read_csv(filename, sep=',',header=None)
array = df.as_matrix()
Nodes = 6

###TransmissionRate and ReceptionRate ###
transmissionRate = [[] for x in xrange(Nodes)]
receptionRate = [[] for x in xrange(Nodes)]

prevMin = -1
currMin = -1
i = 1
'''index for traversing array of csv data;
starting from 1 because the first row of the csv file is empty'''
while(i < len(array)):         #while loop 1
prevMin = currMin
currMin = math.floor(array[i][0]/rateCalcWindow)
#keep adding to same index, else change index or transmission & reception rat
```

```
if not currMin == prevMin :
x=0
while x<Nodes:        #while loop 2
transmissionRate[x].extend([0])
receptionRate[x].extend([0])
x+=1                  #integer increase for while loop 2

#Adding transmission-rates
if array[i][1] == 1:
transmissionRate[0][len(transmissionRate[0])-1]+=1
elif array[i][1] == 3:
transmissionRate[1][len(transmissionRate[1])-1]+=1
elif array[i][1] == 4:
transmissionRate[2][len(transmissionRate[2])-1]+=1
elif array[i][1] == 5:
transmissionRate[3][len(transmissionRate[3])-1]+=1
elif array[i][1] == 6:
transmissionRate[4][len(transmissionRate[4])-1]+=1
else:
transmissionRate[5][len(transmissionRate[5])-1]+=1

#Adding reception -rates
str = array[i][2]
strArray = str.split("|")
for string in strArray:
strInt = int(string)
if strInt == 1:
receptionRate[0][len(receptionRate[0])-1]+=1
elif strInt == 3:
receptionRate[1][len(receptionRate[1])-1]+=1
```

```
elif strInt == 4:
receptionRate[2][len(receptionRate[2])-1]+=1
elif strInt == 5:
receptionRate[3][len(receptionRate[3])-1]+=1
elif strInt == 6:
receptionRate[4][len(receptionRate[4])-1]+=1
else:
receptionRate[5][len(receptionRate[5])-1]+=1

#print strArray
i+=1    #integer increase for while loop 1

#Transposition of arrays
transmissionRate = np.array(transmissionRate).T
receptionRate = np.array(receptionRate).T

#Define headers
header = ['Node #1', 'Node #3', 'Node #4', 'Node #5', 'Node #6', 'Node #7']

#printing transmissionRate to a file
df = pd.DataFrame(transmissionRate, columns = header)
df.to_csv(filename.split(".")[0]+"_transmissionRate.csv", index=False)

#printing receptionRate to a file
df = pd.DataFrame(receptionRate, columns = header)
df.to_csv(filename.split(".")[0]+"_receptionRate.csv", index=False)

###TransmissionRate to ReceptionRate Ratio###
tRrR = [[] for x in xrange(len(transmissionRate))]
```

```
for i in range(len(transmissionRate)):
for j in range(len(transmissionRate[0])):
tRrR[i].extend([transmissionRate[i][j]/receptionRate[i][j]])

df = pd.DataFrame(tRrR, columns = header)        #printing tRrR to a file
df.to_csv(filename.split(".")[0]+"_TR_to_RR.csv", index=False)
```

Appendix B

READER.PY

```
#Author: Pavel Krivopustov and Zach Ruble, UWB
#Date Created: 05/20/2016
#Version: 1.1 (Real time packet stripping was implemented and writting to fil
#Purpose: Extract all layer tags from all packets in real time

from scapy.all import *
import os

try:
interface = sys.argv[1]
except:
interface = 'mon0'

try:
food = open(sys.argv[2], 'w')
except:
food = open('output.txt', 'w')

allTagsList = []
counter = []

try:
def PacketHandler(pkt):
#store all tags from all packets in a list
```

```
print "Packet ID: " + str(len(counter) + 1)
counter.append('1')
s=pkt.command()
curPkt=[]        #the current packet
layerName = []
tagTemp = []
tagName = []
curPkt=s.split(")/")    #split the packet string into layers
#for each of the layers
for i in range(len(curPkt)):
#extract the layer name
tempName = curPkt[i].split("(")
layerName.append(tempName[0])
#making sure the layer contains tags
if len(tempName) != 1:
#split the tag from the data
tagTemp = tempName[1].split("=")
else:
continue
assem = []
allTags = []
for ii in range(len(tagTemp)):
assem = (tagTemp[ii].split(", "))        #split the data and the next tag
#append only the tags to list
allTags.append(assem[-1])
#for all tags
for f in range(len(allTags)-1):
#making sure the tag is consisting of letters
checkTag = allTags[f]
finalTag = layerName[i] + "_" + allTags[f]       #combine layer name and the t
```

```
#prevent repetative layer tags to be added to the list
if finalTag not in allTagsList:
allTagsList.append(finalTag)
food.write(finalTag + "\n")
print "---" + finalTag

except Exception:
pass
print allTagsList
sniff(iface = interface, prn = PacketHandler)


'''Sample Output:
Ether_src, Ether_dst, Ether_type, IP_frag, IP_src, IP_proto,
IP_tos, IP_dst, IP_chksum, IP_len, IP_options, IP_version, IP_flags,
IP_ihl, IP_ttl, IP_id, TCP_reserved, TCP_seq, TCP_ack, TCP_dataofs,
TCP_urgptr, TCP_window, TCP_flags, TCP_chksum, TCP_dport, TCP_sport,
TCP_options, Raw_load, UDP_dport, UDP_sport, UDP_len, UDP_chksum,
NBNSQueryRequest_SUFFIX, NBNSQueryRequest_NSCOUNT,
NBNSQueryRequest_QDCOUNT, NBNSQueryRequest_ARCOUNT,
NBNSQueryRequest_FLAGS, NBNSQueryRequest_NULL,
NBNSQueryRequest_ANCOUNT, IPv6_nh, IPv6_src, IPv6_dst, IPv6_version,
IPv6_hlim, IPv6_plen, IPv6_fl, IPv6_tc, LLMNRQuery_c, LLMNRQuery_qr,
LLMNRQuery_an, LLMNRQuery_nscount, LLMNRQuery_qdcount, LLMNRQuery_ns,
LLMNRQuery_tc, LLMNRQuery_arcount, LLMNRQuery_ar, LLMNRQuery_opcode,
LLMNRQuery_qd, Padding_load, DNS_aa, DNS_qr, DNS_an, DNS_ad, DNS_nscount,
DNS_qdcount, DNS_ns, DNS_tc, DNS_rd, DNS_arcount, DNS_ar, DNS_opcode,
DNS_ra, DNS_cd, DNS_z, DNS_rcode, DNS_id, DNS_ancount, DNS_qd,
DNSRR_rclass, DNSRR_ttl, DNSRR_rrname, DNSRR_rdata, DNSRR_type, DNSRR_tc,
DNSRR_rd, DNSRR_arcount, DNSRR_ar, DNSRR_opcode, DNSRR_ra, DNSRR_cd,
```

DNSRR_z, DNSRR_rcode, DNSRR_id, DNSRR_ancount, DNSRR_qd, DNSRR_ad, DNSRR_nscount, DNSRR_qdcount, DNSRR_ns, Raw_CP, Raw_cid, Raw_mG, Raw_T, Raw_C, Raw_Q, Raw_B, Raw_F, Raw_m, Raw_h ’’’

Appendix C

TP_FP_TN_FN.PY

```
import numpy as np
import pandas as pd

filename = 'prediction.csv'
df = pd.read_csv(filename, sep=',',header=None)
array = df.as_matrix()

predictions = np.asarray(array)
TP = 0; FP = 0; TN = 0; FN = 0

for i in range(1,len(predictions)):
if int(predictions[i,1]) == 0 and int(predictions[i,2]) == 0:
TP += 1
elif int(predictions[i,1]) == 0 and int(predictions[i,2]) == 1:
FP += 1
elif int(predictions[i,1]) == 1 and int(predictions[i,2]) == 0:
FN += 1
else:    # int(predictions[i,1]) == 1 and int(predictions[i,2]) == 1:
TN += 1

confusionMetric = []; confusionMetric.extend([[TP, FP, TN, FN]])
#write to a csv
header = ['True Positives', 'False Positives', 'True Negatives', 'False Negat
```

```
df = pd.DataFrame(confusionMetric, columns = header)
df.to_csv("confusionMetric.csv", index=False)
```

Appendix D

DL_CLASSIFICATION.PY

```
# Create first network with Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
import pandas as pd
import sys

# fix random seed for reproducibility
seed = 147; numpy.random.seed(seed)

start = 0; nmbr_of_epoches = 150

# load pima indians dataset
df = pd.read_csv('dataset.csv', sep = ',', header = None)
dataset = df.as_matrix()

#initializing instances and features from dataset length
instances = len(dataset); features = len(dataset[0])

# split into input (X) and output (Y) variables
X = dataset[start:instances ,1:]
X_Test = dataset[instances: , 1:]
Y = dataset[start:instances ,0]
```

```
# create model
model = Sequential()
model.add(Dense(14, input_dim = features, init='uniform', activation='relu'))
model.add(Dense(10, init='uniform', activation='relu'))
model.add(Dense(2, init='uniform', activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac

# Fit the model
model.fit(X, Y, nb_epoch=nmbr_of_epoches, batch_size=features,
verbose=2)
# calculate predictions
predictions = model.predict(X_Test)
# round predictions
rounded = [round(x) for x in predictions]
print(rounded)
```