6-13-2024

FEDERATED LEARNING BASED AUTOENCODER ENSEMBLE SYSTEM FOR MALWARE DETECTION ON INTERNET OF THINGS DEVICES

Steven Edward Arroyo
Rowan University

FEDERATED LEARNING BASED AUTOENCODER ENSEMBLE SYSTEM FOR MALWARE DETECTION ON INTERNET OF THINGS DEVICES

by
Steven Arroyo

A Thesis

Submitted to the
Department of Computer Science
College of Science and Mathematics
In partial fulfillment of the requirement
For the degree of
Master of Science in Computer Science
at
Rowan University
May 17, 2024

Thesis Chair: Shen-Shyang Ho, Ph.D., Associate Professor, Department of Computer Science

Committee Members:
Anthony Breitzman, Ph.D., Professor, Department of Computer Science
Hieu Nguyen, Ph.D., Professor, Department of Mathematics

Acknowledgements

I want to acknowledge my thesis advisor Dr. Ho for all of his support and guidance. He taught me about the writing process, business, technical theory and most importantly creative out of the box thinking. Through numerous meetings and discussion he encouraged me to challenge myself and brought me to learning more than I ever thought possible.

I also acknowledge my friends (Robert F., 231 Laurel, Dominick V.) for there help in bringing this thesis to fruition. Through lending me resources to run my experiments on, listening to my ideas, and providing feedback. The never ending support was vital in my growth and completion of my thesis.

I also want to acknowledge my partner Phoebe Bennink. Her emotional support and care was vital in aiding the completion of my thesis. Through listening to my ideas and being there when things got busy and tough she encouraged me to keep moving forward.

I want to thank Rowan University for providing me 5 years of both undergraduate and graduate education I needed to bring this thesis into fruition. All the faculty and people separately helped me grow technically and emotionally to become the person that I am today.

Abstract

Steven Arroyo
FEDERATED LEARNING BASED AUTOENCODER ENSEMBLE SYSTEM FOR MALWARE DETECTION ON INTERNET OF THINGS DEVICES
2023-2024
Shen-Shyang Ho, Ph.D.
Master of Science in Computer Science

New technologies are being introduced at a rate faster than ever before and smaller in size. Due to the size of these devices, security is often difficult to implement. The existing solution is a firewall-segmented "IoT Network" that only limits the effect of these infected devices on other parts of the network. We propose a lightweight unsupervised hybrid-cloud ensemble anomaly detection system for malware detection. We perform transfer learning using a generalized model trained on multiple IoT device sources to learn network traffic on new devices with minimal computational resources. We further extend our proposed system to utilize federated learning such that IoT devices feed their output to a cloud server enabling more detection capabilities while keeping the network traffic secure on the device itself maintaining data privacy. We validate this system by creating a simulation testbed to conduct attacks on the IoT devices to evaluate how well the detection system works. We also compare transfer learning using multiple sources to a single source to show how the detection model of a target device is impacted by transfer learning. Empirical results on two datasets, one from the 2016 Mirai botnet attacks on IoT devices and the other from Gafgyt malware attacks on various IoT devices, show the competitiveness and feasibility of our proposed solution.

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

List of Figures

Figure Page

List of Figures (Continued)

Figure Page

List of Tables

Table Page

Chapter 1

Introduction

In this past decade, technology has continually improved exponentially, as people are integrating with it more than ever. Access to state-of-the-art technology at our fingertips will only continue this trend. At the center of this growth, Internet of Things (IoT) devices are the infrastructure delivering this change for both individuals and businesses. Accompanying this growth has been increased exploits and attacks on these IoT devices, which have cost billions of dollars in 2023 alone. Yearly reports from just the past few years have shown a 400 % increase in IoT device attacks, with a 1000% increase targeting universities and institutions alone [1]. Other reports show similar increases over the past few years [2, 3]. Based on these reports coupled with the rapid production of IoT, nearing 201 billion devices, this problem is only going to get worse [4].

There has been research done to solve this IoT device cybersecurity problem. Generally, some of this work involves communications with a cloud and sending network data, which could get intercepted causing a major vulnerability. The latest research focuses on federated learning, a decentralized system of IoT devices that sends model weights to the cloud to update a global model for the device status [5, 6, 7, 8, 9].

In this thesis, we describe and explain our proposed federated learning autoencoder decentralized anomaly detection system. This system uses minimal computationally intensive autoencoders in favor of using feature extraction and a variety of statistics to make an efficient solution. We perform transfer learning using a generalized model trained on a large dataset with millions of packet data from 9 IoT devices to learn network traffic on new devices with minimal computational resources. We perform federated learning to send our device model outputs in a secure manner to the cloud to enable more robust detection capabilities. We also create an attack test bed to evaluate the entire system as this is the

first step in moving the system out of the dataset and into the real world.

The preliminary proof of concept of the architecture presented in this thesis was first published in [10]:

- S. E. Arroyo and S.-S. Ho, "Autoencoder ensemble method for botnets detection on IOT devices," in 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA), December 2022, pp. 715–720.

A poster which provides an overview of the architecture in the thesis and covers the feasibility of this idea was published in [11]:

- S. E. Arroyo and S.-S. Ho, "A Hybrid-Cloud Autoencoder Ensemble Method for BotNets Detection on Edge Devices," in the 8th IEEE International Conference on Fog and Edge Computing (ICFEC), May 2024.

A journal paper titled "Robust Autoencoder Ensemble Method for Botnets Detection on Internet-of-Things Devices" covers the entire architecture to its entirety showing how the system performs on real time systems is currently under reviewed for the "Expert Systems and Applications" journal.

This thesis is organized as follows. In chapter 2 we briefly describe related works and background that makeup the core components of the system. In chapter 3 we describe in detail the design of the entire system from when a packet arrives to when it gets classified and sent to the cloud. In chapter 4, we discuss results of this system and impact of transfer learning using the Kitsune dataset for both Mirai and Gafgyt attacks. Lastly, chapter 5 discusses the creation of the test bed where real attacks are conducted onto simulated IoT Devices to measure performance of the system.

Chapter 2

Background and Related Work

2.1 Anomaly Detection

Anomaly detection of network traffic data is a well studied topic that has been used for the purpose of Intrusion Detection Systems (IDS) firewalls, router and IoT Security. The goal is to create a model of the data, understand the distribution, and flag inputs that vastly vary. The first network intrusion detection algorithm was created in 1986 with a focus on statistics to detect abnormalities in the network traffic [12]. Since then, with the increase availability in compute resources and efficiency along with the advancement of machine learning (ML) technologies, anomaly detection research has pivoted more toward machine learning based approaches [13]. Because predictive model created by these algorithms are powerful at classification tasks and analysis, ML has become an indispensable tool for anomaly detection tasks.

Generally, anomaly detection focuses on determining the differences between what is considered normal and abnormal which becomes a binary classification task. There have been numerous supervised and unsupervised algorithms created for dealing with these tasks [14, 15, 16, 17, 18]. The general architectures used for network intrusion anomaly detection are gated recurrent unit (GRU) [14], Long-Short Term Memory (LSTMs) [15], deep neural Networks (DNNs) [16] and autoencoders [17]. GRU and LSTMs are time-series based neural networks that learn their model weights through time. The advantage of these models is that they account for time at the expense of being more computationally complex. DNNs and autoencoders account for time through feature extraction where DNNs may take an autoencoder output as an input to feature extraction. These models are less computationally intensive and presented as competitive solutions.

2.2 Transfer Learning

Transfer Learning is a machine learning technique used in situations when there is not enough data to create a good predictive model for a target task. This is done typically by improving the model for a target task built off of a dataset related to the target task (e.g., image classification). This model will be trained on both data from the target task and its related tasks resulting in a better representation for the target task. Another advantage for using transfer learning is its ability to learn information in less epochs meaning less computation resources are necessary for learning [19, 20] .

In recent years, pre-trained models built on large datasets exist for many learning problems where transfer learning is necessary [19, 20, 21, 22]. However, there are not many for the IDS anomaly detection task because there is not one standard model being used. There is research being done on transfer learning using raw packet data [23] that shows transfer learning is plausible for the IDS task and can be built upon.

2.3 Federated Learning

Since the introduction of federated learning, researchers have been studying use cases in health, defense, and general cybersecurity [5, 24, 9, 8]. Historically, data would be offloaded to the cloud for compute clusters to create ML models. Federated learning mitigates the need for this by bringing a decentralized learning approach to machine learning that increases security on the devices. To achieve this, machine learning is done on the device itself instead of sending valuable statuses to the cloud. This prevents tampering with the data because the model is localized as well as one leverages compute resources that already exist. Then, the output of the model is encrypted and sent to the cloud for further analysis [5, 6, 7, 8]. In this case, the cloud takes this localized data and forms a generalized understanding of all the network traffic being presented.

There have been multiple algorithms that leverage federated learning for anomaly

detection of network traffic. Generally, these algorithms leverage the use of a machine learning model (GRU, LSTM, Autoencoder) on the device that act as inputs for a global model that updates the weight to create a full picture of the network traffic [5, 6, 7, 8, 9] . With the introduction of federated learning, there has been improved performance when compared to the same model without federated learning.

2.4 Autoencoders

Traditional autoencoders are artificial neural networks that minimize the input layers to a "bottleneck" which is the smallest possible representation of the data in the architecture. This smallest representation gets reconstructed to a full representation of the input data.

Autoencoders are often used in anomaly detection for their ability to create low dimensional representations extracted from high dimensional features. Because of this, many recent algorithms for anomaly detection use autoencoders to feed these representations into another neural network to create a final classification [25, 26, 27, 28]. Autoencoders are also often used for their ability for unsupervised learning. This is because loss is based on the training input data and their output reconstruction without the need for labels.

Much of the latest anomaly detection research takes the basic architecture and modifies it to try and create stronger representations. This often leads to competitive, powerful results that are efficient and can be used for the IoT problem [29, 30].

2.5 Damped Incremental Statistics

Damped Incremental Statistics is a term popularized by the Kitsune model to describe the feature extraction algorithm used to extract network packets into 115 features [31]. Generally, the network packet would get stored into 23 statistics mapping with the Mac Address Layer, IP Layer, and Transport layer of the OSI model. From here, these 23 statistics are extracted with 5 different damping factors from windows of time as far back as

a minute [32]. These make up the 115 features used to describe a network packet. This feature extraction method used is powerful for time series statistics. The extracted features are good for reducing space complexity because only a few features need to be retained over time [33].

Chapter 3

Proposed System and Methodology

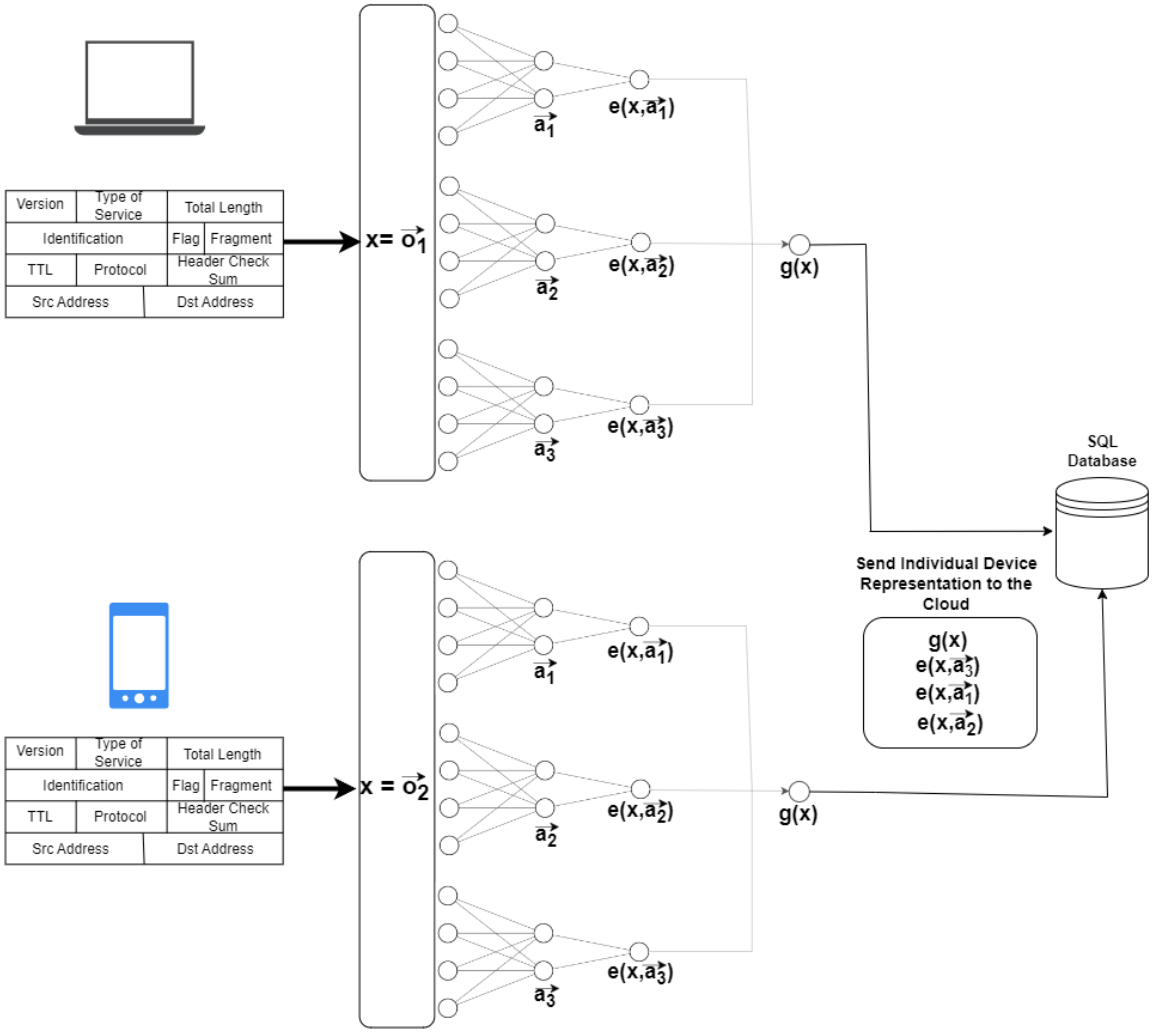
In this chapter, we describe and explain our proposed federated learning autoencoder decentralized anomaly detection system in detail. The system uses minimal computationally intensive autoencoders in favor of using feature extraction and a variety of statistics to achieve an efficient solution.

3.1 System Overview

Figure 1 shows a general overview of how the proposed anomaly detection system works for a single network packet during execution. When a packet is observed, the Damped Incremental Statistics $\vec{o}$ (see subsection 3.2.2) are computed. We cluster these features into multiple non-overlapping feature sets that are fed into their respective autoencoder $\vec{a}_i$. Then all the encoders will make a vote $g(x)$ as to whether the packet is anomalous. Each vote is stored over a period of time and sent to a cloud database and removed from the devices. The vote data sent to the cloud is used to construct a statistical model to happening across the entire network.

Figure 1

Overview of How the Proposed Anomaly Detection System Works on a Simple Two-Device Network When One Network Packet Is Observed



3.2 Feature Extraction and Packet Reader

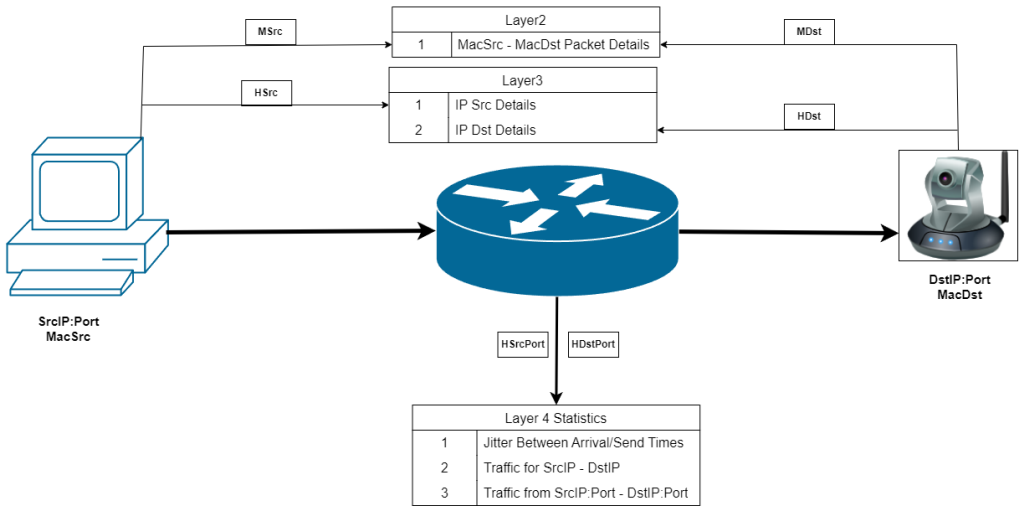
3.2.1 Overview

The feature extractor derives multiple statistics from a packet in a way that is computationally and spatially efficient. The data that comes in is simply the packet size with header information describing the two devices communicating with one another. The ob-

jective is to map this packet data with the header in a way that we can derive meaningful statistics about a packet. Kitsune and our feature extractor map the packet data to the Open Systems Interconnection (OSI) model [32]. The main difference is that our feature extractor extrapolates differences about destination packets as well to create additional features for discovering anomalies. Figure 2 shows how we map to the OSI model using just the packet data and the information from the header. This can be extrapolated to any source-destination combination. MAC addresses are used to generate statistics for packets at Layer 2. IP addresses are used at Layer 3. In Layer 4, we map to the source and destination devices during transmission to track the communications. By using these mappings, we represent what is happening at each of these layers.

Figure 2

Feature Extraction by Mapping Packet Data and the Information from the Header to the OSI Model



3.2.2 Damped Incremental Statistics Usage

Incremental Statistics (*IS*) [32] are snapshots of fundamental statistics about the data without storing all the data over time. Instead, the processed data used to derive

statistics are stored in a tuple form which is much more space-efficient.

Let P be a set of packets $\{p_1, p_2, p_3, ..., p_n\}$ for a particular device. Note that P is of fixed set size when training offline; however, when training online (or real-time) data P is dynamic and typically the number of packets received from $n-1$ seconds ago. This means that when training in real time, p_n is the latest observation.

Table 1 shows the fields we extract from each packet p. In particular, TOA is the time of arrival that we receive p and m is the size of the data in packet p. We use the extracted information to create our Incremental Statistics. These extractions provide features that we can use to create our mappings and descriptors to gain insights into what is happening during packet communication.

Table 1

Information Extracted from Each Packet Data

Packet Field	Description
m	packet size
$isrc$	IP Source Address
$idst$	IP Destination Address
$msrc$	Mac Source Address
TOA	Time of Arrival
$srcprt$	Source Port
$dstprt$	Destination Port

We would like to track how consistent a device is communicating with the hosts and if there are any sudden changes. For this reason, we extract the so-called 1-Dimensional Statistics ($1D$). We focus on monitoring both the TCP/IP connections as well as MAC connections. We monitor TCP/IP and MAC connections separately for the scenarios where a MAC address changes for a particular IP (i.e., ARP poisoning). There are also instances where some devices may be talking with layer 2 protocols that would be missed if we didn't monitor the MAC addresses. We want to use $msrc, mdst, isrc, idst$ (See Table 1) for tracking

and to create at least 2 different 1D instances (MAC and IP). Table 2 shows the extracted $1D$ and the statistics (DS) we want to derive for each packet. Note that $1D$ are computed after each observation p and DS are simply equations that do not need to be calculated until they are necessary which saves space.

Table 2

1-Dimensional Features Extracted for Each Packet

1-Dimensional Statistics ($1D$)	**Formulation**
Number of Instances (NOI)	$\|P\|$
Linear Sum (LS)	$\sum_{i=0}^{\|P\|} m_{p_i}$
Sum of Squares (SS)	$\sum_{i=0}^{\|P\|} m_{p_i}^2$
Time Last Observed (TLO)	$\sum_{i=1}^{n} TOA_{p_i} - TOA_{p_i}, TOA_{p_0} = 0$
Derivable Statistics(DS)	**Formulation**
μ	$\frac{LS}{NOI}$
σ^2	$\frac{SS}{NOI} - \mu$
σ	$\sqrt{\sigma^2}$

The motivation for 2-Dimensional statistics ($2D$) is much more niche than that of a $1D$. We have our features for TCP/IP and MAC that we derived in $1D$; however, how do we detect more subtle differences in traffic? For instance, if you are consistently talking back and forth with an IP sending large masses of data to one another then this IP sends a small packet to a different port. With the $1D$ features, we can't detect something like this because the 1D features do not account for each socket/port on a device. $1D$ features also don't account for situations where the data is consistent but the communications become scattered. $2D$ is introduced to handle situations like the aforementioned where we can track all overall packet communication between 2 devices as a stream that we can derive further statistics from.

Table 3 shows what is within each 2-Dimensional statistics and how they are derived. k, j are $1D$ statistics that describe communications between the src device and destination device respectively. These communications can be IP - IP communications as well as IP:PORT - IP:PORT communications.

Table 3

2-Dimensional Features Extracted for Each Packet

2-Dimensional Statistics ($2D$)	**Formulation**
Number of Instances (NOI)	$\|P\|$
k, j	$k \in 1D, j \in 1D$
Linear Sum (LS)	$LS_{k,j}$
Sum of Squares (SS)	$SS_{k,j}$
Residual Sum Of Squares (RSS)	$\sum_{i=0}^{n}(m_{p_i} - \mu_k) * (m_{p_i} - \mu_j)$
Time Last Observed (TLO)	$\sum_{i=1}^{n} TOA_{p_i} - TOA_{p_i}, TOA_{p_0} = 0$
Derivable Statistics ($DS2D$)	**Formulation**
μ	$\frac{LS}{NOI}$
σ^2	$\frac{SS}{NOI} - \mu$
σ	$\sqrt{\sigma^2}$
$\|2D\|$	$\sqrt{u_i^2 + u_j^2}$
R	$\sqrt{\sigma_i^2 + \sigma_j^2}$
Cov_{ij}	$\frac{SR}{NOI_i + NOI_j}$
r_{ij}	$\frac{Cov_{ij}}{\sigma_i + \sigma_j}$

Lastly, we want to cover Jitter between 2 devices. Jitter is the amount of time that it takes for data from any of the IPs to be received and is integral in detecting Man in The Middle (MITM) and Botnet attacks. This is because both attacks are time-based and can be detected with Jitter observers. To calculate Jitter we make a $J1D$ with a $2D$ input. We do this to extract the TLO which will serve as our primary value to compute LS and SS statistics instead of m. Table 4 shows how we extract Jitter as a $J1D$ observation.

Table 4

Jitter Observations for Each Packet

Jitter Statistics ($J1D$)	**Formulation**
q	$q \in 2D$
Number of Instances (NOI)	$\|P\|$
Linear Sum (LS)	$\sum_{i=0}^{\|P\|} TLO_{q_{p_i}}$
Sum of Squares (SS)	$\sum_{i=0}^{\|P\|} TLO_{q_{p_i}}^2$
Derivable Statistics(JDS)	**Formulation**
μ	$\frac{LS}{NOI}$
σ^2	$\frac{SS}{NOI} - \mu$
σ	$\sqrt{\sigma^2}$

An Incremental Statistic IS is a set of 3 DS (MAC, TCP/IP, Jitter) and 2 $2D$ (IP-IP and IP:Port $\rightarrow$ IP:Port)

$$IS = \{DS_{MAC}, DS_{TCP/IP}, JDS, DS2D_{IP-IP}, DS2D_{IP:Port \rightarrow IP:Port}\} \tag{1}$$

resulting in 23 unique statistics derived for each packet observed. In other words, IS is a

23-element vector.

As is, we gain insights about packets in real time disregarding TOA. We want to look at past and current instances more accurately to predict attacks. Here is where Kitsune introduces a Dampening factor that shows 5 windows of time 100ms, 500ms, 1.5 sec, 10 sec, and 1 min into the past $\lambda = (5, 3, 1, .1, .01)$ [32]. To achieve this, we create the dampening windows for IS as follows.

$$damp(\lambda, IS) = 2^{-\lambda TLO_{IS}} IS \tag{2}$$

We apply this dampening factor to IS 5 times for each window to create our input feature vector $\vec{o}$ of size 115.

$$\vec{o} = \{damp(\lambda_1, IS), damp(\lambda_2, IS), ..., damp(\lambda_5, IS)\}. \tag{3}$$

$\vec{o}$ is what we feed into our autoencoder to gain insights.

3.3 Anomaly Detection Model

3.3.1 Overview

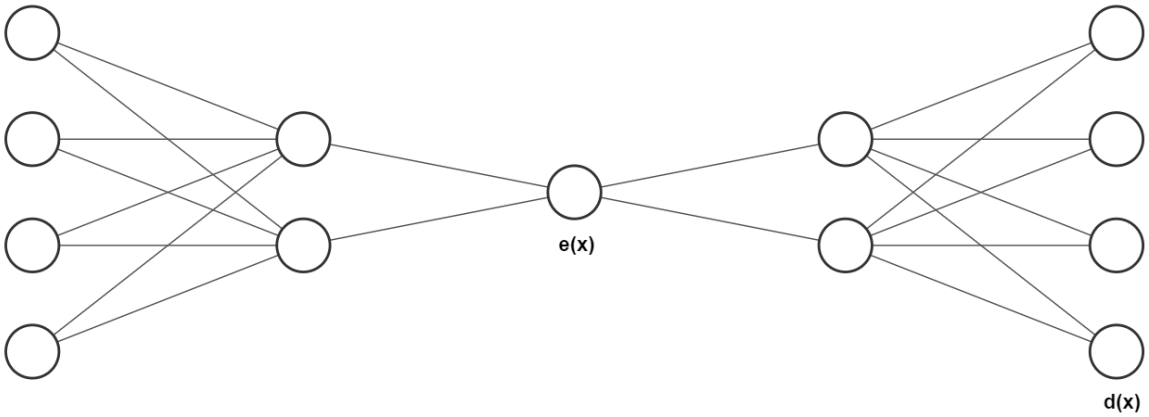
The anomaly detection system is comprised of an ensemble of encoders to find patterns in their respective cluster, an algorithm for deciding the architecture of the neural network, and a statistical model to form a distribution and learn the low dimensional representation. When a new packet is introduced a consensus is taken among the encoders that evaluate if the packet is anomalous. The advantage of this anomaly detection system is to use forward propagation sparingly by introducing a statistical threshold model to determine the anomaly efficiently.

The number of encoders in the ensemble is decided based on the clusters discovered during feature extraction. This describes the unique high-dimensional patterns that are

extracted from the original temporal data. One purpose of autoencoders is to simplify the input data x as much as possible with the least amount of loss for pattern discovery. For this model, the input data x is reduced to one dimension. In Figure 3 we see the size of each layer decrease by half until a single dimension representation $e(x)$ before reconstructing the pattern $d(x)$ of the full extracted dimension . We use the Mean Squared Error (MSE) of $d(x)$ compared to the input data x to optimize the autoencoder. We use $e(x)$ to learn our statistical model.

Figure 3

Autoencoder Architecture



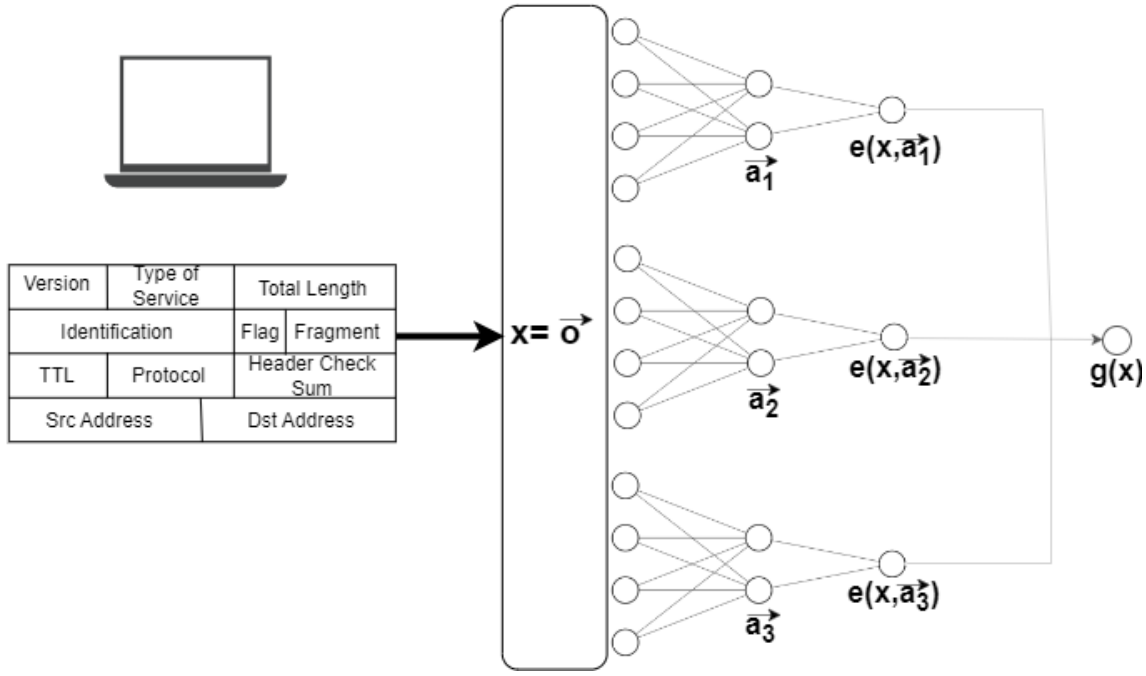
When the autoencoder is trained and the encoder is split off, the encoder makes a prediction for the entire training data. Then we take these predictions and learn the parameters of a Galton distribution which is our threshold model because previous work has shown network traffic is typically log-normal [34]. Using the learned Galton distribution, we obtain thresholds for anomaly detection as opposed to previous work [32] which feed the encoder value into another neural network for detection. Our proposed method reduces computations during real-time scenarios.

Figure 4 shows a general overview of anomaly detection using an ensemble of three autoencoders for a single packet for a single device. When a packet is observed,

we compute the damped Incremental Statistics $\vec{o}$. We cluster these features into multiple non-overlapping feature sets that are fed into their respective autoencoder $\vec{a}_i$. Then all the encoders will make a vote $g(x)$ as to whether the packet is anomalous.

Figure 4

Evaluating a Single Packet p Using the Proposed Autoencoder Ensemble for Anomaly Detection



3.3.2 Ensemble Size Determination based on Clustering

Let T be a matrix where each *row* is a single observation $\vec{o}$ and each column (col) is a feature of $\vec{o}$ (of size 115). $T = \{\vec{o}_1; \vec{o}_2; \vec{o}_3; ...; \vec{o}_n\}$ such that n is the number of observations seen in P. The average for each col is stored in a vector $\vec{MT} = \{\mu_1, \mu_2, \mu_3..., \mu_{|col|}\}$ and each corresponding standard deviation in a vector $\vec{ST} = \{\sigma_1, \sigma_2, \sigma_3..., \sigma_{|col|}\}$. Let T_{norm} be a normalized set of T where we normalize by using $\frac{col - MT_{col}}{ST_{col}}$ for each column of T (col). We use $\vec{MT}$ and $\vec{ST}$ later to normalize our execution matrix E_{norm} (See subsection 3.3.4) as

well.

We use a correlation distance matrix T_{corr} to cluster based on similarity. For this, we use Pearson correlation. We compute each value of T_{corr} as follows.

$$T_{corr}(x,y) = 1 - \frac{\sum_{row=0}^{n}(x_{row}-\bar{x})(y_{row}-\bar{y})}{\sqrt{\sum_{row=0}^{n}(x_{row}-\bar{x})\sum_{row=0}^{n}(y_{row}-\bar{y})}} \tag{4}$$

such that x and y are features of T and $\bar{x}$ and $\bar{y}$ are the means of the feature values. We use Wards algorithm [35] to minimize variability between clusters to create our dendrogram for hierarchical clustering. We choose a threshold that defines the minimum distance or "cutoff" for a feature to be a part of each cluster c_i. This threshold is used to decide the number of clusters, K which corresponds to the number of autoencoders in the ensemble.

Let $C = \{c_1, c_2, c_3 ... c_K\}$ be the set of K clusters. Each cluster c_i has a corresponding data matrix D_i and an autoencoder $\vec{a}_i$ to be trained using D_i. Prepossessing and clustering takes place once before training. During execution, these clusters discovered are applied to the new feature set.

Algorithm 1 Learning Ensemble of Autoencoders

Require: $iter \geq 0$

0: **for each** $\vec{a}_i$ **do**{Go through each autoencoder}

0: **for** $t \leq iter$ **do**

0: $y = d(r^0, \vec{a}_i)$ {Forward Propagation (r^0 is a random sample of D_i)}

0: $\hat{y} \leftarrow r^0$

0: $E_{in} = MSE(y, \hat{y})$

0: $W(t+1) = SGD(E_{in}, W(t))$

0: **end for**

0: Set $\mu_{\vec{a}_i}, \sigma_{\vec{a}_i}$

0: **end for**=0

3.3.3 *Autoencoder Training*

The input layer of $\vec{a}_i$ takes the data from D_i as input. Let r^0 be a single data instance of D_i fed into the input layer with dimensionality d equal to the dimensionality of D_i. For layer l,

$$r^l = R(W^l r^{l-1} + b^l) \tag{5}$$

such that the Rectified Linear Unit (ReLU) activation function

$$R(m) = \begin{cases} m, & m > 0 \\ 0, & otherwise. \end{cases}$$

and W^l and b^l are weights and bias at layer l and r^{l-1} is the output of the previous layer $(l-1)$. The bottleneck is calculated as

$$e(x, \vec{a}_i) = e(x) = W^{\frac{L}{2}} r^{\frac{L}{2}-1} + b^{\frac{L}{2}} \tag{6}$$

and the final decode layer,

$$d(x, \vec{a}_i) = d(x) = W^L r^{L-1} + b^L \tag{7}$$

L is the number of the autoencoder layers excluding $l = 0$ and L is assumed to be even, without loss of generality. The above equations describe how the model forward propagates to give an estimated output.

To update $W(t)$, we use Stochastic Gradient Descent $SGD(E_{in}, W(t-1))$ with $E_{in} = MSE(d(x), y)$ such that y is the input data from D_i which happens to be r^0 in this case and $d(x)$ is our autoencoder output. Once W is learned for $\vec{a}_i$, we estimate the thresholds used

to classify anomalies for the particular autoencoder $\vec{a}_i$ as

$$\mu_{\vec{a}_i} = \frac{\sum_{j=1}^{|M|} e(x_j, \vec{a}_i)}{|M|} \tag{8}$$

and

$$\sigma_{\vec{a}_i} = \frac{\sum_{j=1}^{|M|} e(x_j, \vec{a}_i) - \mu_{\vec{a}_i}}{|M|} \tag{9}$$

where $|M|$ is the total number of instances of M and x_j is an instance of M.

3.3.4 Classifying Unseen Instances

Let E be the set of unseen instances. E is represented as a matrix where each *row* is a single observation $\vec{o}$ and each feature column (col) is a feature of $\vec{o}$ similar to T in subsection 3.3.2. Let E_{norm} be a normalized version of E using $\frac{col - MT_{col}}{ST_{col}}$. Similarly, for each cluster $c_i \in C$, $\vec{a}_i$ is an autoencoder trained on D_i. $E_i \subseteq E_{norm}$ such that E_i contains the features that are specific to cluster c_i. To classify an unseen instance x, we give each autoencoder $\vec{a}_i$ a vote,

$$h(x, \vec{a}_i) = \begin{cases} 1, & |e(x, \vec{a}_i) - \mu_{\vec{a}_i}| > df * \sigma_{\vec{a}_i} \\ 0, & otherwise. \end{cases} \tag{10}$$

df is a user-defined tuning parameter that decides if the anomaly threshold exceeds a certain number of standard deviations. x is a single execute instance of E_i. Next, we get a total vote count from the ensemble of autoencoders,

$$c(x) = \sum_{i=1}^{K} h(x, \vec{a}_i) \tag{11}$$

Then we make the final classification decision for x

$$g(x) = \begin{cases} Anomaly, & c(x) > \frac{K}{2} \\ \\ Benign, & otherwise. \end{cases} \tag{12}$$

3.4 Incremental Learning System

3.4.1 Overview

Network traffic is variable in terms of distribution and frequency across even similar devices. However, training a model from scratch takes time. Transfer learning is introduced to take a pretrained model and tweak it to the specifications of the digital fingerprint of the new device's network traffic. The incremental learning system creates an adaptive anomaly detector model by introducing new data to the pretrained model. Overtime, a new fingerprint is created that adapts to the devices traffic.

To create a robust detection model, we first have to introduce a model that is good at detecting abnormalities for all IoT devices across the data set. To do this, we create a larger data set D combining multiple IoT device fingerprints. For this, there are 9 IoT Devices with roughly 1 million packets (anomalies included) per device. Roughly 143,000 thousand packets are taken across each IoT device to create our detection model. Then an Anomaly Detection Model is formed using the data from the 7 IoT devices with evaluations of anomalies for each device. Because the model is general, the distribution threshold will also be general. As the model adapts to the device, this threshold may need to be adjusted.

When the detection model is trained and ready for deployment, Incremental learning to a target data device is the next phase. As new packets are introduced, the encoder goes through a learning phase for 15 epochs. At the end of learning, the distribution is recompiled by taking a weighted average of the new model outputs and detection model

outputs. The standard deviations are combined as well as shown in Equation 13. This combined distribution ensures that the adaptive model doesn't forget the previous distribution it learned on.

3.4.2 Transfer Learning

Earlier results [10] show that with extended training time, the model can learn traffic for any particular device. Transfer learning is performed to minimize training time by adapting a general pre-trained model that can detect anomalies for all devices with data in D. We again use the approach described in subsection 3.3.2 to determine the number of clusters and ensemble size for our general model that will be applied to each target device. For each cluster $c_i \in C$, there exists a TL_i containing all the feature columns and instances associated with c_i at some (future) time instance t' for a target device that does not exist in D. $\vec{a}_i$ is a detection model at some time instance t such that $t < t'$. We repeat Algorithm 1 where r^0 is now an instance of TL_i.

We modified Line 8 in Algorithm 1 to replace the computed sample means and standard deviations with the weighted means and standard deviations computed using Equation 13 to account for the weights of the autoencoder and the distribution of all the IoT devices. These modified statistics describe the one detection model for the target device. This weighted model is constructed using the weighted mean and standard deviation to combine the previous and new distributions. To minimize space complexity, we use Incremental Statistics to update $\mu_{\vec{a}_i}, \sigma_{\vec{a}_i}$ of the current model derived from data observed so far. In our description below, we use $|\vec{a}_i|$ to define the total number of instances observed so far for an autoencoder.

Let the sample mean and the variance of the new model for each autoencoder be $\delta\mu_{\vec{a}_i} = \frac{\sum_{j=1}^{|TL_i|} e(x_j,\vec{a}_i)}{|TL_i|}$ and $\delta\sigma^2_{\vec{a}_i} = \frac{\sum_{j=1}^{|TL_i|} (e(x_j,\vec{a}_i)-\delta\mu_{\vec{a}_i})^2}{|TL_i|-1}$ of TL_i for the target device at time t', respectively. With this adjustment, we can now combine distributions by calculating the weighted means and standard deviations as follows.

The updated weighted means for autoencoder $\vec{a}_i$ at time t' is

$$\mu_{\vec{a}_i}(t') = \frac{|\vec{a}_i|}{|TL_i| + |\vec{a}_i|}\mu_{\vec{a}_i} + \frac{|TL_i|}{|TL_i| + |\vec{a}_i|}\delta\mu_{\vec{a}_i}.$$

The updated weighted standard deviation for autoencoder $\vec{a}_i$ at time t' is

$$\sigma_{\vec{a}_i}(t') = \sqrt{\frac{(|\vec{a}_i| - 1)\sigma^2_{\vec{a}_i} + |\vec{a}_i|(\mu_{\vec{a}_i} - \mu_{\vec{a}_i}(t'))^2 + (|TL_i| - 1)\delta\sigma_{\vec{a}_i} + |TL_i|(\delta\mu_{\vec{a}_i} - \mu_{\vec{a}_i}(t'))}{|\vec{a}_i| + |TL_i| - 1}}. \tag{13}$$

This updates the standard deviation and sample mean as the device is receiving network packages. Some benefits to this method are not having to store the package data and a reasonable model is learned with limited amount of data quickly. As new packets are introduced, the weights and thresholds update together to create an updated normal network data model for detection purpose for the target device. In our study, this model is trained from 4 IoT devices outside of D.

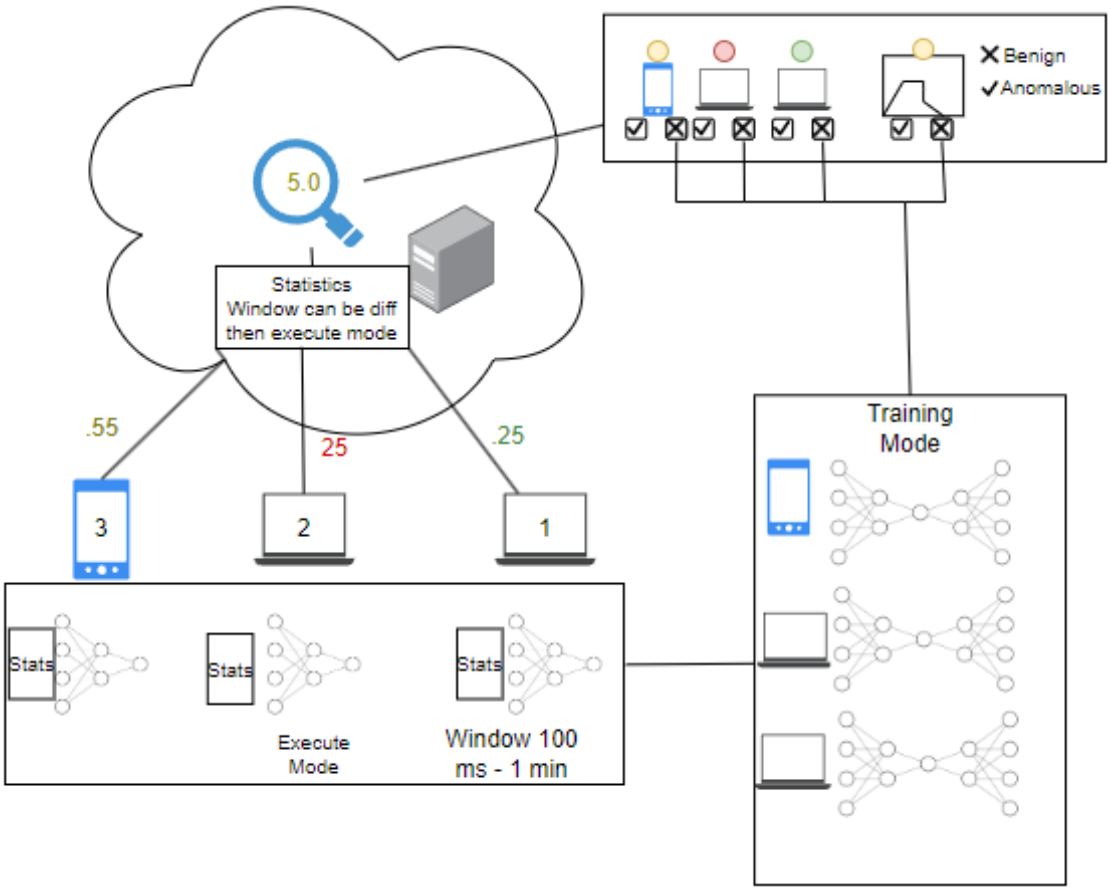
3.5 Federated Learning Anomaly Detector

3.5.1 Overview

To utilize federated learning for our proposed anomaly detection system, the IoT devices in question must be able to perform on-device machine learning tasks (with some minimal computing resources). Each device on the network has a personalized anomaly detection model that only listens to its data. This creates a real-time, scalable, secure system that uses computing resources of multiple devices. The model outputs are shared with the cloud to fully understand the device behaviors and send alerts when a system experiences abnormalities. Figure 5 provides an overview of how model outputs from multiple devices are sent to the cloud to learn a detection model.

Figure 5

Federated Learning Model for Anomaly Detection



Incremental learning is done across each edge device on a network to learn a local anomaly detection model. The data for this model is taken in real-time and extracted. The device and distribution are trained over 15 epochs every 30 seconds for 5 minutes. After this model is finished training, it goes into ”execute” mode; Every 30 seconds a batch of data is fed into the local model to evaluate the state of the device. Also, an anomaly score $g(x)$ is sent to the cloud to learn the status of the device. The advantage of this approach is in the model's ability to train a new edge device in 5 minutes and gain a good understanding of the traffic introduced.

Upon receipt of the status of each device in the cloud, a global statistical distribu-

tion of network quality and abnormalities is learned. The objective is to create a full map of device statuses across an infrastructure. If multiple devices on a segmented line are experiencing anomalies the model can warn the user that a segment is experiencing anomalies for evaluations to be done. The idea is to stop the domino effect of malicious devices taking over the network. During training mode, the anomaly status is taken over time to learn a global distribution. If this distribution exceeds an expected threshold the whole network can be classified as under attack.

3.5.2 Federated Learning Hyperparameters

We utilize federated learning to leverage decentralized input from various devices on our network to create a global mapping of network statuses. Let FL be a network with a set of edge devices $ED = \{ed_1, ed_2, ed_3, ..., ed_n\}$ such that n is the number of edge devices in ED, a prediction database PR and also centralized server CS. For each ed_i there exists a detector that contains the ensemble of transfer learned models $TD = \{\vec{a}_1, \vec{a}_2, \vec{a}_3, ...\vec{a}_k\}$ where k is the size of the ensemble. TD is trained on all the devices D_T and *uid* is the unique name given to each edge device *ed* on the network. An observation $\vec{o}$ is extracted each time a packet is read and added to an observation matrix O (See subsection 3.2.2) such that $O = \{\vec{o}_1; \vec{o}_2; \vec{o}_3; ...; \vec{o}_m\}$ and m is the number of packets observed after a time interval l. Because we are adding to our observation scope and want to retain what the model predictor has stored we use the weighted update algorithm (See subsection 3.4.2) to train on each observation after l.

For execution, after interval l , we store the predicted observations into a prediction array PO as follows. Let $PO = \{g(\vec{o}_1), g(\vec{o}_2), g(\vec{o}_3), ..., g(\vec{o}_m)\}$ for each observation $\vec{o}_i$ that exists in O. We then populate our prediction database PR with each PO across all *ed* that exists in ED. Let $PR = \{PO_1, PO_2, PO_3, ..., PO_n\}$ such that each instance PO_j corresponds with an edge device ed_j using a key *uid* specific to each *ed*.

The centralized server CS iterates through PR after l. Based on the total number of

predicted observations for all edge devices in ED, we compute a threshold Tp to decide whether a network is under attack during the "execute" mode using the following:

$$class = \begin{cases} UnderAttack, & \frac{totAnom}{totObs} \geq Tp \\ \\ Normal, & otherwise. \end{cases} \tag{14}$$

a $class$ is gathered after each interval l and is stored each time to help the user understand the status of their network, $totAnom$ is the number of anomalies detected for all devices in ED, and $totObs$ is the total number of observations for all edge devices in ED.

Figure 1 shows how the system works when there is only one $\vec{o}$ in O. When m increases in size beyond 1, the algorithm works the same except it runs over all of O captured after the time interval l. Also, the full data gets sent all at once before getting erased, not one at a time. One advantage of this is the encoders can run in parallel (3 separate threads in this example). This is because each encoder does not need each other to make their predictions, they run independently. Also if necessary, the encoder outputs $e(x)$ could be configured to be sent directly with $g(x)$ to the central server to do processing instead. This can lighten the load depending on the flexibility of the edge device executing forward propagations. The most prominent advantage is the anomaly detection model is adapted to their own individual device and network packets are never shared with the cloud. This is because the statistics is extracted from the packet data, and used only during the weight update algorithm (See subsection 3.4.2).

Chapter 4

Experimental Results

To evaluate how well the model can generalize a system, we utilize transfer learning on data from one IoT device source and adapt it to all IoT device targets. We also perform transfer learning on data from multiple sources and adapt to the same set of target devices. We compare their performance after the same amount of epochs to evaluate how well the predictive model adapt to new target IoT devices.

4.1 Experimental Design and Performance Evaluation

The dataset used in our experiments is the popular Kitsune IoT dataset [32]. This dataset is provided in an Incremental Statistics format similar to what we described in subsection 3.3.2. They generate this dataset for 9 IoT devices creating about 7 million packets between anomaly and benign examples. That dataset contains an equal amount of benign and anomaly examples. The anomaly samples are split into two primary vectors of botnet attack, Mirai and Gafgyt. Both botnets are used today to cause damage to IoT infrastructure [36]. A few devices in the dataset do not contain Mirai data (e.g., Samsung Webcam). We train our ensemble on Damni Doorbell to create a control model that we use for both single (Damni Doorbell) and multiple source (Damni Doorbell, Simple Home Security Camera 2, Provision Security 1-2, Ennino Doorbell) transfer learning scenarios.

Based on results from previous work [10], we build our control model by training 45 epochs of our control device data on an ensemble consisting of 3 autoencoders. For each source we introduce to our control model, we give 15 epochs of training time to that device's data to avoid overfitting. Similarly, for the remaining target devices, we also give 15 epochs of training time.

Before we compare multiple sources to a single source, we determine the threshold

for our anomaly detection model. To do this, we use our control model and apply transfer learning to a single target device and determine between the thresholds of 3 standard deviations and 5 standard deviations. These values were chosen because they are the thresholds commonly used for Normal Distributions and Galton Distributions, respectively. We compare their anomaly detection performance using False Positive Rates (FPR) and False Negative Rates (FNR) on Mirai attack data before and after transfer learning. Our results are the average FNR and FPR using 5-fold cross validation.

To compare the performance for target devices utilizing transfer learning using models created from single and multiple data sources, we evaluate their FPR and FNR (Mirai & Gafgyt Attacks) on all the data and a tenth of the data. We do this to measure how the combined distribution (Equation 13) affects the model's performance when all data is present and just a fraction. This is necessary when we simulate the system so we can get an idea of how much data is needed to make a notable impact on performance.

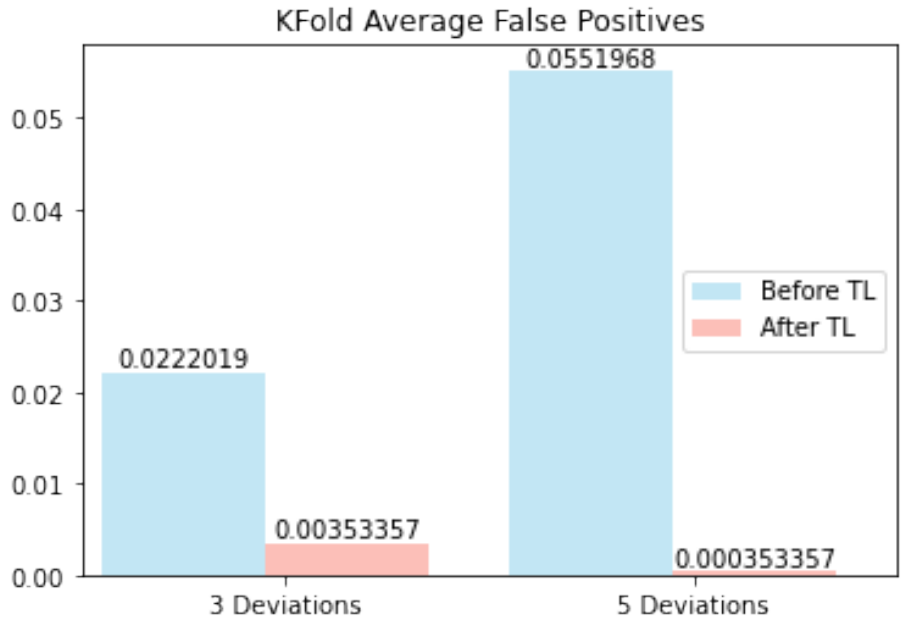
4.2 Experimental Results and Discussion

Figure 6 shows the impact that the deviation factor (i.e., number of standard deviation) has on transfer learning performance. When evaluating false positives, we observe that both instances (i.e., 3 and 5 standard deviations) experience a drop after transfer learning occurs. Before transfer learning, using 3 deviations performs better than 5 with an FPR of 2.22% and 5.52%, respectively. This makes sense because no data has been introduced or adapted to the network traffic of the target device yet. However, after transfer learning, we observe that 5 deviations becomes a better approximation by an order of magnitude. This is likely because as the autoencoders adapt to the target traffic, the approximations better suit a Galton distribution as opposed to a Normal distribution. With false negatives, we observe a peculiar trend where we see a small increase before and after transfer learning for 5 deviations. This could be something that happened when creating the combined distribution (Equation 13) but is more likely to be some statistical fluctuation on the data

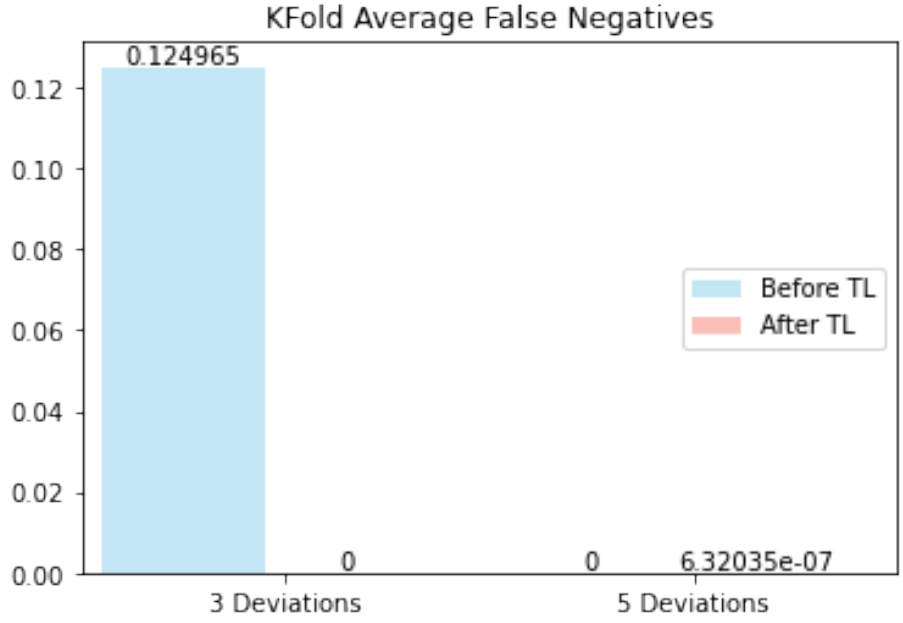
since the increase was so minimal. With 3 deviations, we see a large decrease in FNR after transfer learning which makes sense after the model began approximating to the network traffic. Based on these findings, the ideal deviation range is between 4.9 - 5.0 (Based on FNR trend on 5) so we use deviation factor of 5 for our subsequent experiments.

Figure 6

Target Device Simple Home Evaluated on a Single Source



(a) *False Positives*



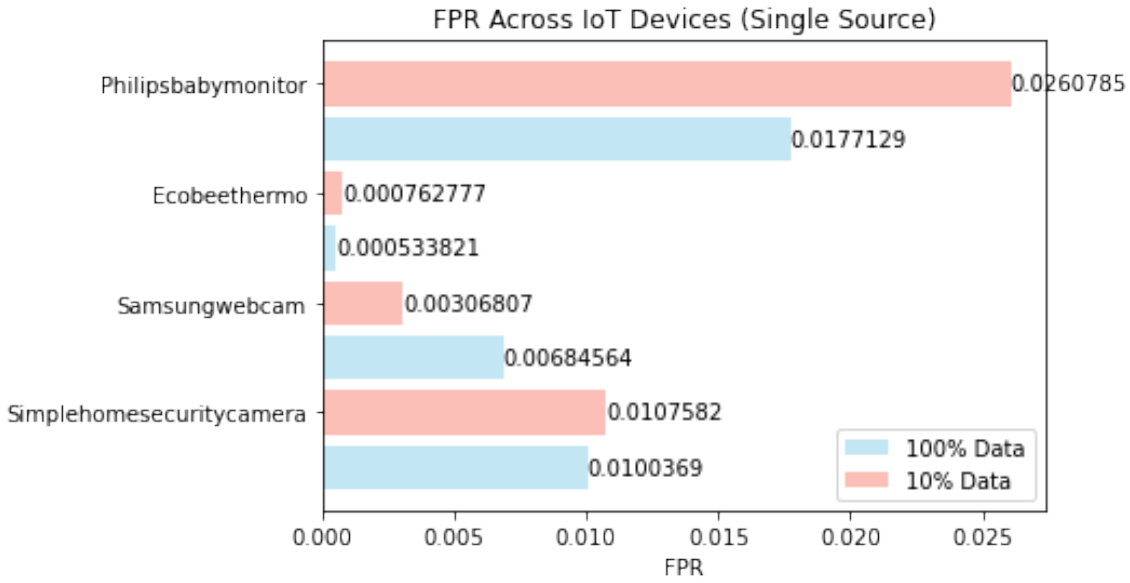
(b) *False Negatives*

Figure 7 compares the misclassifications of benign data when using single source

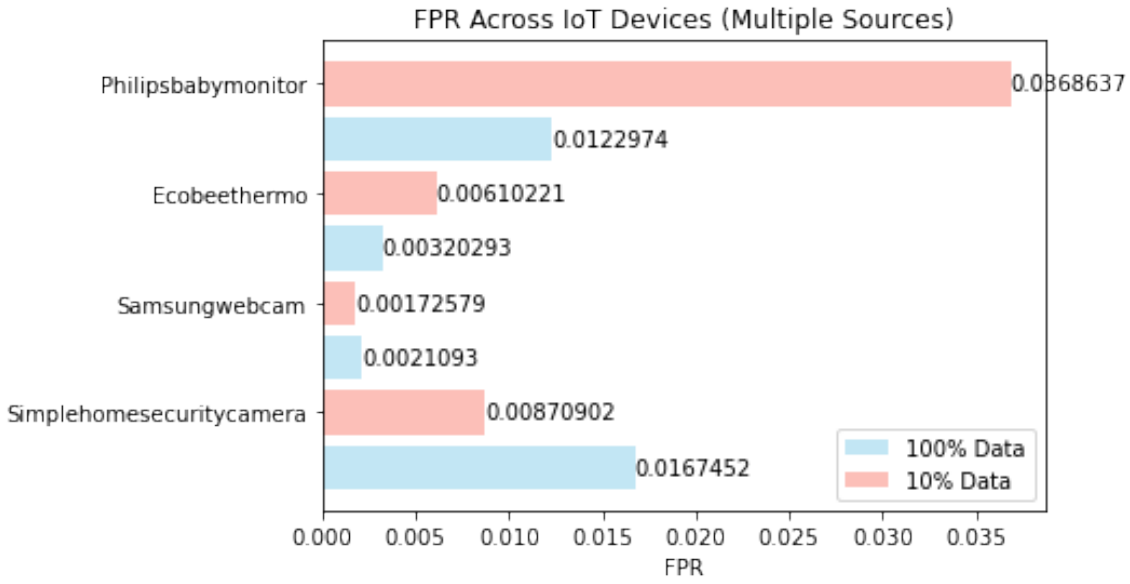
and multiple sources transfer learning. When evaluating the multiple sources transfer learning using 10% of the data, we observe instances where the model has difficulty generalizing, it's not until after 100% of the data is trained that the model classifies benign data correctly. This is likely due to the distribution formed by the model (See Equation 13). As more data are used for training the model, $|TL_i|$ also increases in size meaning that an imbalanced data problem could occur where if the amount of data used for multiple sources is a million data points and the target is a few thousand those will barely impact the distribution formed. This is shown by observing that after 100% of the data is used the model becomes better at generalizing and the FPR drops off significantly.

Figure 7

False Positives of Single vs Multiple Source Transfer Learning



(a) *Single Source*



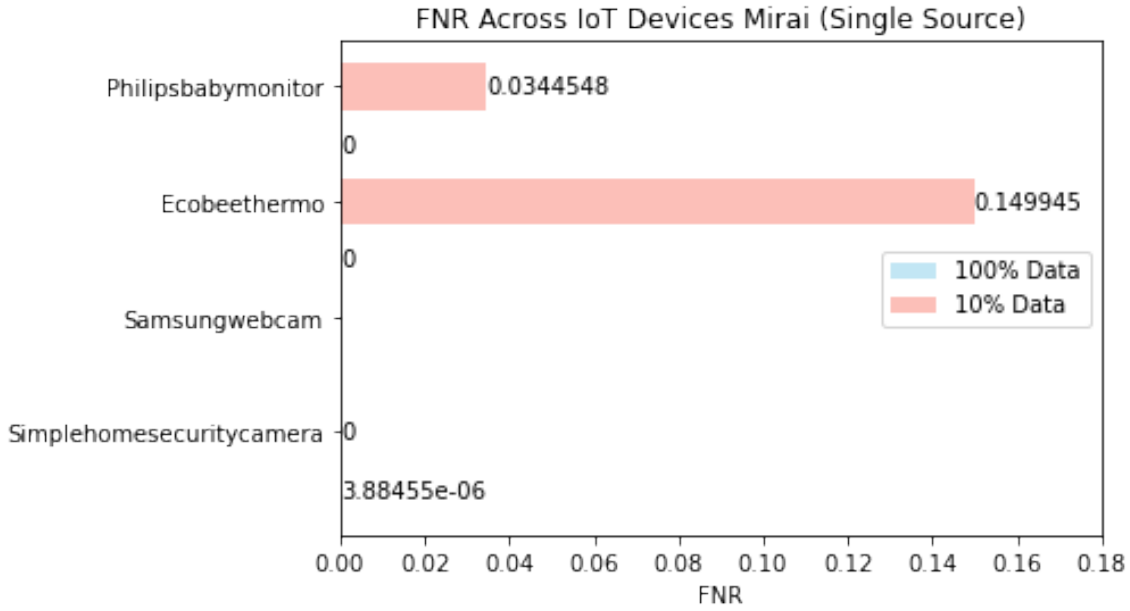
(b) *Multiple Sources*

For single source, we see the same trend with 10% data and increasing to 100%. When comparing the multiple and single source transfer learning false negatives in Figure 8

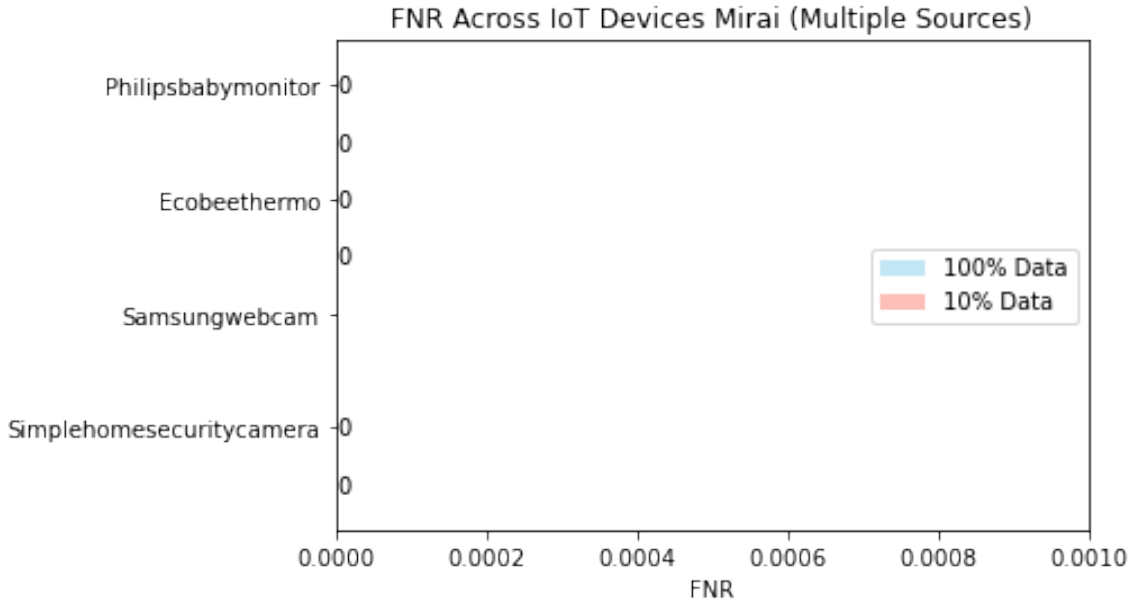
and Figure 9, we observe the same trend where the single source had low false positives, resulting in false negatives above 5% for Gafgyt and above 3.4% for Mirai anomaly attacks. This trend is not the same for the robust model where the false negatives are less than 1×10^{-3}. This result is expected because multiple sources are trained on more varying data than a single source implying that the encoder distribution determines a better model in less time.

Figure 8

Mirai False Negatives of Single vs Multiple Source Transfer Learning



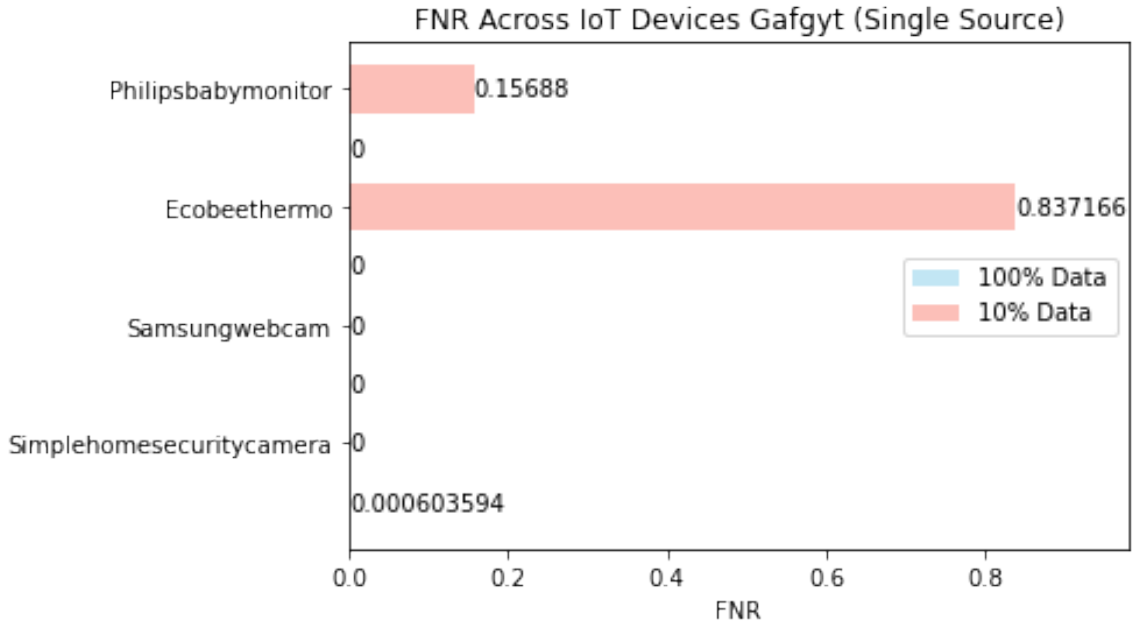
(a) *Single Source*



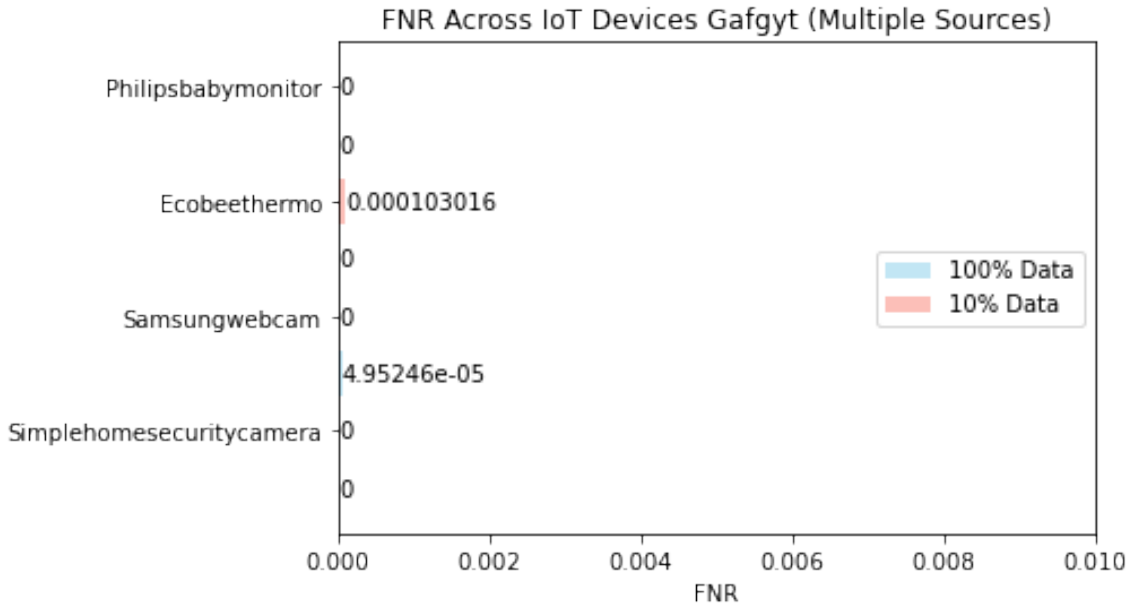
(b) *Multiple Sources*

Figure 9

Gafgyt False Negatives of Single vs Multiple Source Transfer Learning



(a) *Single Source*



(b) *Multiple Sources*

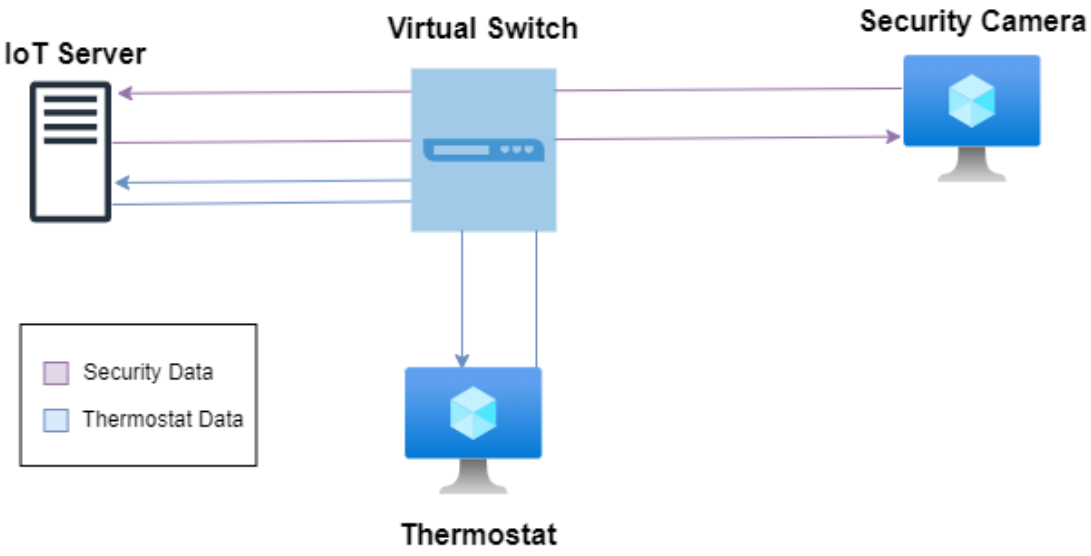
Chapter 5

System Simulation and Evaluation

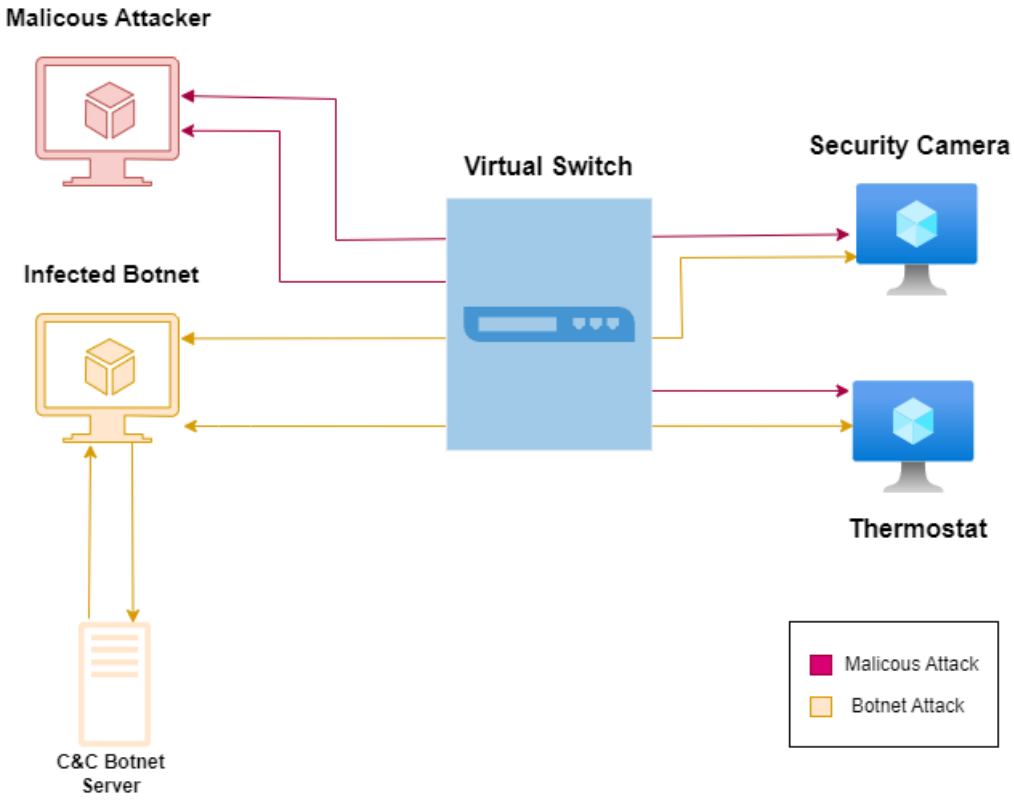
To test the capabilities of the proposed system, we create a simulation testbed to evaluate anomaly detection performance in real-world scenarios. Figure 10 shows an overview of the components in the testbed on a virtual LAN (VLAN). "Security Camera" and "Thermostat" are simulations of those devices' behaviors on a virtual machine. Figure 10a shows the background processes that are always happening on the testbed. The "IoT Server" acts as a Hub to receive IoT information from the two devices and provides feedback that can get intercepted during MITM attacks. Figure 10b shows the 2 virtual machines that are attacking the network. Malicious Attacker is our attack suite that does most of the attacks on the IoT suite (MITM, SYN FLOOD, HPING, SSDP FLOOD) (see Table 6). "Infected Botnet" is the botnet on the network that is reporting to a C&C Botnet Server. The "Infected Botnet" conducts Mirai attacks on the system attempting to turn the Security Camera and Thermostat devices to become part of the botnet. We put this network under 2 tests. The first test creates a baseline to see how the model performs when no attack is happening. In the second test, we perform a variety of attacks with the 2 additional attack virtual machines.

Figure 10

Overview of Simulation Testbed



(a) *Simulated Background Processes*



(b) *Simulated Attack in the Test Bed*

5.1 Real World System Baseline

This experiment creates a baseline for the Federated Learning Anomaly Detector on our testbed. We start by creating 2 virtual machines that simulate IoT device behavior. Then, we install our developed predictive model validated in chapter 4 onto each virtual machine. Finally, we run the system for a long time with no explicit anomaly attempt.

Figure 10a shows the regular processes happening. In summary, the IoT devices share their statuses with the IoT Server that will return a response to them. Each device will also have an SSDP (Simple Service Discovery Protocol) with SSL (Secure Sockets Layer) server communicating with other IoT devices on the system.

5.2 Simulating IoT Behavior

To simulate device behavior, we want to make the simulation as realistic as possible. To do this, we decide to simulate security camera data along with thermostat data. They both are a part of the real device dataset, so model performance should be similar to the expectation shown in our transfer learning experimental results (See subsection 3.4.2).

To simulate a virtual machine security camera, a server sends a TCP (Transmission Control Protocol) request between 0 and 10 seconds requesting for a status. The device will respond with a status measuring the availability of the camera (Up, Degraded, Down). These statuses are determined based on the usage of the VM CPU resources ($>$75% = Up, 25-75% = Degraded, $<$25% = Down). Upon receipt of this status, the server sleeps for 1 second between camera events to process the data. Note that the measurement statuses are not something evaluated during the training of the system, it is simply just the packet sizes and time intervals (See subsection 3.3.2).

To simulate a virtual machine thermostat, a server sends a TCP request between 2-4 seconds requesting temperature. The device will respond with a synthetic measurement of temperature in the house over to the server that will then process that data and give a

response. This data is formatted in the same way for both IoT simulations.

Each IoT device sends a multicast announcement to let the network (other devices) know that it is ready for communication. After this announcement, the devices listen to simulated SSDP requests to respond with information about the device. Each device also has a TLS (Transport Layer Security) server to simulate what would be a connection to the IoT device to sign in. Table 5 shows the specifications of each virtual machine used. The decision to put 4GB on each virtual machine is because with less than that the virtual machine does not start.

Table 5

Specifications of Simulated Devices

IP Address	Device Simulation	Threads Intel i9900k	GB of Ram
192.168.68.55	Security Cam	1	4
192.168.68.54	Thermostat	1	4
192.168.68.56	Primary Attack Suite	2	4
192.168.68.57	Infected Botnet	3	4

5.3 Installation of Prediction Model on Devices

We install the trained predictive model onto each individual virtual machine. The anomaly detection system goes through 1 hour of incremental learning before going into

detection mode to detect anomalies. We set the detection threshold to 5 standard deviations based on results from section 4.2 and have the ensemble of autoencoders to vote whether the device is anomalous or not.

During training, the device communicates and populates data to the prediction database *PR*. This communication will get flagged as anomalous or benign if it varies significantly from what the model learns. We also set a threshold for $Tp = 0.2$ (See Equation 14) as a comparison to see how the model trained overnight and compare the same models when anomalies are introduced.

5.4 Evaluation Metrics

We evaluate the system using the percentage of anomalies over time as follows.

$$percentAnomalies = \frac{NumAnomalies}{TotalPacketsSeen} \tag{15}$$

where $NumAnomalies$ is the number of anomalous packets detected over approximately 20 minutes and $TotalPacketsSeen$ is the total number of packets observed over approximately 20 minutes.

5.5 Baseline (No Anomalies) Model Results and Discussions

The first model that ran overnight with one hour of training, the model kept the full size of all the devices trained on ($|\vec{a}|$ = approximately 130,000 trained across 4 devices). After the training is completed, the model trained on 123,256 additional packets. The evaluation (See Figure 11) shows the model was able to keep below 2.5% detection rate.

To see how the model performed with less training packets, a model was ran overnight after 20 minutes of training time (See Figure 12). Roughly $46,000$ packets were used for training for each device. Little seemed to happen and the model showed far less than a 2.5% detection rate. This could be because it had less training time and was closer to the optimal

model than the 1-hour model. To evaluate this further, we ran the 1-hour model once more overnight where instead of keeping the size, we lower it so $\mu_{\vec{a}_i}, \sigma_{\vec{a}_i}$ stays the same however $|\vec{a}_i|$ is $13,000$ instead. This is to establish an importance weight of maintaining the previous distribution.

The model that ran overnight with changing $|\vec{a}_i|$ ($|\vec{a}_i|$ = approximately 13,000) shows that the model was successful in learning the simulated traffic. For the majority of the night, the model stayed below 20% of the average anomalous traffic (See Figure 13). We do see a spike in that traffic during one interval, and this could be due to turning on and off lights because SSDP picks up on all IoT devices and this was connected to a live network with other IoT devices functioning, or, this was an actual mistake of the model. However, this spike was still below the threshold and would not have been classified as an anomaly. We use this model for the rest of the experiments. This model was chosen because regardless of the spike, it showed promise and would be interesting to evaluate how this model handles anomalies.

Figure 11

1 Hour Model Ran Overnight on 100% Data

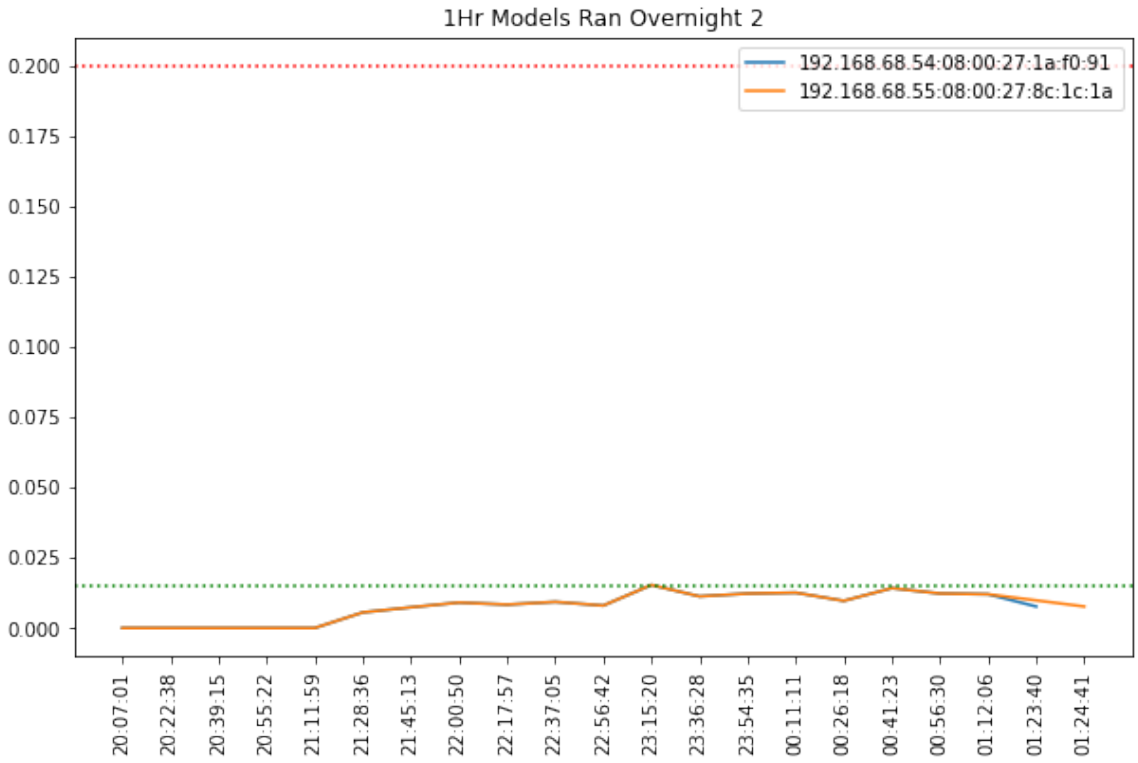


Figure 12

20 Minute Model Ran Overnight on 100% Data

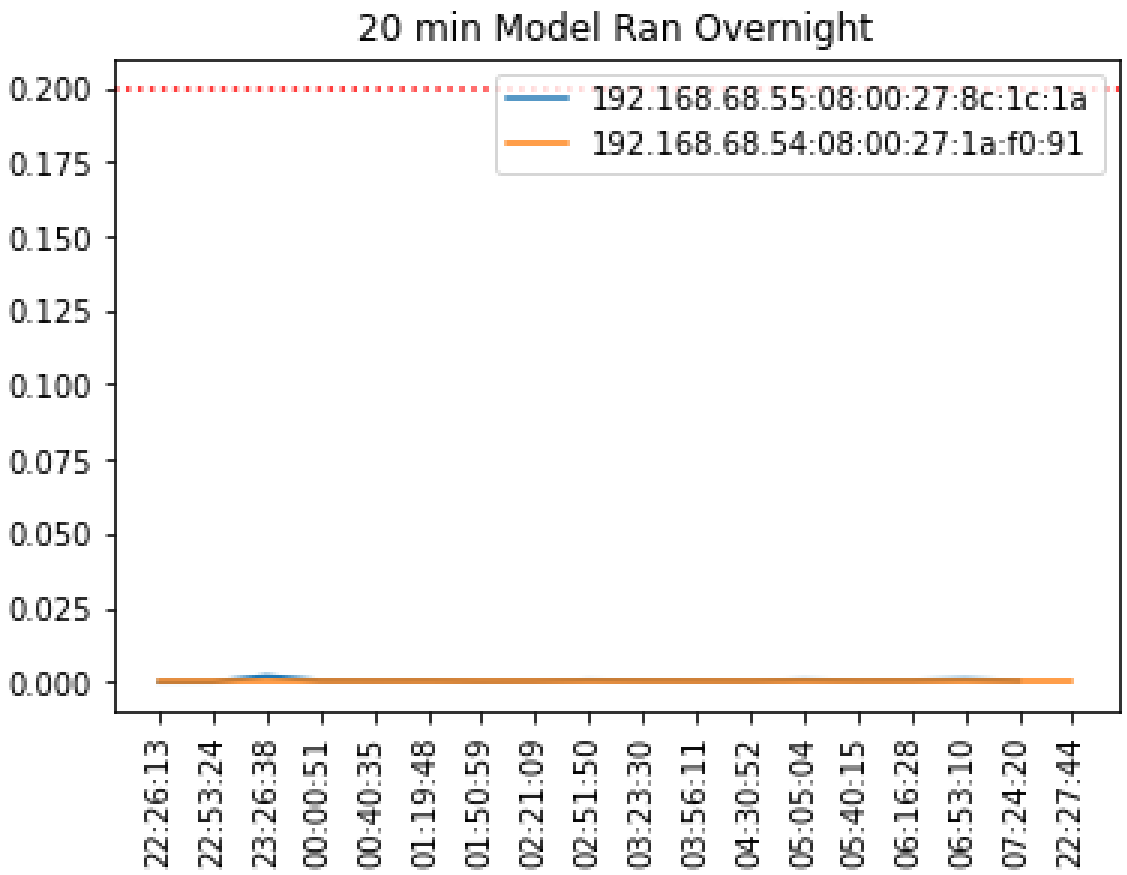
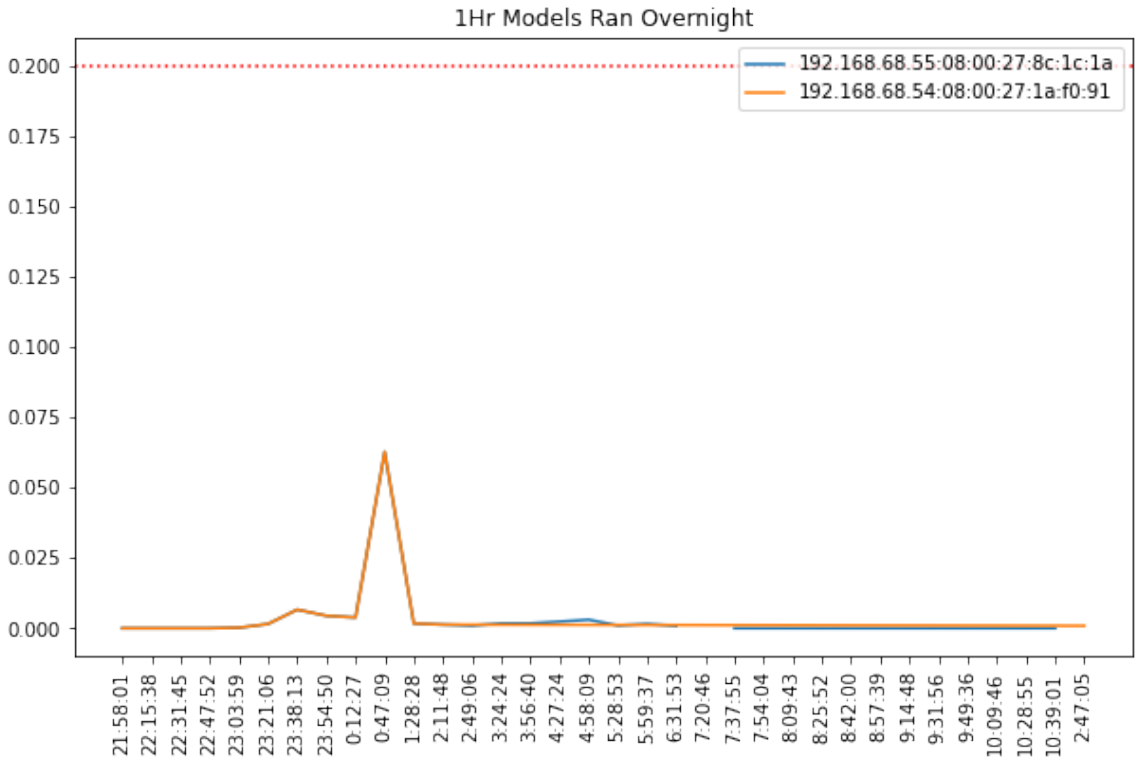


Figure 13

1 Hour Model Ran Overnight on 10% Data



5.6 Injecting Anomalies in Real Time

This experiment consists of simulated attacks on the testbed. After training on the simulated IoT devices, we conduct attacks on them to see how the proposed system responds under attack. We conduct a Mirai botnet attack to make the IoT devices become part of the botnet and use it to scan for other machines on the system. SSDP, SYN FLOOD, and HPING attacks are flooding attacks meant to overwhelm the system with information to cause a crash or make the system vulnerable to some other kind of attack. We use MITM attacks to intercept packets between two devices by jumping in between the flow of traffic and acting as a "middleman" to forward whatever message to the same system. For our MITM attacks, we do ARP Poisoning to pretend we are a Mac Address to intercept

packets.

Both malicious and botnet attacks are never done at the same time and the entire system is reloaded for the sake of consistency in testing each type of attack. Malicious attacks cover a large variety of attacks (See Table 6) while botnet attacks consist strictly of taking over the device and reporting it to the C&C server to do a scan of the network or attacking another device on the network.

Table 6

Tools for each Attack Vector Used

Malicous Attacker	**Tool Used For Attack**
MITM	EtterCap
SSL DOS Attack	THC
SYN Flood	hping
SSDP Attack	DDOS RootSec
Infected Botnet	**Tool Used For Attack**
Mirai	Mirai Source Code

5.7 Creating Attack Virtual Machines (VM)

The "Malicious Attacker" (See Figure 10b) is a Kali Linux VM with 2 Cores of an Intel i9900k with 4GB of RAM. Tool used for each type of attack (See Table 6) was installed onto the suite. The second is the "Infected Botnet", which is intended to handle strictly botnet attacks. The "Infected Botnet" was created on an Ubuntu Linux VM With 3 Cores of an Intel i9900k and 8GB of RAM designated to the VM. The Mirai Source Code was put onto this VM and configured to run the C&C server to send commands over to the infected devices for them to run. In this instance, we tell the devices to SYN Flood the other device on the network.

5.8 Results and Discussions

In the benign scenario, we showed over 20 minutes of results in section 5.1 because there was no activity happening that was causing immense spikes. It was just a normal background process. In this scenario, as we are attacking these devices in a small window, we are showing results at every 30 seconds as it shows what the traffic looks while the attack is happening.

Generally, it seems that the federated learning anomaly detector is able to detect variability in anomalies. During the Mirai Attack (Figure 14) we see that within 30 seconds after the attack begins the system is able to detect abnormality in the network data. Across 3 of the attacks (Figure 14, Figure 15, and Figure 16), we observe that the Thermostat (see Table 5 for device specifications) experiences a lower detection rate or fewer spikes as opposed to the Security Camera. This could be because Thermostat data might have been too simple for the model to learn so it could be more difficult to decipher anomalies. With that being said, the detector is still able to pick up on deviations in the network traffic. Also, for a more realistic use case of this model, the threshold would be much lower than 20% and still be able to be detected well.

Figure 14

Mirai Detection After 1 Hour Model Training

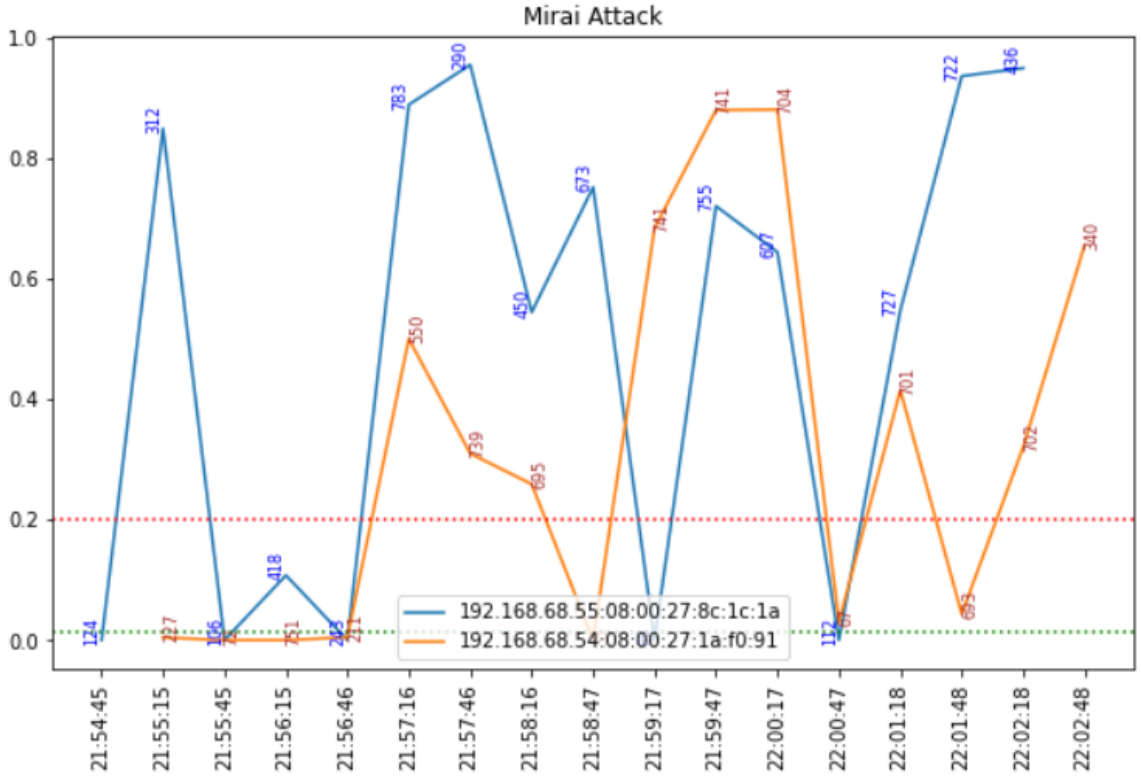


Figure 15

Hping Detection After 1 Hour Model Training

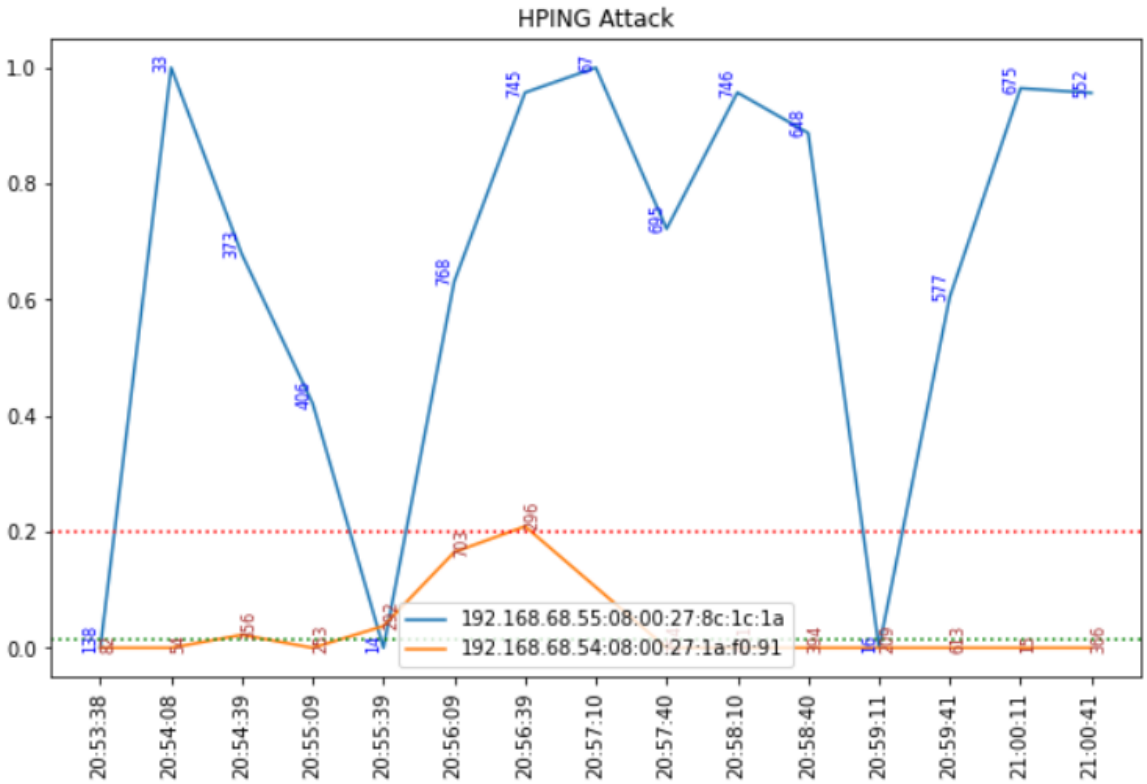
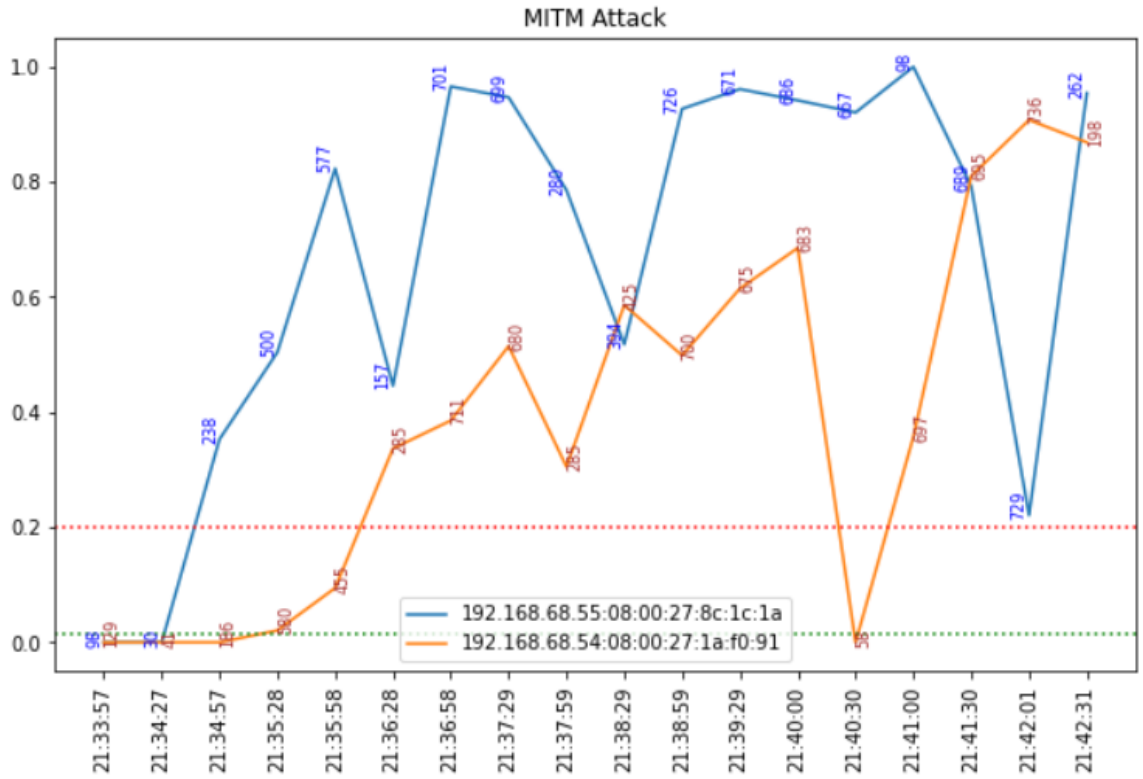


Figure 16

MITM Detection After 1 Hour Model Training



Finally, we observe spikes as the system monitor the network traffic (Figure 14, Figure 15, Figure 16,Figure 17 and Figure 18). There are instances when there are no anomaly and instances when there are. This is likely because of the asynchronicity between the Central Server *CS* and the devices sending packet updates. The timer interval l that *CS* reads from the prediction database *PR* is the same as the interval the individual devices send in batches to *PR*. If *CS* starts a few seconds after the individual devices, there will be an inconsistency between what is actually being read and what is being sent. For instance, during the Mirai attack (See Figure 14) we see 312 packets at 21:55:15, however at 21:55:45 ($l = 30s$), we see the packets drop to 106 and increase once more. This is an example of a sync issue between the devices.

Figure 17

SSDP Detection After 1 Hour Model Training

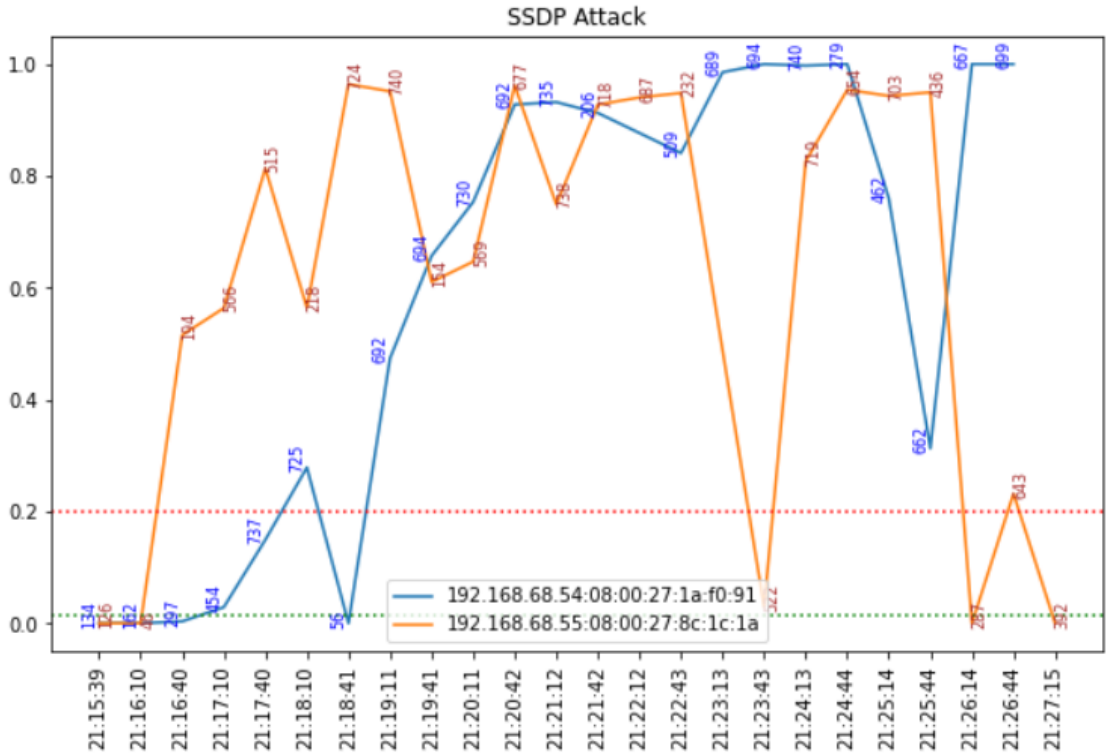
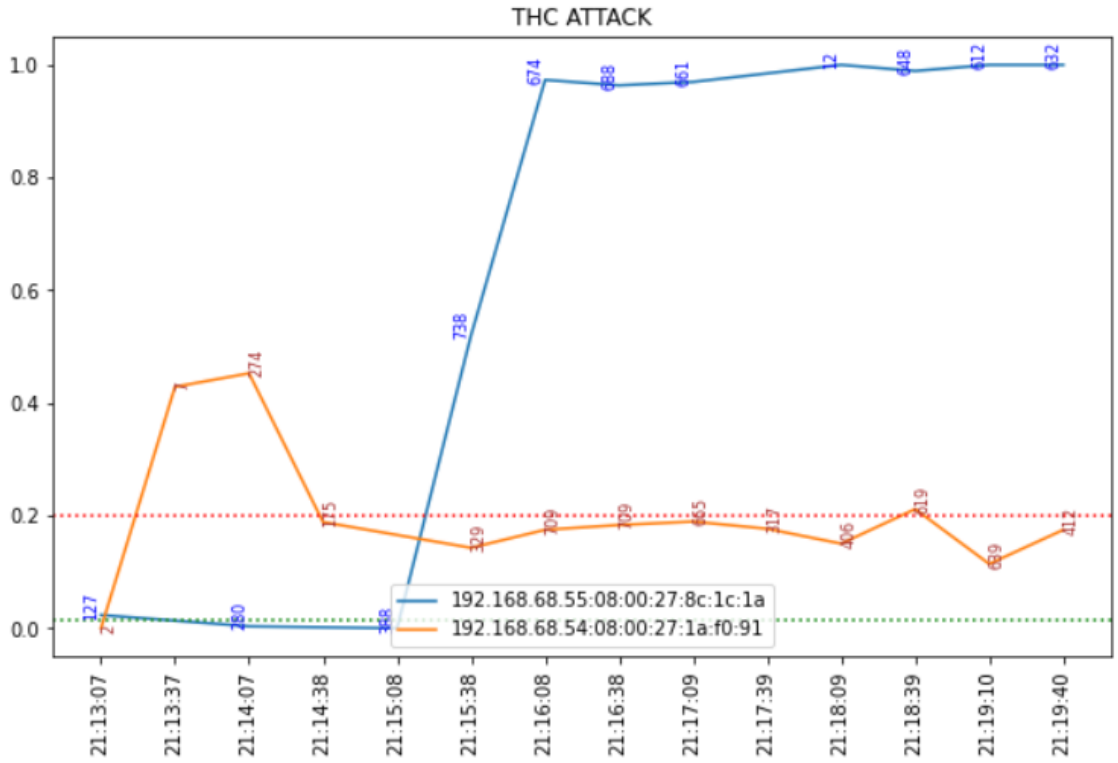


Figure 18

TCP SYN Detection After 1 Hour Model Training



Chapter 6

Conclusion and Future Work

We propose a lightweight unsupervised hybrid-cloud ensemble anomaly detection system for malware detection. We perform transfer learning using a generalized model trained on multiple IoT device sources to learn network traffic on new devices with minimal computational resources. We further extend our proposed system to utilize federated learning such that IoT devices feed their output to a cloud server enabling more detection capabilities while keeping the network traffic secure on the device itself maintaining data privacy. We validate this system by creating a simulation testbed to conduct different attacks on the IoT devices to evaluate how well the detection system works. We also compare transfer learning using multiple sources to a single source to show how the detection model of a target device is impacted by transfer learning. Empirical results on two datasets, one from the 2016 Mirai botnet attacks on IoT devices and the other from Gafgyt malware attacks on various IoT devices, show the competitiveness and feasibility of our proposed solution.

For future work, we intend to test the system further by doing more runs with varying training times. We also plan on adding more capabilities to the cloud platform to introduce multi-classification of attacks and to provide more analytics. This will also include taking this system out of the testbed and moving it to a real network for conducting experiments and attacks there. More experiments will also be introduced testing the combined distribution for transfer learning and testing other ways to update the prediction model.

References

[1] Z. ThreatLabz, *Zscaler threatlabz 2023 Enterprise IoT & OT Threat Report*, 2023.

[2] A. Marton and S. Systems, "IoT malware attacks up by 37% in the first half of 2023," *Safepay Systems News*, Aug. 2023.

[3] E. Altares, J. Salvio, and R. Tay, *2022 IoT Threat Review*, Fortinet, Jan. 2023.

[4] B. Marr, "2024 IoT And Smart Device Trends: What You Need To Know For The Future," *Forbes*, 2024, Contributor.

[5] S. S. Sepasgozar and S. Pierre, "Fed-NTP: A federated learning algorithm for network traffic prediction in Vanet," *IEEE Access*, vol. 10, pp. 119 607–119 616, Jan. 2022. DOI: 10.1109/access.2022.3221970. [Online]. Available: https://doi.org/10.1109/access.2022.3221970.

[6] Y. Zhao, J. Chen, D. Wu, J. Teng, and S. Yu, "Multi-task network anomaly detection using federated learning," in *Proceedings of the 10th international symposium on information and communication technology*, 2019, pp. 273–279.

[7] L. Cui *et al.*, "Security and privacy-enhanced federated learning for anomaly detection in iot infrastructures," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3492–3500, 2021.

[8] M. J. Idrissi *et al.*, "Fed-anids: Federated learning for anomaly-based network intrusion detection systems," *Expert Systems with Applications*, vol. 234, p. 121 000, 2023.

[9] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated-learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, vol. 9, no. 4, pp. 2545–2554, 2021.

[10] S. E. Arroyo and S.-S. Ho, "Autoencoder ensemble method for botnets detection on iot devices," in *21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022, pp. 715–720. DOI: 10.1109/ICMLA55696.2022.00119.

[11] S. E. Arroyo and S.-S. Ho, "Hybrid-cloud autoencoder ensemble method for botnets detection on edge devices," in *8th IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2024.

[12] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987. DOI: 10.1109/tse.1987.232894. [Online]. Available: https://doi.org/10.1109/tse.1987.232894.

[13] Q. Mao, F. Hu, and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 2595–2621, Jan. 2018. DOI: 10.1109/comst.2018.2846401. [Online]. Available: https://doi.org/10.1109/comst.2018.2846401.

[14] G. ALMahadin *et al.*, "Vanet network traffic anomaly detection using gru-based deep learning model," *IEEE Transactions on Consumer Electronics*, 2023.

[15] H. C. Altunay and Z. Albayrak, "A hybrid cnn+ lstm-based intrusion detection system for industrial iot networks," *Engineering Science and Technology, an International Journal*, vol. 38, p. 101 322, 2023.

[16] S. Alzughaibi and S. El Khediri, "A cloud intrusion detection systems based on dnn using backpropagation and pso on the cse-cic-ids2018 dataset," *Applied Sciences*, vol. 13, no. 4, p. 2276, 2023.

[17] S. Ryu, B. Jeon, H. Seo, M. Lee, J.-W. Shin, and Y. Yu, "Development of deep autoencoder-based anomaly detection system for hanaro," *Nuclear Engineering and Technology*, vol. 55, no. 2, pp. 475–483, 2023.

[18] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in iot devices," *Computer Networks*, vol. 204, p. 108 693, 2022.

[19] Q. Pan, Y. Bao, and H. Li, "Transfer learning-based data anomaly detection for structural health monitoring," *Structural Health Monitoring*, vol. 22, no. 5, pp. 3077–3091, 2023.

[20] A. W. Salehi *et al.*, "A study of cnn and transfer learning in medical imaging: Advantages, challenges, future scope," *Sustainability*, vol. 15, no. 7, p. 5930, 2023.

[21] J. Y.-L. Chan, K. T. Bea, S. M. H. Leow, S. W. Phoong, and W. K. Cheng, "State of the art: A review of sentiment analysis based on sequential transfer learning," *Artificial Intelligence Review*, vol. 56, no. 1, pp. 749–780, 2023.

[22] Z. Wei, J. K. Calautit, S. Wei, and P. W. Tien, "Real-time clothing insulation level classification based on model transfer learning and computer vision for pmv-based heating system optimization through piecewise linearization," *Building and Environment*, vol. 253, p. 111 277, 2024.

[23] D. A. Bierbrauer, M. J. De Lucia, K. Reddy, P. Maxwell, and N. D. Bastian, "Transfer learning for raw network traffic detection," *Expert Systems with Applications*, vol. 211, p. 118 641, 2023.

[24] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020.

[25] P. Wei, B. Wang, X. Dai, L. Li, and F. He, "A novel intrusion detection model for the can bus packet of in-vehicle network based on attention mechanism and autoencoder," *Digital Communications and Networks*, vol. 9, no. 1, pp. 14–21, 2023.

[26] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset," *IEEE Access*, vol. 9, pp. 140 136–140 146, 2021.

[27] D. Novoa-Paradela, O. Fontenla-Romero, and B. Guijarro-Berdiñas, "Fast deep autoencoder for federated learning," *Pattern Recognition*, vol. 143, p. 109 805, 2023.

[28] J. Jithish, B. Alangot, N. Mahalingam, and K. S. Yeo, "Distributed anomaly detection in smart grids: A federated learning-based approach," *IEEE Access*, vol. 11, pp. 7157–7179, 2023.

[29] X. Sáez-de-Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Clustered federated learning architecture for network anomaly detection in large scale heterogeneous iot networks," *Computers & Security*, vol. 131, p. 103 299, 2023.

[30] N. Lin, Y. Wang, E. Zhang, K. Yu, L. Zhao, and M. Guizani, "Feedback delay-tolerant proactive caching scheme based on federated learning at the wireless edge," *IEEE Networking Letters*, vol. 5, no. 1, pp. 26–30, 2023.

[31] Y. Zhong *et al.*, "HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning," *Computer Networks*, vol. 169, p. 107 049, Mar. 2020. DOI: 10.1016/j.comnet.2019.107049. [Online]. Available: https://doi.org/10.1016/j.comnet.2019.107049.

[32] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," Jan. 2018. DOI: 10.14722/ndss.2018.23204. [Online]. Available: https://doi.org/10.14722/ndss.2018.23204.

[33] M. Al-Fawa'reh, J. Abu-Khalaf, P. Szewczyk, and J. J. Kang, "Malbot-drl: Malware botnet detection using deep reinforcement learning in iot networks," *IEEE Internet of Things Journal*, 2023.

[34] M. Alasmar, R. Clegg, N. Zakhleniuk, and G. Parisis, "Internet traffic volumes are not gaussian—they are log-normal: An 18-year longitudinal study with implications for modelling and prediction," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1266–1279, 2021.

[35] R. C. de Amorim, "Feature relevance in ward's hierarchical clustering using the l p norm," *Journal of Classification*, vol. 32, pp. 46–62, 2015.

[36] A. Affinito, S. Zinno, G. Stanco, A. Botta, and G. Ventre, "The evolution of mirai botnet scans over a six-year period," *Journal of Information Security and Applications*, vol. 79, p. 103 629, 2023.