DEVELOPMENT AND EVALUATION OF A SECURITY AGENT FOR INTERNET OF THINGS

THESIS

Youngjun Park, 2nd Lieutenant, USAF

AFIT-ENG-MS-20-M-053

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-053

DEVELOPMENT AND EVALUATION OF A SECURITY AGENT FOR INTERNET OF THINGS

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Cyber Operations

Youngjun Park,

2nd Lieutenant, USAF

March 26, 2020

AFIT-ENG-MS-20-M-053

DEVELOPMENT AND EVALUATION OF A SECURITY AGENT FOR INTERNET OF THINGS

THESIS

Youngjun Park,
2nd Lieutenant, USAF

Committee Membership:

Barry E. Mullins, Ph.D., P.E.
Chair

Scott R. Graham, Ph.D.
Member

Timothy H. Lacey, Ph.D., CISSP
Member

Stephen J. Dunlap, M.S., CISSP
Member

AFIT-ENG-MS-20-M-053

Abstract

Recent surges of the IoT market and the unregulated manufacturing of devices have created a network of vulnerable devices that can be leveraged to launch global-scale attacks, causing massive amounts of damage worldwide. As a result, the U.S. Government Accountability Office (GAO) has identified ensuring security of these devices as a critical challenge for national security, requiring immediate attention for government agencies that deploy them, including the Department of Defense.

This research identifies two primary sources where breach of confidentiality occur in IoT and proposes an architecture of a security agent capable of protecting these areas. Prior research has demonstrated that many devices lack the computational capacity to perform meaningful encryption of their data, and that even variations in encrypted Wi-Fi frames can be used to extract information about the devices. The proposed security agent, Internet of Things Active Management Unit (IoTAMU), provides confidentiality for these two areas via the following capabilities: (1) authentication, (2) firewall, (3) encryption, and (4) spoofing. By acting as a proxy between the IoT devices and the gateway router, IoTAMU encrypts the traffic between itself and the Remote Interface (RI) (e.g., smart phone) of the devices, and spoofs network packets to hide the underlying network patterns, substantially decreasing the likelihood of unwanted leakage of knowledge to the public. This research provides an implementation of the firewall, encryption, and the spoofing functionalities of the agent. The experiments measure the effectiveness of IoTAMU's ability to uniquely modify the observed network signatures of each device via spoofing, and its potential effect on network congestion.

To test the spoofer's effect, an Identical Device Model Classifier (IDMC) is devel-

oped, which measures the similarities of the observed network signatures of each pair of devices and recognizes identical-model devices. After performing Principal Component Analysis (PCA) on the device signatures, a Similarity Score (SS) metric is derived by calculating the ratios of euclidean distances between the principal component scores of each pair of devices. The IDMC correctly identifies all identical-model devices in baseline network settings without the spoofer, achieving 100% precision, recall, and specificity at high threshold (SS>0.9). When the spoofer is enabled, none of the identical pairs are identified at high threshold, and up to 66% identical pairs are identified at lower thresholds (SS>0.8, 0.7). Overall, the spoofer is able to sufficiently modify the observed network signatures of each device; the observed differences between each pair increase overall (p-value = 0.01132) at 120 spoofed samples, making it more difficult to identify similar devices. Finally, the experiments in this research show the spoofer has a negligible effect on network congestion. As the number of spoofed samples increases, there is a linear increase in the number of additional packets created and its throughput, averaging an increase of one packet per second and 1.546 kbps per sample of network signature spoofed. Furthermore, there are no dropped packets across all trials, and the calculated network latency times remain relatively consistent as more packets are added into the network.

Acknowledgements

I am thankful for AFIT and the Air Force for giving me this unique opportunity. But this would not have been possible without the helpful guidance of Dr. Barry Mullins, who continue to inspire me to be as passionate about the subject as he is. I'd like to express my gratitude for Dr. Scott Graham, Dr. Timothy Lacey, Mr. Stephen Dunlap, Mr. Ryan Harris, and the rest of the AFIT faculty whom I've had the pleasure to work with, for their expertise and input throughout this process. I am grateful for my fellow AFIT students for their friendship and collaboration, and making this experience unforgettable.

I'd like to thank my family and friends for always supporting me, no matter what I set out to do. Lastly, I owe this to my dog Luna for being by my side day and night.

Youngjun Park

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

List of Acronyms

Abbreviation		Page
AES	Advanced Encryption Standard	13
AFIT	Air Force Institute of Technology	19
AGPS	Artificially Generated Packet Signatures	55
ANOVA	Analysis of Variance	73
AP	Access Point	9
BSSID	Basic Service Set Identifier	9
CA	Certificate Authority	14
CUT	Components Under Test	59
DHCP	Dynamic Host Configuration Protocol	32
DoD	Department of Defense	1
FN	False Negatives	63
FP	False Positives	63
GAO	Government Accountability Office	18
IAT	Inter-arrival Time	44
ICMP	Internet Control Message Protocol	38
IDMC	Identical Device Model Classifier	24
IoT	Internet of Things	x
IoTAMU	Internet of Things Active Management Unit	2
IP	Internet Protocol	29
ISP	Internet Service Provider	34

Abbreviation		Page
IV	Initialization Vector	39
LAN	Local Area Network	20
MAC	Media Access Control	9
MIC	Message Integrity Check	7
MTU	Maximum Transmission Unit	50
NAT	Network Address Translation	29
NL	Network Latency	64
NT	Network Throughput	64
OUI	Organizational Unique Identifier	49
OWASP	Open Web Application Security Project	18
PC	Packet Count	64
PCA	Principal Component Analysis	3
PCS	Principal Component Score	15
PD	Packets Dropped	64
PKE	Public Key Encryption	12
PMK	Pairwise Master Key	10
PNP	Plug-and-Play	2
POC	Proof-of-Concept	5
PSD	Pairwise Signature Distance	72
PSK	Preshared Key	10
PTK	Pairwise Transient Key	9

Abbreviation		Page
RI	Remote Interface	x
SCADA	Supervisory Control and Data Acquisition	6
SDN	Software-Defined Networking	20
SKE	Symmetric Key Encryption	12
SS	Similarity Score	51
SSID	Service Set Identifier	9
SUT	System Under Test	59
TCP	Transmission Control Protocol	38
TLS	Transport Layer Security	14
TN	True Negatives	63
TP	True Positives	63
UDP	User Datagram Protocol	3
VPN	Virtual Private Network	37
WAN	Wide Area Network	34
WEP	Wired Equivalent Privacy	11
WNIC	Wireless Network Interface Controller	10
WPA2	Wireless Protected Access 2	9

DEVELOPMENT AND EVALUATION OF A SECURITY AGENT
FOR INTERNET OF THINGS

I. Introduction

1.1 Background

The IoT industry is currently one of the fastest growing in the world, projected to surpass global spending of $1 trillion by the year 2022 [1]. However, this rapidly expanding market is saturated with unregulated devices without the necessary security measures to protect user information [2]. Information leakage caused by vulnerabilities can have privacy implications for individual users, and can escalate to compromise of assets and personnel for organizations. As more of these devices are deployed in operational settings of critical national organizations such as the Department of Defense (DoD), immediate investigations to develop defensive measures to protect their confidentiality are warranted.

1.2 Problem Statement

Prior IoT research has demonstrated vulnerabilities that exist in these devices and the potential information that can be extracted from them. Two major weaknesses that expose information are the focus of this research. First, many of these devices lack the hardware capacity to perform meaningful encryption, meaning any insiders with access to the internal network may have unauthorized access to the data [2]. Second, data mining on the unencrypted fields of encrypted Wi-Fi frames can lead to unintentional information leakage [3]. As the amount of resources and the machine

learning capabilities of the adversaries grow, the information that can be extracted from encrypted wireless traffic will continue to increase [4]. These vulnerabilities have dangerous implications for personal households and organizations that deploy IoT. Although significant strides have been made in the field of IoT security, limited progress has been made in improving the confidentiality of these devices, especially for wireless communications. Therefore, this research seeks to answer whether a Plug-and-Play (PNP)-style security agent can be developed with minimal changes to an existing network, that can offer modification of the observed patterns of wireless traffic and encryption of the wired traffic without imposing a significant burden on the network.

1.3 Research Objectives

This research proposes the architecture of Internet of Things Active Management Unit (IoTAMU), a security agent capable of encryption and spoofing. It seeks to demonstrate its soundness of design by conducting all experiments on a network set up with IoTAMU serving as the gateway for all IoT devices, and via a implementation of the firewall and encryption functionalities. Then, it attempts to illustrate an area of information leakage from Wi-Fi traffic patterns that can be concealed by a spoofer. This research aims to evaluate the spoofer's ability to hide the device-specific signatures by modifying the observed patterns of unencrypted fields of each device's encrypted Wi-Fi frames (e.g., lengths of payload, direction, time between sent/received packets; hereafter referred to as network signatures). Finally, spoofing's potential negative effect on network congestion is investigated. In summary, this research has the following main objectives:

1. Measure the accuracy of a classifier in identifying devices of identical models from their network signatures.

2. Determine the effectiveness of a spoofing algorithm in changing the observed network signatures of pre-existing devices.

3. Examine the impact on network congestion from the additional traffic created by the spoofer.

1.4 Hypothesis

It is hypothesized that a security agent can be developed in the form of a PNP-style device to provide additional confidentiality via encryption for IoT networks. Additionally, it is hypothesized that the observed network signatures of the devices can be uniquely modified to be unrecognizable via spoofing, without contributing to a substantial congestion of the network.

1.5 Approach

First, a controlled network is set up to model realistic internal and external subnets where the IoT and their RI (i.e., smart phone) reside. IoTAMU is then installed in the smart home (internal) network and connects to various Wi-Fi IoT devices. Its encryption and spoofing capabilities are evaluated separately. Encryption functionality is tested using two different laptops posing as an IoT device and its RI that exchange User Datagram Protocol (UDP) messages via IoTAMU. Then the spoofing functionality is tested by collecting the network traffic with and without the spoofer enabled for comparison. To measure the performance of the spoofer, an identical device model classifier is developed, which works by measuring the similarities in the observed network signatures of the devices via Principal Component Analysis (PCA). Lastly, the spoofer's impact on network congestion is measured by incrementally increasing the load of spoofed traffic.

1.6 Assumptions/Limitations

The limitations and the assumptions made throughout this research are summarized below:

- The network modeled in this research is assumed to be representative of a realistic smart home network.
- The behavior of the observed network signatures of the IoT devices are assumed to be similar in other network environments.
- The devices are limited to three different types of devices and three identical-model pairs.

1.7 Contributions

This research contributes to the field of IoT and Wi-Fi security via the following four contributions:

1. **IoTAMU:** An architecture of a PNP-style IoT security agent capable of authentication, firewall, encryption, and spoofing is presented. The presented architecture helps isolate the vulnerable devices and offer additional protection while minimally disrupting the existing network.
2. **Remote Computing for IoT:** This research demonstrates applications of utilizing dedicated external agents to provide additional computing resources for less powerful devices. IoTAMU's capabilities developed in this research provide added layers of security for IoT devices.
3. **Identical Device Model Classifier (IDMC):** IDMC provides a methodology to measure the similarities in the observed network signatures and identify identical-model devices.

4. **Network Signature Modification Algorithm:** This research presents an algorithm to uniquely modify the observed network signatures of the devices based on the distributions of previously observed signatures.

1.8 Thesis Overview

The rest of this research is organized in five chapters. Chapter II provides a summary of the current state of IoT security, the Wi-Fi protocol, and other related work. Then, in Chapter III, the design process of the IoTAMU is described and its encryption capability is demonstrated via an independent Proof-of-Concept (POC) experiment. It is followed by a description of the methodology used to evaluate the spoofing capability of IoTAMU in Chapter IV. Next, Chapter V presents and analyzes the results from the experiments. And lastly, Chapter VI highlights the major conclusions drawn from this research and provides potential future work.

II. Background and Literature Review

2.1 Overview

This chapter provides an overview of the current challenges facing IoT security in Section 2.2. Next, it describes the Wi-Fi protocol and how it contributes to vulnerabilities in IoT. It also provides an overview of the encryption schemes and PCA used in this research. Finally, it concludes with a discussion of related research efforts in IoT security.

2.2 Internet of Things Security

The IoT represent the new wave of embedded technologies with the added functionality of remote connectivity. Examples of IoT include the sensory devices in Supervisory Control and Data Acquisition (SCADA) networks used for remote control, medical devices with Internet connectivity for remote monitoring, and baby monitors used in smart home networks. Since its inception, the IoT industry has been one of the fastest growing in the world, projected to reach $1 trillion by the year 2022 [1]. However, the rapid growth of the industry is rivaled by the increasing number of vulnerabilities found on these devices. Examples of vulnerabilities that have been discovered are listed in public websites such as Common Vulnerabilities and Exposures (CVE) [5]. There are many vulnerabilities in its hardware and software implementation; Section 2.2.1 follows the work of Park et al. [2] to provide a high-level overview of the different classes of security risks associated with IoT.

2.2.1 Classes of Vulnerabilities Found in IoT

2.2.1.1 Insecure Storage and Communication

One of the main vulnerabilities in IoT is insecure storage and communication. The devices often lack the hardware capacity to perform expensive computations required for encryption; even if encryption is supported, many schemes use lightweight algorithms that can be bypassed with relative ease. In fact, a 2014 study by HP found that 70 percent of IoT devices used in the study communicated without encryption [6]. This vulnerability compromises both confidentiality and integrity of the device by allowing an adversary to eavesdrop on the network communication, and manipulate the data to their advantage, respectively. Strong encryption is able to mitigate these risks through the encryption of the message itself and Message Integrity Check (MIC), which verifies that the message has not been altered [7]. Cloud services such as Google [8] and Amazon [9] provide remote computing resources that can aid in securing the devices, including data encryption. However, using third party services introduce the problem of disclosure of sensitive information discussed in Section 2.2.1.2, and other vulnerabilities associated with cloud networks, which is out of scope of this research. This research focuses on mitigating the breach of data confidentiality on devices with limited/lack of encryption.

2.2.1.2 Collection and Disclosure of Sensitive Information

IoT devices both knowingly and unknowingly collect information on their surroundings. A couple of most prominent examples include medical devices [10], which collect information about patients, and embedded SCADA systems [11], which collect information about various processes being monitored. If the devices are not properly secured, sensitive information stored on these devices is vulnerable to attacks from malicious actors, which can result in information leakage. The implications of the in-

formation leakage can range from invasion of privacy to a national security threat, if the compromised device served a critical function such as those in a SCADA network.

Unfortunately, it is not just the data directly stored and exchanged on the devices that are vulnerable. One of the main challenges in dealing with IoT is their varying interactions within the network, which can lead to information leakage for which the user may not be aware. Examples of information leakage in IoT include device fingerprinting, and pattern-of-life modeling as seen in the work of Beyer et al. [3] discussed in Section 2.6.

2.2.1.3 Poor Authentication Mechanism

One of the most common vulnerabilities leveraged by an adversary is a device's poor authentication mechanism. Many devices are allowed to operate with pre-installed default admin credentials. In addition, even if the users are required to change the credentials before operating, many devices do not support setting secure passwords through password policies that limit the number of characters or the type of characters allowed in the password. This vulnerability allows a malicious actor to take control of the device, allowing them to gain access to any stored data or use the device as a pivot into the internal network to launch further attacks.

2.3 Wireless Protocols

2.3.1 Wi-Fi

Wi-Fi is one of the most common communication protocols used by the IoT [12]. The IEEE 802.11 (hereafter referred to as 802.11) standard specifies wireless local area network protocol for the physical and the link layers in the TCP/IP network architecture [13]. As seen in Figure 1, a wireless network consists of the following components: (1) wireless hosts, (2) wireless communication links, (3) base station, and

(4) external network [7]. Within the network, wireless devices (i.e., hosts) connect to the external network via different wireless communication link technologies by associating with a base station or an Access Point (AP). While an AP connects to the external network via a router, many modern APs designed for home networks integrate a router. When the AP is set up in a secure network, it is assigned a Service Set Identifier (SSID), the name visible to the hosts in proximity, a channel number that defines the frequency range of communication, and a secret passphrase. If a client (i.e., host) wishes to associate with an AP in a secure network, the client must first authenticate itself to the AP with the passphrase.

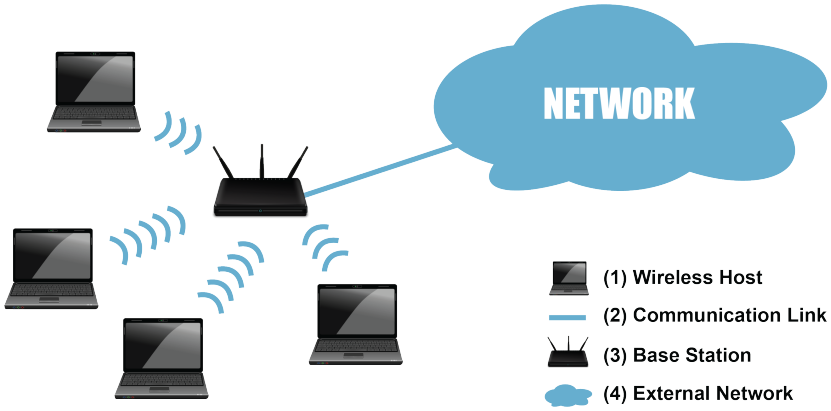


Figure 1: An architecture of a wireless network.

The unit of data exchanged in a network is represented by a 802.11 frame (Figure 2). Only the following fields are of interest in this research: destination Media Access Control (MAC) address (Address 1), source MAC address (Address 2), and Basic Service Set Identifier (BSSID), which is the MAC address of the AP's wireless interface (Address 3). A MAC address is a 48-bit number that uniquely identifies the device that is assigned by the manufacturer.

In a protected network, the payload of the 802.11 frame remains encrypted. The encryption scheme used in the network setup of this research is Wireless Protected Access 2 (WPA2). Under WPA2, the payload is encrypted using the Pairwise Tran-

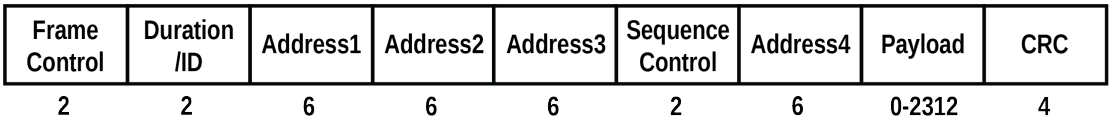


Figure 2: The 802.11 frame (numbers indicate the length of field in bytes).

sient Key (PTK), crafted using the WPA2 Preshared Key (PSK) (derived from the passphrase and SSID), MAC address of AP, MAC address of client, a nonce created by the AP, and a nonce created by the client. Figure 3 depicts the initial WPA2 4-way handshake between the client and the AP, in which the components of the PTK are exchanged. Once the client and the AP are authenticated and associated, they each derive the Pairwise Master Key (PMK) from the preinstalled PSK. Then, the AP sends a nonce to the client, with which the client derives the PTK. Next, the client sends its nonce to the server with the MIC encrypted with the PTK. Using the client nonce, the server generates the PTK and uses it to verify the encrypted MIC received from the client. If the MIC checks out, the AP installs the PTK and sends a confirmation message to the client, to which the client responds with an acknowledgement to complete the 4-way handshake. In this research, it is assumed that the only encryption that is performed in the IoT network is that of Wi-Fi, substantiated by [14] discussed in Section 2.6.

Each wireless station has a Wireless Network Interface Controller (WNIC) through which it sends and receives wireless traffic. In a traditional setting, the WNIC is in managed mode, only picking up wireless traffic addressed to itself. However, there are two other modes in which the WNIC can be set to pick up additional traffic: promiscuous mode and monitor mode. In promiscuous mode, the WNIC captures all traffic associated with the BSSID of the associated AP. Similarly in monitor mode, the WNIC captures all wireless traffic regardless of the BSSID associated with the traffic. In this research, the WNIC is set to monitor mode to capture all wireless

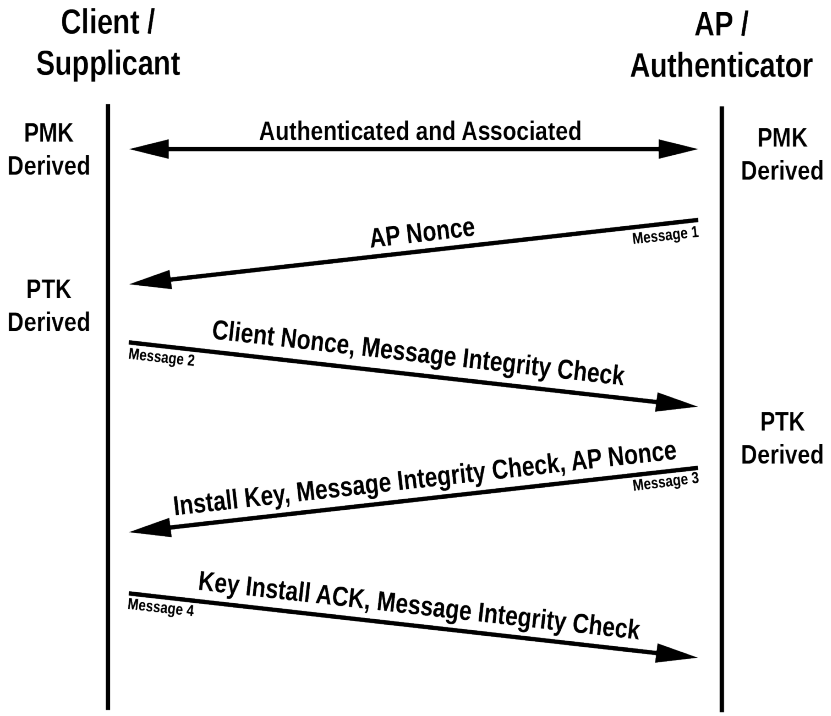


Figure 3: The WPA2 4-way handshake between the client and the AP.

traffic [15].

The security provided by the WPA2 standard is a significant improvement over that of its predecessor, Wired Equivalent Privacy (WEP) [16]. However, there are still studies that demonstrate the effectiveness of WPA2 password cracking attacks that allow an adversary to gain access to the network such as those of [17][18]. The WPA3 standard currently in development by the Wi-Fi Alliance is able to mitigate some of the vulnerabilities that exist in WPA2, including dictionary attacks [16]. However, discussion of specific protocols used in the WPA3 standard is out of scope of this research.

2.3.2 Other Wireless Protocols

In addition to Wi-Fi, there are other wireless protocols frequently used by IoT devices for communication, including the IEEE 802.15.1 (Bluetooth) [19] and 802.15.4

(Zigbee) [20] standards, and the ITU-T G.9959 recommendation (Z-Wave) [21]. Vulnerabilities similar to those discussed here are present in these protocols and pose a challenge when securing the IoT devices [22][23]. In Bluetooth, device-specific information such as the device name, services, and technical specifications are exchanged, which can be used in device fingerprinting. Likewise, ZigBee and Z-Wave devices advertise identifiers similar to the MAC addresses in Wi-Fi which uniquely identifies the device.

2.4 Public Key Encryption (PKE) and Symmetric Key Encryption (SKE)

There are two main encryption schemes being used in networking: Symmetric Key Encryption (SKE), and Public Key Encryption (PKE) [7]. Whereas SKE uses the same key to encrypt and decrypt messages, PKE uses one key to encrypt and another key to decrypt.

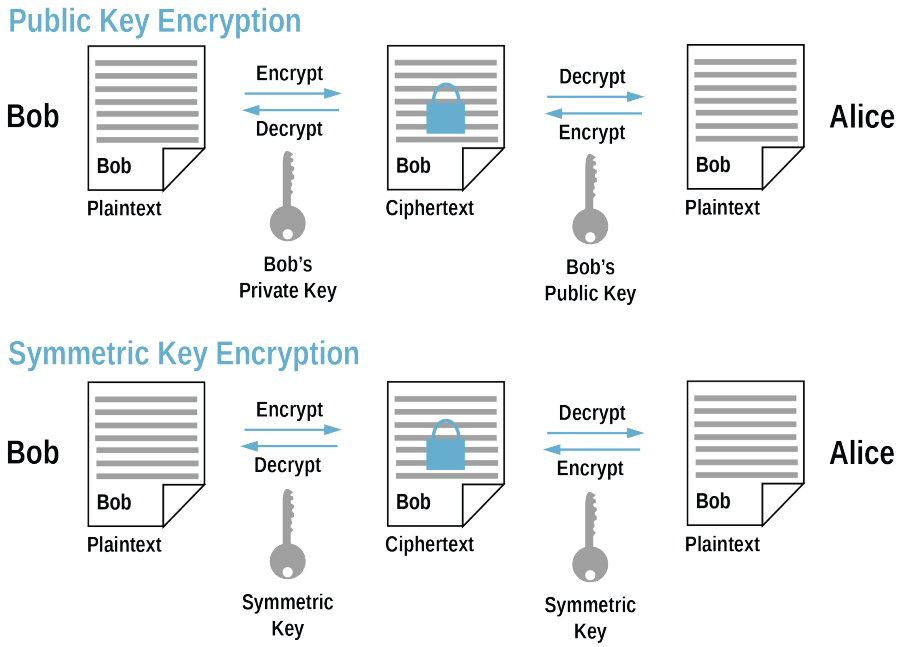


Figure 4: An overview of the Public and Symmetric Key Encryption.

PKE consists of a public and a private key pair, and the decryption and encryption

algorithm. A private key is kept secret by the user who generated the key, and the associated public key is distributed to the public. Messages can be encrypted using either a public or a private key, but they must be decrypted using the other corresponding key. Popular PKE algorithms include Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC). An example of this exchange is depicted in Figure 4. Here, Bob wants to exchange a secret message with Alice. In this scenario, Bob has already shared his public key with Alice. Bob first encrypts his plaintext secret message using his private key, converting into a ciphertext. After Alice receives the ciphertext, she decrypts it using Bob's public key. She then composes a secret response and encrypts the message using Bob's public key. Upon receiving the encrypted message, Bob decrypts it using his private key to read Alice's secret message.

Unlike PKE, SKE consists of one symmetric key and the decryption and encryption algorithm. Therefore, for the recipient to read the encrypted message, they must have access to the same symmetric key used for encryption. Popular SKE algorithms include Blowfish and Advanced Encryption Standard (AES). This research uses AES in the implementation of the encryption capability for IoTAMU. In the SKE scheme depicted in Figure 4, Bob uses a symmetric key shared with Alice to encrypt his secret message. Once Alice receives the ciphertext, she decrypts the message with the symmetric key. Then, she generates a response and encrypts the message using the same key. Finally, Bob decrypts the encrypted response with the symmetric key to read Alice's message.

On top of confidentiality, PKE is also used for digital signatures and certificates to prove the identity of the originator of the message. Because a private key is unique to an entity, a message encrypted with a unique private key undeniably proves the identity of the originator. Therefore, a digital signature created using the private key

of the sender provides authenticity and non-repudiation for the message.

An important application of digital signatures is found in digital certificates signed by a Certificate Authority (CA). Digital certificates are used as a means to validate the identity of the sender. It contains the public key of the sender, the information of the issuer (CA), as well as the digital signature of the issuer. If the receiver trusts the issuer of the digital certificate and validates the digital signature of the issuer using the issuer's public key, then the receiver can trust the sender's public key contained in the certificate. An example of this mechanism is seen in the Transport Layer Security (TLS) protocol, where the authenticity of a web server is validated via the use of digital certificates [24]. The authentication component of IoTAMU developed in this research is modeled after this protocol.

2.5 Principal Component Analysis (PCA)

PCA is a dimension reduction technique that transforms a set of potentially correlated features into a smaller set of orthogonal features, otherwise known as principal components [25]. It is a linear transformation where each coordinate (i.e., principal component) points to the direction of highest variance in the data, such that each coordinate is orthogonal to one another. Performing PCA on a set of data has a few important merits. First, it reduces the dimension of a large feature set down to a manageable size. Second, it helps extract the parts of features that retain the greatest amount of information (i.e., variation) between the samples. Lastly, because it is a dimension reduction technique, it allows visualization of a high dimensional data; this allows clustering of similar samples, giving it a characteristic similar to that of unsupervised machine learning techniques such as k-means clustering.

Traditionally, before the derivation of the principal components, the data is centered around the origin and normalized by subtracting the mean and dividing by the

standard deviation for each feature. As illustrated in Figure 5, the first principal component is derived by finding a line through the origin, where the distance from the projection of each point onto the line to the origin is maximized, or where each point's orthogonal distance to the line is minimized. This method can also be expressed by the problem of maximizing the variance around the origin by choosing a vector v such that the following condition is satisfied

$$Variance = max \left(\frac{\sum_{i} (x_i^T v)^2}{n-1} \right) \tag{1}$$

where n is the number of points, x_i^T is the transpose of the x_i matrix describing the i^{th} point, and $x_i^T v$ is the length of the projection of point x_i onto v. Each subsequent principal components are derived by choosing a line through the origin orthogonal to the previous principal components that satisfies the above conditions. The maximum number of principal components that can be derived is the smaller of the number of samples or the number of features. Once all of the principal components are derived, the new data is represented by

$$\hat{x_i} = \begin{bmatrix} v_1^T x_i \\ v_2^T x_i \\ \cdots \\ v_k^T x_i \end{bmatrix} \tag{2}$$

where the new data $\hat{x_i}$ is represented by the projections of the original data x_i onto k principal components. The set of projections of each original data onto the k^{th} principal component are known as the Principal Component Score (PCS); the newly derived data can be visualized by plotting the PCSs of the first two or three principal

components. Steps one and two in Figure 5 shows the derivation of the PCS for principal components one and two from points $x1$, $x2$, and $x3$. Then, in step three, each point is visualized in the newly transformed space. Whereas the example in Figure 5 depicts the derivation of the principal components in a 2-dimensional space, PCA becomes a powerful visualization tool on higher-dimensional data by allowing the multi-dimensional data to be represented in a 2 or 3-dimensional space.

Furthermore, through the creation of the principal components, PCA allows for the abstraction of the numerous features in the original data. This becomes useful in a system where little is known about which features are important in the system's overall behavior and how they interact. Therefore, it facilitates a shift in focus from determining which features are important in the behaviors of different systems, to how the overall behaviors of systems differ. The classifier implemented in this research takes advantage of this characteristic of PCA to measure the closeness of the network signatures (comprised of 36 features) of pairs of devices.

A scree plot is a bar graph that depicts the percent variation of each component; it is useful in determining the number of subset of principal components to select for analysis. The percent variation of data accounted by each principal component can be calculated by taking the value derived in (1) and dividing by the sum of variance for all principal components. It is expressed by

$$\begin{aligned} \%Variation_k &= \frac{Variance_k}{TotalVariation} \\ TotalVariation &= \sum_k Variance_k \end{aligned} \tag{3}$$

where k is the k^{th} principal component. In practice, selection of the first set of principal components whose sum of percent variations account for 70% to 90% of the total is recommended for analysis [25]. For this research, the midway point (80%) is selected as the threshold for selection of principal components.

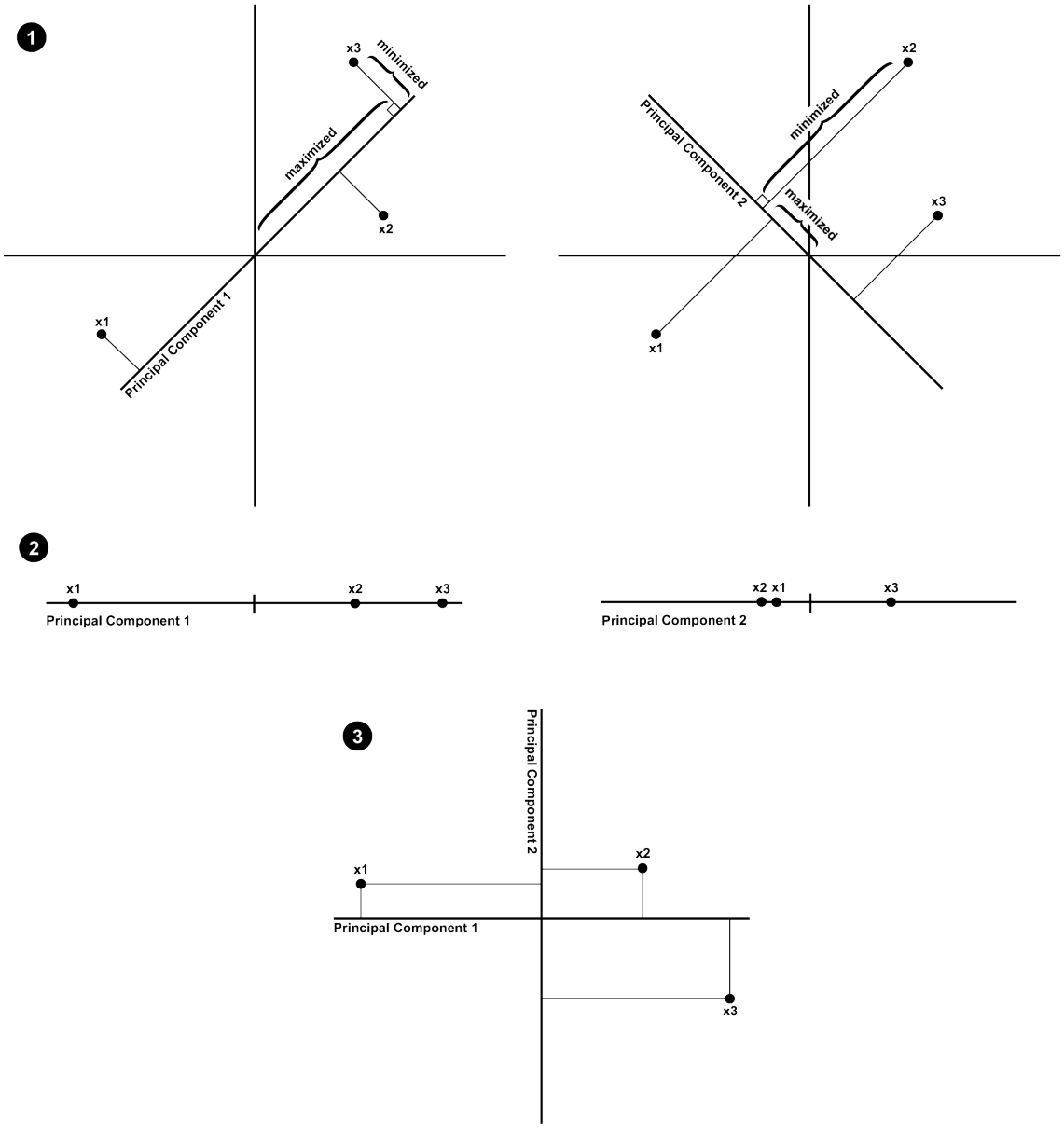


Figure 5: Derivation of the principal components in a 2-dimensional space.

2.6 Related Research

Recognizing IoT security flaws, many security experts have researched numerous attack surfaces of IoT. Current efforts, including the Internet of Things Project by Open Web Application Security Project (OWASP) [26], provide valuable resources to help manufacturers and end users secure their devices. However, as discussed in Section 2.2.1.1, due to physical limitations of the devices, meaningful security measures that often require large computational power such as encryption remain elusive for many devices. Unfortunately, this leaves security, in large part, to those who deploy them.

Sectors that deploy IoT, such as the U.S. government, have identified the vulnerabilities and have recommended mitigation policies and guidelines with limited success. The Government Accountability Office (GAO) is an entity that audits and researches various issues pertaining to the U.S. government, including the use of IoT devices within the DoD. Their 2017 report highlighted multiple risks associated with the IoT including poor built-in security of the devices, limited encryption, and possible exploitation from supply chain [27]. In addition, the report concluded that the DoD's current policies on information systems and cybersecurity are neither effectively implemented nor adequately address the growing security concerns with the IoT. In the following year, the GAO identified establishing a cybersecurity strategy for IoT as one of the DoD's most critical challenges. Although the DoD's senior leadership has recognized the problem, no tangible actions have been taken at the time of the 2018 report [28]. Most recently in Septempter of 2019, the Internet of Things Cybersecurity Improvement Act of 2019 has been reintroduced in the U.S. Senate to revitalize the failed Internet of Things Cybersecurity Improvement Act of 2017 and the Internet of Things Federal Cybersecurity Improvement Act of 2018. But, at the time of this research, it has yet to pass the Senate [29].

Numerous studies so far have demonstrated the vulnerabilities that exist in IoT devices. Examples include those that exist in network cameras as seen in [30–32]. In particular, Ostrom and Sambamoorthy [31] showcased a series of attacks that can be launched against IoT cameras via Address Resolution Protocol (ARP) cache poisoning, a common technique used to eavesdrop on network traffic between hosts [15]. More recently, a study by Park et al. used passive network sniffing to demonstrate the pervasiveness of the vulnerability to eavesdropping 10 years after Ostrom and Sambamoorthy's work, by extracting the video feed from unencrypted data packets of a network camera [14].

Likewise, a 2015 study by Valasek and Miller demonstrated the risks of using the infotainment system found in modern vehicles [33]. Their study demonstrated a remote control attack that was able to successfully gain control of a vehicle while the driver was onboard. This finding led to the recall of 1.4 million vehicles by the automobile company [34].

In addition, the unintentional information leakage from an aggregate collection of data has the potential to reveal knowledge about its surroundings. Strava's fitness tracker application uses GPS information to determine the exercise patterns of the user. However, an analyst was able to show that by cross referencing publicly-available maps and the location data from the application, undisclosed information regarding U.S. military installations such as security guards' patrol routes could be inferred [35]. In addition, a recent study at Air Force Institute of Technology (AFIT) illustrated that by only using the unencrypted fields found in Wi-Fi frame headers, an adversary is able to sniff a network of over-the-counter IoT devices to accurately collect pattern-of-life information [3]. The authors were able to recognize certain patterns of encrypted Wi-Fi traffic to fingerprint the different devices present in the network, and determine the specific duration and time the user was present in the

smart home.

The threat posed by information leakage becomes more complex when statistical tools are utilized. Following the study by Beyer et al., Aragon improved the device-type classifier using several machine learning techniques including Random Forests, K-Nearest Neighbors (KNN), and Linear Discriminant Analysis (LDA) [4]. Aragon's classifiers were relatively successful in correctly identifying the device types with above $\sim 80\%$ precision for the smart home Wi-Fi devices. However, the classifier relied too heavily on the vendor information extracted from the first three bytes of the MAC address, resulting in a significant drop in performance ($\sim 50\%$ precision) after removal of the vendor information.

Similarly, Atkinson et al. used Random Forests on passively sniffed Wi-Fi traffic to infer different characteristics about a user (e.g., age, sex, hobbies, etc.) [36]. In their study, the authors downloaded some of the most popular applications from different genres, and created a list of characteristics one could infer from the usage of the application. After ensuring their machine learning classifier was able to classify the applications from sniffed traffic, the authors created a list of personas for the users based on different combinations of applications detected. Although the machine learning classifier suffered significantly in a live environment, it nonetheless demonstrated the feasibility of an adversary being able to infer various characteristics about a user based only on sniffed encrypted Wi-Fi traffic.

Over the years, several solutions have been proposed to mitigate the threats that exist in IoT networks. Examples of these approaches include using Software-Defined Networking (SDN) to develop Local Area Network (LAN) management schemes [37] [38], deploying edge gateways to encrypt the network traffic [39][40], and injecting spoofed network packets to modify the observed network signatures [3]. Although the SDN approach is often used to prevent a rogue device that has been compromised

from further infecting a network, it does not protect the confidentiality of a network from an adversary passively eavesdropping.

Data confidentiality of IoT can be protected through the use of an encryption agent. Doukas et al. presented a model which connected an external gateway to each IoT device. The gateway was paired with a cloud server that allowed the model to encrypt and decrypt the traffic before reaching the intended recipient [39]. Similarly, Hsu et al. developed a framework which utilized the enhanced capabilities of existing edge nodes on the network (e.g., router), to provide additional security for IoT, including encryption.

The aforementioned AFIT study demonstrated the effectiveness of injecting spoofed packets in disguising the smart home network [3]. In order to defeat the information leakage found in their study, the authors developed a spoofer on a Raspberry Pi that periodically sent out crafted network packets to mimic the devices present in the smart home. By sending packets of varying length and MAC addresses, they were able to both prevent their device classifier from correctly predicting the type of device, and hide the pattern-of-life information of the user.

Beyer et al., developed an algorithm that relied on packet generation from hard-coded information such as the packet length. However, this creates network signatures that are different than those of the underlying network, which can be isolated using data mining techniques. In the field of knowledge hiding in disguised data sets, Huang et al. demonstrated that disguising existing data with values sampled from a uniform distribution is an ineffective method that can be reversed through statistical analysis [41]. Instead, they illustrate that injection of noise that retains the overall distribution of the existing data is a more effective technique for knowledge hiding. Following their research, the spoofer for IoTAMU is implemented by generating artificial frames that mimic the signatures of the underlying network.

Table 1 provides a summary of the related research. Despite ongoing efforts in the field, no standout solution has been proposed that can be widely deployed in the near future. This research investigates the data confidentiality deficiencies present in IoT networks and seeks to develop an easily configurable model to enhance the confidentiality of an IoT network by synthesizing the encryption and spoofing agents discussed in this section. Ultimately, this thesis explores the feasibility of a PNP-style security agent that can be used to secure both the wireless and wired communications used in operations that rely on IoT, with minimal disruption of the existing network.

Table 1: Summary of Related Research

	Vuln	**Smart Home**	**Wi-Fi Analysis**	**Mitigation**		
				Spoofing	**Encryption**	**SDN**
Heffner (2013) [30]	X					
Ostrom and Sambamoorthy (2011) [31]	X					
Stanislav and Beardsley (2015) [32]	X					
Valasek and Miller (2015) [33]	X					
Atkinson et al. (2018) [36]	X		X			
Miettinen et al. (2017) [37]	X	X				X
Demetriou et al. (2017) [38]	X	X				X
Doukas et al. (2012) [39]	X				X	
Hsu et al. (2018) [40]	X	X			X	
Beyer et al. (2018) [3]	X	X	X	X		
Aragon (2019) [4]	X	X	X			
Park et al. (2019) [2]	X					
Park et al. (2019) [14]	X	X	X	X	X	
Park (2020)	X	X	X	X	X	
Vuln: vulnerability assessment						

2.7 Background Summary

This chapter begins with a discussion of the high-level vulnerabilities that exist in IoT. It also provides an overview of the Wi-Fi protocol commonly used as the

standard communication protocol for IoT. A brief description of the SKE and PKE schemes, and PCA are also provided. The chapter concludes with a survey on recent research efforts that highlight the different types of vulnerabilities that exist in IoT, as well as potential solutions to secure exposed networks. Although a wide spectrum of solutions have been proposed, many approaches require changes to the existing protocols, which is difficult to implement in large networks such as those of the DoD. This thesis contributes to the field of IoT security through the development of a PNP-style security agent and demonstrating the feasibility of its deployment in large networks.

III. IoTAMU Design

3.1 Overview

To mitigate the vulnerable wireless communications innate in many IoT devices, this research develops and examines the effectiveness of a novel security agent architecture: IoTAMU. It serves as a proxy deployed in between a traditional smart home router and IoT devices to secure their communication. This is accomplished by using encryption to protect the traffic flowing between the agent and the RI of the IoT, and using spoofing to conceal the wireless traffic between the agent and IoT devices. IoTAMU performs automatic classification of active and passive states to generate the network signatures of each device. New network signatures are generated from those of existing devices, which is used by IoTAMU to spoof packets to mask the underlying communication of the devices. The Identical Device Model Classifier (IDMC), capable of classifying identical model devices, is developed to measure the effectiveness of the spoofer's ability to modify the observed signatures of the devices. This chapter describes the functionality of IoTAMU in a smart home network, list of tools used, network setup of the research, the architecture of IoTAMU, and a series of pilot studies that helped guide its design decisions.

3.2 Internet of Things Active Management Unit (IoTAMU)

IoTAMU is a security agent developed to mitigate the eavesdrop vulnerabilities found in IoT devices. Figure 6 illustrates IoTAMU's impact in a hypothetical smart home network. Most often, a router serves as the default gateway in a smart home network and connects all devices to the external network. In this model, the communication between the IoT device and the RI may or may not be encrypted. Even if wireless encryption such as WPA2 is assumed, an eavesdropper sniffing on the net-

work may be able to intercept the data by cracking the WPA2 key, if the application data itself is not encrypted. Unfortunately, as demonstrated by Park et al., it is still common to find IoT devices that send traffic in cleartext [14]. Furthermore, as discussed in Section 2.6, encrypted IEEE 802.11 frames contain unencrypted fields (e.g., MAC address, packet length) that allow an adversary to infer various information about the network based on the observed traffic patterns, including but not limited to device fingerprinting, activity identification, and pattern-of-life modeling.

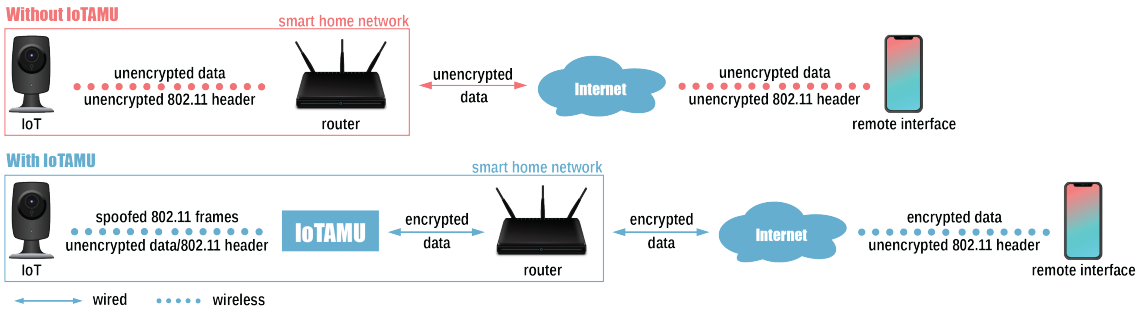


Figure 6: Depiction of a typical smart home network with and without IoTAMU.

With the introduction of IoTAMU, the security agent acts as a central gateway for all IoT, effectively isolating them from the rest of the hosts present in the network. This direct communication between the IoT devices and IoTAMU can be accomplished by setting the IoTAMU as an AP and having the IoT devices associate with it. Additionally, its two primary capabilities, encryption and spoofing, secure the application level data and mask the network traffic patterns, respectively, to secure the IoT network.

In a typical communication between an IoT device and its RI, there are two sides of wireless communication that are vulnerable to passive sniffing: communication between an IoT and its associated AP and that of the RI and its associated AP. Spoofing and encryption capabilities of IoTAMU respectively provide an additional layer of security for each side of wireless communication.

IoTAMU spoofs crafted 802.11 frames to conceal device-specific patterns of network signatures that can be observed in wireless communications between an IoT device and its AP (i.e., IoTAMU), preventing information leakage from unencrypted fields of encrypted 802.11 frames using data mining techniques. However, if encryption of data is not directly performed by the IoT device, its application-layer data remains unencrypted and is vulnerable to eavesdropping from those with access to the wireless network. When packets are received at IoTAMU's wireless interface, their application-layer data are encrypted and forwarded to the RI. Once the RI receives the packet, a decryption agent residing in the RI (in the form of a background process) decrypts the payload and forwards it to the intended application. Packets generated from the RI are encrypted by the same agent before being forwarded to IoTAMU, where they are decrypted and forwarded to the IoT device. Therefore, payloads in wireless communications between the RI and its AP are encrypted and secured against eavesdropping from those with access to the wireless network. Implementations of IoTAMU's capabilities are described in Section 3.6.

3.3 Tools

Table 2 provides the list of open source tools used in this research. Wireshark and the aircrack-ng suite are used for network reconnaissance and analysis. Python, iptables, ebtables, hostapd, and Scapy are used to implement IoTAMU. Finally, R is used to generate the network statistics and analyze the experimental results.

Table 2: List of Tools Used

Name	Version	Description
Wireshark	3.2.0	Software used for packet analysis [42]
airodump-ng	1.5.2	Software used to capture network traffic [43]
airmon-ng	1.5.2	Software used to configure the wireless card for passive network sniffing [43]
Python	3.5.2	Programming language used to implement functions of IoTAMU [44]
Scapy	2.4.3	Python package used to inject network packets [45]
R	3.5.0	Programming language used for statistical analyses [46]
iptables	1.8.3	Software used for network layer packet filtering [47]
ebtables	1.8.3	Software used for link layer packet filtering [48]
hostapd	2.9	Software used to enable the AP functionality on IoTAMU [49]

3.4 Devices

Table 3 provides the list of devices used in this research. The IoT devices are selected to maximize the variety in each type of devices (e.g., camera). When possible, devices from different manufacturers are selected for analysis. Otherwise, different models from the same manufacturers are selected. Two devices of the same manufacturer and model are used for the purpose of classifying identical-model devices. In this research, two different configurations of IoT device and RI pairs are used. In the first configuration, `Phone1` acts as the RI for the actual IoT devices; in the second configuration, two laptops are used to pose as an IoT device (`Laptop1`) and its RI (`Laptop2`). IoTAMU is implemented on `Laptop3`. Lastly, three routers are used to set up simulated internal and external networks with respect to the IoT devices.

Table 3: List of Hardware Used

Name	Model	Description
Wansview Wireless Camera	Q3S X Series	IoT camera
Belkin Wireless Camera	F7D7602v2	IoT camera
Dropcam Wireless Camera	DROPCAM3-HD/B	IoT camera
Belkin Wireless Switch	F7C027	IoT switch
Belkin Wireless Switch	F7C063	IoT switch
TP-Link Wireless Switch	HS100	IoT switch
TP-Link Wireless Light Bulb	LB100	IoT light bulb
LIFX Wireless Light Bulb	A19	IoT light bulb
Samsung Smartphone	Galaxy S8 Android version 9	Smartphone used to remotely control the IoT devices in configuration 1
Microsoft Laptop (Laptop1)	Surface Book 2 Windows 10 Pro version 1903	Laptop used to pose as an IoT device in configuration 2
Dell Laptop (Laptop2)	Alienware 15 R2 Windows 10 Home version 1903	Laptop used to pose as a RI in configuration 2
Lenovo Laptop (Laptop3)	Thinkpad W541 Kali 2018.4	Laptop used to implement IoTAMU
Ubiquiti Router (Router1)	ER-X	Router that bridges the Internet, external, and smart home networks
Netgear Router (Router2)	Nighthawk(R) X4S R7800	Router that connects the RI to the smart home network
Motorola Router (Router3)	MG7540	Router that connects the internal/ external network to the Internet
Alfa Card	AWUS036NHA	WNIC used to sniff Wi-Fi traffic
Device specs: Laptop1 – Intel i5-7300U CPU, 8 GB RAM; Laptop2 – Intel i7-6700HQ CPU, 16 GB RAM; Laptop3 – Intel i7-4910MQ CPU, 8 GB RAM		

3.5 Network Setup

Figure 7 depicts the network setup, and examples of the packet flow from the RI to IoTAMU via direct communication and via cloud server. At each hop, any changes in the Internet Protocol (IP) address and port number from Network Address Translation (NAT) on `Router1`, `Router2`, and `Router3` are highlighted in red. The NAT table in the figure summarizes these changes, with the changes for packets routing from the RI to the cloud server on the left and those routing from the cloud server to IoTAMU on the right. This research simulates an external and an internal (smart home) network via three subnets: (1) `192.168.1.0/24`, (2) `172.16.0.0/24`, and (3) `10.0.0.0/24`. A fourth subnet `192.168.0.0/24` is used to enable a connection to the Internet as select IoT devices need to connect to cloud servers to establish a connection with the RI. In a direct communication from an RI to an IoT device, the packet flows from the RI, `Router2`, `Router1`, and IoTAMU before reaching the IoT device. If a cloud server is utilized by the IoT device, the packet is routed to the server from the RI via `Router2`, `Router1`, `Router3`, and the Internet; then routed to the IoT device from the server via `Router3`, `Router1`, and IoTAMU. A summary of the MAC and IP addresses of all devices set up in the network is provided in Table 4.

Figures 8 and 9 depict configuration 1 and 2 of the two IoT device and RI pairs used in this research. In configuration 1, `Phone1` is used as the RI to interact with the actual IoT devices; in configuration 2, `Laptop1` and `Laptop2` are posed as an IoT device and its RI, respectively. Configuration 1 is used in experiments 1 and 2 (see Sections 4.8.2 and 4.8.3), and configuration 2 is used in the POC experiment of IoTAMU's encryption functionality (see Section 3.6.3.1) and experiment 3 (see Section 4.8.4).

Table 4: Network Address of Devices

Name	**Manufacturer**	**MAC**	**IP**
Camera1[a] (C1)	Belkin	EC:1A:59:E4:FD:41	192.168.1.24
Camera2 (C2)	Dropcam	30:8C:FB:3A:1A:AD	192.168.1.18
Camera3 (C3)	Wansview	28:AD:3E:38:6F:B6	192.168.1.3
CameraTest[a] (CT)	Belkin	EC:1A:59:E5:02:0D	192.168.1.25
LightBulb1[a] (LB1)	LIFX	D0:73:D5:26:B8:4C	192.168.1.6
LightBulb2 (LB2)	TP-Link	B0:4E:26:C5:2A:41	192.168.1.5
LightBulbTest[a] (LBT)	LIFX	D0:73:D5:26:C9:27	192.168.1.20
Switch1[a] (S1)	Belkin	14:91:82:CD:DF:3D	192.168.1.21
Switch2 (S2)	Belkin	60:38:E0:EE:7C:E5	192.168.1.12
Switch3 (S3)	TP-Link	70:4F:57:F9:E1:B8	192.168.1.8
SwitchTest[a] (ST)	Belkin	B4:75:0E:0D:94:65	192.168.1.22
Phone1 (P1)	Samsung	DC:EF:CA:CF:7E:6E	10.0.0.3
Laptop1 (L1)	Microsoft	C4:9D:ED:2D:AC:83	192.168.1.17
Laptop2 (L2)	Dell	9C:B6:D0:0D:6E:05	10.0.0.4
Laptop3 (IoTAMU)	Lenovo	8C:AE:4C:FF:99:98 A4:C4:94:3E:20:9C	192.168.1.2 (WD) 192.168.1.254 (WL)
Router1 (R1)	Ubiquiti	80:2A:A8:9E:45:5A 80:2A:A8:9E:45:5B 80:2A:A8:9E:45:5C	192.168.1.1 (eth0) 172.16.0.1 (eth1) 192.168.0.2 (eth2)
Router2 (R2)	Netgear	78:D2:94:4D:AB:3F 78:D2:94:4D:AB:3E	172.16.0.2 (WD) 10.0.0.1 (WL)
Router3 (R3)	Motorola	E0:98:61:FB:07:17	192.168.0.1
WD: wired, WL: wireless			

[a] Test devices within each type are same models as the first devices in their respective types

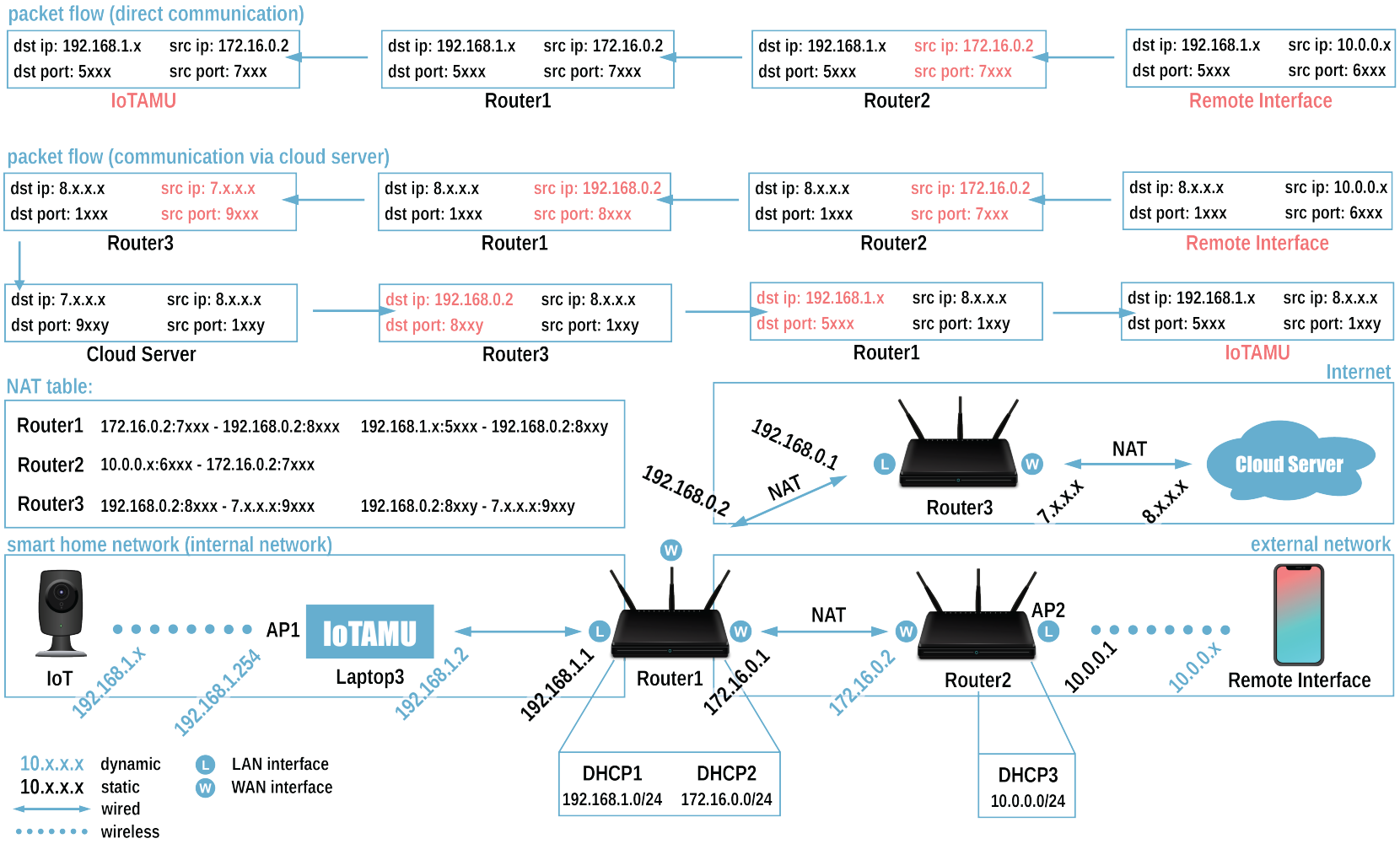


Figure 7: Examples of packet flow from the RI to IoTAMU and the network setup of the experiment.

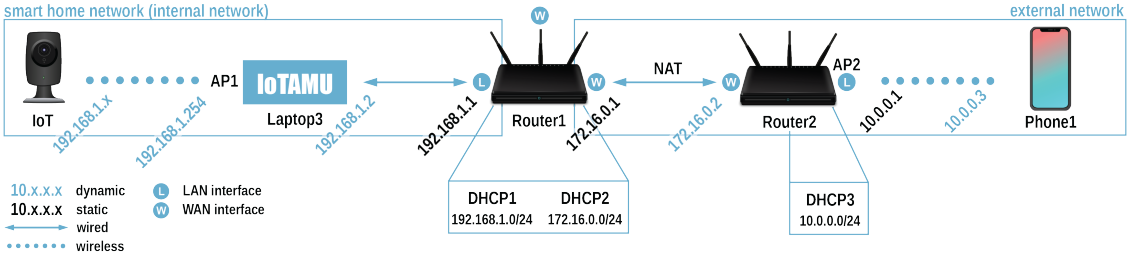


Figure 8: The network setup using Phone1 as the RI to interact with the actual IoT devices (Configuration 1).

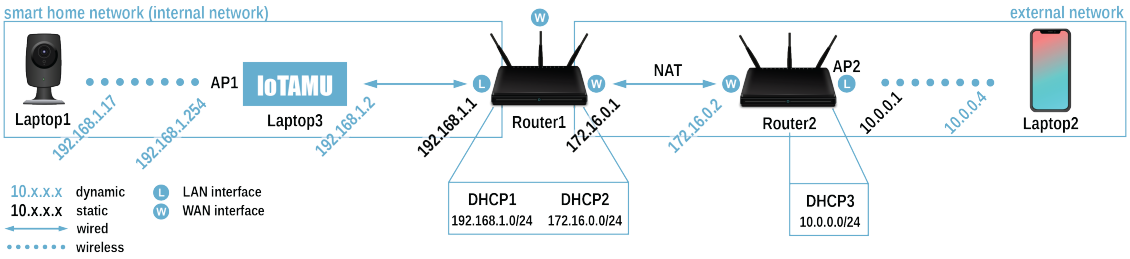


Figure 9: The network setup using Laptop1 as the IoT and Laptop2 as its RI (Configuration 2).

The first subnet `192.168.1.0/24` is the internal network where the IoT devices and IoTAMU reside. `Router1` acts as the gateway router for the smart home network with one of its interfaces (`eth0`), with a static IP address of `192.168.1.1`, representing the LAN interface for the simulated internal network (the routing table for `Router1` is shown in Figure 10). The interface is directly connected via an Ethernet cable to the interface on IoTAMU, which has a dynamic IP address of `192.168.1.2` assigned by `DHCP1`, the Dynamic Host Configuration Protocol (DHCP) server residing in `Router1` (Figure 11). In addition, IoTAMU has a WNIC which is set as the AP for the IoT devices. The WNIC has a static IP address of `192.168.1.254`. Connecting to the AP are the IoT devices whose IP addresses are dynamically assigned by `DHCP1`. In an actual deployment of IoTAMU, the gateway router (`Router1`) as well as IoTAMU should have NAT functionalities enabled to prevent external devices from communicating directly with the internal devices. However, for ease of access and

analysis in this research, NAT has been turned off on the internal side of the network setup.

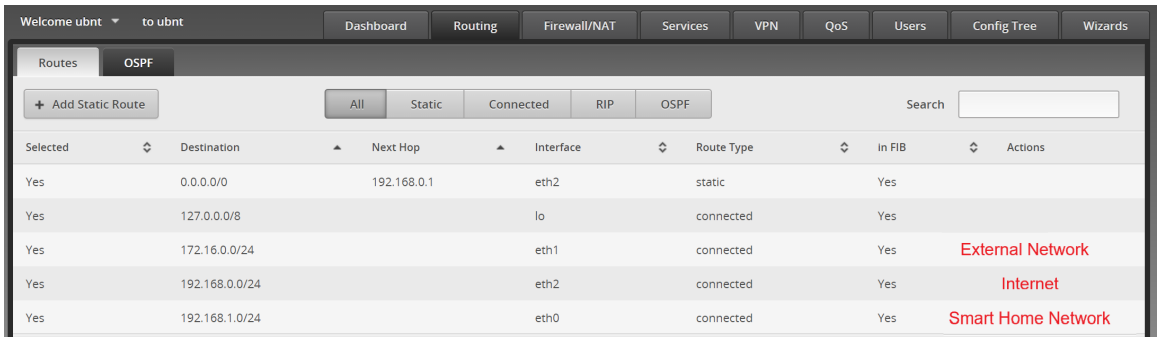


Figure 10: Routing table for the three interfaces on Router1.

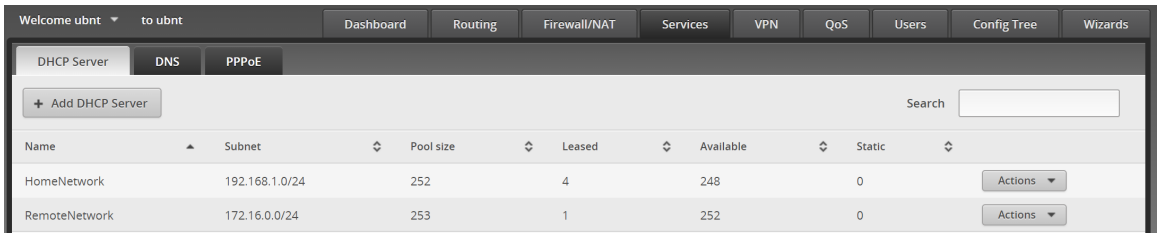


Figure 11: DHCP servers residing on Router1.

AP functionality of IoTAMU is implemented using hostapd. To enable this, `wlan0` interface is first set to AP mode by editing the `/etc/network/interfaces` config file, and creating the `hostapd.conf` file to be used as an input for hostapd. Both configuration files are provided in Appendix A. The AP is set up to use the WPA2 encryption scheme with a passphrase for security. Either a bridged or a NAT-enabled wireless network can be configured via hostapd. Although a NAT-enabled network is better for security, IoTAMU is set up as a bridged network for ease of access to the connected devices and analysis for the purpose of this research.

IoTAMU is implemented on the Lenovo Thinkpad (`Laptop3`) model W541 I7-4910MQ running on Kali version 2018.4 as its operating system. It must be initialized on startup via the script provided in Appendix B for AP and firewall, Appendix C for encryption, and Appendix D for spoofer. The scripts are run via the following

commands:

```
sh iotamu.sh
python3 [proxy_server.py|proxy_server2.py]
python3 spoofer.py [csv file containing network signatures]
```

where `proxy_server.py` and `proxy_server2.py` perform decryption on packets originating from the RI, and encryption on those from IoT devices, respectively. For the remainder of this research, it is assumed that the first script is always running, and the latter two are only enabled when specified.

The second interface on `Router1` (`eth1`) represents the Wide Area Network (WAN) interface connecting to the external network (`172.16.0.1`). It is connected to the WAN interface with a dynamically assigned IP address of `172.16.0.2` on `Router2` via an Ethernet cable. To mimic the WAN address being assigned by an Internet Service Provider (ISP) for a typical smart home network, the IP address of `Router2`'s WAN interface is assigned by a second DHCP server residing in `Router1` for the `172.16.0.0/24` subnet. `Router2` also acts as the AP to associate with the RI via a wireless connection. In this research, the internal subnet created by `Router2` (`10.0.0.0/24`) acts as the external network. IP addresses of the smartphone (`Phone1`) and the laptop (`Laptop2`) that acts as the RI are dynamically assigned by `DHCP3` residing in `Router2`.

Finally, the third interface on `Router1` (`eth2`) with the IP address `192.168.0.2` enables the connection to the Internet from the simulated internal and external networks. The interface is connected via an Ethernet cable to a LAN interface (`192.168.0.1`) on `Router3`, which connects to a local ISP. Because the particular model of router does not have a routing table that can be modified, NAT is enabled on `eth2` of `Router1` to correctly route the packets destined for the simulated network from the Internet.

The APs in `Router2` and IoTAMU operate in the 2.4 GHz band under the IEEE 802.11n standard, with SSIDs "NETGEAR20" and "IoTAMU", respectively. In addition, they are set up on channels one and six, respectively, to minimize their interference and model two distinct networks. `Router3`'s AP, which is not a part of this research, is set up on channel 11 to prevent interfering with those of `Router2` and IoTAMU.

3.6 IoTAMU Architecture

There are four components to IoTAMU: (1) authentication, (2) firewall, (3) encryption, and (4) spoofer. Its architecture and how each component interacts with the rest of the network is depicted in Figure 12. This research primarily focuses on the spoofing component of IoTAMU, as other components can be easily implemented using existing protocols. The following sections describe each component in detail.

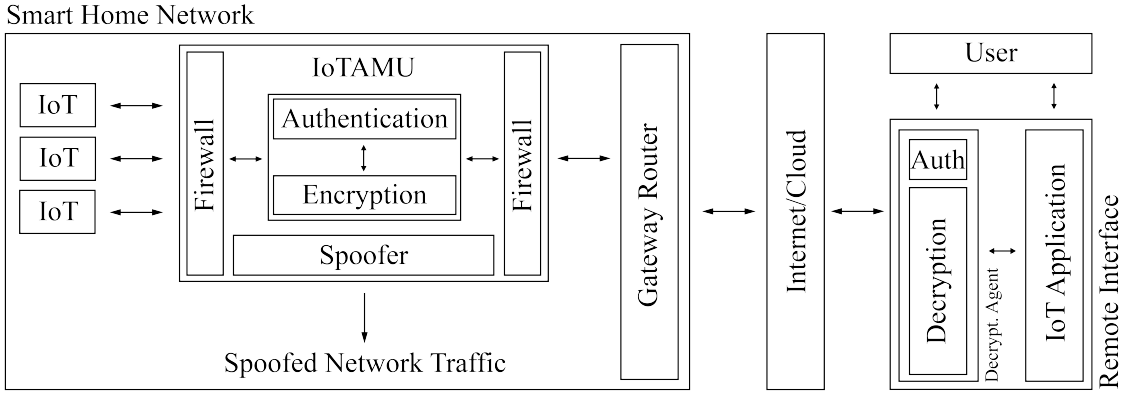


Figure 12: The proposed architecture of IoTAMU.

3.6.1 IoTAMU: Authentication

The purpose of the authentication component is to ensure that the encryption agent in IoTAMU is only communicating with the intended decryption counterpart

in the RI (e.g., smartphone, laptop). As previously discussed in Section 2.4, digital certificates provide great utility in authenticating the communicating hosts. For IoTAMU, the certificates should be signed by a common CA as configured by the network administrator. Because there are well established implementations of this mechanism publicly available, such as that of [50], the implementation of the authentication functionality is not included in this research.

3.6.2 IoTAMU: Firewall

IoTAMU also acts as a firewall between the IoT devices and the rest of the network to ensure only the intended devices are communicating. One way to accomplish this is through filtering the connections that did not originate from the internal network on the wireless interface. Figure 13 shows a snippet of code from Appendix B that provides the implementation of the firewall using iptables [47] via whitelisting only connections of interest. IoTAMU's firewall first sets all incoming connections to be dropped. There are three different chains of filters configurable by iptables. First is the `INPUT` chain which deals with connections destined for local sockets, second is the `OUTPUT` chain which deals with connections created by local sockets, and last is the `FORWARD` chain which deals with connections that are not destined for a local socket but are forwarded to their destinations by the local machine. Since the firewall should not directly receive or send any packets, all connections in the `INPUT` and `OUTPUT` chain are set to `DROP`. Initially, `FORWARD` chain's policy is set to `DROP`, which is then overwritten by another rule allowing connections generated from an internal IP address (192.168.1.0/24) to go through.

Other filtering options include using specific IP addresses and port numbers. Although the IP addresses may change over time, the port numbers associated with listening sockets on IoT devices are often fixed; this information can be used to provide

tighter control over the network flow. Unfortunately, this method does not prevent an adversary from creating a series of spoofed packets to brute force a response from one of the devices. However, it becomes increasingly difficult for the attacker to gain useful knowledge on the network as more layers of rules specific to the devices in the network are added. Therefore, it behooves the network administrator to have an in-depth understanding of their devices as best practice for better protection against malicious actors.

```
IPTABLES=/sbin/iptables
INT_NET=192.168.1.0/24
### flush existing rules and set chain policy setting to DROP
echo "[+] Flushing existing iptables rules..."
$IPTABLES -F
$IPTABLES -F -t nat
$IPTABLES -X
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

### ACCEPT rule for packets originating from the internal network
$IPTABLES -A FORWARD -i wlan0 -s $INT_NET -j ACCEPT
```

Figure 13: An example implementation of the firewall using iptables.

3.6.3 IoTAMU: Encryption

One of the primary functions of IoTAMU is encryption. This functionality provides additional security for IoT devices with weak/no encryption available. There are two different ways of accomplishing this: (1) creating a separate secure connection between IoTAMU and the remote server and (2) only encrypting a portion of the packet (e.g., application layer data), while maintaining the original connection between the IoT device and the RI. The first method can be implemented using existing protocols such as Virtual Private Network (VPN) technology or TLS. To demonstrate IoTAMU's capability as a PNP-style agent with minimal overhead, this research implements the second method as a POC using Python's cryptography pack-

age [51]. The full implementation of the encryption function of IoTAMU is available in Appendix C.

However, it should be noted that the encryption functionality may not work as intended for devices that communicate via cloud servers. If any data is processed in the cloud, encrypting the data via IoTAMU prevents the cloud server from recognizing the data.

3.6.3.1 Proof of Concept Experiment: Encryption

This section demonstrates the encryption functionality of IoTAMU as an independent proof-of-concept experiment. To intercept and parse the raw packets coming into the two network interfaces of IoTAMU, another Python script is used (see Appendix E) which is called by the main encryption scripts (`proxy_server.py` and `proxy_server2.py`) as a module. When a packet is received at an interface, it determines whether it is a UDP packet, Transmission Control Protocol (TCP) packet, or neither (e.g., Internet Control Message Protocol (ICMP) packets). If it is the first two with a payload, the script proceeds to encrypt the payload, replace the original payload, recalculate the checksums, then forward the packet to the intended recipient; if not, the packet is forwarded without any change. The network and transport layer checksums are recalculated and modified in the packets to account for the modified payloads from encryption. Because the encryption script effectively creates a second packet to be sent with the encrypted data, the original unencrypted packet needs to be dropped by the kernel after it is received at the interface. If encryption is enabled, packets received on the wireless interface can be dropped by disabling the `ACCEPT` rule in Figure 13, allowing the default `DROP` rule for the `FORWARD` chain to drop the packet. However, because a bridged network is used in this research, the packets received on the wired interface must be dropped at the link layer. Therefore, for this

proof-of-concept demonstration, this is implemented using ebtables by setting the drop rules for the packets with the appropriate MAC address in the `FORWARD` chain (Figure 14). In an actual deployment of IoTAMU, the use of ebtables is unnecessary and commenting out the `ACCEPT` rule in Figure 13 suffices as IoTAMU should be set up as a NAT-enabled network.

```
### DROP rules
# these packets need to be dropped after routing if encryption is enabled
$EBTABLES -P FORWARD ACCEPT
$EBTABLES -A FORWARD -i eth1 -d  C4:9D:ED:2D:AC:83 -j DROP
```

Figure 14: Drop rule set using ebtables.

The encryption functionality of IoTAMU is implemented using SKE. Modeled after the encryption standards in TLS, AES is selected as the encryption algorithm [24]. Because the implementation of the symmetric key exchange mechanism is out of scope of this research, a 256-bit key and a 128-bit Initialization Vector (IV) are hard coded on IoTAMU and the RI (see Appendix C); the actual keys are arbitrarily chosen as they are irrelevant to the purpose of this experiment.

Its functionality is tested using two laptops posing as an IoT device and its RI using configuration 2 of the network setup (Figure 9). `Laptop1` (client) is connected to IoTAMU (`Laptop3`), and `Laptop2` (server) is connected to `Router2`. The two Python scripts used for the client and server sides of the communication are available in Appendix F. As seen in Figure 15, `Laptop1` first sends a plaintext UDP message to `Laptop2`. When the message passes through IoTAMU, the plaintext message is encrypted using the shared symmetric key and IV, then the modified packet is forwarded to the RI. After receiving the message, the RI decrypts the ciphertext and sends an encrypted acknowledgement message back to the client. Finally, the message is decrypted at IoTAMU, and the communication is completed when the client receives the plaintext acknowledgement. As discussed in Section 3.2, although the messages exchanged over Wi-Fi between the IoT device (`Laptop1`) and IoTAMU remain in

cleartext, those exchanged between the RI (`Laptop2`) and its AP are encrypted and protected against eavesdropping.

Figure 15: The flow of messages in the POC experiment testing IoTAMU's encryption functionality.

3.6.4 IoTAMU: Spoofer

The second primary function of IoTAMU is spoofing. By spoofing crafted wireless frames, IoTAMU is able to deter device fingerprinting and pattern-of-life modeling of the user. Beyer et al. performed an investigation of these capabilities via spoofing. In their study, they were able to defeat their classifier and demonstrated the feasibility of this technique being used to secure a wireless IoT network by spoofing hard-coded packets based on the observed network signatures of specific devices.

This research extends their work through the development of the following functionalities: (1) automatic classification of active and passive states of IoT devices, (2) automatic generation of observed network signatures of the devices, and (3) artificial packet generation and spoofing algorithm that hardens the network against data mining by uniquely modifying the observed network signatures of each device.

The following sections describe a series of pilot studies that influenced the design and implementation of each function.

3.6.4.1 Passive Network Sniffing

Before the spoofing algorithm can be implemented, the network patterns (e.g., MAC addresses, packet length) generated by the IoT devices must be analyzed. Using the passive network sniffing capability provided by the aircrack-ng suite, a preliminary network reconnaissance is performed.

First, an Alfa WNIC is connected to `Laptop3` and set to monitor mode using airmon-ng (Figure 16) with the command

```
airmon-ng start wlan1
```

Next, any interfering processes are killed via

```
airmon-ng check kill
```

As seen in Figure 17, an initial scan of the network using the command

```
airodump-ng wlan1mon
```

identified the SSID of interest (IoTAMU) as well as its BSSID (A4:C4:94:3E:20:9C). Then, the network packets associated with the AP's BSSID are captured on channel six and saved for analysis (Figure 18) via

```
airodump-ng wlan1mon -c 6 -o pcap -w base --bssid a4c4943e209c
```

where `-c` specifies the channel (6) `-o` specifies the output file format (pcap), `-w` specifies the output file prefix (base), and `--bssid` specifies the BSSID of the AP (a4c4943e209c).

To isolate the traffic from each IoT device, separate captures are performed for the following devices: `Camera1`, `CameraTest`, `Camera3`, `LightBulb1`, `LightBulbTest`, `LightBulb2`, `Switch1`, `SwitchTest`, and `Switch3`. Each device type is composed of two identical models and one other model to test whether the similarities in network

```
root@kali:~/iot/iotamu/Captures3# airmon-ng start wlan1

Found 3 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

  PID Name
  598 NetworkManager
  643 wpa_supplicant
  820 dhclient

PHY     Interface       Driver          Chipset

phy0    wlan0           iwlwifi         Intel Corporation Wireless 7260 (rev 83)
phy1    wlan1           ath9k_htc       Qualcomm Atheros Communications AR9271 802.11n
root@kali:~/iot/iotamu/Captures3# airmon-ng check kill

Killing these processes:

  PID Name
  643 wpa_supplicant
  820 dhclient
```

Figure 16: Command used to set the WNIC to monitor mode.

```
root@kali:~/iot/iotamu/Captures3# airodump-ng wlan1mon

 CH  2 ][ Elapsed: 6 s ][ 2019-12-03 22:00

 BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

 30:FD:38:E5:4C:64  -86        1        0    0   6  130  WPA2 CCMP   PSK  Liahona
 94:57:A5:C6:D0:F8  -83        0        0    0   6   65  WPA2 CCMP   PSK  DIRECT-F7-HP OfficeJet 46
 C0:89:AB:14:0E:D0  -82        2        1    0  11  195  WPA2 CCMP   PSK  Jalapeno
 A4:C4:94:3E:20:9C  -40       30       13    0   6   54  WPA2 CCMP   PSK  IoTAMU
 78:D2:94:4D:AB:3E  -46       38        0    0   1  720  WPA2 CCMP   PSK  NETGEAR20
```

Figure 17: Command used to identify the AP beacons that are broadcasting.

signatures of identical models can be identified. As shown in Table 5, the network traffic of each device is captured for one minute; the duration is divided into three equal time frames to sequentially capture the device's passive state, active state, and passive state, consisting of 20 seconds each. During each device's active state, the device is activated once and the remainder of the time frame allows it to return to its passive state. Here, an 'activation of device' for switches and light bulbs is defined as turning them on (devices are set to off before each run), and that of cameras is defined as streaming the video feed for 10 seconds. Although not precise, the time frames of the three states are generalized in 20-second windows to serve as ground

```
root@kali:~/iot/iotamu/congestion# airodump-ng wlan1mon -c 6 -o pcap -w base --b
ssid a4c4943e209c


 CH  6 ][ Elapsed: 30 s ][ 2019-12-09 16:14

 BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH E

 A4:C4:94:3E:20:9C  -33 100      256      719   16   6  54   WPA2 CCMP   PSK  I

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 A4:C4:94:3E:20:9C  C4:9D:ED:2D:AC:83  -32    1 -54      1        48
 A4:C4:94:3E:20:9C  EC:1A:59:E4:FD:41  -44    2 - 1    267        65
 A4:C4:94:3E:20:9C  EC:1A:59:E5:02:0D  -51    2 -54    293        58
 A4:C4:94:3E:20:9C  B0:4E:26:C5:2A:41  -51    0 -54      0         1
 A4:C4:94:3E:20:9C  70:4F:57:F9:E1:B8  -53    0 -54      0         1
 A4:C4:94:3E:20:9C  28:AD:3E:38:6F:B6  -61   48 -54      0       373
 A4:C4:94:3E:20:9C  30:8C:FB:3A:1A:AD  -63   48 -54      2        77
 A4:C4:94:3E:20:9C  60:38:E0:EE:7C:E5  -65   12 - 1      1       107
 A4:C4:94:3E:20:9C  D0:73:D5:26:C9:27  -67    1 -54      0        12
 A4:C4:94:3E:20:9C  D0:73:D5:26:B8:4C  -70    1 -54      0        12
```

Figure 18: Command used to capture the wireless frames associated with the AP with BSSID IoTAMU.

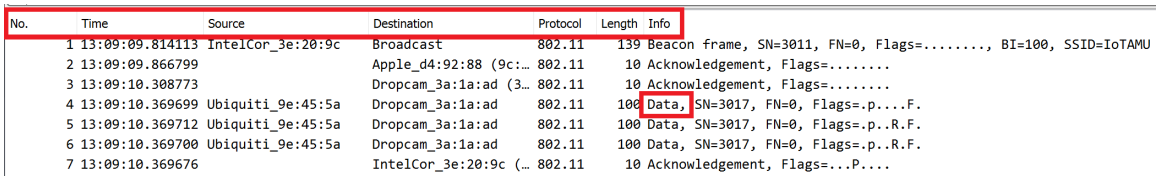
truth that can help define the device state classifications discussed in Section 3.6.4.3. As shown in Table 5, during each capture, the sniffer is first run for 20 seconds to capture the device's passive state. At 20 seconds, the device is activated to capture its active state. Finally, the device is allowed to return to its passive state and the sniffer is stopped at 60 seconds.

Table 5: Timeline of Network Capture for Each Device

Time(sec)	**Device State**	**Action**
0	Passive	Start sniffer
20	Active	Activate device
40	Passive	-
60	-	Stop sniffer

3.6.4.2 Preprocessing Data

After the network sniffing is complete, the captured pcap-formatted file needs to be parsed for analysis. As seen in Figure 19, human-readable fields in the wireless frames include the time, source and destination MAC addresses, protocol, and length. It also specifies the type of frame including beacon, acknowledgement, and data. This research focuses on the `DATA` frames where the source and the destination MAC addresses of the packets are those of `Router1`, IoTAMU, or the IoT devices.



No.	Time	Source	Destination	Protocol	Length	Info
1	13:09:09.814113	IntelCor_3e:20:9c	Broadcast	802.11	139	Beacon frame, SN=3011, FN=0, Flags=........, BI=100, SSID=IoTAMU
2	13:09:09.866799		Apple_d4:92:88 (9c:…	802.11	10	Acknowledgement, Flags=........
3	13:09:10.308773		Dropcam_3a:1a:ad (3…	802.11	10	Acknowledgement, Flags=........
4	13:09:10.369699	Ubiquiti_9e:45:5a	Dropcam_3a:1a:ad	802.11	100	Data, SN=3017, FN=0, Flags=.p....F.
5	13:09:10.369712	Ubiquiti_9e:45:5a	Dropcam_3a:1a:ad	802.11	100	Data, SN=3017, FN=0, Flags=.p..R.F.
6	13:09:10.369700	Ubiquiti_9e:45:5a	Dropcam_3a:1a:ad	802.11	100	Data, SN=3017, FN=0, Flags=.p..R.F.
7	13:09:10.369676		IntelCor_3e:20:9c (…	802.11	10	Acknowledgement, Flags=...P....

Figure 19: An example of the captured network traffic viewed in Wireshark.

To filter the capture file, a Python script (see Appendix G) extracts the following packet information: (1) packet number, (2) time at which the packet was sniffed, (3) direction (i.e., incoming/outgoing from IoT device's perspective), (4) length, and (5) Inter-arrival Time (IAT) of the incoming and outgoing packets. An IAT is defined as the time in between the previous incoming/outgoing packet to the next arriving incoming/outgoing packet, respectively. The script is run using the following command:

```
python3 preprocess.py [outputfile in csv format] [device name]
```

where `[device name]` is one of those listed in Table 4 (e.g., Camera1).

When the script is run, the packet information is parsed and filtered based on the MAC addresses of the devices and separately saved in a csv file format. An example of a preprocessed output for `Switch1` is shown in Figure 20.

```
No.,Time,Direction,Length,IAT
865,1575482965.514581000,INCOMING,108,1575482965.51458096504211425781
869,1575482965.515632000,INCOMING,108,0.00105094909667968750
873,1575482965.517666000,INCOMING,100,0.00203418731689453125
875,1575482965.518690000,INCOMING,100,0.00102400779724121094
882,1575482965.519729000,INCOMING,100,0.00103878974914550781
888,1575482965.524826000,INCOMING,88,0.00509715080261230469
```

Figure 20: A snippet of the preprocessed file for Switch1.

3.6.4.3 Device State Classification

While observing the network traffic patterns acquired after the preprocessing step, it was hypothesized that the devices exhibit different traffic patterns in their active and passive states. It was further hypothesized that the differences between the two states among different devices would be valuable in device model classification. The analyses in Sections 3.6.4.3 and 3.6.4.4 are performed using R version 3.5.0 (see Appendix H).

Two different graphs are plotted for each device to determine the differentiating factors between the two states (Figures 21 and 22). The first is a histogram of the number of packets sent over time with a one second bin width; the second is a scatter plot of the lengths of packets sent versus time in seconds. One particular example where the activation window is well differentiated can be seen in Figure 21. In this example from `Camera1`, around the 20 second mark both the number of packets sent over time and the lengths of packets increase drastically. However, it can be observed that there is around a five second window where small but significant increases for both plots are observed before the sharp jumps in magnitude at times between 22 to 27 seconds. This window coincides with the loading time in the application after the video feed is activated (i.e., play button is pressed in the application). Because the device is actively exchanging data during this window, it is classified as `ACTIVE` for the purpose of this research.

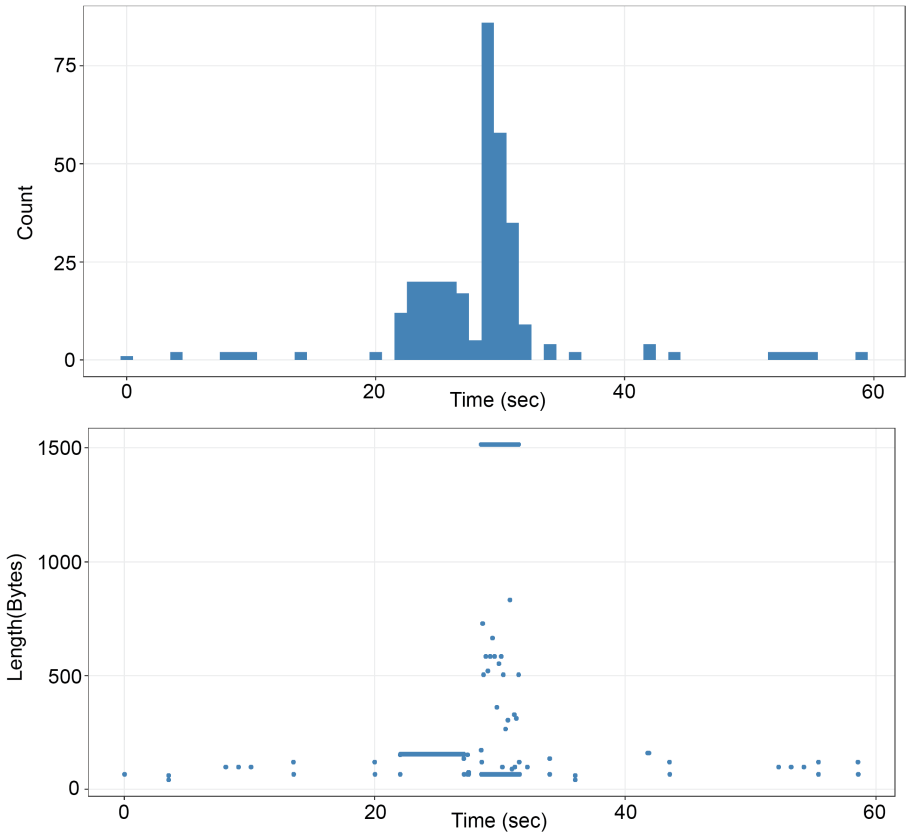


Figure 21: Histogram of the number of packets sent over time (top) and scatter plot of the length of packets sent (bottom) by Camera1.

In the second set of examples from `LightBulb1` seen in Figure 22, the activation window appeared to be indifferentiable from the rest. Because of the similarities in the traffic patterns throughout, it is assumed that the IoT device's state is stored in a cloud server and its state is periodically checked by the device, rather than resulting from direct communication with the RI. Further investigation of the protocol is out of scope of this research. Although there is a lack of a definite activation window, it can be construed as there being multiple periodic activation windows throughout.

When the devices are activated, all of their respective traffic patterns share the following characteristics: presence of packets with increased payload length and increase in the number packets sent over time (or shorter IAT). Although both of these characteristics are important markers of the device's activation states, because each

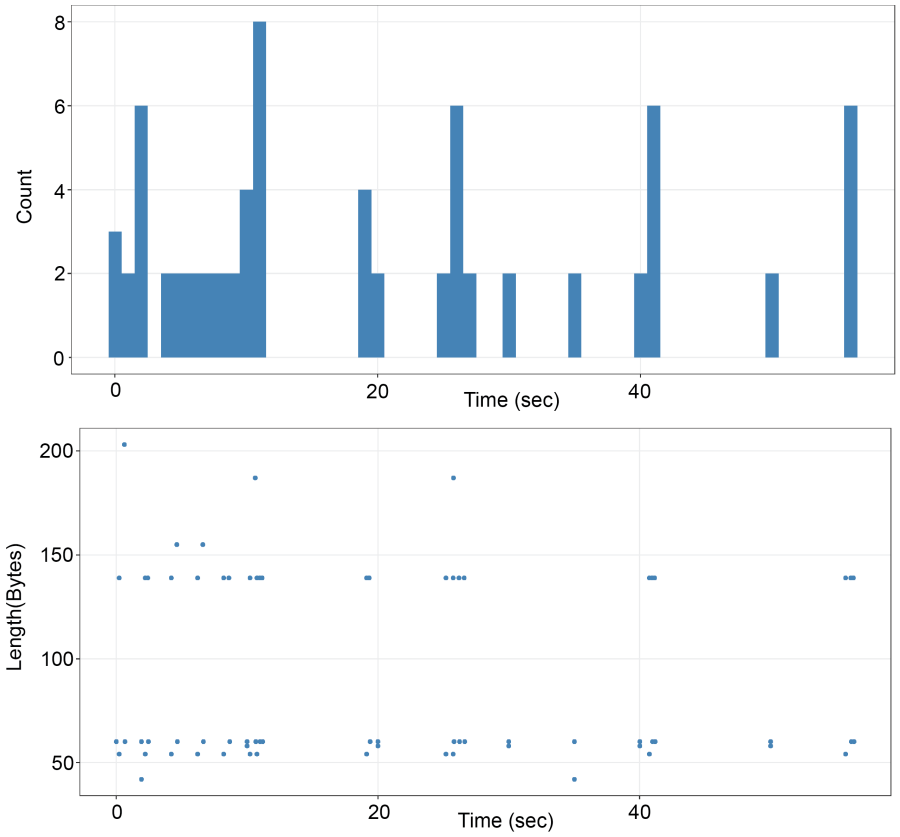


Figure 22: Histogram of the number of packets sent over time (top) and scatter plot of the length of packets sent (bottom) by LightBulb1.

histogram bin provides a discrete time interval, the number of packets sent is used in this research to identify the device state information. If the number of packets sent over time is used, two questions remain to be answered – how wide should the bins be and how should the threshold that defines each state be set. Upon inspection of the spread of the histograms, 0.5 second bin width is empirically determined as an acceptable range that retains precision and the characteristics of the two different states.

The second question is answered with the assumptions that the default state of the devices is passive and that there would be significantly less packets exchanged during this state. This assumption allows the use of outliers to define active states. Using

each bin of the histogram as a data point, the outliers are defined by the inequality

$$\begin{aligned} Outlier &> 1.5 \times IQR + Q_3 \\ IQR &= Q_3 - Q_1 \end{aligned} \tag{4}$$

where IQR represents the interquartile range and Q_1 and Q_3 represents the first and third quartiles of the data. Q_1 and Q_3 are found by taking the median of the first and second half of the data, respectively. The resulting 0.5 second windows of the outliers are labeled as the `ACTIVE` states of the device. Figure 23 shows the histograms of `Camera1` and `LightBulb1` with bin widths of 0.5, where their respective active states are highlighted in red. In the figures, the distinct active state window for `Camera1` is accurately labeled; all valid windows for `LightBulb1` where packets were exchanged is labeled as `ACTIVE`. The performance for `LightBulb1` is deemed acceptable as it was assumed that the device periodically communicated with the cloud server to update its state. Furthermore, because the ultimate goal is identical device model classification, even if portions of time frames are misidentified, as long as it remains consistent across the like-model devices, it has little impact in the overall classification algorithm.

3.6.4.4 Identical Device Model Classifier (IDMC)

As discussed in Section 2.6, previous efforts in classifying device type has produced limited success. However, these results are to be expected from a highly diverse pool of samples as that of IoT. Therefore, instead of device types, this research focuses on investigating the feasibility of classifying identical model devices (i.e., devices with same manufacturer and model number).

There are limited sources of variation that can be observed from encrypted wireless frames, which include the source and destination MAC addresses, direction (from

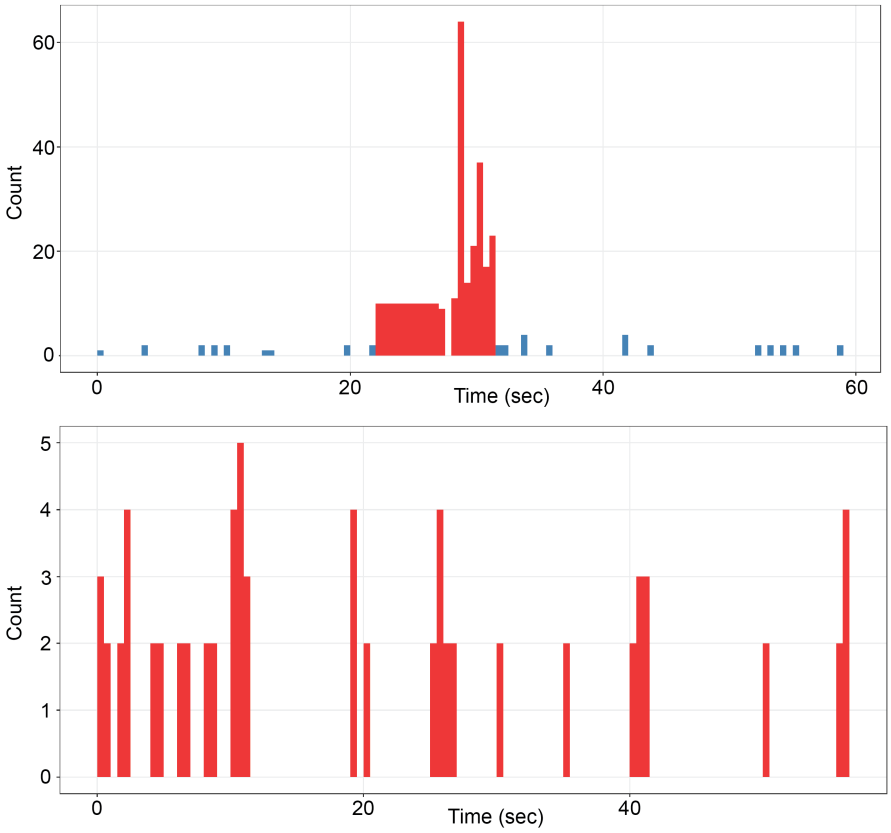


Figure 23: Histograms of the number of packets sent over time for Camera1 (top) and LightBulb1 (bottom).

perspective of a particular device), length of the packets, and IAT. One of the easiest sources of information in these packets is the first three bytes of a MAC address, also known as the Organizational Unique Identifier (OUI). Unique sets of OUI are specific to each vendor, prescribing a range of MAC addresses that can be assigned to a device. Therefore, the manufacturer of a device can be easily determined by looking up the OUI of the device via online tools such as that of Wireshark [52]. Although it is an important feature for identifying potentially identical devices, it can be easily modified, and should not be the primary source of information. Therefore, this research focuses on the variations in packet length and IAT that have been proven as important features in classification studies on encrypted Wi-Fi traffic [4][36][53]. It should be noted that the OUI can vary even if the devices are from the same vendor,

and in some cases, identical models. For example, `Switch1` and `SwitchTest` are identical model devices with different OUIs. But, using the OUI lookup tool provided by Wireshark does result in identical entries for both (BelkinIn Belkin International Inc.).

Before variations in packet lengths are considered, the packets are divided into four different states: (1) active, outgoing, (2) active, incoming, (3) passive, outgoing, and (4) passive, incoming. The analyses in Section 3.6.4.3 have already shown that the devices exhibit different traffic patterns in their active and passive states. It can also be assumed that the outgoing and incoming traffic patterns are different because they are generated by different devices (i.e., RI or IoT device). Then, the packets are categorized based on their lengths in 200-byte increments up to 1600, to account for a Maximum Transmission Unit (MTU) of 1500 bytes.

In addition, the variability introduced by the difference in activation windows from human error and environment factors must be considered. For example, wireless signals may be deformed or weakened, causing the device to retransmit the data, or the video feeds may not be started and stopped at exact experimental parameters. These factors cause an unintentional increase in traffic load that may change the overall outcome of the classification. Therefore, rather than using the actual packet counts as a statistic, the density of each packet-size window is calculated for each state. The packet density for a particular state and packet length combination is described by

$$PacketDensity_{state,length} = \frac{PacketCount_{state,length}}{\sum_{length} PacketCount_{state,length}} \tag{5}$$

Finally, the average IAT for each state is calculated by taking the mean of IAT times in each state. For example, mean IAT for the active, outgoing state of a device is calculated by first filtering its preprocessed file for the rows with `OUTGOING`

entries and times that fall within the calculated `ACTIVE` histogram bins (as defined in Section 3.6.4.3), then taking the mean of the `IAT` column. The calculated statistics for each device is summarized in Table 6. Each feature (i.e., each row in the table) is labeled using the following scheme:

Packet Density: [Packet Length][State] ex. [200][ActiveOut]

Averange IAT: [State]IAT ex. [ActiveOut]IAT

Using the computed statistics as the feature set, PCA is performed to identify clusters of like devices. It was hypothesized that the network signatures from same device models will be clustered significantly closer than others. The `prcomp` function in the `stats` package for R is used to perform PCA. Table 7 shows the PCS values derived from PCA, and Figure 24 shows the scree plot summarizing the percent variations for each principal component. The first three components, accounting for a total of 78% of variation, are used for visualization of the data in a 3-dimensional plot shown in Figure 25. In the plot, the same colored points represent the same device types (e.g., camera), and the same colored triangle shapes represent identical device models (e.g., Belkin camera model F7D7602v2). As expected, the 3-D plot identifies three distinct clusters composed of the same device models, with the identical switch models (gray triangles) almost overlapping one another.

Next, a Similarity Score (SS) metric is devised to quantify the closeness of each pair of devices. The SS for $device_i$ and $device_j$ is defined as

$$SS(device_i, device_j) = 1 - \frac{distance(PCS_{device_i}, PCS_{device_j})}{\max_{i,j \in n}(distance(PCS_{device_i}, PCS_{device_j}))} \quad (6)$$

To calculate the SS, the euclidean distance between the PCSs of a pair of devices (PCS_{device_i}) is first measured using the Pythagorean formula. As discussed in Section 2.5, the scores of the first principal components whose sum of percent variations exceed 80% of the total are used to calculate the distances. In this particular case,

Table 6: Summary of Packet Densities and Average IATs for Each IoT

	C1	CT	C3	LB1	LBT	LB2	S1	ST	S3
200ActiveOut	0.5066	0.1394	0.06122	0.9737	1	0.7541	0.2615	0.3429	0.7895
400ActiveOut	0.02203	0.002286	0.02041	0.02632	0	0.04918	0.1231	0.1	0.01754
600ActiveOut	0.03965	0.003429	0.04082	0	0	0	0.06154	0.05714	0.03509
800ActiveOut	0.008811	0.01714	0.02041	0	0	0.03279	0.03077	0.01429	0
1000ActiveOut	0.004405	0.04	0	0	0	0	0.01538	0.01429	0.1579
1200ActiveOut	0	0.01029	0.04082	0	0	0	0	0	0
1400ActiveOut	0	0	0.02041	0	0	0.1639	0	0	0
1600ActiveOut	0.4185	0.7874	0.7959	0	0	0	0.5077	0.4714	0
200ActiveIn	1	1	1	1	1	0.5455	0.9211	0.825	0.5192
400ActiveIn	0	0	0	0	0	0.2	0.02632	0.025	0.25
600ActiveIn	0	0	0	0	0	0.03636	0.05263	0.05	0
800ActiveIn	0	0	0	0	0	0	0	0	0
1000ActiveIn	0	0	0	0	0	0.07273	0	0.025	0.07692
1200ActiveIn	0	0	0	0	0	0	0	0	0
1400ActiveIn	0	0	0	0	0	0.03636	0	0	0.03846
1600ActiveIn	0	0	0	0	0	0.1091	0	0.075	0.1154
ActiveOutIAT	0.04186	0.0118	0.01008	1.518	1.638	0.9672	0.3505	0.209	1.03
ActiveInIAT	0.06484	0.01699	0.01874	1.813	2.088	1.075	0.1592	0.1283	1.131
200PassiveOut	1	1	0.03132	0	0	0	0	0	0
400PassiveOut	0	0	0.01139	0	0	0	0	0	0
600PassiveOut	0	0	0.2342	0	0	0	0	0	0
800PassiveOut	0	0	0.1189	0	0	0	0	0	0
1000PassiveOut	0	0	0.07687	0	0	0	0	0	0
1200PassiveOut	0	0	0.06762	0	0	0	0	0	0
1400PassiveOut	0	0	0.03132	0	0	0	0	0	0
1600PassiveOut	0	0	0.4285	0	0	0	0	0	0
200PassiveIn	1	1	1	0	0	0	0	0	0
400PassiveIn	0	0	0	0	0	0	0	0	0
600PassiveIn	0	0	0	0	0	0	0	0	0
800PassiveIn	0	0	0	0	0	0	0	0	0
1000PassiveIn	0	0	0	0	0	0	0	0	0
1200PassiveIn	0	0	0	0	0	0	0	0	0
1400PassiveIn	0	0	0	0	0	0	0	0	0
1600PassiveIn	0	0	0	0	0	0	0	0	0
PassiveOutIAT	1.815	1.547	0.03821	0	0	0	0	0	0
PassiveInIAT	3.796	3.43	0.05244	0	0	0	0	0	0

the first four components that sum to 91.2% are used. Calculated distances between the selected PCSs from each pair is then divided by the maximum distance among all unique pairs and subtracted from 1, resulting in a value ranging from 0 to 1. This value is the similarity score, where a value closer to one indicates a higher resemblance

Table 7: PCS of Each Device (Pilot Study)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
C1	-1.14	3.581	-0.6427	-1.447	0.07939	1.173	-0.2782	-0.01733	-1.11e-16
CT	-1.701	3.347	-0.8112	-1.864	0.4517	-1.112	0.2521	0.008741	-9.437e-16
C3	-7.499	-3.248	0.9035	-0.2354	-0.01025	0.07332	-0.01436	-0.0003797	6.384e-16
LB1	1.499	0.7236	2.848	1.19	0.1447	0.02927	0.08631	0.3893	1.277e-15
LBT	1.71	0.6869	3.454	1.088	0.2523	-0.09896	-0.01095	-0.347	4.441e-16
LB2	3.12	-2.806	-1.354	-0.9193	2.73	0.1094	0.0006459	0.001694	-1.388e-15
S1	0.04711	0.2703	-2.305	2.866	-0.4113	-0.3743	-0.7763	0.002643	-1.11e-16
ST	0.5916	-0.1862	-2.114	1.873	-0.9178	0.3162	0.9242	-0.04451	2.498e-16
S3	3.373	-2.369	0.02134	-2.551	-2.319	-0.1156	-0.1835	0.006791	-2.776e-16
PC: principal component									

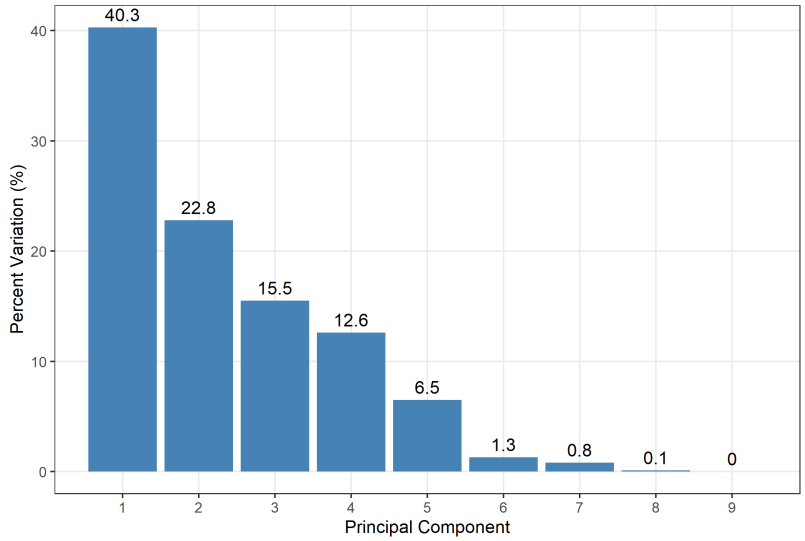


Figure 24: The percentage of variation accounted by each principal component.

in the pair's network signature. Table 8 provides a summary of the SS for each pairs of devices. As hypothesized, the scores for the same model devices with values 0.9324, 0.9418, and 0.8895 (highlighted in bold) are significantly higher than those of others.

3.6.4.5 Spoofer Implementation

One of the first steps in implementing the spoofer is generating the packets to be spoofed. As discussed in Section 2.6, this research creates artificial packets that mimic the signatures of the underlying network to defeat data mining through sta-

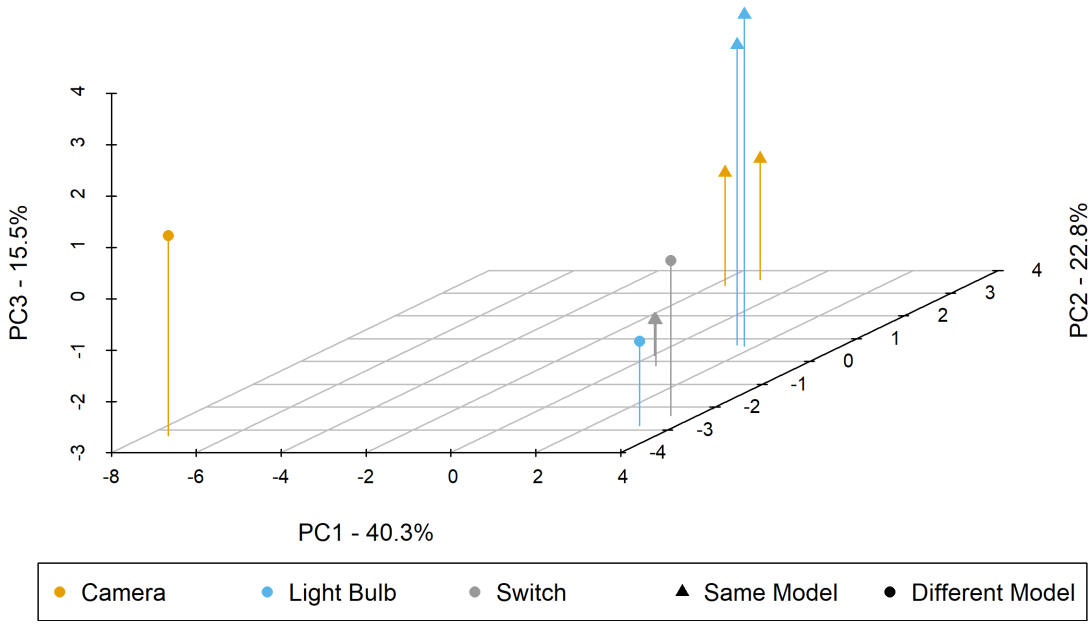


Figure 25: Visualization of the first three principal components of the nine IoT devices.

Table 8: Summary of Similarity Scores for Each IoT Pair

	C1	CT	C3	LB1	LBT	LB2	S1	ST
CT	**0.9324**							
C3	0.1475	0.1869						
LB1	0.4767	0.4357	0.09465					
LBT	0.4367	0.3962	0.06858	**0.9418**				
LB2	0.3091	0.2945	0.02671	0.4548	0.4253			
S1	0.4807	0.4552	0.1555	0.4968	0.4398	0.4776		
ST	0.5073	0.4833	0.1595	0.5377	0.4815	0.5842	**0.8895**	
S3	0.3225	0.3099	0	0.4706	0.4553	0.8040	0.3505	0.4589

tistical analyses. Generating the packets from the existing signatures first requires a training stage where the current network must be monitored. This is completed in Section 3.6.4.4 where Table 6 containing the packet densities and average IATs of the devices is generated. Next, using the R script provided in Appendix I, the entries for IAT are transformed into log scale to better capture the variance from fea-

tures that are orders of magnitudes smaller. Then, a covariance matrix is calculated from the data, which is in turn used to generate a pool of samples that resemble the signatures of the original data using the `mvrnorm` function from the MASS package in R. Because the new data is generated from random sampling from a multivariate normal distribution, negative numbers may be produced. For the IAT entries, undoing the log transformation rescales the negative numbers into positive numbers. For packet density features, if a negative number is generated for a certain feature (e.g., 200ActiveOut), a value twice the magnitude of the minimum within the feature is added to the original data. Since the relative distances between the values within each feature does not change from this operation, the rescaling does not affect their variance. Lastly, because the packet length statistics capture the density within each state (as defined in Section 3.6.4.4), values for each state are rescaled such that their sum is equal to one. This is accomplished by taking the original value and dividing by the sum of the values in the state it belongs (e.g., ActiveOut). Nine samples of the Artificially Generated Packet Signatures (AGPS) are shown in Table 9.

The spoofer functionality of IoTAMU, implemented in Python version 3.5.2 (see Appendix D), uses the AGPS to inject spoofed packets to conceal the existing network signatures. Algorithm 1 displays pseudocode describing the implementation of the spoofer. The spoofer takes as input the table of AGPS in CSV file format. It first creates a thread for each sample that is being spoofed. Each sample is defined by a unique set of network signatures (i.e., packet length density information, average IAT for different states). Then, they are assigned a MAC address of one of the existing devices in the network and either a passive or an active state with probabilities 90% and 10%, respectively. This is done to designate the devices' predominant state as passive. For each of the incoming and outgoing directions, the following information is configured. First, the duration of the segment to be spoofed is chosen at ran-

dom between 5 and 15 seconds (defined as `Duration`). This value is then divided by the IAT for the particular state and direction to calculate the number of packets in the segment (defined as `NumberOfPackets`). Then, the number of times the segment is repeated is chosen between 5 and 15 (defined as `NumberOfRepeat`). Finally, `NumberOfPackets` different packet lengths for a segment are chosen and stored in a list (`PacketList`), by sampling from a list of packet lengths windows (i.e., 0 to 200, 200 to 400, etc.), using the density statistics from the AGPS as their respective probabilities. `PacketList`, IAT, and `NumberOfRepeat` are passed on to two subthreads (one for each direction) that perform the actual spoofing via Scapy. After spoofing each packet in `PacketList`, the subthread sleeps for the duration of the IAT, and the list is repeated `NumberOfRepeat` times. Once the subthreads are joined, the thread for the particular AGPS sleeps for a random duration between 5 to 10 seconds, and the process is repeated with a selection of a passive or an active state. All duration parameters and passive/active state probabilities are empirically determined from pilot studies; optimizations of these parameters are out of scope of this research. Ultimately, the goal of the spoofer is to uniquely modify the observed network signatures for each device to decrease the likelihood of unwanted leakage of information.

3.7 Summary

This chapter discusses the architecture of IoTAMU and the network setup of this research. It also describes how each component of IoTAMU is implemented in detail. Proposed components of IoTAMU provide applications of how PNP-style remote computing agents can be utilized to provide additional security. In addition, its spoofing functionality offers a prototype mechanism that can be used to conceal the device specific patterns of unencrypted fields in encrypted wireless communications.

Algorithm 1 Pseudocode Implementation of Spoofer

```
1: function MAIN(NetworkSignatures)
2:     for Signature in NetworkSignatures do
3:         NewThread(GeneratePackets(Signature))
4:     end for
5:     Join Threads
6:     return
7: end function

8: function GeneratePackets(Signature)
9:     loop
10:        State ← [PASSIVE|ACTIVE] with probability (90%, 10%)
11:        for Direction in [INCOMING, OUTGOING] do
12:            Duration ← random(5, 15)
13:            NumberOfPackets ← Duration/IAT
14:            NumberOfRepeat ← random(5, 15)
15:            PacketList ←
16:            repeat(Sample(200, 400, 600, 800, 1000, 1200, 1400, 1600) with probability (density statistics for State and Direction from Signatures))
17:            until length(PacketList) == NumberOfPackets
18:            NewThread(Spoof(PacketList, IAT, NumberOfRepeat))
19:        end for
20:        Join Threads
21:        Sleep(random(5, 10))
22:    end loop
23:    return
24: end function

25: function Spoof(PacketList, IAT, NumberOfRepeat)
26:    repeat
27:        for Packet in PacketList do
28:            ConstructPacket
29:            SpoofPacket
30:            Sleep(IAT)
31:        end for
32:    until NumberOfRepeat
33:    return
34: end function
```

Table 9: Summary of Artificially Generated Packet Signatures

	Samples								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
200ActiveOut	0.3941	0.5047	0.3954	0.3186	0.3973	0.4197	0.4664	0.4565	0.4369
400ActiveOut	0.05263	0.09548	0.0545	0.07073	0.06237	0.05618	0.08411	0.06421	0.06613
600ActiveOut	0.1148	0.06482	0.1134	0.1215	0.1084	0.1069	0.08011	0.09385	0.09722
800ActiveOut	0.01299	0.003034	0.01254	0.008207	0.01062	0.01224	0.005634	0.01045	0.009892
1000ActiveOut	0.02561	0.02116	0.02569	0.03338	0.02633	0.02394	0.02304	0.02183	0.02357
1200ActiveOut	0.09734	0.1559	0.1003	0.1366	0.1133	0.1002	0.1414	0.1093	0.1148
1400ActiveOut	0.002695	0.0009194	0.002642	0.002794	0.002444	0.002433	0.001453	0.001989	0.00208
1600ActiveOut	0.2999	0.154	0.2955	0.3081	0.2793	0.2784	0.1978	0.2419	0.2494
200ActiveIn	0.67	0.8024	0.6772	0.776	0.7091	0.6742	0.7709	0.6925	0.7084
400ActiveIn	0.2505	0.1218	0.2442	0.1726	0.217	0.243	0.1542	0.2219	0.2115
600ActiveIn	0.01424	0.01351	0.01406	0.008614	0.01312	0.0149	0.01331	0.01546	0.01436
800ActiveIn	0.01697	0.01623	0.01678	0.01134	0.01585	0.01763	0.01604	0.01819	0.01708
1000ActiveIn	0.005523	0.005523	0.005523	0.005523	0.005523	0.005523	0.005523	0.005523	0.005523
1200ActiveIn	0.01642	0.01569	0.01624	0.01079	0.0153	0.01708	0.01549	0.01764	0.01654
1400ActiveIn	0.0157	0.01496	0.01551	0.01007	0.01458	0.01635	0.01477	0.01691	0.01581
1600ActiveIn	0.01068	0.009947	0.0105	0.005055	0.009563	0.01134	0.009753	0.0119	0.0108
ActiveOutIAT	1.687	1.458	1.488	0.02692	0.7857	2.884	1.125	4.795	2.101
ActiveInIAT	1.517	1.349	1.431	0.2313	1.064	1.923	1.217	2.392	1.647
200PassiveOut	0.5174	0.6587	0.5266	0.6809	0.5681	0.5151	0.6288	0.5279	0.5551
400PassiveOut	0.4826	0.3413	0.4734	0.3191	0.4319	0.4849	0.3712	0.4721	0.4449
600PassiveOut	0	0	0	0	0	0	0	0	0
800PassiveOut	0	0	0	0	0	0	0	0	0
1000PassiveOut	0	0	0	0	0	0	0	0	0
1200PassiveOut	0	0	0	0	0	0	0	0	0
1400PassiveOut	0	0	0	0	0	0	0	0	0
1600PassiveOut	0	0	0	0	0	0	0	0	0
200PassiveIn	0.9672	0.9824	0.9677	0.9664	0.9694	0.9695	0.9778	0.9732	0.9725
400PassiveIn	0.03277	0.01765	0.03232	0.03362	0.03063	0.03054	0.02219	0.02676	0.02753
600PassiveIn	0	0	0	0	0	0	0	0	0
800PassiveIn	0	0	0	0	0	0	0	0	0
1000PassiveIn	0	0	0	0	0	0	0	0	0
1200PassiveIn	0	0	0	0	0	0	0	0	0
1400PassiveIn	0	0	0	0	0	0	0	0	0
1600PassiveIn	0	0	0	0	0	0	0	0	0
PassiveOutIAT	2.794	1.186	2.527	0.2258	1.563	3.477	1.272	3.93	2.454
PassiveInIAT	1.202	0.7697	1.153	0.4678	0.9485	1.282	0.8197	1.302	1.097

IV. Methodology

4.1 Problem/Objective

The experiments in this research seek to evaluate the effectiveness of injecting packets generated using AGPS in modifying the observed signatures of devices. Furthermore, they attempt to measure the negative impact of the additional packets generated in contributing to network congestion. Ultimately, the experiments evaluate IoTAMU's ability to obscure the existing patterns of network signatures to deter information extraction from the encrypted fields in the current Wi-Fi standards. This is accomplished by measuring IDMC's performance in a controlled IoT network, and comparing with its performance in a spoofed network. In summary, this research has the following main objectives:

1. Measure the accuracy of a classifier in identifying devices of identical models from their network signatures.

2. Determine the effectiveness of a spoofing algorithm in changing the observed network signatures of pre-existing devices.

3. Examine the impact on network congestion from the additional traffic created by the spoofer.

The results of this research help identify an area of information leakage from encrypted Wi-Fi communications and provide a potential solution to mitigate this vulnerability.

4.2 System Under Test

The System Under Test (SUT) and Components Under Test (CUT) for this research is depicted in Figure 26. This research examines the effect of IoTAMU on

different observable network statistics. The CUT for this experiment are the IDMC and the spoofer. Section 4.3 summarizes the assumptions made throughout the experiment. Next, the Wi-Fi traffic collected, or the uncontrolled variable in this experiment, is described in Section 4.4. Section 4.5 examines the different experimental parameters including the physical layout of the experiment and the number of devices. Then, Section 4.6 describes the response variables, or metrics, consisting of the identical device model classification, network latency, the number of packets dropped, total packet count, and the network throughput. The IoTAMU status and device events, discussed in Section 4.7, are the main controlled variables for this experiment. Finally, Section 4.8 concludes this chapter with a discussion of the experimental design.

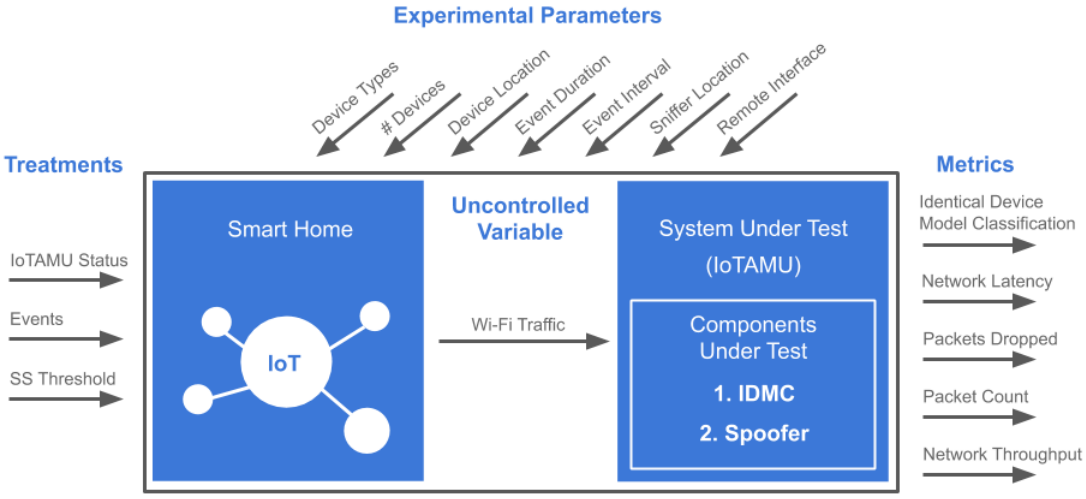


Figure 26: Diagram depicting the System Under Test and Components Under Test of the experiment.

4.3 Assumptions

The following assumptions are made throughout the design and the execution of the experiment.

1. The duration and the sequence of device activation performed in this experiment

are reflective of a real smart home environment.

2. The relative locations of the devices in the smart home models those in a real smart home environment.

3. The external and internal networks abstracted in the network setup of the experiment models the actual flow of information between an IoT device and the RI.

4. Network sniffing in this experiment is conducted on the same device as IoTAMU using an external WNIC. It is assumed that the network traffic observed under a more realistic scenario (i.e., using a separate device from farther distance) is similar to that of the experiment. This assumption is substantiated through previous work by Beyer et al. [3].

5. The eavesdropper has enough prior knowledge about the network to sniff the Wi-Fi traffic, including the SSID, and the channel of the AP.

6. The network interference between the devices and from environmental factors are negligible.

4.4 Uncontrolled Variables

Although efforts are made in the network setup to isolate the Wi-Fi communications as much as possible, interference from environmental factors that cause degradation of signals remain. However, it is assumed that this interference is negligible and does not have a significant impact on the experiment results.

4.5 Experimental Parameters

The following variables are held constant throughout the experiment to limit the source of variability caused by external factors. Detailed descriptions of the variables are provided in Section 4.8.

- **Device Types:** The device types, including their models and manufacturers, remain constant throughout the experiment (see Table 3).
- **Number of Devices:** The number of devices remains constant throughout the experiment.
- **Device Location:** The location of the devices are held constant throughout the experiment.
- **Event Duration:** The duration of each activation event remains constant throughout the experiment.
- **Event Interval:** The interval between each activation event remains constant throughout the experiment.
- **Sniffer Location:** Sniffing is performed on the same device where IoTAMU is implemented; its location remains constant throughout the experiment.
- **Remote Interface:** The device for the remote interface as well as its location remain constant throughout the experiment.

4.6 Metrics

The overall objective of this experiment is to measure the effects of IoTAMU's spoofer on the existing network. Its ability to obscure the existing network signatures of the devices is indirectly measured through the lowered accuracy of IDMC. Its effect

on network congestion is determined by measuring the added network latency and the number of packets dropped from the added traffic. Detailed descriptions of the response metrics as they pertain to each objective is provided below.

1. **Objective 1:** Measure the accuracy of a classifier in identifying devices of identical models from their network signatures.

   Identical Device Model Classification: The following metrics are clustered under "Identical Device Model Classification" (as seen in Figure 26); they are used to calculate the classification status:

   - **True Positives (TP):** The TP metric quantifies the number of correctly classified identical model pairs exceeding the SS threshold.
   - **True Negatives (TN):** The TN metric quantifies the number of correctly classified nonidentical model pairs below the SS threshold.
   - **False Positives (FP):** The FP metric quantifies the number of incorrectly classified nonidentical model pairs exceeding the SS threshold.
   - **False Negatives (FN):** The FN metric quantifies the number of incorrectly classified identical model pairs below the SS threshold.

2. **Objective 2:** Determine the effectiveness of a spoofing algorithm in changing the observed network signatures of pre-existing devices.

   - Same response variables as those of Objective 1 are used to measure the change in their values before and after the addition of the spoofer in the experiment.

3. **Objective 3:** Examine the impact on network congestion from the additional traffic created by the spoofer.

- **Network Latency (NL):** The NL metric, measured in seconds, represents the time it takes to send a message from an IoT device and receive a response back from its RI.
- **Packets Dropped (PD):** The PD metric quantifies the number of responses from the RI not received by the IoT device, for each packet generated by the IoT device. Although either the initial packet generated by the IoT device or the response from the RI can be dropped, either or both events count as one drop event for this metric.
- **Packet Count (PC):** The PC metric is the total number of data packets observed in the network. It is used to estimate the number of packets generated by IoTAMU for each sample spoofed.
- **Network Throughput (NT):** The NT metric is the rate at which data is sent over the network. It is used to estimate the amount of data that is generated by IoTAMU per second for each sample spoofed. NT is calculated by dividing the total amount of observed data from all packets (bits) by the duration of the capture (seconds).

4.7 Treatments

The main components that are varied in the experiment are the `IoTAMU Status`, the order in which the devices are activated (labeled `Events`), and the `SS Threshold`. Table 10 provides a summary of the three treatments and their levels. `IoTAMU Status` is defined as the number of samples spoofed by IoTAMU and `Events` are defined as the predefined orders in which the devices are activated for each run. The specific orders of device activation are discussed in Section 4.8.2.1. Data from the three runs are pooled into one for calculation of overall performance metrics. SS is a measure of confidence from 0 to one, with a value closer to one conveying a stronger confidence,

that determines whether the packet signatures observed from two devices are those of identical models. Its derivation from the euclidian distance of the PCS as discussed in Section 3.6.4.4 is reproduced below.

$$SS(device_i, device_j) = 1 - \frac{distance(PCS_{device_i}, PCS_{device_j})}{\max_{i,j \in n}(distance(PCS_{device_i}, PCS_{device_j}))}$$

Two network signatures are considered to be from identical models if their SS value exceeds a predefined threshold.

Three separate experiments are conducted to meet each of the objectives. The first experiment measures classifier accuracy; therefore, 0 samples are spoofed by IoTAMU. The second experiment investigates spoofing effectiveness and uses 11 or 120 spoofed samples. The third experiment examines network congestion resulting from spoofed packets; 0 to 120 samples are spoofed in increments of 10. There are no events varied for the third experiment; all the devices are set to run in their passive states. Instead, each level of IoTAMU status is repeated three times and the resulting data are pooled into one for calculation of overall performance metrics. Three different SS thresholds are used in experiments 1 and 2 to classify identical model pairs: 0.9, 0.8, and 0.7. These values are chosen around the observed SS values of identical model pairs (see Section 3.6.4.4).

Table 10: Summary of Experimental Treatments and Their Levels

Treatment	**Experiment**	**Levels**
IoTAMU Status	1	0
	2	11/120
	3	0/10/20/30/40/50/60/70/80/90/100/110/120
Events	1,2	Run1/Run2/Run3
	3	None
SS Threshold	1,2	0.9/0.8/0.7
	3	None
Experiment 1: classifier accuracy, Experiment 2: spoofing effectiveness, Experiment 3: network congestion		

4.8 Experimental Design

The pipelines of the three experiments are depicted in Figure 27, which are discussed in the rest of this section.

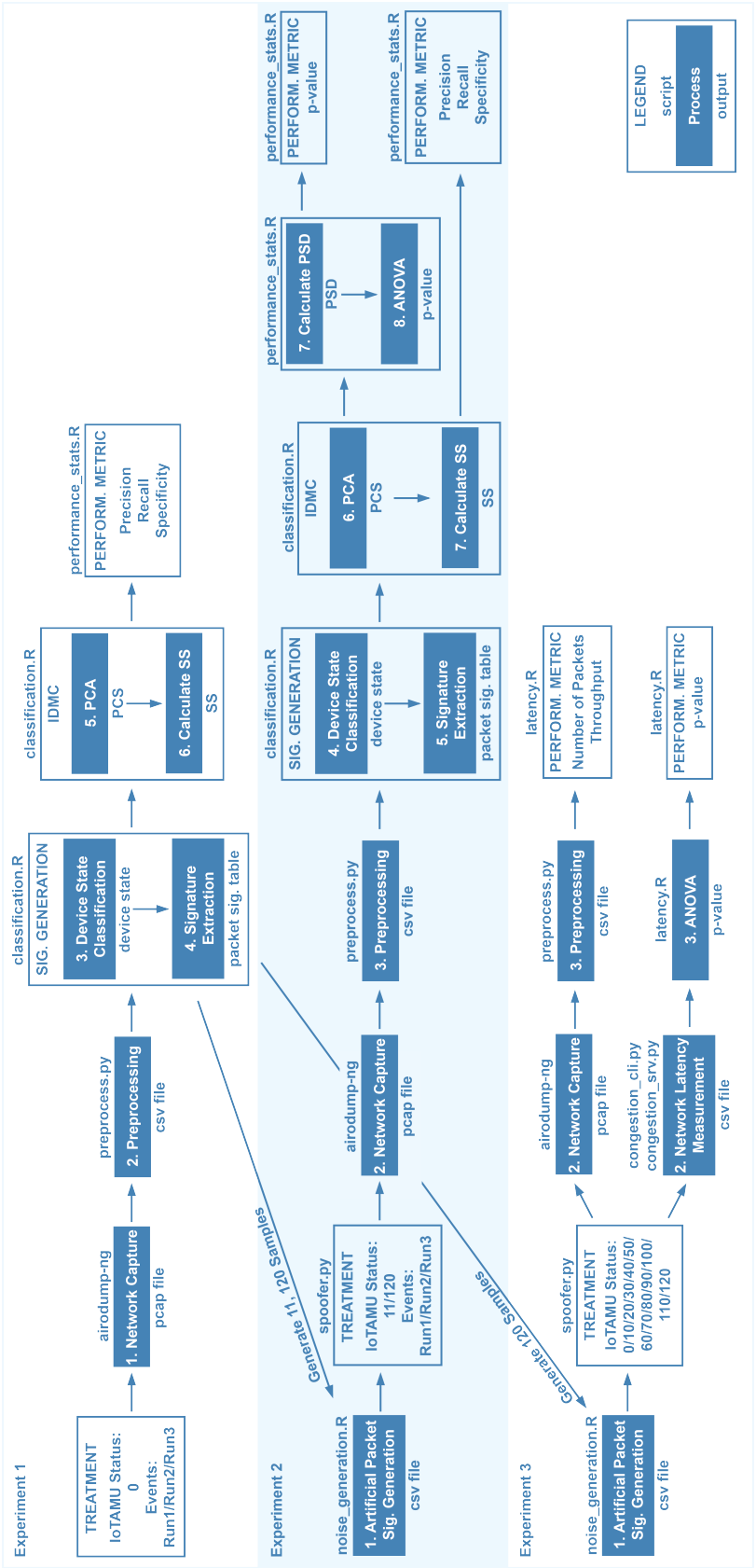


Figure 27: The pipelines of the three experiments.

4.8.1 Device Setup

Figure 28 depicts the relative locations of the devices in the experiment. These locations remain unchanged throughout the three experiments.

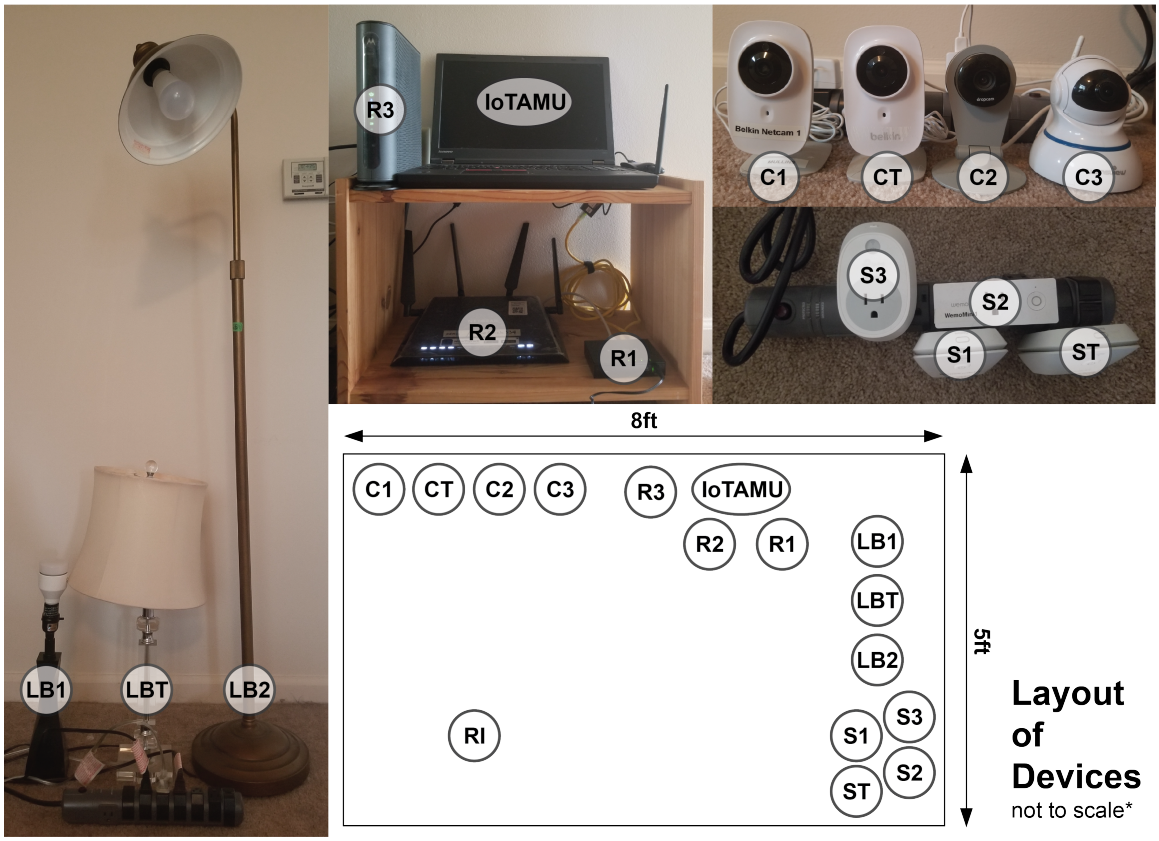


Figure 28: The relative locations of the devices in the experiment.

4.8.2 Experiment 1: Measuring the Performance of IDMC

The first experiment is conducted to measure the baseline performance of the IDMC. It is measured through its ability to correctly classify both identical and nonidentical device models; its overall performance metrics are calculated by pooling the data gathered from the three runs in this experiment. Network configuration 1 described in Figure 8 is used in experiment 1.

4.8.2.1 Data Collection Process

First, to set up a baseline network state, the 11 IoT devices are turned on and activated once from the RI (`Phone1`) before being left to run in their passive states for five minutes. This is performed before each run to create a consistent behavior for the devices. After reaching their steady states, the IoT devices are assigned predefined orders to be activated for three different runs as shown in Table 11. Similar to the pilot studies discussed in Section 3.6.4, the devices are manually activated in one minute intervals. During each run, the sniffer is set to run for 55 seconds before the first RI application is opened. At the 55^{th} second of each minute, the application is opened, allowing five seconds of buffer time for the application to load on the RI. At the 0^{th} second of each minute, the IoT devices are activated in the predetermined order. Here, 'activation of device' for switches and light bulbs is defined as turning them on (devices are set to off before each run), and that of cameras is defined as streaming the video feed for 15 seconds. After the activation of the last device in each run, the sniffer is turned off at the 12^{th} minute. A detailed timeline of each run is available in Appendix J.

4.8.2.2 Data Analysis

Once the pcap files are created from the network capture, they are passed into the preprocessing script as described in Section 3.6.4.2. Its outputs (csv-formatted files) are used to generate the packet signatures in R (see Appendix K). First, the data from the three runs are pooled into one set to be analyzed. Then timestamps in the data are recalculated to represent a relative time since the observation of the first packet during each run. 720 and 1440 seconds are added to the recalculated timestamps for runs two and three respectively, to stagger the three runs into one time-series data. To generate the signatures, each packet is assigned a device state (see Section 3.6.4.3),

Table 11: Order of Device Activation

No.	Run 1	Run 2	Run 3
1	SwitchTest	LightBulb1	LightBulb2
2	Switch3	SwitchTest	Switch2
3	Camera1	Camera2	Camera3
4	LightBulbTest	LightBulbTest	Switch1
5	CameraTest	Switch3	CameraTest
6	Switch1	Camera3	SwitchTest
7	Lightbulb2	Lightbulb2	LightBulb1
8	Camera3	Camera1	LightBulbTest
9	Switch2	Switch2	Camera1
10	Camera2	Switch1	Switch3
11	LightBulb1	CameraTest	Camera2

then the packet signatures for each state (e.g., density of packet lengths, IAT) are generated. Finally, PCA is performed on the signatures to generate the PCS; the first principal components whose sum of variations account for at least 80% of the total variation are used to calculate the SS (see Section 3.6.4.4). Three different thresholds for SS, 0.7, 0.8 and 0.9, are used to define a positive classification event. Next, the number of TP, TN, FP, and FN are first determined by using SS thresholds only, then they are cross validated with the OUI information for each device. For example, in the second set of results, device pairs with SS values above the threshold but with different OUIs will be classified as a negative event (i.e., non-identical models).

4.8.2.3 Performance Metric

Metrics used to determine the performance of IDMC are precision, recall, and specificity.

- **Precision**: It is the percentage of IDMC's positive identical-model classifica-

tions that are actually positive. Precision is defined by

$$Precision = \frac{TP}{(TP + FP)} \times 100 \tag{7}$$

- **Recall**: It measures the percentage of all of the identical-model pairs that are identified by IDMC. Recall is defined by

$$Recall = \frac{TP}{(TP + FN)} \times 100 \tag{8}$$

- **Specificity**: It measures the percentage of all of the non-identical-model pairs that are identified by IDMC. Specificity is defined by

$$Specificity = \frac{TN}{(TN + FP)} \times 100 \tag{9}$$

4.8.3 Experiment 2: Measuring the Effectiveness of IoTAMU's Spoofing Capability

The second experiment measures the IoTAMU's ability to change the observed packet signatures of IoT devices. Two different methods are used to estimate its effectiveness. Using the same metrics as that of the first experiment, the first method measures IDMC's performance in the spoofer-enabled network. The second method calculates the change in the pairwise distances between devices from the PCS of the devices. Network configuration 1 described in Figure 8 is used in experiment 2.

4.8.3.1 Data Collection Process

Data collection is performed using the same methodology as that of the first experiment with the addition of the spoofer. Two different configurations of the spoofer are investigated, with three runs performed at each configuration. The first

configuration spoofs AGPS of 11 samples to test IoTAMU at its minimum settings (one sample spoofed for each device in the network), and the latter spoofs 120 AGPS to test its maximum setting (120 is the maximum number of samples that can be reliably spoofed using the experiment setup, without exceeding the maximum number of threads or running out of memory). As discussed in Section 3.6.4.5, the AGPS are created by sampling from a multivariate normal distribution with the covariance matrix generated from the packet signatures of the existing network. For each run of experiment 2, a unique set of AGPS is calculated from the packet signatures obtained during experiment 1. Before the sniffer is started for each run, the spoofer is run for five seconds to allow it to fully initialize; the rest of the experiment follows the same sequence of events as defined in Section 4.8.2.

4.8.3.2 Performance Metric

The first set of metrics used for the experiment is the same as those of the first experiment. Differences in IDMC's precision, recall, and specificity from those calculated in the baseline network indicate a change in the packet signatures of the IoT devices.

Additionally, if the spoofing algorithm is working as intended, it should change the observed signatures of the IoT devices such that there are greater differences between each pair. To quantify this change, the following metric is used:

- **Pairwise Signature Distance (PSD):** PSD measures the euclidean distances between the PCS of a pair of device signatures. Because the distance values are used in SS to calculate the degree of similarity between pairs of devices, an increase in PSD directly translates to greater differences in the observed packet

signatures. PSD for $device_i$ and $device_j$ is simply defined by

$$PSD(device_i, device_j) = distance(PCS_{device_i}, PCS_{device_j}) \tag{10}$$

where PCS is the set of PCS of first principal components whose sum of variations add up to at least 80% of the total variation.

Lastly, the calculated PSD from the baseline network (from experiment 1), and those with 11 and 120 spoofed samples is used in Analysis of Variance (ANOVA) to determine whether there is a statistically significant difference in their mean distances. A p-value threshold of 0.05 is used.

4.8.4 Experiment 3: Measuring IoTAMU's Impact on Network Congestion

Experiment 3 measures the potential negative effects of injecting additional packets into the network. NL, PD, PC, and NT measured during a simulated communication between a client and a server are the metrics used in this experiment. As discussed in Section 4.7, each level of IoTAMU status is repeated three times. Then, the results from three trials are combined and used in ANOVA to determine whether there are significant differences in both NL and PD for increasing number of devices spoofed. Network configuration 2 described in Figure 9 is used for experiment 3, where `Laptop1` and `Laptop2` are posed as an IoT device and its RI, respectively.

4.8.4.1 Data Collection Process

This experiment compares the congestion metrics acquired from networks with increasing amounts of spoofed traffic. A pair of UDP server and client scripts written in Python are used to help measure the NL and PD (see Appendix L). The 11 IoT devices used in experiments 1 and 2 are left to run in their steady states throughout

the experiment. While the spoofer is running, the client (`Laptop1`) generates a series of 50 packets with payload content ranging from 0 to 49 (which serve as sequence numbers), and sends to the server. Upon receiving the packet, the server (`Laptop2`) replies with the same payload content to serve as an acknowledgement. The duration in seconds for the client to send a packet and receive the appropriate response from the server is recorded as the NL (output of `congestion_cli.py`). A timeout period of two seconds is set while the client waits for the response to account for any packets dropped due to network congestion. At the end of the test, total number of dropped packets is calculated by subtracting the number of received packets from the number of expected packets (50).

The timeline of events for each treatment is shown in Table 12. To measure the congestion statistics for the baseline network (i.e., 11 IoT devices running in steady states with 0 spoofed samples), the sniffer is started and run for 15 seconds before the server and client scripts are run on the respective laptops. The two scripts are run via the commands:

```
python3 congestion_srv.py
python3 congestion_cli.py
```

where the server script is run first before running the client script. 15 seconds after starting the sniffer, the UDP scripts are run, exchanging 50 request and response pairs; 15 seconds after the scripts are run, the sniffer is stopped, completing the capture. Subsequent treatments involve spoofing samples in increments of 10 on top of the baseline network, up to 120 samples. As previously mentioned, the number of samples that can be spoofed is limited by IoTAMU's implementation. To create the 120 AGPS to be spoofed, the covariance matrix calculated from the overall packet signatures acquired in experiment 1 is used. Each treatment spoofing N devices uses the first N entries from the list of AGPS. This causes an overlap in the signatures

that are spoofed between the treatments, resulting in a more consistent outcome. Capturing the data for the spoofer-enabled treatments uses the same method as the one used to capture the baseline network, with the addition of a five second window where the spoofer is initialized before the sniffer is started. The pcap-formatted output files from the sniffer are preprocessed as discussed in Section 3.6.4.2; the resulting csv-formatted files are used to visualize the PC and the throughput at increasing number of samples spoofed.

Table 12: Timeline of Events for Each Treatment in Experiment 3

Time(sec)	**Number of Samples Spoofed**												
	0	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**
0	-	Start spoofer											
5	Start capture												
20	Start server/client scripts												
35	End capture/stop spoofer												

4.8.4.2 Performance Metric

After the NL measurements are acquired, ANOVA is performed to determine whether a statistically significant difference in the mean NL times exists between the networks with increasing amounts of spoofed samples. Because the network interference from environmental factors are assumed to be negligible, data from all three trials are pooled into one. Then, NL, PD, PC, and NT are compared at each number of samples spoofed to visualize the amount of traffic introduced from each additional sample and their effect on congestion.

4.9 Methodology Summary

This chapter provides the methodology used to measure the performance of the spoofing capability of IoTAMU. The differences in precision, recall, and specificity of IDMC with and without spoofing enabled is used to measure IoTAMU's effectiveness. To quantify the change in network signatures caused by spoofing, PSD is calculated between each pair of devices. Lastly, IoTAMU's potential impact on network congestion is estimated though NL, PD, PC, and NT statistics acquired from spoofing increasing number of devices.

V. Results and Analysis

5.1 Overview

This chapter presents the results obtained from the experiments as described in Chapter IV. The following sections discuss the results from each experiment. Section 5.2 describes the performance of IDMC in a baseline network. Its performance with the spoofer enabled is discussed in Section 5.3 to determine the effectiveness of IoTAMU's spoofing capability. Lastly, the network congestion metrics are examined in Section 5.4 evaluate the spoofer's effect on the existing network.

5.2 Performance of IDMC (Experiment 1)

This section evaluates the performance of IDMC through its precision, recall, and specificity. The results are calculated using the R script provided in Appendix M.

Figure 29 shows a 3-D scatter plot of the first three PCS of each device. Raw PCS calculated in experiments 1 and 2 are provided in Appendix N. The three principal components shown are responsible for 70% of the total variation in the data. Each device type is assigned a unique color, and the identical models within each type are denoted as triangles. Three distinct clusters are identified in the graph, each consisting of identical model pairs (i.e., triangles of same color), with identical light bulb models (blue triangles) almost completely overlapping one another. This indicates a high probability that the clustered devices are identical models, which is quantified via the SS.

The overall SS values calculated from the PCS is summarized in Table 13. The bolded values indicate the scores of actual identical model pairs, (`Camera1`, `CameraTest`), (`LightBulb1`, `LightBulbTest`), and (`Switch1`, `SwitchTest`), with values 0.9532, 0.9968, and 0.9511, respectively. Based on three different thresholds, the SS values

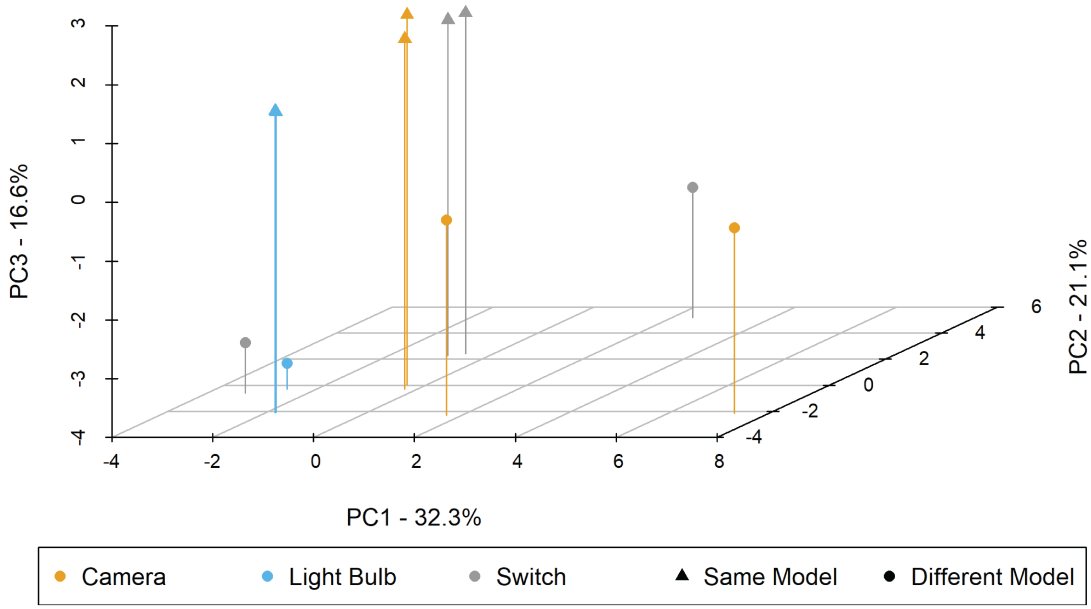


Figure 29: Visualization of the first three principal components of the 11 devices (Experiment 1).

are used to calculate the performance metrics.

Table 13: Summary of SS for Each IoT Pair (Experiment 1)

	C1	CT	C2	C3	LB1	LBT	LB2	S1	ST	S2
CT	**0.9532**									
C2	0.23	0.1983								
C3	0.4596	0.4405	0.2379							
LB1	0.6956	0.6835	0.07828	0.5805						
LBT	0.6964	0.6842	0.07931	0.5819	**0.9968**					
LB2	0.4512	0.4265	0.03666	0.3384	0.4879	0.4906				
S1	0.752	0.774	0.1179	0.2995	0.516	0.5175	0.4392			
ST	0.7478	0.7734	0.1232	0.3212	0.5171	0.5187	0.4207	**0.9511**		
S2	0.2807	0.287	0.0738	0.2999	0.1799	0.1826	0.2355	0.3784	0.4136	
S3	0.4507	0.4262	0	0.2715	0.4782	0.4804	0.8923	0.4387	0.4125	0.1553

Table 14 shows the corresponding performance metrics at each threshold SS value with and without cross validation with OUI. While significant changes are observed in IDMC's precision across the threshold values, its recall and specificity are consistently high, with 100% recall and >90% specificity throughout. This indicates that

Table 14: IDMC Performance Metrics at Each Threshold Value

OUI	**Threshold**	**TP**	**TN**	**FP**	**FN**	**Prec**$_{\%}$	**Recall**$_{\%}$	**Spec**$_{\%}$
NO	>0.7	3	47	5	0	37.5	100	90.38
NO	>0.8	3	51	1	0	75	100	98.08
NO	>0.9	3	52	0	0	**100**	**100**	**100**
YES	>0.7	3	47	5	0	37.5	100	90.38
YES	>0.8	3	51	1	0	75	100	98.08
YES	>0.9	3	52	0	0	**100**	**100**	**100**
TP: true positives, TN: true negatives, FP: false positives, FN: false negatives, Prec: precision, Spec: specificity								

IDMC is able to correctly identify all identical models at high SS thresholds, and lowering the threshold yields more FPs. At the >0.9 threshold, IDMC successfully identifies all three out of three identical-model pairs with no FP, yielding 100% precision. As expected, each step of the lowered threshold levels introduces additional FPs, decreasing both precision and specificity. Precision of IDMC suffers the largest, dropping as low as 37.5% at 0.7 threshold. However, the significant changes in precision across the different threshold levels are reflective of the small number of the actual positive classifications compared to that of actual negative classifications, and does not necessarily reflect poorly on IDMC's performance. Overall, IDMC performs better at higher thresholds, and its 100% performance metrics measured at the >0.9 threshold is indicative of its reliability as a classifier.

The pairs that are identified as FP are (`Switch3`, `LightBulb2`) at threshold >0.8, (`Camera1`, `Switch1`), (`Camera2`, `Switch1`), (`Camera1`, `SwitchTest`), and (`Camera2`, `SwitchTest`) at threshold >0.7. Because all FP pairs are from the same vendors, validating the classification results with OUI does not produce any changes in results. The similarities between the two TP-Link devices `Switch3` and `LightBulb2` are to be expected, as they are both instantaneously activated devices with two different

states (on/off) from the same vendor. It is likely that the same protocols are used to activate the devices, resulting in similar network signatures. Similarly, the four additional FPs introduced at >0.7 threshold are from the same vendor and are likely to use similar network protocols. On the contrary, the two different switch models from Belkin (`Switch1`/`SwitchTest` and `Switch2`) have fingerprints that greatly differ from each other ($SS \approx 0.4$). These results demonstrate IDMC's ability to differentiate the nonidentical model devices from the same manufacturer that likely use similar network protocols.

5.3 Effectiveness of IoTAMU's Spoofing Capability (Experiment 2)

This section discusses IoTAMU's ability to modify the observed network fingerprints of the devices. The changes in observed network signatures across the treatments of `Camera1`, `CameraTest`, `LightBulb1`, and `LightBulbTest` are illustrated in Figures 30 and 31. Those of other devices are available in Appendix O. Histograms on the left under each device show the number of packets observed over time with bin widths of 0.5 seconds. The time frames classified as `ACTIVE`, and `PASSIVE` by IDMC are denoted in red and blue, respectively. On the other side are scatter plots indicating the lengths of packets received over time. The first row under each device shows the distributions observed in the network without spoofing (from experiment 1), and subsequent rows depict networks with 11 and 120 samples spoofed, respectively. The sharp increases in the number of packets sent during the active states of cameras are retained in the spoofed networks. However, increasing the number of spoofed samples better masks the underlying network patterns of the devices overall, each device displaying a unique pattern of added signatures (i.e., different patterns are observed comparing the plots across the same rows of each device). Furthermore, the network patterns observed in baseline network are completely masked for all light

bulb models, `Switch2`, and `Switch3` by the additional frames generated by spoofing 120 samples.

Two different sets of metrics are used to quantify the observed changes in network signatures: The same metrics as that of experiment 1 are used to observe the decline in IDMC's performance in a spoofed network, and PSD is calculated for each pair of devices to measure the relative changes in network fingerprints. An overall PSD is calculated for the network by taking an average of all device pairs.

5.3.1 IDMC Performance

Figures 32 and 33 illustrate the relative positions of the devices using the first three principal components derived from the networks with 11 and 120 samples spoofed, respectively. Compared to the distribution observed in Figure 29 from experiment 1, the clusters are less defined. Although the signatures of `Switch1` and `SwitchTest` remains relatively close in Figure 32, those of other identical-model devices show significant changes. The observed differences are quantified through SS in Tables 15 and 16. As expected, SS for identical-model pairs are significantly lower than those from experiment 1.

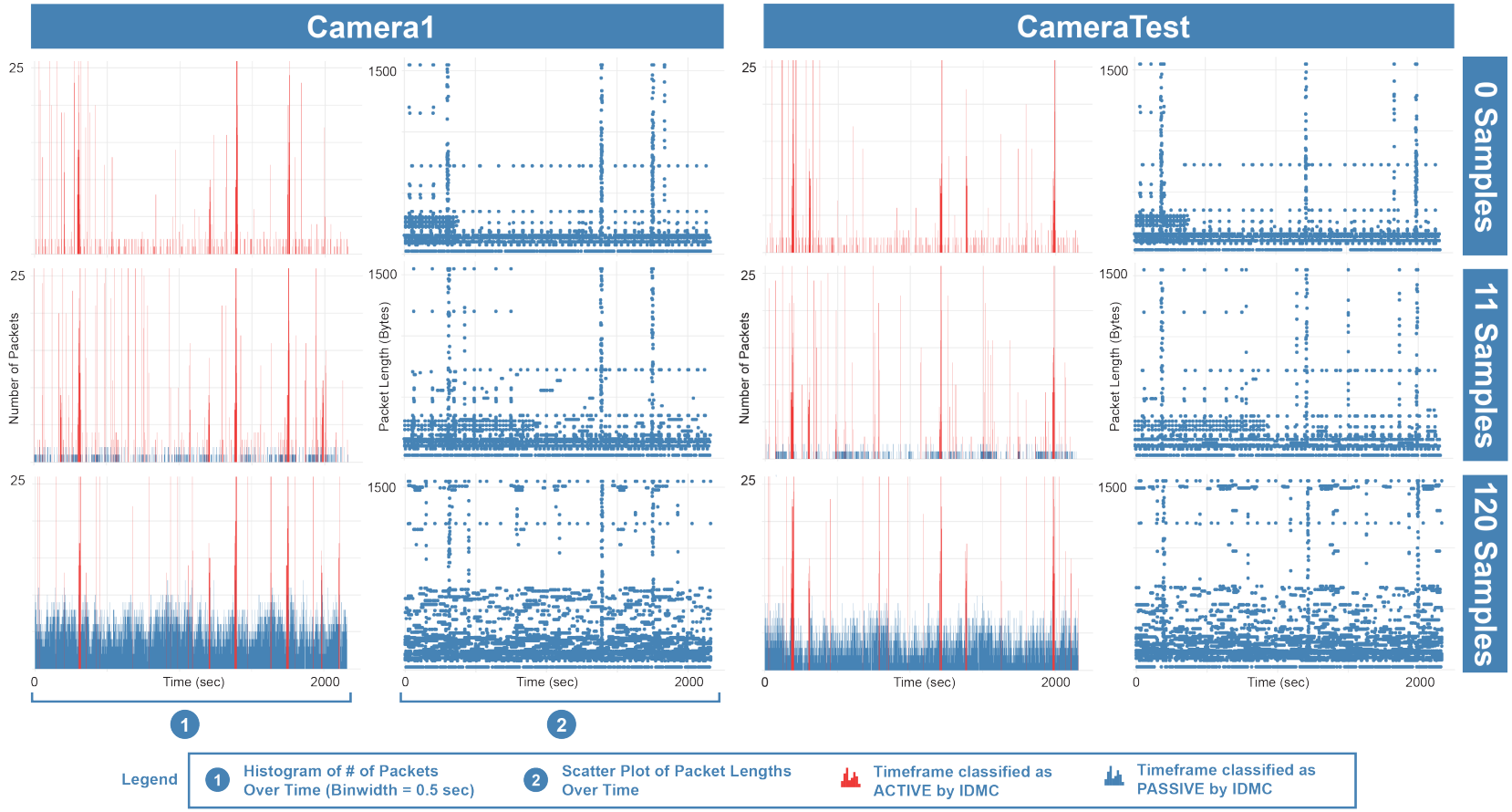


Figure 30: The observed network fingerprints of Camera1 and CameraTest at three different configurations of IoTAMU.

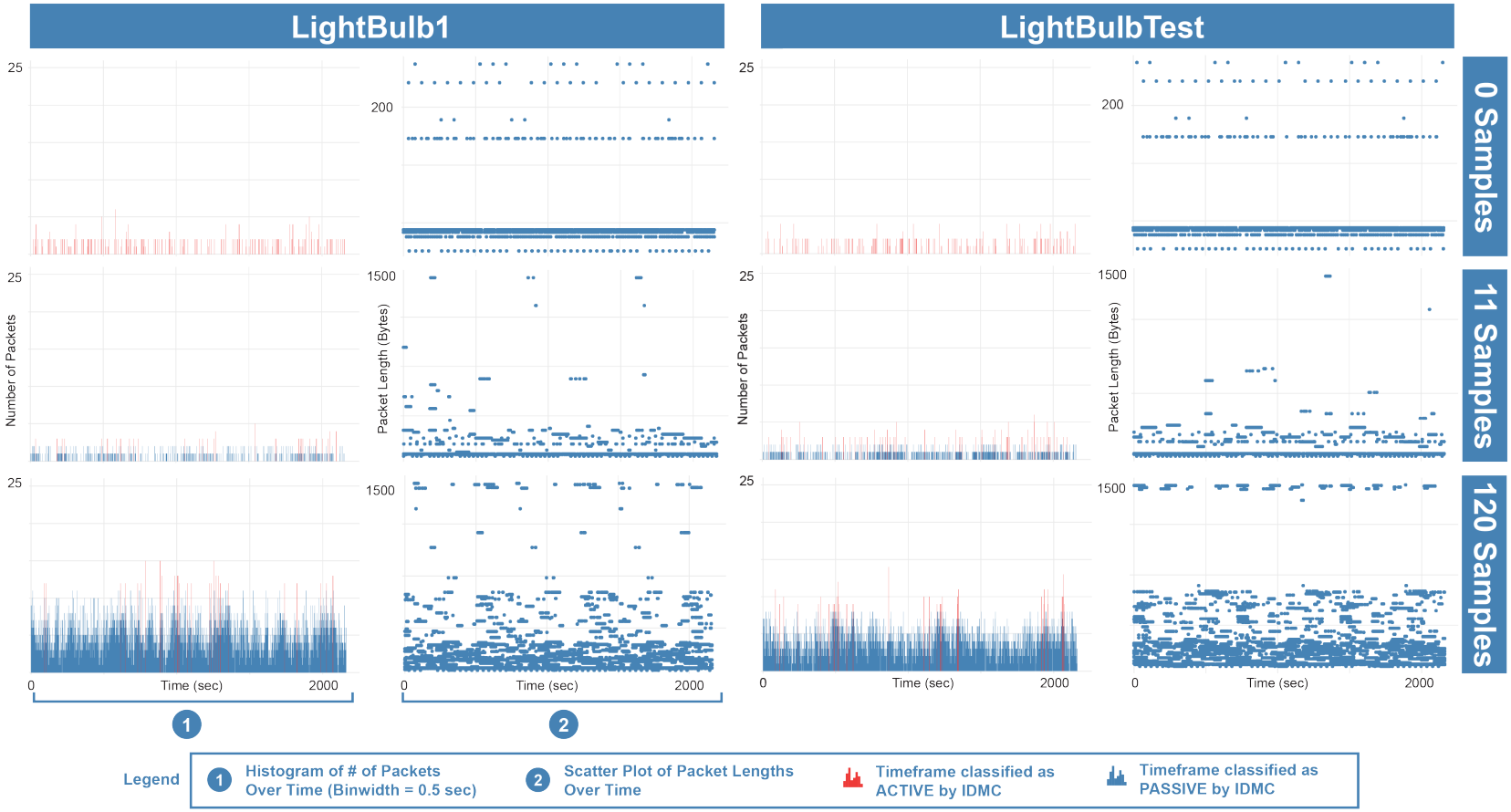


Figure 31: The observed network fingerprints of LightBulb1 and LightBulbTest at three different configurations of IoTAMU.

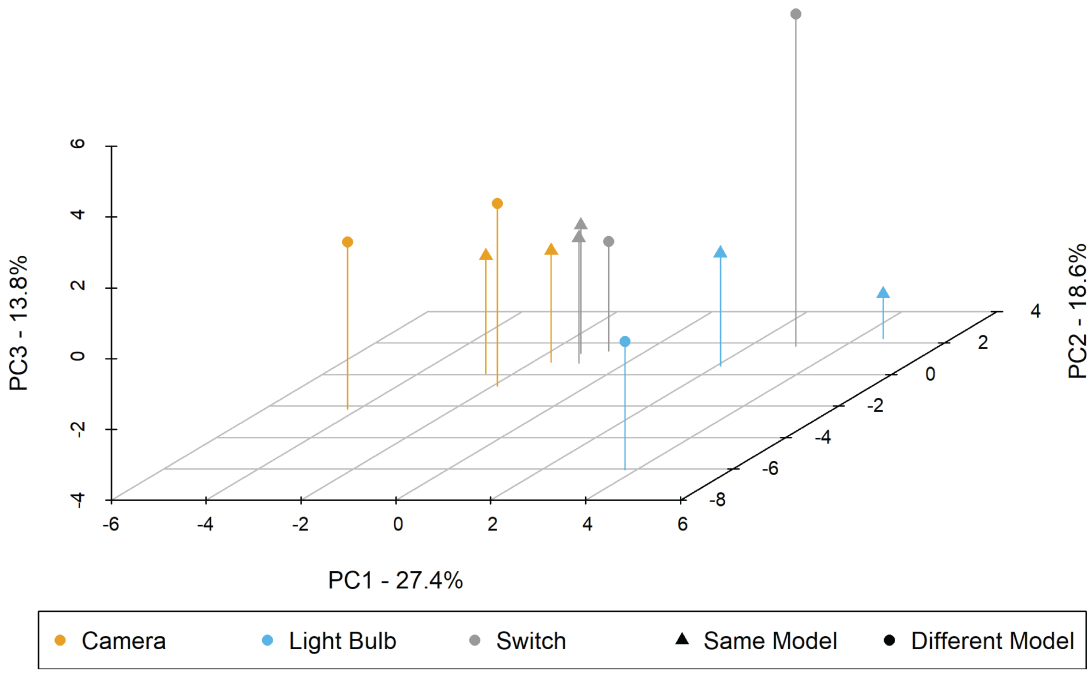


Figure 32: Visualization of the first three principal components in the network with 11 spoofed samples (Experiment 2).

Table 15: Summary of SS for Each IoT Pair (Experiment 2: 11 Spoofed Samples)

	C1	CT	C2	C3	LB1	LBT	LB2	S1	ST	S2
CT	**0.8712**									
C2	0.4461	0.5568								
C3	0.3588	0.3806	0.4616							
LB1	0.2385	0.1675	0	0.1138						
LBT	0.5216	0.4385	0.1988	0.4502	**0.443**					
LB2	0.1608	0.146	0.05757	0.09556	0.06243	0.3193				
S1	0.7601	0.7099	0.4439	0.5624	0.3025	0.6854	0.2011			
ST	0.7786	0.6967	0.3452	0.4409	0.2094	0.6237	0.1431	**0.8369**		
S2	0.7247	0.6322	0.2811	0.4187	0.2355	0.6673	0.1487	0.8085	0.9146	
S3	0.2473	0.1932	0.05201	0.1724	0.1451	0.3345	0.05872	0.2987	0.2814	0.2538

Performance statistics for IDMC in both networks are summarized in Table 17. At the highest threshold, IDMC fails to identify any of the actual identical model devices, while one FP is identified in the network with 11 spoofed samples. The identified FP pair (`SwitchTest`, `Switch2`) saw an increase of roughly 0.5 in SS compared to that of

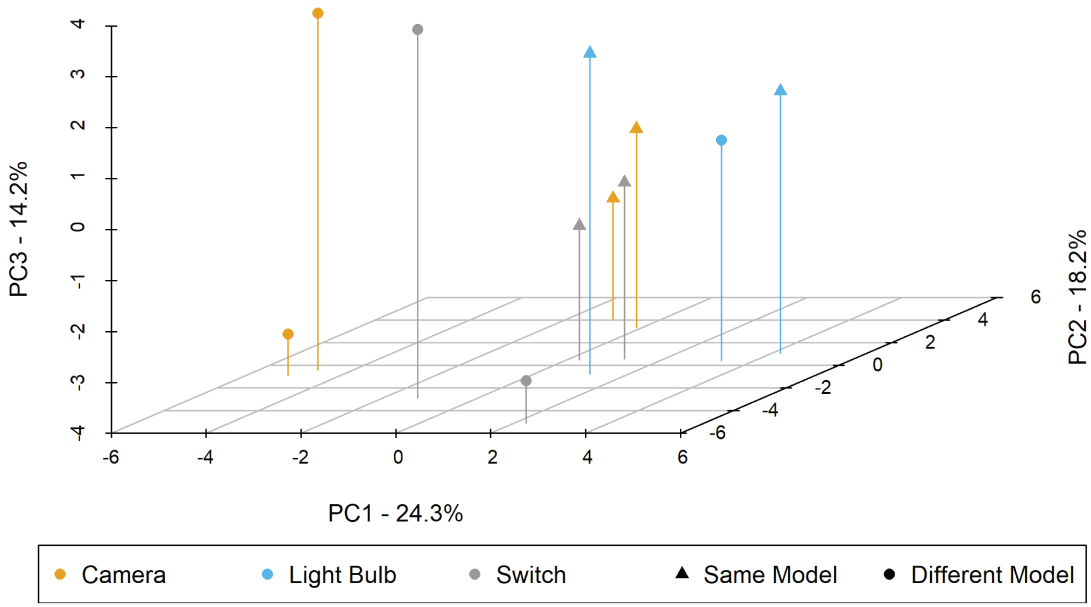


Figure 33: Visualization of the first three principal components in the network with 120 spoofed samples (Experiment 2).

Table 16: Summary of SS for Each IoT Pair (Experiment 2: 120 Spoofed Samples)

	C1	CT	C2	C3	LB1	LBT	LB2	S1	ST	S2
CT	**0.7209**									
C2	0.2525	0.2424								
C3	0.1841	0.1569	0.1288							
LB1	0.4959	0.2932	0.2978	0.2009						
LBT	0.4141	0.2326	0.06596	0.0208	**0.6355**					
LB2	0.3229	0.2833	0.04924	0.07434	0.4483	0.4597				
S1	0.6427	0.5194	0.314	0.2891	0.594	0.5216	0.3366			
ST	0.6802	0.4946	0.2568	0.2436	0.6847	0.6109	0.3968	**0.8587**		
S2	0.3267	0.1688	0.261	0.171	0.551	0.2038	0.2363	0.3288	0.3908	
S3	0.1358	0.1052	0.03779	0	0.256	0.1651	0.1779	0.3877	0.3558	0.1815

the baseline network. Similarly, several FP pairs identified in lower thresholds from the network with 11 spoofed samples saw significant increases in SS, which demonstrate the spoofer's ability to sufficiently change the observed network fingerprints of the devices. Lowering the threshold level increases the number of the correctly identified positives, but the classifier fails to reach 100% recall. Although spoofing 120 samples does not introduce more FP pairs, the ultimate goal of the spoofer is

to magnify the differences in the overall observed network signatures. Thus, the decrease in recall is a better metric of the spoofer's performance. Overall, a decline in performance is observed for IDMC after the introduction of the spoofer.

Table 17: IDMC Performance Metrics at Each Threshold Value

#SpSam	**Threshold**	**TP**	**TN**	**FP**	**FN**	**Prec$_{\%}$**	**Recall$_{\%}$**	**Spec$_{\%}$**
11	>0.7	2	46	6	1	25	66.67	88.46
11	>0.8	2	50	2	1	50	66.67	96.15
11	>0.9	0	51	1	3	0	0	98.08
120	>0.7	2	52	0	1	100	66.67	100
120	>0.8	1	52	0	2	100	33.33	100
120	>0.9	0	52	0	3	0	0	100
#SpSam: number of spoofed samples, TP: true positives, TN: true negatives, FP: false positives, FN: false negatives, Prec: precision, Spec: specificity								

5.3.2 Pairwise Signature Distance (PSD)

PSD measures the relative similarities between the network signatures of each pair of devices. Differences in its value measured from each level of spoofing can be used to quantify the changes introduced via spoofing, with positive differences representing greater observed differences between the network signatures of a pair of devices. The calculated PSD for the three networks are provided in Appendix P. Figure 34 shows the distribution of the PSD observed in the baseline network, and the networks with 11 and 120 samples spoofed. Each point within each level of IoTAMU status (i.e., number of spoofed samples) represents the PSD for a unique pair of devices. The identical-model pairs are separately highlighted in different colors (green, orange, and red). As illustrated in the figure, an increase in the PSD values for the identical model pairs at each stage of spoofing is apparent. Furthermore, spoofing additional samples increases the mean PSD, suggesting that greater differences in network signatures

are observed overall for each pair. Performing ANOVA at a significance threshold of 0.05 shows that the observed difference in mean PSD in the networks with 0 and 11 spoofed samples are not statistically significant (p-value=0.8253). However, spoofing 120 samples is able to bring sufficient changes to the observed signatures compared to those of the baseline network, producing a statistically significant increase in PSD with a p-value of 0.01132.

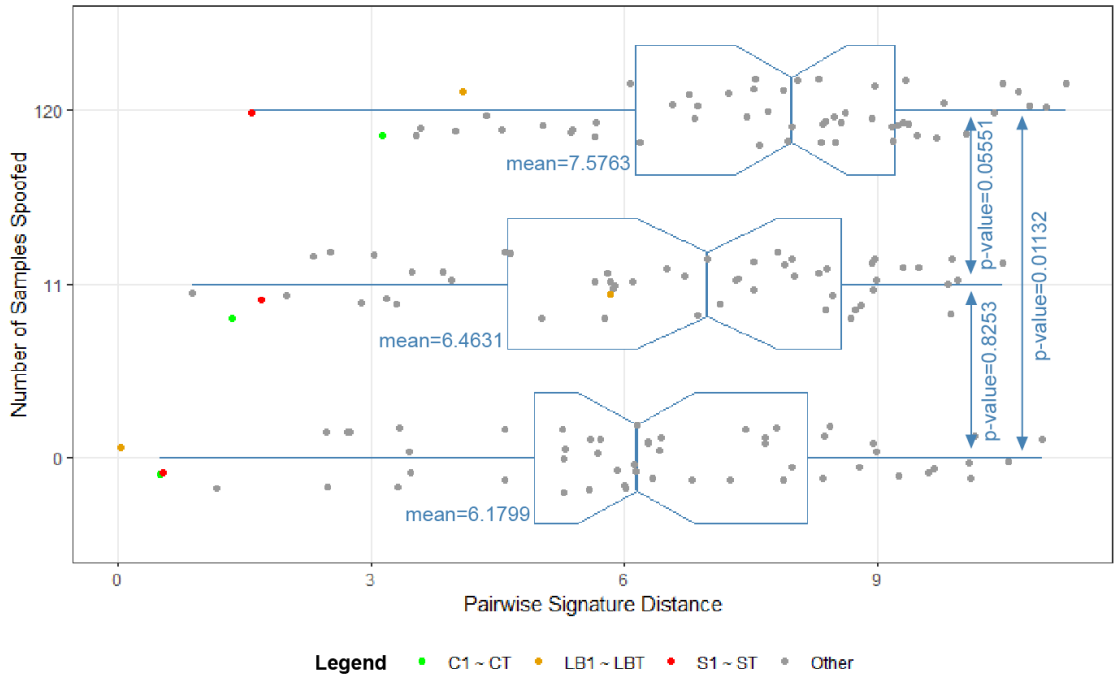


Figure 34: The distribution of pairwise signature distances observed in networks with different numbers of spoofed samples.

5.4 Experiment 3: IoTAMU's Impact on Network Congestion (Experiment 3)

To determine IoTAMU's potential impact on network congestion, NL, PD, PC, and NT are measured in a controlled environment. The metrics are calculated using the R script provided in Appendix Q. Figure 35 depicts the measured metrics for each

treatment. No dropped packets are observed across all treatments; therefore, PD is excluded from the results. Throughout all trials, the NL remains relatively consistent with an overall mean of 0.01489 seconds; with the exception of a few outliers, no significant deviations are observed. This is confirmed via ANOVA, where the null hypothesis (H_0: there is no difference in mean NL times between the treatments), is failed to be rejected with an overall p-value of 0.283. In other words, there is not enough evidence that suggest that different mean NL times are observed in networks with increasing number of spoofed samples. Both PC and NT, however, saw linear increases as more samples are spoofed by IoTAMU.

In the network with 120 spoofed samples, the observed PC from the additional network traffic reaches an average of 4181 packets across the three trials over 30 seconds, equating to a rate of 0.9167 (≈ 1) packets per second per sample spoofed after subtracting the PC observed in the baseline network:

$$\frac{(4181packets - 881packets)}{30sec/120samples} = 0.9167packets/sec/sample$$

Additionally, the average throughput for the additional traffic reaches 234.0 kbps, or 1.546 kbps per sample after subtracting the throughput observed in the baseline network:

$$\frac{8bits}{1Byte} * \frac{(2632434Bytes - 544847Bytes)}{90sec/120samples} * \frac{1kilobit}{1000bits} = 1.546kbps/sample$$

This per-sample average equates to roughly 0.001546% of the theoretical maximum throughput of the IEEE 802.11n standard ($\approx$ 100 Mbps) [54] and 0.000002209% that of the more recent IEEE 802.11ac standard ($\approx$ 7 Gbps) [55]. Therefore, the additional overhead created by each sample is largely negligible and is scalable to a much larger network. Ultimately, the relatively steady NL times as well as no packets dropped

across all trials in the experiment indicate that IoTAMU is able to produce the results found in Section 5.3 without imposing a significant burden on the network.

5.5 Results Summary

This section provides a summary of the three experiments conducted, testing the performance of the IDMC, IoTAMU's spoofing capability, and measuring the spoofer's impact on the network. Precision, recall and specificity are used to measure IDMC's ability to classify identical-model pairs. In the baseline network, IDMC is highly successful at stringent thresholds, achieving a 100% rate for all metrics at the SS threshold of 0.9. Although the introduction of FPs at lower thresholds negatively impacts IDMC's precision, its 100% recall and high specificity nonetheless demonstrate its ability to classify devices with identical/similar pairs of network signatures. With the introduction of spoofing, IDMC suffers a significant decrease in its recall, identifying no identical pairs at the highest threshold (0.9), and improving to two out of three at the lowest threshold (0.7). Furthermore, a statistically significant increase in PSD is observed in the network where 120 samples are spoofed. Finally, while steady increases in PC and NT are observed, there are no PD nor statistically significant differences in mean NL times between networks with increasing numbers of samples spoofed by IoTAMU. These results demonstrate IoTAMU's ability to sufficiently modify the observed network signatures of the existing devices, while imposing a negligible amount of additional traffic into the network. The experiments in this research show that IoTAMU's spoofer is able to uniquely modify the observed patterns of network signatures of devices, decreasing the likelihood of information leakage from unencrypted fields of encrypted Wi-Fi traffic.

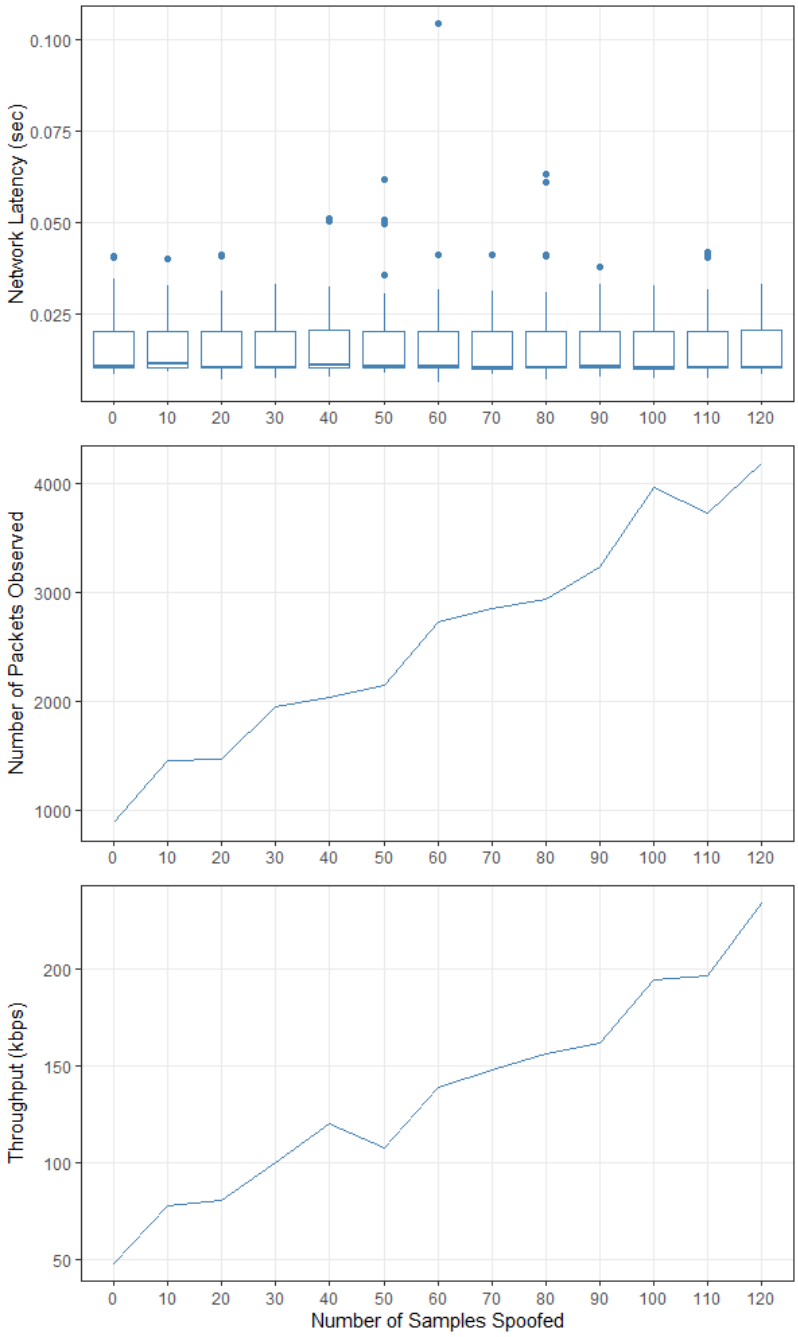


Figure 35: The observed congestion metrics for each device at different number of samples spoofed by IoTAMU.

VI. Conclusion

6.1 Overview

This section summarizes the research and the results of the experiments in this research. Section 6.2 highlights the significant conclusions drawn from this research's findings, and they are synthesized in Section 6.3 to underline their implications in IoT security. Lastly, Section 6.4 discusses potential future work for IoTAMU.

6.2 Research Conclusions

This research successfully evaluates the effectiveness of a novel security agent in improving the data confidentiality of IoT networks via the following contributions: (1) proposal of the architecture of a PNP-style IoT security agent, (2) demonstration of utilizing a remote computing node to provide added layers of security for IoT devices, (3) development of a classifier that can accurately identify identical-model devices and measure the similarities in network signatures of devices, and (4) development of a spoofing algorithm that can uniquely modify the observed network signatures of the devices based on the distributions of previously observed signatures.

First, the soundness of IoTAMU's design is demonstrated by conducting all experiments for this research on a network set up with IoTAMU serving as the gateway for all IoT devices. IoTAMU serves as the AP as well as the firewall, routing all traffic for the devices. Its encryption functionality is also demonstrated via a proof-of-concept experiment where a simulated communication between an IoT device and its RI successfully exchanges encrypted and decrypted traffic via IoTAMU.

To test the effectiveness of IoTAMU's spoofer, IDMC is developed to characterize and classify the observed network traffic of the IoT devices. At a SS threshold of 0.9, IDMC is able to correctly classify all identical-device pairs in the network with

100% precision, recall, and specificity. The classifier's precision suffers from FPs at lower threshold levels. However, the identified FP pairs are devices from same manufacturers which likely use the same network protocols for communication. IDMC's performance at stringent threshold (SS>0.9) installs high credibility in its ability to measure the similarities in network signatures and classify identical device models.

As hypothesized, IDMC's performance declines after enabling IoTAMU's spoofing capability. At high threshold (SS>0.9), the classifier is not able to identify any of the positive pairs; its performance improves at lower thresholds (SS>0.8,0.7) identifying up to two out of three identical-model pairs. Then, the PSDs are measured for each device pairs to characterize the overall affect of the spoofing algorithm on their network signatures. Although not statistically significant, spoofing 11 samples on average yields greater distances between each pair of devices. More importantly, the distances between the identical-model devices are significantly higher than those observed in the baseline network. Spoofing 120 samples does produce a statistically significant increase in the mean distance between devices as hypothesized. These results successfully demonstrate IoTAMU's ability to modify the network signatures of the devices to help prevent information leakage from variations in observed network traffic patterns.

Finally, measuring the additional traffic created by the spoofer shows a negligible impact on network congestion. While increasing the number of spoofed samples linearly increases throughput and number of observed packets, there is no statistically significant increase in NL. On average, each spoofed sample produces approximately one packet per second, with a throughput of 1.546 kbps. The additional load of traffic created by IoTAMU is largely negligible in the current IEEE 802.11 standards, and is scalable to a much larger network.

6.3 Research Significance and Synthesis

From light bulbs to smart TVs, IoT devices are replacing more everyday objects in an ordinary household. While they provide new capabilities, they also increase the potential attack surface for adversaries. Numerous prior research as well as the experiments in this research have demonstrated the potential information leakage from the unprotected fields available to the public in wireless protocols, including those not studied in this research such as Bluetooth and Zigbee. Because significant portions of today's communications are wireless, these studies raise serious concerns about the privacy of the users.

Ordinarily, an adversary does not necessarily have the motive to target a regular smart home. Furthermore, the vulnerabilities found in IoT are often device specific, and a vulnerability found in one particular model will not have any effect in another. However, the classifier developed for this research helps identify a case where Wi-Fi leakage from ordinary smart homes can make them easier targets. Consider the following scenario. A malicious actor is aware of a vulnerability that exists in a particular Wi-Fi door lock. To find out if such a lock is present in a neighborhood, the actor first trains the classifier with the network signature of the door lock. Then the actor can sniff the network traffic for similar signatures from a remote location by employing a drone or simply wardriving. This reconnaissance technique enables the actor to cover a much larger area to identify potential targets without drawing suspicion. In this scenario, modifying the observed network signatures of the devices becomes the first line of defense against these reconnaissance methods.

Although developed for Wi-Fi IoT, the security agent proposed in this research has implications for all wireless protocols and devices. Data mining in wireless communications remains a threat as long as the unprotected portions of wireless frames can be intercepted. Therefore, unless the signals are kept within a perfect Faraday

cage, an adversary with enough time and motive is able to obtain potentially valuable information. In case of encrypted Wi-Fi, the primary sources of information come from the source and destination addresses, length of the payload, and the time the packet is observed. The specific ways in which these fields vary over time enable certain knowledge to be inferred. Therefore, uniquely changing the observed traffic for each device substantially decreases the likelihood of unwanted leakage of knowledge to the public.

Similarly, the security agent proposed in this research has important applications in the operations of the Air Force and the DoD. From security cameras to personal devices owned by DoD personnel, countless facets of their day-to-day operations are aided by IoT. Leakage of information from any sources can lead to endangerment of assets, personnel, and ultimately the mission. The capabilities implemented in IoTAMU enable greater protection of information from both insider threats and outsiders via isolation of potentially vulnerable devices and modification of the network signatures to be unrecognizable. As organizations become more reliant on wireless communications to conduct their operations, it becomes paramount to investigate the security implications of their publicly-available byproducts.

Deployment of IoTAMU with a few additional security measures can offer greater protection of information. First is changing the MAC addresses of the devices. The OUI in a MAC address inevitably provides valuable information about the device. If possible, changing the first three bytes of MAC addresses to be identical, or randomizing them for all devices can help harden the network against information leakage. In addition to changing the observed patterns of network signatures of the existing devices, IoTAMU can also be used to mimic the signature of a nonexistent device by spoofing with a MAC address that is not present in the network. This generates greater complexity within the network and provides an additional layer of security.

If this feature is to be used to spoof multiple additional devices, spoofing from multiple antennas with different strengths of attenuators can prevent the adversary from recognizing the source of the packets by comparing their signal strengths.

Lastly, the proposed architecture of IoTAMU paves the groundwork for the development of additional functionalities for IoT devices via remote computing, where dedicated computing nodes can add custom features for devices with less computational capacity. The capabilities demonstrated in this research are just few of many applications of this scheme; further research can provide more advanced layers of security and functionalities for IoT.

Many prior research efforts have stressed the importance of designing IoT devices with security in mind, but has led to a limited improvement on the quality of the devices over the years. This research not only proposes a potential solution to protect the confidentiality of the devices, but also offers an alternate way to approach the problem. Instead of relying on the manufacturers to individually secure the devices using different standards, a deeper investigation of hardening the universal communication protocols used by the devices could produce more fruitful results.

6.4 Future Work

The architecture of IoTAMU is a prototype, and future work is warranted to better evaluate its performance. Several areas of future work are described below.

- The encryption functionality tested in this research directly performs encryption on the received packets via a custom script. Comparing its overhead to that of a traditional VPN technique could offer additional insight to its design and implementation.

- Currently, the different functionalities of IoTAMU exist as separate scripts that are individually run on a laptop. Migrating the functionalities to a dedicated

unit (i.e., Raspberry Pi), and packaging them into one module could help test its functionalities in an actual deployment setting.

- All functioning scripts for IoTAMU are written in Python. To minimize the overhead created from processing, they should be migrated to a lower-level language such as C++.

- The maximum number of samples that could reliably be spoofed by IoTAMU is 120. Optimizing the implementation to support additional samples could help obscure the signatures from extended activation events of streaming devices (e.g., camera).

- The spoofing algorithm in this research only considers the data frames of Wi-Fi. Incorporating beacon frames into the equation is a more sophisticated approach.

- All experiments in this research are conducted in a relatively controlled environment with minimal external interference. Conducting additional experiments in different network environments is necessary to fully test IoTAMU's performance.

- There are only three identical-model pairs tested in this research. Acquiring more devices and reproducing this research's results is necessary.

- The spoofing algorithm's parameters (e.g., duration of a segment, number of repeats, etc.) are fixed throughout the experiments. Investigating the change in network signatures using different parameters can provide more insight to the spoofer's performance.

- This research is only concerned with devices that use Wi-Fi. The techniques used in this research can be extended to other wireless protocols such as Bluetooth and Zigbee.

Appendix A. IoTAMU Access Point Configuration Files

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).
# /etc/network/interfaces

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# set wlan0 in access point mode
auto wlan0
iface wlan0 inet static
hostapd /etc/hostapd/hostpad.conf
address 192.168.1.254
netmask 255.255.255.0

auto br0
iface br0 inet dhcp
bridge_ports eth1 wlan0
```

etc/network/interfaces

```
### /etc/hostapd/hostapd.conf

interface=wlan0
bridge=br0

# SSID to be used in IEEE 802.11 management frames
ssid=IoTAMU
# Driver interface type (hostap/wired/none/nl80211/bsd)
```

```
9  driver=nl80211
10 # Country code (ISO/IEC 3166-1)
11 country_code=US
12
13 # Operation mode (g = IEEE 802.11g )
14 hw_mode=g
15 # Enable 802.11n support
16 ieee80211n=1
17 # Channel number
18 channel=6
19 # Maximum number of stations allowed
20 max_num_sta=15
21
22 # Bit field: bit0 = WPA, bit1 = WPA2
23 wpa=2
24 # Bit field: 1=wpa, 2=wep, 3=both
25 auth_algs=1
26
27 # Set of accepted cipher suites
28 rsn_pairwise=CCMP
29 wpa_pairwise=TKIP
30 # Set of accepted key management algorithms
31 wpa_key_mgmt=WPA-PSK
32 wpa_passphrase=IoTAMUPassword!
33
34 # hostapd event logger configuration
35 logger_stdout=-1
36 logger_stdout_level=2
37
38 #misc.
39 ctrl_interface=/var/run/hostapd
40 ctrl_interface_group=0
```

```
41
42 #wps
43 eap_server=1
44 wps_state=2
45 ap_setup_locked=1
46 wps_pin_requests=/var/run/hostapd.pin-req
47 config_methods=label display push_button keypad
48
49 ## QoS support
50 #wmm_enabled=1
51 ## Use "iw list" to show device capabilities and modify ht_capab
       accordingly
52 #ht_capab=[HT40+][SHORT-GI-40][TX-STBC][RX-STBC1][DSSS_CCK-40]
```

Code/hostapd.conf

Appendix B. IoTAMU Setup Code

```
1  #!bin/sh
2  ### hostapd.conf : /etc/hostapd/hostapd.conf -> ./hostapd/hostapd.conf;
       /etc/default/hostapd;
3  ### /etc/network/interfaces
4  ### dnsmasq.conf : /etc/dnsmasq.conf
5  #IoTAMU initialization code
6  #uses iptables for firewall and hostapd to start the AP
7
8  IPTABLES=/sbin/iptables
9  EBTABLES=/sbin/ebtables
10
11 INT_NET=192.168.1.0/24
12 REMOTE=172.16.0.2
13 IOT=192.168.1.17
14
15 ### flush existing rules and set chain policy setting to DROP
16 echo "[+] Flushing existing iptables rules..."
17 $IPTABLES -F
18 $IPTABLES -F -t nat
19 $IPTABLES -X
20 $IPTABLES -P INPUT DROP
21 $IPTABLES -P OUTPUT DROP
22 $IPTABLES -P FORWARD DROP
23
24 ### ACCEPT rule for packets originating from the internal network
25 $IPTABLES -A FORWARD -i wlan0 -s $INT_NET -j ACCEPT
26
27 ### DROP rules
28 # used for proof of concept encryption test
29 # drop packets with the MAC address of Laptop1 after routing:
```

```
30 $EBTABLES -P FORWARD ACCEPT
31 $EBTABLES -A FORWARD -i eth1 -d  C4:9D:ED:2D:AC:83 -j DROP
32
33 ### enable forwarding
34 echo "[+] Enabling IP forwarding..."
35 echo 1 > /proc/sys/net/ipv4/ip_forward
36
37 ### enable AP
38 hostapd ./hostapd/hostapd.conf
```

Code/iotamu.sh

Appendix C. IoTAMU Encryption Code

```
1  #! /user/bin/env python
2  #
3  # Decrypt data received from the RI
4  #
5  # encryption source: https://cryptography.io/en/latest/hazmat/primitives
       /symmetric-encryption/
6
7  import socket, sys, parse, os, binascii, struct
8
9  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
       modes
10 from cryptography.hazmat.backends import default_backend
11
12 max_conn = 3 # Max Connection Queues To Hold
13 buf_size = 4096
14
15 def main():
16     #create socket
17     try:
18         s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.
       ntohs(3))
19         t = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.
       ntohs(3))
20     except Exception as e:
21         sys.exit(2)
22
23     s.bind(("eth1",0))
24     t.bind(("wlan0",0))
25
26     encryptor, decryptor = initialize_enc()
```

```
27      while 1:
28          eth_proto = 0
29          ip_proto = 0
30          payload = 0
31          try:
32              #create another thread to receive data going other direction
33              raw_data,addr=s.recvfrom(65536) #data received from eth0
34              dst_mac, src_mac, eth_proto, ip_data = parse.ethernet_frame(
    raw_data)
35              if eth_proto==8:
36                  version,header_length,ttl,ip_proto,src_ip,dst_ip,data=
    parse.ipv4_packet(ip_data)
37                  if ip_proto == 17: #UDP packet received
38                      src_port,dst_port,length,payload = parse.udp_packet(
    data)
39                  elif ip_proto == 6: #TCP packet received
40                      src_port,dst_port,seq,ack,flag_urg,flag_ack,flag_psh
    ,flag_rst,flag_syn,flag_fin,offset,payload = parse.tcp_packet(data)
41
42              if payload != 0:
43                  if parse.ipv4(src_ip) != "172.16.0.2":
44                      continue
45                  print("Message received: " + str(payload))
46                  new_data = dec_data(payload, decryptor)
47                  print("Message decrypted: " + str(new_data))
48
49                  #MAC address of the gateway router
50                  dst_mac = b"\xc4\x9d\xed\x2d\xac\x83"
51
52                  #construct new headers
53                  checksum = 0
```

```
54                length = 8 + len(new_data) #calculate new length for
    Transport layer header
55                p_header = struct.pack('!2H', ip_proto, length)
56                p_header = src_ip + dst_ip + p_header
57
58                if ip_proto == 17: #new headers
59                    #new IP header
60                    checksum = 0
61                    total_length = header_length + length
62                    i_header = struct.pack('!2s H', ip_data[:2],
    total_length) + ip_data[4:10] + struct.pack('!H', checksum) +
    ip_data[12:20] + ip_data[20:header_length]
63                    checksum = parse.checksum(i_header)
64                    i_header = struct.pack('!2s H', ip_data[:2],
    total_length) + ip_data[4:10] + struct.pack('!H', checksum) +
    ip_data[12:20] + ip_data[20:header_length]
65
66                    #new UDP header
67                    checksum = 0
68                    t_header = struct.pack('!4H', src_port, dst_port,
    length, checksum)
69                    checksum = parse.checksum(p_header + t_header +
    new_data)
70                    t_header = struct.pack('!4H', src_port, dst_port,
    length, checksum)
71                i_offset = 14 #offset to ip layer
72                t.send(dst_mac+raw_data[6:i_offset]+i_header+t_header+
    new_data)
73
74            except KeyboardInterrupt:
75                t.close()
76                s.close()
```

```
77                sys.exit(1)
78        s.close()
79        t.close()
80
81  #initializer for encryption
82  def initialize_enc():
83        backend = default_backend()
84        key = b"11111111111111111111111111111111" #Same key is used for RI
      and IoTAMU
85        iv = b"2222222222222222"
86        cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
87        encryptor = cipher.encryptor()
88        decryptor = cipher.decryptor()
89        return encryptor, decryptor
90
91  #encrypt data
92  def enc_data(data, encryptor):
93        ctext = encryptor.update(data) + encryptor.finalize()
94        return ctext
95
96  #decrypt data
97  def dec_data(data, decryptor):
98        pt = decryptor.update(data) + decryptor.finalize()
99        return pt
100
101 #debug function used to test if encryption is working
102 def debug():
103       encryptor, decryptor = initialize_enc()
104       data = b"12345678901234567890123456789012"
105       encrypted = enc_data(data, encryptor)
106       decrypted = enc_data(encrypted, decryptor)
107       print(data)
```

```
108     print(encrypted)
109     print(decrypted)
110
111 if __name__ == "__main__":
112     main()
113     #debug()
```

Code/proxy_server.py

```
1 #! /user/bin/env python
2 #
3 # Encrypt data received from the IoT devices
4 #
5 # encryption source: https://cryptography.io/en/latest/hazmat/primitives
    /symmetric-encryption/
6
7 import socket, sys, parse, os, binascii, struct
8
9 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
    modes
10 from cryptography.hazmat.backends import default_backend
11
12 def main():
13     #create socket
14     try:
15         s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.
    ntohs(3))
16         t = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.
    ntohs(3))
17     except Exception as e:
18         sys.exit(2)
19
20     s.bind(("eth1",0))
```

```
21      t.bind(("wlan0",0))
22
23      encryptor, decryptor = initialize_enc()
24      while 1:
25          eth_proto = 0
26          ip_proto = 0
27          payload = 0
28          try:
29              raw_data,addr=t.recvfrom(65536) #data received from eth0
30              dst_mac, src_mac, eth_proto, ip_data = parse.ethernet_frame(
    raw_data)
31              if eth_proto==8:
32                  version,header_length,ttl,ip_proto,src_ip,dst_ip,data=
    parse.ipv4_packet(ip_data)
33                  if ip_proto == 17: #UDP packet received
34                      src_port,dst_port,length,payload = parse.udp_packet(
    data)
35                  elif ip_proto == 6: #TCP packet received
36                      src_port,dst_port,seq,ack,flag_urg,flag_ack,flag_psh
    ,flag_rst,flag_syn,flag_fin,offset,payload = parse.tcp_packet(data)
37
38              if payload != 0:
39                  if parse.ipv4(src_ip) != "192.168.1.17":
40                      continue
41                  print("Message received: " + str(payload))
42                  new_data = enc_data(payload, encryptor)
43                  print("Message encrypted: " + str(new_data))
44
45                  #MAC address of the IoT
46                  dst_mac = b"\x80\x2a\xa8\x9e\x45\x5a"
47
48                  #construct new headers
```

```
49                checksum = 0
50                length = 8 + len(new_data) #calculate new length for
    Transport layer header
51                p_header = struct.pack('!2H', ip_proto, length)
52                p_header = src_ip + dst_ip + p_header
53
54                if ip_proto == 17: #new headers
55                    #new IP header
56                    checksum = 0
57                    total_length = header_length + length
58                    i_header = struct.pack('!2s H', ip_data[:2],
    total_length) + ip_data[4:10] + struct.pack('!H', checksum) +
    ip_data[12:20] + ip_data[20:header_length]
59                    checksum = parse.checksum(i_header)
60                    i_header = struct.pack('!2s H', ip_data[:2],
    total_length) + ip_data[4:10] + struct.pack('!H', checksum) +
    ip_data[12:20] + ip_data[20:header_length]
61
62                    #new UDP header
63                    checksum = 0
64                    t_header = struct.pack('!4H', src_port, dst_port,
    length, checksum)
65                    checksum = parse.checksum(p_header + t_header +
    new_data)
66                    t_header = struct.pack('!4H', src_port, dst_port,
    length, checksum)
67                    i_offset = 14 #offset to ip layer
68                    s.send(dst_mac+raw_data[6:i_offset]+i_header+
    t_header+new_data)
69
70            except KeyboardInterrupt:
71                s.close()
```

```
72                t.close()
73                sys.exit(1)
74        s.close()
75        t.close()
76
77    def initialize_enc():
78        backend = default_backend()
79        key = b"11111111111111111111111111111111"#same key used in RI and
        IoTAMU
80        iv = b"2222222222222222"
81        cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
82        encryptor = cipher.encryptor()
83        decryptor = cipher.decryptor()
84        return encryptor, decryptor
85
86    #encrypt data
87    def enc_data(data, encryptor):
88        ctext = encryptor.update(data) + encryptor.finalize()
89        return ctext
90
91    #decrypt data
92    def dec_data(data, decryptor):
93        pt = decryptor.update(data) + decryptor.finalize()
94        return pt
95
96    #debug function
97    def debug():
98        encryptor, decryptor = initialize_enc()
99        data = b"12345678901234567890123456789012"
100       encrypted = enc_data(data, encryptor)
101       decrypted = enc_data(encrypted, decryptor)
102       print(data)
```

```
103         print(encrypted)
104         print(decrypted)
105 
106 if __name__ == "__main__":
107         main()
108         #debug()
```

Code/proxy_server2.py

Appendix D. Spoofer Code

```
1  #! /user/bin/env python
2  #
3  # FLAG: PARAMETER (variables that can be adjusted to better mimic live
       device signatures)
4  #
5  # Spoof network traffic based on AGPS
6  # USEAGE: python3 spoofer.py [AGPS file in csv format]
7
8  import socket, sys, csv, random, time, threading
9  import numpy as np
10 from scapy.all import *
11
12 #list of device names and respective MAC addresses
13 #comment out devices not being used for analysis
14 IoT = {
15           "Camera3": "28:AD:3E:38:6F:B6",
16           "Camera2": "30:8C:FB:3A:1A:AD",
17           "Camera1": "EC:1A:59:E4:FD:41",
18           "CameraTest": "EC:1A:59:E5:02:0D",
19           "Lightbulb2": "B0:4E:26:C5:2A:41",
20           "Lightbulb1": "D0:73:D5:26:B8:4C",
21           "LightbulbTest": "D0:73:D5:26:C9:27",
22           "Switch3": "70:4F:57:F9:E1:B8",
23           "Switch2": "60:38:E0:EE:7C:E5",
24           "Switch1": "14:91:82:CD:DF:3D",
25           "SwitchTest": "B4:75:0E:0D:94:65"
26       }
27
28 IoTAMU = "80:2A:A8:9E:45:5A"
29 STATE = ["active", "passive"]
```

```
30 DIST = {}
31 #encryption can be enabled for payload
32 SA = SecurityAssociation(ESP, spi=0x00000100, crypt_algo='AES-CBC',
       crypt_key=b'iotamukey16bytes')
33
34 # creates the actual packet and sends to network
35 def spoof(repeat, iat, spoof_list, direction, device):
36     #generate payloads of specified sizes
37     global SA
38     payloads = []
39     if direction == 1:
40         eth_dst = IoT[device]
41         eth_src = IoTAMU
42     else:
43         eth_dst = IoTAMU
44         eth_src = IoT[device]
45     ip_src = "1.2.3.4" #bogus IP
46     ip_dst = "1.2.3.4" #bogus IP
47     ttl = 1
48     iface = "wlan0"
49     for data in spoof_list:
50         payloads.append("a"*data)
51     payloads = payloads * repeat
52     for payload in np.random.choice(payloads, len(payloads)):
53         #packet = SA.encrypt(IP(src=ip_src,dst=ip_dst,ttl=ttl)/UDP()/
       payload)
54         packet = IP(src=ip_src,dst=ip_dst,ttl=ttl)/UDP()/payload
55         #sendp(Dot11(addr1=eth_dst,addr2=eth_src,addr3=eth_src,type = 2,
       subtype = 0)/packet,iface=iface, verbose=0)
56         sendp(Ether(src=eth_src,dst=eth_dst)/packet,iface=iface, verbose
       =0)
57             #debug lines
```

```
58              #elapsed_time += time.time() - start
59              #print "%s; ELASPED TIME: %f; SENT PACKET LENGTH: %d\n" % (
    state, elapsed_time, len(payload))
60          time.sleep(iat)
61      #debug lines
62          #elapsed_time += iat
63  #     print "TOTAL DURATION: %f\n" % (elapsed_time)
64
65  def generate_packets(device, seed):
66      #device = name of device
67      #direction = [0 | 1]; 0:outgoing, 1:incoming
68      #seed = seed for random number generators
69      #algorithm:
70      #infinite loop,
71      #1. get state [passive/active]
72      #2. for incoming and outgoing states
73      #3. get interarrival time (IAT), length, duration [1:10 (seconds)]
74      #4. spoof
75      #5. sleep for some time
76      while True:
77          r_random = random.Random(seed)
78          n_random = np.random.RandomState(seed)
79          sub_threads = []
80          #assumed the device is active 10% of the time
81          s = n_random.choice(STATE, 1, p = (0.1, 0.9))[0]
82          if s == "active":
83              offset = 0
84          else:
85              offset = 18
86          #duration of a segment
87          #packets of sizes specified in the list "spoof_list" is spoofed
    during a segment
```

```
88          duration = r_random.randrange(5,15) #PARAMETER
89          #construct packets for outgoing(0) and incoming(1) states
90          for direction in range(0,2):
91              t_offset = offset + (8 * direction)
92              device_dist = DIST[device][t_offset:t_offset+8]
93              iat = float(DIST[device][offset + 16 + direction])
94              iat = r_random.uniform(0.5*iat, 1.5*iat)
95              #choose packet sizes at random in each range
96              packet_200 = r_random.randrange(0,200)
97              packet_400 = r_random.randrange(200,400)
98              packet_600 = r_random.randrange(400,600)
99              packet_800 = r_random.randrange(600,800)
100             packet_1000 = r_random.randrange(800,1000)
101             packet_1200 = r_random.randrange(1000,1200)
102             packet_1400 = r_random.randrange(1200,1400)
103             packet_1600 = r_random.randrange(1400,1441)
104             packet_list = [packet_200, packet_400, packet_600,
    packet_800,
105                            packet_1000, packet_1200, packet_1400,
    packet_1600]
106             #number of repeats of the segment
107             repeat = r_random.randrange(1,10) #PARAMETER
108             n_packets = int(duration/iat)
109             spoof_list = n_random.choice(packet_list, n_packets, p =
    device_dist)
110             device_info = device + s + str(direction)
111             t = threading.Thread(target=spoof, args=(repeat,iat,
    spoof_list, direction, device))
112             sub_threads.append(t)
113         #start subthreads at the same time
114         for t in sub_threads:
115             t.start()
```

```
116         #join threads
117         for t in sub_threads:
118             t.join()
119         seed += 1
120     #sleep for a random duration between 5-10 seconds
121         time.sleep(r_random.randrange(5,10))
122
123 def main():
124     #read in AGPS file
125     in_file = sys.argv[1]
126     with open(in_file, "r") as in_file:
127         f_input = csv.reader(in_file)
128         target_dist = [rows for rows in f_input]
129     devicenames = [name[0] for name in target_dist][1:]
130     for row in target_dist:
131         del row[0]#remove header
132   #dictionary containing the distribution of each sample
133     global DIST
134     DIST = dict(zip(devicenames, target_dist[1:]))
135
136     #rotate random number seeds during loop
137     seed = 42 #PARAMETER
138     main_threads = []
139   #create a thread for each sample being spoofed
140     for device_name in devicenames:
141         m = threading.Thread(target=generate_packets, args=(device_name,
      seed))
142         main_threads.append(m)
143         m.start()
144         seed += 100
145     for m in main_threads:
146         m.join()
```

```
147
148 if __name__ == "__main__":
149     main()
```

Code/spoofer.py

Appendix E. Raw Packet Parsing Code

```
1  # Parse received packet from an interface to extract header information
       and payload
2  # source:
3  # github.com/koehlma/snippets/blob/master/python/network/sniffer.py
4  # github.com/vduddu/Malware/blob/master/Sniffer/Ethernet/
       ethernet_sniffer.py
5
6
7  import socket
8  import struct
9
10 #parse link layer
11 def ethernet_frame(data):
12     dst_mac,src_mac,protocol=struct.unpack('! 6s 6s H',data[:14])
13     return dst_mac, src_mac, socket.htons(protocol), data[14:]
14
15 #retrieve mac address from ethernet frame
16 def get_mac_address(bytes_addr):
17     bytes_str=map('{:02x}'.format, bytes_addr)
18     mac_addr=':'.join(bytes_str).upper()
19     return mac_addr
20
21 #parse IP layer
22 def ipv4_packet(data):
23     version_header_length=data[0]
24     version=version_header_length >> 4
25     header_length = (version_header_length & 15) * 4
26     ttl,protocol,src_ip,dst_ip=struct.unpack('! 8x B B 2x 4s 4s',data
       [:20])
```

```
27      return version,header_length,ttl,protocol,src_ip,dst_ip,data[
        header_length:]
28
29 #retrieve IP address from IP datagram
30 def ipv4(addr):
31      return '.'.join(map(str,addr))
32
33 #parse UDP packet
34 def udp_packet(data):
35      src_port,dst_port,size=struct.unpack('! H H H 2x',data[:8])
36      return src_port,dst_port,size,data[8:]
37
38 #parse TCP packet
39 def tcp_packet(data):
40      src_port,dst_port,seq,ack,flags=struct.unpack('! H H L L H',data
        [:14])
41      offset=(flags >> 12) * 4
42      flag_urg=(flags & 32) >> 5
43      flag_ack=(flags & 16) >> 4
44      flag_psh=(flags & 8) >> 3
45      flag_rst=(flags & 4) >> 2
46      flag_syn=(flags & 2) >> 1
47      flag_fin=(flags & 1)
48      return src_port,dst_port,seq,ack,flag_urg,flag_ack,flag_psh,flag_rst
        ,flag_syn,flag_fin,offset,data[offset:]
49
50 #create checksum from packet
51 def checksum(data):
52      checksum = 0
53      data_len = len(data)
54      if (data_len % 2):
55          data_len += 1
```

```
56          data += struct.pack('!B', 0)
57
58      for i in range(0, data_len, 2):
59          w = (data[i] << 8) + (data[i + 1])
60          checksum += w
61
62      checksum = (checksum >> 16) + (checksum & 0xFFFF)
63      checksum = ~checksum & 0xFFFF
64      return checksum
65
66 #parse contents of received packet and print header information
67 #this is used for debugging purpose
68 def parse(sock):
69      raw_data,addr=sock.recvfrom(65536)
70      dst_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
71      print("\nEthernet Frame:")
72      print("Destination: {}, Source: {}, Protocol: {}".format(
       get_mac_address(dst_mac),get_mac_address(src_mac),eth_proto))
73
74    #check for IP packets
75      if eth_proto==8:
76          version,header_length,ttl,ip_proto,src_ip,dst_ip,data=
       ipv4_packet(data)
77          print("IPv4 Packet:")
78          print("Version: {}, Src IP: {}, Dst IP: {}, Protocol: {}".format
       (version,ipv4(src_ip),ipv4(dst_ip),ip_proto))
79
80      #check for UDP packets
81          if ip_proto == 17:
82              src_port,dst_port,length,data = udp_packet(data)
83              print("UDP Segment:")
```

```
84              print("Source Port: {}, Destination Port {}, Length: {}".
    format(src_port,dst_port, length))
85              print(data)
86
87      #check for TCP packets
88          elif ip_proto == 6:
89              src_port,dst_port,seq,ack,flag_urg,flag_ack,flag_psh,
    flag_rst,flag_syn,flag_fin,offset,data = tcp_packet(data)
90              print("TCP Segment:")
91              print(data)
92
93  def main():
94      s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3)
    )
95      while True:
96          parse(s)
97
98  if __name__ == '__main__':
99      main()
```

Code/parse.py

Appendix F. Code for Proof-of-Concept Experiment Testing Encryption Functionality of IoTAMU

```
#client code used during proof of concept encryption test
#sends a message to the RI and waits for a response

import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

server_address = ("172.16.0.2", 12345)
client_address = ("192.168.1.17", 12345)
sock.bind(client_address)
#send message to RI
message = b"this is a secret message 1234567"
sock.sendto(message, server_address)
print('Client sent: ' + str(message))
#wait for response from the RI
while True:
    data,address = sock.recvfrom(4096)
    print('Client received: ' + str(data))
```

Code/encryption_client.py

```
#server code used during proof of concept encryption test
#decrypts received encrypted message and sends an encrypted response

import socket
import sys
import proxy_server

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
9
10 server_address = ("10.0.0.4", 12345)
11 client_address = ("192.168.1.17", 12345)
12 sock.bind(server_address)
13
14 encryptor, decryptor = proxy_server.initialize_enc()
15 message = "This is a secret response 123456"
16 while True:
17     data,address = sock.recvfrom(4096)
18     if data:
19         print("Received encrypted message: " + str(data))
20     #decrypt encrypted message
21     decrypted = proxy_server.dec_data(data, decryptor)
22     #encrypt response
23         encrypted = proxy_server.enc_data(message, encryptor)
24         print("Message decrypted: " + str(decrypted))
25     print("Sending response . . .")
26     #send encrypted response to client
27         sock.sendto(encrypted, address)
```

Code/encryption_server.py

Appendix G. Preprocessor Code

```
1  import pyshark as ps
2  import binascii
3  import sys
4
5  IoT = {
6      "Camera1": "EC:1A:59:E4:FD:41",
7      "CameraTest": "EC:1A:59:E5:02:0D",
8      "Camera2": "30:8C:FB:3A:1A:AD",
9      "Camera3": "28:AD:3E:38:6F:B6",
10     "Lightbulb1": "D0:73:D5:26:B8:4C",
11     "LightbulbTest": "D0:73:D5:26:C9:27",
12     "Lightbulb2": "B0:4E:26:C5:2A:41",
13     "Switch1": "14:91:82:CD:DF:3D",
14     "SwitchTest": "B4:75:0E:0D:94:65",
15     "Switch2": "60:38:E0:EE:7C:E5",
16     "Switch3": "70:4F:57:F9:E1:B8"
17 }
18 IoTMU = "A4:C4:94:3E:20:9C"
19
20 def main():
21     if len(sys.argv) != 4:
22         print("python iot_table.py [inputfile] [outputfile] [Type]\n")
23         sys.exit()
24     f_input = sys.argv[1]
25     f_output = sys.argv[2]
26     f_type = sys.argv[3]
27
28     cap = ps.FileCapture(f_input, display_filter='frame contains %s &&
       frame contains %s' % (IoT[f_type], IoTMU))
29
```

```
30      f = open(f_output, "w")
31      f.write('No.,Time,Direction,Length,IAT\n')
32      lastOutPacket = 0
33      lastInPacket = 0
34      for packet in cap:
35          #if the destination MAC addr is IoTAMU, then the packet is an
        outgoing packet
36          if "80:2A:A8:9E:45:5A" in packet.wlan.da.upper():
37              direction = "OUTGOING"
38              iat = float(packet.sniff_timestamp) - float(lastOutPacket)
39              lastOutPacket = packet.sniff_timestamp
40          else:
41              direction = "INCOMING"
42              iat = float(packet.sniff_timestamp) - float(lastInPacket)
43              lastInPacket = packet.sniff_timestamp
44          f.write('%s,%s,%s,%s,%s\n' % (packet.number, packet.
        sniff_timestamp, direction, packet.length, '{:.20f}'.format(iat)))
45      f.close()
46
47  if __name__ == '__main__':
48      main()
```

Code/preprocess.py

Appendix H. Classifier Code Used During Initial Setup

```
#script used during pilot study to create network distribution graphs,
#identify active passive states, PCA, and calculate SS
#
#The script takes as input the csv files created after the preprocessing
    step

library(plyr)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggthemes)
library(broom)
library(stats)
library(RColorBrewer)
library(scatterplot3d)
library(DMwR)

setwd("C:\\Users\\Youngjun Park\\Desktop\\IoTMU\\Captures2")

#flag to enable plots (1 to enable)
plots = 0

devices.df <- NULL
devices.stats <- NULL
devicelist = c("Camera1", "CameraTest", "Camera3", "LightBulb1", "
    LightBulbTest", "LightBulb2", "Switch1", "SwitchTest", "Switch3")
for (device in devicelist){
  devicename <- read.csv(paste(device, ".csv", sep = ""))
  devicename$Name <- device
  devicename$Time <- devicename$Time - min(devicename$Time)
```

```
29  if (plots == 1){
30    #create scatter plot of lengths of packets sent vs time
31    p <- ggplot(devicename, aes(x = Time, y = Length)) + geom_point(
      colour = "steelblue")
32    p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
33    p <- p + ggtitle(paste("Lengths of Packets Sent Over Time", device,
      sep=" - "))
34    plot(p)
35    ggsave(file=paste(device, "_time_length.png", sep = trial), height =
       4, width = 6)
36
37    #create histogram of number of packets sent vs time
38    p <- ggplot(devicename, aes(Time)) + geom_histogram(binwidth = 0.5,
      fill = 'steelblue')
39    p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
40    p <- p + ggtitle(paste("Number of Packets Sent Over Time", device,
      sep=" - "))
41    plot(p)
42    ggsave(file=paste(device, "_time_hist.png", sep = trial), height =
      4, width = 6)
43  }
44
45  #determine breaks for histogram
46  break.time <- seq(0, round_any(max(devicename$Time), 0.5, ceiling),
      0.5) ###
47  break.length <- seq(0, 1600, 200)
48
49  # determine active and passive states
50  devicename.hist = hist(devicename$Time, breaks = break.time, plot =
      FALSE)
51  #The outliers in the number of packets sent will represent the active
      state
```

```
52  #For a given continuous variable, outliers are those observations that
       lie outside 1.5*IQR,
53  #   where IQR, the 'Inter Quartile Range' is the difference between 75
     th and 25th quartiles.
54  #extract outliers
55  devicename.times <- devicename.hist$breaks[-1][(devicename.hist$counts
       >= min(boxplot.stats(devicename.hist$counts)$out))]#remove [-1]
56  if (length(devicename.times) == 0){ devicename.times <- devicename.
     hist$breaks[-1] }
57
58  ############################
59  # activation window graph
60  ############################
61  if (plots == 1){
62    state <- rep('#4682b4', length(break.time)-1)
63    state[break.time[-1] %in% devicename.times] <- '#ff3232'
64
65    p <- ggplot(devicename, aes(Time)) + stat_bin(boundary = 0.5,
     binwidth = 0.5, fill = state) ###
66    p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
67    p <- p + ggtitle("Number of Packets Sent Over Time")
68    plot(p)
69    ggsave(file=paste(device, "_active_passive.png", sep = trial),
     height = 4, width = 6)
70  }
71  ###################################
72  # Divide active and passive states
73  ###################################
74  # remove edge cases
75  devicename <- devicename[-1, ]
76  devicename <- devicename[!devicename$IAT > max(devicename$Time),]
77
```

```
78  devicename$State <- "Passive"
79  devicename[round_any(devicename$Time, 0.5, ceiling) %in% devicename.
      times,]$State <- "Active" ### floor
80
81  #remove edge case
82  devicename.active <- devicename[devicename$State == "Active",]
83  devicename.passive <- devicename[devicename$State == "Passive",]
84  ############################
85  # create distribution plots
86  # 1: density of packets of size x for each state
87  # 2: mean incoming IAT
88  # 3: mean outgoing IAT
89  devicename.dist <- data.frame(c(hist(devicename.active[devicename.
      active$Direction == "OUTGOING",]$Length, breaks = break.length, plot
      = FALSE)$counts/(nrow(devicename.active[devicename.active$Direction
      == "OUTGOING",])),
90                                  hist(devicename.active[devicename.
      active$Direction == "INCOMING",]$Length, breaks = break.length, plot
      = FALSE)$counts/(nrow(devicename.active[devicename.active$Direction
      == "INCOMING",])),
91                                  mean(devicename.active$IAT[devicename.
      active$Direction == "OUTGOING"]),
92                                  mean(devicename.active$IAT[devicename.
      active$Direction == "INCOMING"]),
93                                  hist(devicename.passive[devicename.
      passive$Direction == "OUTGOING",]$Length, breaks = break.length,
      plot = FALSE)$counts/(nrow(devicename.passive[devicename.passive$
      Direction == "OUTGOING",])),
94                                  hist(devicename.passive[devicename.
      passive$Direction == "INCOMING",]$Length, breaks = break.length,
      plot = FALSE)$counts/(nrow(devicename.passive[devicename.passive$
      Direction == "INCOMING",])),
```

```
95                                      mean(devicename.passive$IAT[devicename
    .passive$Direction == "OUTGOING"]),
96                                      mean(devicename.passive$IAT[devicename
    .passive$Direction == "INCOMING"])
97                                      ))
98
99  rownames(devicename.dist) <- c(map(hist(devicename.active$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "ActiveOut",
     sep = ""),
100                                     map(hist(devicename.active$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "ActiveIn",
    sep = ""),
101                                     "ActiveOutIAT",
102                                     "ActiveInIAT",
103                                     map(hist(devicename.passive$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "PassiveOut"
    , sep = ""),
104                                     map(hist(devicename.passive$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "PassiveIn",
     sep = ""),
105                                     "PassiveOutIAT",
106                                     "PassiveInIAT"
107                                     ) %>% unlist
108
109 colnames(devicename.dist) <- device
110 devicename.dist[apply(devicename.dist, 1, is.nan),] <- 0
111 if (is.null(devices.df)){
112   devices.df <- devicename.dist
113   devices.stats <- cbind(devicename$Time, devicename$Length, device)
114   devices.hist <- cbind(devicename.hist$counts, device)
115 }
116 else{
```

```
117      devices.df <- cbind(devices.df, devicename.dist)
118      devices.stats <- rbind(devices.stats, cbind(devicename$Time,
         devicename$Length, device))
119      devices.hist <- rbind(devices.hist, cbind(devicename.hist$counts,
         device))
120    }
121 }
122
123 table.out <- signif(devices.df, 4)
124 write.csv(table.out, paste("tableout", ".csv", sep = trial), quote =
       FALSE)
125
126 #remove features with only 0 for PCA
127 devices.filtered.df <- devices.df[apply(devices.df, 1, sum) != 0,]
128 write.csv(t(devices.df), "device_distributions.csv", quote = FALSE)
129
130 #perform PCA
131 pca <- prcomp(t(devices.filtered.df), scale = TRUE)
132
133 ## make a scree plot
134 pca.var <- pca$sdev^2
135 pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
136
137 pca.var.per.df <- as.data.frame(pca.var.per)
138 if (plots == 0)
139 {
140   p <- ggplot(pca.var.per.df, aes(x=as.integer(rownames(pca.var.per.df))
        ,y=pca.var.per)) +
141   geom_bar(stat="identity", fill = "steelblue")+
142   geom_text(aes(label=pca.var.per),position="stack",vjust=-0.5)+
143   ggtitle("Percent Variation from Each Principal Component")+
144   theme_bw() + theme(panel.grid.minor = element_blank())+
```

```
145   labs(y="Percent Variation (%)", x = "Principal Component")+
146   scale_x_continuous(breaks=seq(1,9,1))
147   plot(p)
148   ggsave(file=paste("pca_scree", ".png", sep = ""), height = 5, width =
        7)
149 }
150
151 pca.data <- data.frame(Sample=rownames(pca$x),
152                        Type=devicelist,
153                        X=pca$x[,1],
154                        Y=pca$x[,2],
155                        Z=pca$x[,3])
156
157 ##################
158 #GRAPH 1ST 3 PC #
159 ##################
160 if (plots == 0){
161   shapes = c(17, 17, 16, 17, 17, 16, 17, 17, 16)
162   colors <- c("#E69F00", "#E69F00", "#E69F00", "#56B4E9", "#56B4E9", "
        #56B4E9", "#999999", "#999999", "#999999")
163   graphics.off()
164   png(filename = paste("PCA", ".png", sep = ""), width = 8, height =
        5.9, units = "in", res = 300)
165
166   p <- scatterplot3d(pca.data[3:5],
167                      pch = shapes,
168                      type = "h",
169                      color = colors,
170                      main="First Three Principal Components",
171                      xlab = paste("PC1 - ", pca.var.per[1], "%", sep="")
        ,
```

```
172                         ylab = paste("PC2 - ", pca.var.per[2], "%", sep="")
    ,
173                         zlab = paste("PC3 - ", pca.var.per[3], "%", sep="")
    ,
174                         box = FALSE)
175   legend("bottom", legend = c("Camera", "Light Bulb", "Switch", "Same
    Model", "Different Model"),
176           col =  c("#E69F00", "#56B4E9", "#999999", "#000000", "#000000")
    , pch = c(16, 16, 16, 17 ,16),
177           inset = -0.25, xpd = TRUE, horiz = TRUE)
178   dev.off()
179 }
180
181 #find the number of principal components needed to meet minimum %
    variation
182 #pca.per <- list with % variation for each principal component
183 #value <- minimum threshold to meet (e.g., 0.8 for 80%)
184 find.threshold <- function(pca.per, value){
185   i <- 1
186   while(i <= length(pca.per)){
187     score <- sum(pca.per[1:i])
188     if (score > value){break}
189     i <- i + 1
190   }
191   return(i)
192 }
193 threshold <- find.threshold(pca.var.per, 80)
194 devices.dist <- dist(pca$x[,1:threshold])
195
196 #measure the distance between points -> determine SS
197 #calculate SS
198 dist.out <- 1 - devices.dist/max(devices.dist)
```

```
199 dist.out <- signif(dist.out, 4)
200 write.csv(as.matrix(dist.out), paste("simscore", ".csv", sep = trial),
        quote = FALSE)
201
202 #write PCA result
203 pca.out <- signif(pca$x, 4)
204 write.csv(pca.out, paste("pcascore", ".csv", sep = trial), quote = FALSE
        )
```

Code/Classification_pilot.R

Appendix I. Artificial Packet Signature Generation Code

```
#Generate AGPS from network distribution of devices
#source: https://stats.stackexchange.com/questions/164471/generating-a-simulated-dataset-from-a-correlation-matrix-with-means-and-standard

library(MASS)

#read network signature distribution file
setwd("C:/Users/Youngjun Park/Desktop/IoTMU/Captures3")
distribution <- read.csv("device_distributions3.csv", row.names = 1)
#replace 0 with NA so they are not used in analyses
distribution[c(17,18,35,36)][distribution[c(17,18,35,36)] == 0] <- NA
distribution[c(17,18,35,36)] <- log(distribution[c(17,18,35,36)]) #log scale IAT
distribution.means <- apply(distribution, 2, mean, na.rm = TRUE)
#create covariance matrix from the distribution
distribution.cov <- cov(distribution, use="complete.obs")
#sample 120 times from multivariate normal distribution
noise <- mvrnorm(n = 120, mu = distribution.means, Sigma = distribution.cov) %>% data.frame

#undo log scale
noise[c(17,18,35,36)] <- exp(noise[c(17,18,35,36)])


#apply for every column
#if there is a negative number, add 2x the magnitude of minimum value to the column
rescale.col <- function(dist){
  dist[abs(dist) < 0.0001] <- 0
  if (TRUE %in% (dist < 0)){
```

```
27      dist <- dist + 2*abs((min(dist)))
28    }
29    return(dist)
30  }
31
32  #recalculate percentage for each feature:
33  #new percentage = value/row_total
34  rescale.row <- function(dist){
35    total <- sum(dist[1:8])
36    dist[1:8] <- (dist[1:8])/total
37    total <- sum(dist[9:16])
38    dist[9:16] <- (dist[9:16])/total
39    total <- sum(dist[19:26])
40    dist[19:26] <- (dist[19:26])/total
41    total <- sum(dist[27:34])
42    dist[27:34] <- (dist[27:34])/total
43    return(dist)
44  }
45
46  #apply the functions to the noise data
47  new.noise <- data.frame(apply(noise, 2, rescale.col))
48  new.noise <- data.frame(t(apply(new.noise, 1, rescale.row)))
49  outfile <- cbind(rep(rownames(distribution), 46), new.noise)
50  #write file
51  write.csv(outfile, "target_distributions_120.csv", quote = FALSE, row.
      names = FALSE)
```

Code/noise_generation.R

Appendix J. Timeline of Device Activation

Table 18: Timeline of Device Activation (Run1)

Run1	
Time (min:sec)	**Activity**
0:00	Sniffer started
0:55	Application opened for SwitchTest
1:00	SwitchTest turned on
1:55	Switch3 application opened
2:00	Switch3 turned on
2:55	Camera1 application opened
3:00	Camera1 video feed started
3:15	Camera1 video feed stopped
3:55	LightBulbTest application opened
4:00	LightBulbTest turned on
4:55	CameraTest application opened
5:00	CameraTest video feed started
5:15	CameraTest video feed stopped
5:55	Switch1 application opened
6:00	Switch1 turned on
6:55	LightBulb2 application opened
7:00	LightBulb2 turned on
7:55	Camera3 application opened
8:00	Camera3 video feed started
8:15	Camera3 video feed stopped
8:55	Switch2 application opened
9:00	Switch2 turned on
9:55	Camera2 application opened
10:00	Camera2 video feed started
10:15	Camera2 video feed stopped
10:55	LightBulb1 application opened
11:00	LightBulb1 turned on
12:00	Sniffer stopped
Switch and light bulb device states reset to off	

Table 19: Timeline of Device Activation (Run2)

Run2	
Time (min:sec)	**Activity**
0:00	Sniffer started
0:55	LightBulb1 application opened
1:00	LightBulb 1 turned on
1:55	SwitchTest application opened
2:00	SwitchTest turned on
2:55	Camera2 application opened
3:00	Camera2 video feed started
3:15	Camera2 video feed stopped
3:55	LightBulbTest application opened
4:00	LightBulbTest turned on
4:55	Switch3 application opened
5:00	Switch3 turned on
5:55	Camera3 application opened
6:00	Camera3 video feed started
6:15	Camera3 video feed stopped
6:55	LightBulb2 application opened
7:00	LightBulb2 turned on
7:55	Camera1 application opened
8:00	Camera1 video feed started
8:15	Camera1 video feed stopped
8:55	Switch2 application opened
9:00	Switch2 turned on
9:55	Switch1 application opened
10:00	Switch1 turned on
10:55	CameraTest application opened
11:00	CameraTest video feed started
11:15	CameraTest video feed stopped
12:00	Sniffer stopped
Switch and light bulb device states reset to off	

Table 20: Timeline of Device Activation (Run3)

Run2	
Time (min:sec)	**Activity**
0:00	Sniffer started
0:55	LightBulb2 application opened
1:00	LightBulb2 turned on
1:55	Switch2 application opened
2:00	Switch2 turned on
2:55	Camera3 application opened
3:00	Camera3 video feed started
3:15	Camera3 video feed stopped
3:55	Switch1 application opened
4:00	Switch1 turned on
4:55	CameraTest application opened
5:00	CameraTest video feed started
5:15	CameraTest video feed stopped
5:55	SwitchTest application opened
6:00	SwitchTest turned on
6:55	LightBulb1 application opened
7:00	LightBulb1 turned on
7:55	LightBulbTest application opened
8:00	LightBulbTest turned on
8:55	Camera1 application opened
9:00	Camera1 video feed started
9:15	Camera1 video feed stopped
9:55	Switch3 application opened
10:00	Switch3 turned on
10:55	Camera2 application opened
11:00	Camera2 video feed started
11:15	Camera2 video feed stopped
12:00	Sniffer stopped
Switch and light bulb device states reset to off	

Appendix K. Classifier Code Used During Experiment

```
#script used during the experiment to create network distribution graphs
    ,
#identify active passive states, PCA, and calculate SS
#
#The script takes as input the csv files created after the preprocessing
     step

library(plyr)
library(dplyr)
library(purrr)
library(ggplot2)
library(ggthemes)
library(broom)
library(stats)
library(RColorBrewer)
library(scatterplot3d)
library(DMwR)

setwd("C:\\Users\\Youngjun Park\\Desktop\\IoTMU\\Captures4")

#flag to enable plots
plots = 0

trial = "_COMBINED5"
devices.df <- NULL
devices.stats <- NULL
devicelist = c("Camera1", "CameraTest", "Camera2", "Camera3", "
    Lightbulb1","LightbulbTest", "Lightbulb2", "Switch1", "SwitchTest",
    "Switch2", "Switch3")
for (device in devicelist){
```

```
27  devicename <- read.csv(paste(device, ".csv", sep = "_1")) #run 1
28  devicename$Name <- device
29  devicename$Time <- devicename$Time - min(devicename$Time)
30  tempdevice <- read.csv(paste(device, ".csv", sep = "_2")) #run 2
31  tempdevice$Name <- device
32  tempdevice$Time <- tempdevice$Time - min(tempdevice$Time) +  720 #add
     time offset
33  devicename <- rbind(devicename, tempdevice)
34  tempdevice <- read.csv(paste(device, ".csv", sep = "_3")) #run 3
35  tempdevice$Name <- device
36  tempdevice$Time <- tempdevice$Time - min(tempdevice$Time) +  1440 #add
     time offset
37  devicename <- rbind(devicename, tempdevice)
38
39  if (plots == 1){
40    #create scatter plot of lengths of packets sent vs time
41    p <- ggplot(devicename, aes(x = Time, y = Length)) + geom_point(
     colour = "steelblue")
42    p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
43    p <- p + ggtitle(paste("Lengths of Packets Sent Over Time", device,
     sep=" - "))
44    plot(p)
45    ggsave(file=paste(device, "_time_length.png", sep = trial), height =
     4, width = 6)
46
47    #create histogram of number of packets sent vs time
48    p <- ggplot(devicename, aes(Time)) + geom_histogram(binwidth = 0.5,
     fill = 'steelblue')
49    p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
50    p <- p + ggtitle(paste("Number of Packets Sent Over Time", device,
     sep=" - "))
51    plot(p)
```

```
52      ggsave(file=paste(device, "_time_hist.png", sep = trial), height =
        4, width = 6)
53    }
54
55    #determine breaks for histogram
56    break.time <- seq(0, round_any(max(devicename$Time), 0.5, ceiling),
        0.5) ###
57    break.length <- seq(0, 1600, 200)
58
59    # determine active and passive states
60    devicename.hist = hist(devicename$Time, breaks = break.time, plot =
        FALSE)
61    #The outliers in the number of packets sent will represent the active
        state
62    #For a given continuous variable, outliers are those observations that
        lie outside 1.5*IQR,
63    #   where IQR, the 'Inter Quartile Range' is the difference between 75
        th and 25th quartiles.
64    #extract outliers
65    outliers <- boxplot.stats(devicename.hist$counts)$out
66    outliers <- outliers[outliers > 0]
67    devicename.times <- devicename.hist$breaks[-1][(devicename.hist$counts
        >= min(outliers))]#remove [-1]
68    if (length(devicename.times) == 0){ devicename.times <- devicename.
        hist$breaks[-1] }
69
70    ############################
71    # activation window graph
72    ############################
73    if (plots == 1){
74      state <- rep('#4682b4', length(break.time)-1)
75      state[break.time[-1] %in% devicename.times] <- '#ff3232'
```

```
76      p <- ggplot(devicename, aes(Time)) + stat_bin(boundary = 0.5,
        binwidth = 0.5, fill = state) ###
77      p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
78      p <- p + coord_cartesian(xlim=c(0, 2160), ylim=c(0, 25))
79      p <- p + ggtitle("Number of Packets Sent Over Time")
80      plot(p)
81      ggsave(file=paste(device, "_active_passive.png", sep = trial),
        height = 4, width = 6)
82    }
83    ###################################
84    # Divide active and passive states
85    ###################################
86    # remove edge cases
87    devicename <- devicename[-1, ]
88    devicename <- devicename[!devicename$IAT > max(devicename$Time),]
89
90    devicename$State <- "Passive"
91    devicename[round_any(devicename$Time, 0.5, ceiling) %in% devicename.
        times,]$State <- "Active" ### floor
92
93    #remove edge case
94    devicename.active <- devicename[devicename$State == "Active",]
95    devicename.passive <- devicename[devicename$State == "Passive",]
96    ##########################
97    # create distribution plots
98    # 1: density of packets of size x for each state
99    # 2: mean incoming IAT
100   # 3: mean outgoing IAT
101   devicename.dist <- data.frame(c(hist(devicename.active[devicename.
        active$Direction == "OUTGOING",]$Length, breaks = break.length, plot
        = FALSE)$counts/(nrow(devicename.active[devicename.active$Direction
        == "OUTGOING",]))),
```

```
102                                     hist(devicename.active[devicename.
    active$Direction == "INCOMING",]$Length, breaks = break.length, plot
    = FALSE)$counts/(nrow(devicename.active[devicename.active$Direction
    == "INCOMING",])),
103                                     mean(devicename.active$IAT[devicename.
    active$Direction == "OUTGOING"]),
104                                     mean(devicename.active$IAT[devicename.
    active$Direction == "INCOMING"]),
105                                     hist(devicename.passive[devicename.
    passive$Direction == "OUTGOING",]$Length, breaks = break.length,
    plot = FALSE)$counts/(nrow(devicename.passive[devicename.passive$
    Direction == "OUTGOING",])),
106                                     hist(devicename.passive[devicename.
    passive$Direction == "INCOMING",]$Length, breaks = break.length,
    plot = FALSE)$counts/(nrow(devicename.passive[devicename.passive$
    Direction == "INCOMING",])),
107                                     mean(devicename.passive$IAT[devicename
    .passive$Direction == "OUTGOING"]),
108                                     mean(devicename.passive$IAT[devicename
    .passive$Direction == "INCOMING"])
109   ))
110
111   rownames(devicename.dist) <- c(map(hist(devicename.active$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "ActiveOut",
     sep = ""),
112                                    map(hist(devicename.active$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "ActiveIn",
    sep = ""),
113                                    "ActiveOutIAT",
114                                    "ActiveInIAT",
115                                    map(hist(devicename.passive$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "PassiveOut"
```

```
    , sep = ""),
116                                   map(hist(devicename.passive$Length,
    breaks = break.length, plot = FALSE)$breaks[-1], paste, "PassiveIn",
     sep = ""),
117                                   "PassiveOutIAT",
118                                   "PassiveInIAT"
119   ) %>% unlist
120
121   colnames(devicename.dist) <- device
122   devicename.dist[apply(devicename.dist, 1, is.nan),] <- 0
123   if (is.null(devices.df)){
124     devices.df <- devicename.dist
125     devices.stats <- cbind(devicename$Time, devicename$Length, device)
126     devices.hist <- cbind(devicename.hist$counts, device)
127   }
128   else{
129     devices.df <- cbind(devices.df, devicename.dist)
130     devices.stats <- rbind(devices.stats, cbind(devicename$Time,
    devicename$Length, device))
131     devices.hist <- rbind(devices.hist, cbind(devicename.hist$counts,
    device))
132   }
133 }
134
135 #write distributions
136 if (plots == 1){
137   table.out <- signif(devices.df, 4)
138   write.csv(table.out, paste("tableout", ".csv", sep = trial), quote =
     FALSE)
139 }
140
141 #remove features with only 0 for PCA
```

```
devices.filtered.df <- devices.df[apply(devices.df, 1, sum) != 0,]

#perform PCA
pca <- prcomp(t(devices.filtered.df), scale = TRUE)

## make a scree plot
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)

pca.var.per.df <- as.data.frame(pca.var.per)
if (plots == 1)
{
  p <- ggplot(pca.var.per.df, aes(x=as.integer(rownames(pca.var.per.df))
    ,y=pca.var.per)) +
    geom_bar(stat="identity", fill = "steelblue")+
    geom_text(aes(label=pca.var.per),position="stack",vjust=-0.5)+
    ggtitle("Percent Variation from Each Principal Component")+
    theme_bw() + theme(panel.grid.minor = element_blank())+
    labs(y="Percent Variation(%)", x = "Principal Component")
  plot(p)
  ggsave(file=paste("pca_scree", ".png", sep = trial), height = 5, width
    = 7)
}

pca.data <- data.frame(Sample=rownames(pca$x),
                       Type=devicelist,
                       X=pca$x[,1],
                       Y=pca$x[,2],
                       Z=pca$x[,3])

##################
#GRAPH 1ST 3 PC #
```

```
172 ################
173 if (plots == 1){
174   shapes = c(17, 17, 16, 16, 17, 17, 16, 17, 17, 16, 16)
175   colors <- c("#E69F00", "#E69F00", "#E69F00","#E69F00", "#56B4E9", "#56
      B4E9", "#56B4E9", "#999999", "#999999", "#999999", "#999999")
176   graphics.off()
177   png(filename = paste("PCA", ".png", sep = trial), width = 8, height =
      5.9, units = "in", res = 300)
178
179   p <- scatterplot3d(pca.data[3:5],
180                      pch = shapes,
181                      type = "h",
182                      color = colors,
183                      main="First Three Principal Components",
184                      xlab = paste("PC1 - ", pca.var.per[1], "%", sep="")
      ,
185                      ylab = paste("PC2 - ", pca.var.per[2], "%", sep="")
      ,
186                      zlab = paste("PC3 - ", pca.var.per[3], "%", sep="")
      ,
187                      box = FALSE)
188   legend("bottom", legend = c("Camera", "Light Bulb", "Switch", "Same
      Model", "Different Model"),
189          col =  c("#E69F00", "#56B4E9", "#999999", "#000000", "#000000")
      , pch = c(16, 16, 16, 17 ,16),
190          inset = -0.25, xpd = TRUE, horiz = TRUE)
191   dev.off()
192 }
193
194 #find the number of principal components needed to meet minimum %
      variation
195 #pca.per <- list with % variation for each principal component
```

```r
#value <- minimum threshold to meet (e.g., 0.8 for 80%)
find.threshold <- function(pca.per, value){
  i <- 1
  while(i <= length(pca.per)){
    score <- sum(pca.per[1:i])
    if (score > value){break}
    i <- i + 1
  }
  return(i)
}
threshold <- find.threshold(pca.var.per, 80)
devices.dist <- dist(pca$x[,1:threshold])

#measure the distance between points -> determine SS
#calculate SS
dist.out <- 1 - devices.dist/max(devices.dist)
dist.out <- signif(dist.out, 4)

write.csv(as.matrix(dist.out), paste("simscore", ".csv", sep = trial),
    quote = FALSE)

#write PCA result
pca.out <- signif(pca$x, 4)
write.csv(pca.out, paste("pcascore", ".csv", sep = trial), quote = FALSE
    )
```

Code/Classification.R

Appendix L. Network Congestion Measure Server/Client Code

```
# Server code responding to UDP messages during congestion testing

import socket, sys

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

server_address = ('192.168.1.17', 12346)
client_address = ('172.16.0.2', 12345)
sock.bind(server_address)
#listen for messages from client and respond with same payload
while True:
    data,address = sock.recvfrom(512)
    print(data)
    sock.sendto(data, client_address)
```

Code/congestion_srv.py

```
# Client code to send series of UDP messages during congestion testing
# Measures NL and PD

import socket, sys, time

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

client_address = ('localhost', 12345)
server_address = ("172.16.0.2", 12345)
sock.bind(client_address)
#send 50 UDP messages with payload from "0" to "49"
for x in range(50):
    message = str(x)
```

```
14      start = time.time()
15      sock.sendto(message.encode('utf-8'), server_address)
16      sock.settimeout(2) #set 2 seconds timeout while waiting for a
        response
17      data,address = sock.recvfrom(512)
18      if data == str(x): #ensure that response is for the message sent
19          print(time.time() - start) #print NL time
```

Code/congestion_cli.py

Appendix M. Performance Metric Analysis Code

```
#Calculate performance statistics

library(ggplot2)

#calculate precision
precision <- function(tp, fp){
  return(100 * tp/(tp+fp))
}
#calculate recall
recall <- function(tp, fn){
  return(100 * tp/(tp+fn))
}
#calculate specificity
specificity <- function(tn, fp){
  return(100 * tn/(tn+fp))
}

########################
# EXP 1
########################

#threshold 0.9 results
tp = 3
tn = 52
fp = 0
fn = 0
precision(tp, fp)
recall(tp, fn)
specificity(tn, fp)

```

```
31 #threshold 0.8 results
32 tp = 3
33 tn = 51
34 fp = 1
35 fn = 0
36 precision(tp, fp)
37 recall(tp, fn)
38 specificity(tn, fp)
39
40 #threshold 0.7 results
41 tp = 3
42 tn = 47
43 fp = 5
44 fn = 0
45 precision(tp, fp)
46 recall(tp, fn)
47 specificity(tn, fp)
48
49 ##########################
50 # EXP 2
51 ##########################
52 # 11 samples
53 #threshold 0.9 results
54 tp = 0
55 tn = 51
56 fp = 1
57 fn = 3
58 precision(tp, fp)
59 recall(tp, fn)
60 specificity(tn, fp)
61
62 #threshold 0.8 results
```

```
63 tp = 2
64 tn = 50
65 fp = 2
66 fn = 1
67 precision(tp, fp)
68 recall(tp, fn)
69 specificity(tn, fp)
70
71 #threshold 0.7 results
72 tp = 2
73 tn = 46
74 fp = 6
75 fn = 1
76 precision(tp, fp)
77 recall(tp, fn)
78 specificity(tn, fp)
79
80 # 120 samples
81 #threshold 0.9 results
82 tp = 0
83 tn = 52
84 fp = 0
85 fn = 3
86 precision(tp, fp)
87 recall(tp, fn)
88 specificity(tn, fp)
89
90 #threshold 0.8 results
91 tp = 1
92 tn = 52
93 fp = 0
94 fn = 2
```

```
precision(tp, fp)
recall(tp, fn)
specificity(tn, fp)

#threshold 0.7 results
tp = 2
tn = 52
fp = 0
fn = 1
precision(tp, fp)
recall(tp, fn)
specificity(tn, fp)

# Calculate PSD
#exp1 <- devices.dist #SS matrix from experiment 1
ss1 <- ss(exp1)
#exp2.1 <- devices.dist #SS matrix from experiment 2 with 11 samples
ss2.1 <- ss(exp2.1)
#exp2.2 <- devices.dist #SS matrix from experiment 2 with 120 samples
ss2.2 <- ss(exp2.2)

#create dataframe for analysis
dist.df <- data.frame(rbind(cbind(exp1, 0), cbind(exp2.1, 11), cbind(
    exp2.2, 120)))
colnames(dist.df) <- c("Distance", "Treatment")
dist.df$Treatment <- factor(dist.df$Treatment)
dist.df$Device <- "Other"
dist.df$Device[c(1, 35, 50, 56, 90, 105, 111, 145, 160)] <- c("C1 ~ CT",
     "LB1 ~ LBT", "S1 ~ ST", "C1 ~ CT", "LB1 ~ LBT", "S1 ~ ST", "C1 ~ CT
    ", "LB1 ~ LBT", "S1 ~ ST")
dist.df$Device <- factor(dist.df$Device, levels = c("C1 ~ CT", "LB1 ~
    LBT", "S1 ~ ST", "Other"))
```

```
123
124 #perform ANOVA on PSD for 0, 11, 120 spoofed as treatments
125 dist.aov <- aov(Distance~Treatment, dist.df)
126 dist.aov %>% TukeyHSD %>% tidy
127 #verify QQ plot
128 qqnorm(dist.aov$residuals)
129 qqline(dist.aov$residuals)
130 #verify residuals
131 dist.df$Residuals <- dist.aov$residuals
132 p <- ggplot(dist.df, aes(x = Treatment, y = Residuals)) + geom_point(
        color="steelblue")
133 p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
134 plot(p)
135
136 #calculate mean PSD for each treatment
137 means <- aggregate(Distance~Treatment, dist.df, mean)
138
139 #create boxplot of PSD for each number of samples spoofed
140 p <- ggplot(dist.df, aes(Treatment, Distance)) + geom_boxplot(notch=TRUE
        , colour = "steelblue", outlier.shape = NA) + geom_jitter(width=0.2,
         aes(color = Device)) + coord_flip()
141 p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
142 p <- p + labs(x = "Number of Samples Spoofed", y = "Pairwise Signature
        Distance") + scale_color_manual(values=c("#00FF00", "#E69F00", "#
        ff0000", "#999999"))
143 p <- p + theme(legend.position="bottom")
144 plot(p)
```

Code/performance_stats.R

Appendix N. Calculated Principal Component Scores in Experiments 1 and 2

Table 21: PCS of Each Device (Experiment 1)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
C1	-0.2358	-0.3166	1.962	-0.7974	0.4844	0.4182	-1.129	0.3836	0.4825	-5.523e-05	5.915e-16
CT	-0.3773	0.01338	2.296	-0.6478	0.5868	0.4621	-1.19	0.3832	-0.4564	0.0005816	4.666e-16
C2	7.323	-2.164	-0.8424	-2.424	0.4358	-0.1927	0.3971	-0.09209	-0.01978	1.924e-06	5.759e-16
C3	1.684	-2.289	-0.6821	3.723	-2.545	0.2876	-0.5118	-0.05957	-0.007274	-0.000231	-3.296e-16
LB1	-1.815	-2.109	1.135	1.374	1.362	-0.4779	0.7996	-0.08883	-0.002258	-0.03723	6.609e-16
LBT	-1.808	-2.086	1.108	1.374	1.317	-0.473	0.805	-0.1128	0.005592	0.03749	8.691e-16
LB2	-2.586	-0.2939	-3.564	-0.9611	0.1987	2.758	0.3753	0.2216	-0.01067	-5.45e-06	7.303e-16
S1	-0.8089	2.257	1.713	-1.395	-2.137	-0.6311	1.243	0.6667	-0.001804	1.958e-05	1.475e-16
ST	-0.5543	2.417	1.797	-0.9598	-0.656	0.8351	-0.00733	-1.252	0.01095	-0.0005287	-2.619e-16
S2	2.424	5.162	-1.783	2.504	1.375	-0.5522	-0.07437	0.1588	0.01396	6.546e-05	4.372e-16
S3	-3.246	-0.5915	-3.141	-1.789	-0.4219	-2.434	-0.7077	-0.2089	-0.01473	-0.0001096	6.748e-16
PC: principal component											

Table 22: PCS of Each Device (Experiment 2: 11 Spoofed Samples)

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
C1	-1.602	0.7789	-0.8602	1.402	-1.45	0.4368	-1.323	0.1924	-0.5632	-1.655	-1.037e-15
CT	-2.572	0.0459	-0.6834	0.925	-1.744	0.1966	-1.919	-0.1536	1.151	1.13	-4.372e-16
C2	-4.245	-2.194	0.7138	-2.514	-1.761	-1.322	1.68	0.3788	0.007035	-0.1404	-6.661e-16
C3	-1.897	-0.7473	1.163	-2.801	3.135	1.45	-1.098	-0.2711	0.3396	-0.2715	5.829e-16
LB1	4.563	2.278	-2.732	-3.068	-1.36	0.8026	0.3593	-0.2033	0.0879	0.06841	8.049e-16
LBT	2.089	0.5446	-0.8295	0.3353	1.766	-2.912	-0.6033	1.577	0.4234	-0.101	4.302e-16
LB2	3.75	-6.053	-0.3768	1.337	-0.08834	0.6218	0.1184	-0.2551	-0.09288	0.01884	1.055e-15
S1	-0.9929	0.7215	-0.4738	0.247	0.6634	-0.7885	-0.4007	-1.041	-2.118	0.8045	2.776e-16
ST	-1.289	1.328	-0.387	1.816	0.6928	1.995	1.532	2.012	-0.2447	0.4257	5.829e-16
S2	-0.7836	1.487	-0.9124	1.967	1.164	-0.4386	1.645	-2.078	0.9644	-0.3054	8.292e-16
S3	2.98	1.811	5.379	0.3536	-1.016	-0.04139	0.01081	-0.1578	0.04478	0.02634	1.055e-15
PC: principal component											

Table 23: PCS of Each Device (Experiment 2: 120 Spoofed Samples)

	PC1	**PC2**	**PC3**	**PC4**	**PC5**	**PC6**	**PC7**	**PC8**	**PC9**	**PC10**	**PC11**
C1	-0.1095	3.317	-0.09739	0.85	0.3488	-1.782	1.629	-1.621	0.09608	-1.191	8.57e-16
CT	-1.034	4.103	-1.64	1.937	-1.698	-1.016	-1.744	0.9828	0.9196	0.561	1.632e-17
C2	-4.746	-0.4199	3.015	2.901	-0.492	2.722	0.2309	0.1562	-0.1999	-0.2685	4.645e-16
C3	-5.129	-0.8696	-3.192	-4.246	0.5024	0.572	-0.1495	0.0651	0.2544	-0.2516	3.886e-16
LB1	1.184	-0.7671	2.296	-1.134	1.175	0.03561	1.161	-0.6103	2.045	0.9245	8.704e-16
LBT	4.161	1.095	1.136	-1.29	1.928	1.602	-2.249	0.379	0.07361	-0.8199	-4.817e-16
LB2	3.283	0.4331	0.3204	-2.384	-3.867	1.299	0.9084	-0.2936	-0.68	0.1653	1.055e-15
S1	0.275	0.4703	-1.361	1.027	1.694	0.3133	-0.5271	-1.869	-1.526	1.123	9.506e-16
ST	1.159	0.5888	-0.5428	0.346	1.613	-0.4471	2.017	2.706	-0.8569	0.2044	1.648e-16
S2	-1.285	-2.86	3.234	-0.984	-0.6161	-3.25	-1.241	0.2359	-0.714	-0.03304	1.627e-15
S3	2.241	-5.09	-3.169	2.979	-0.5881	-0.04904	-0.03663	-0.1314	0.5878	-0.4144	9.454e-16
PC: principal component											

Appendix O. Observed Network Fingerprints of Cameras, Light Bulbs and Switches

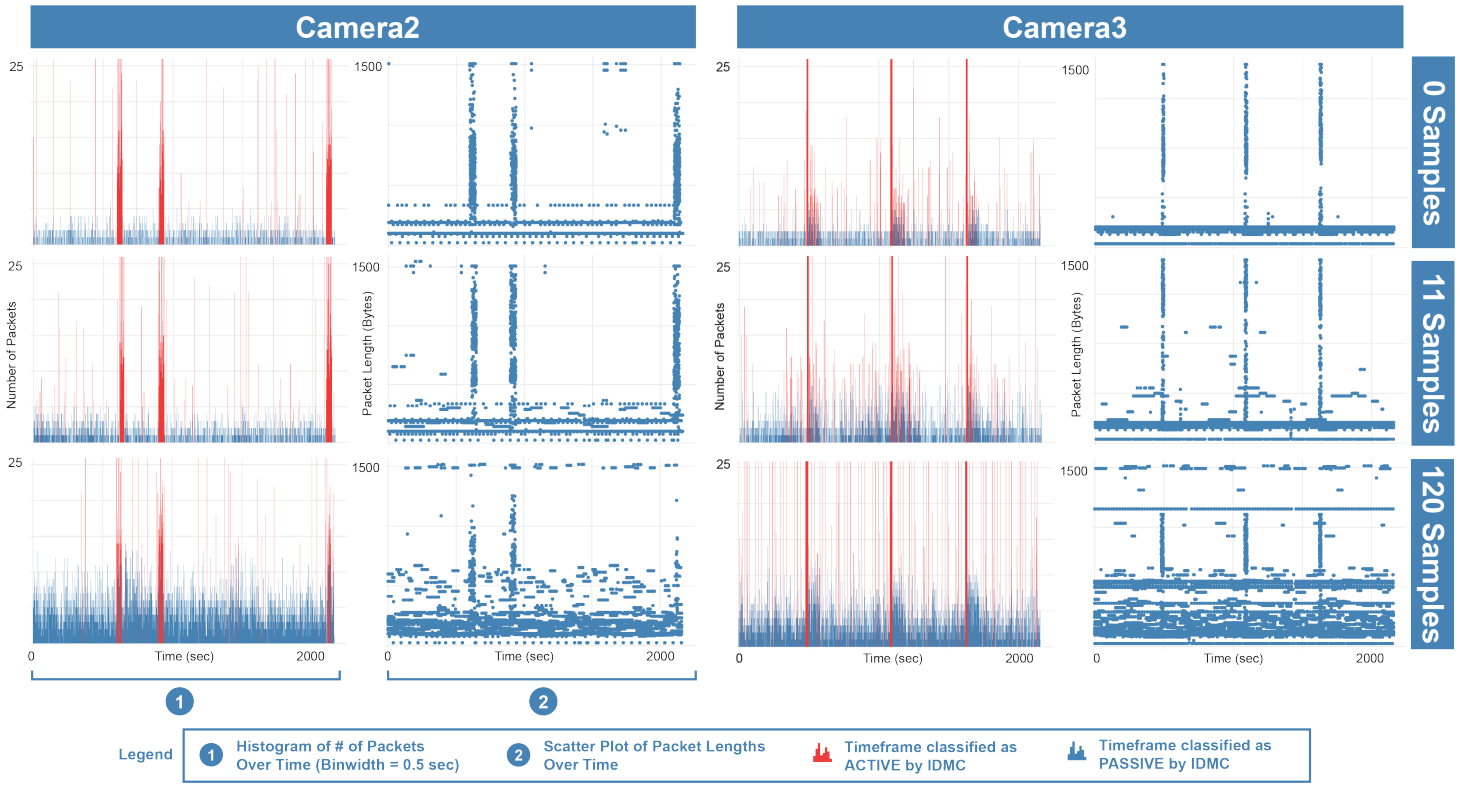


Figure 36: The observed network fingerprints of Camera2 and Camera3 at three different configurations of IoTAMU.

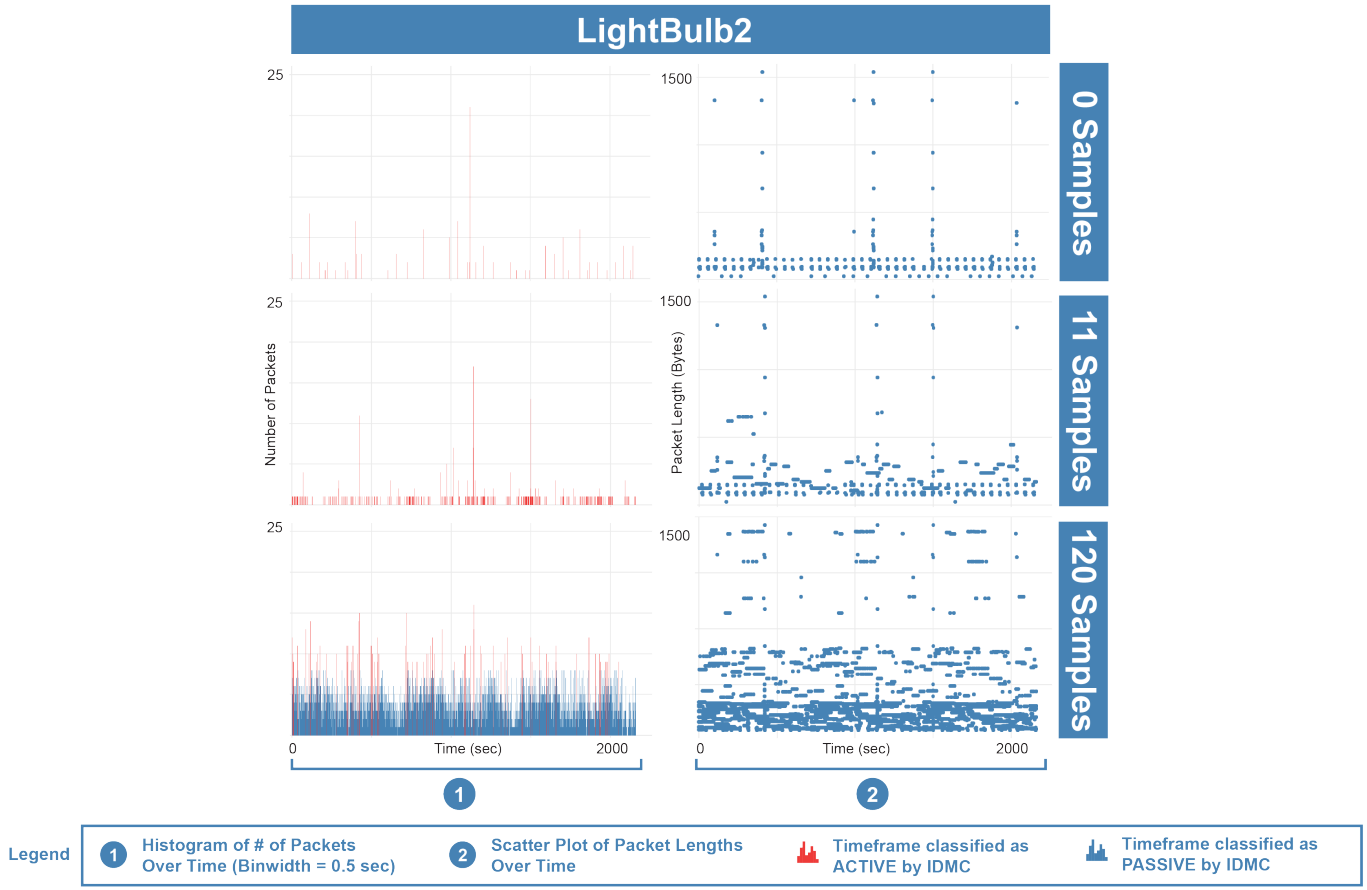


Figure 37: The observed network fingerprints of LightBulb2 at three different configurations of IoTAMU.

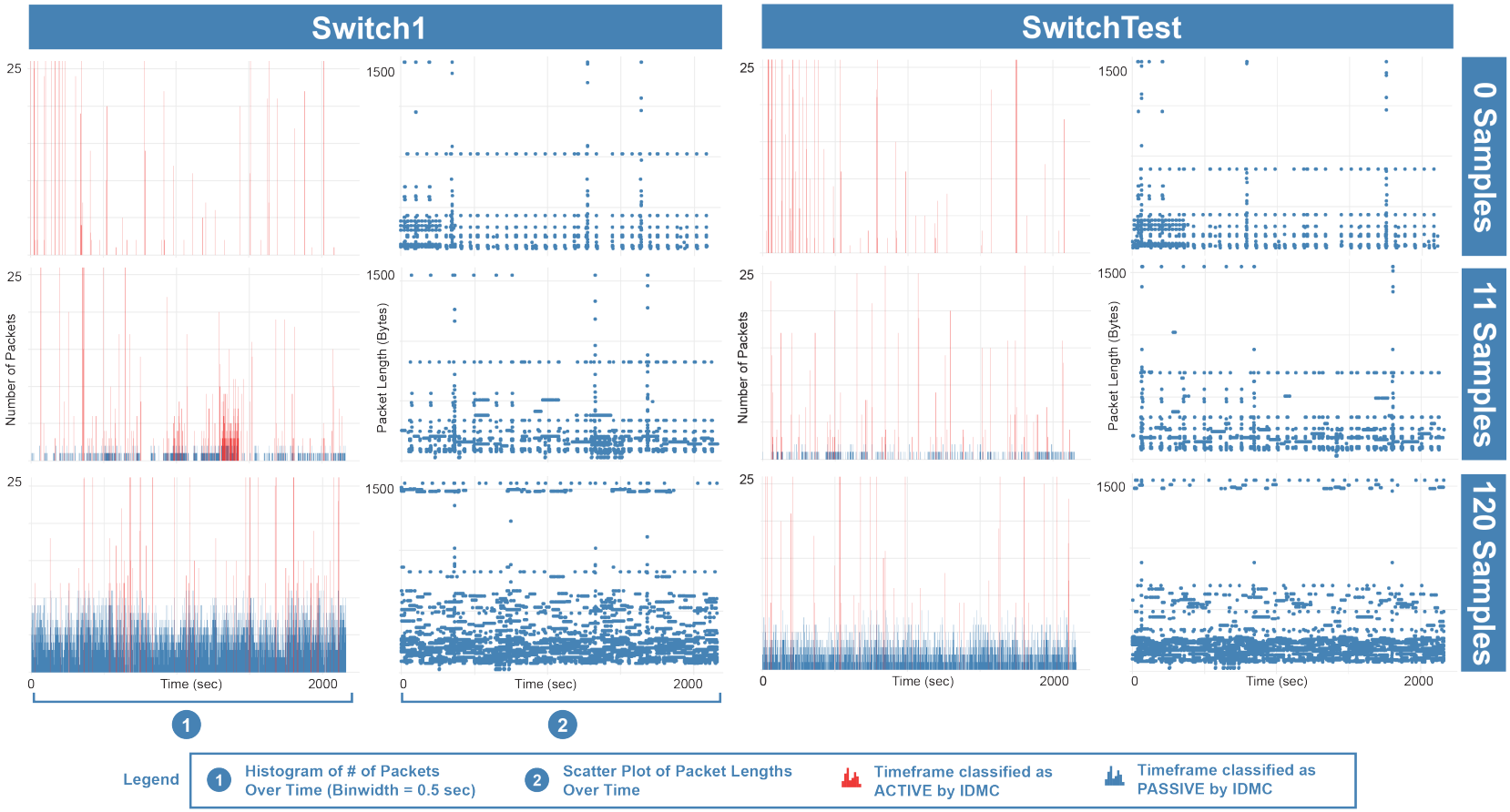


Figure 38: The observed network fingerprints of Switch1 and SwitchTest at three different configurations of IoTAMU.

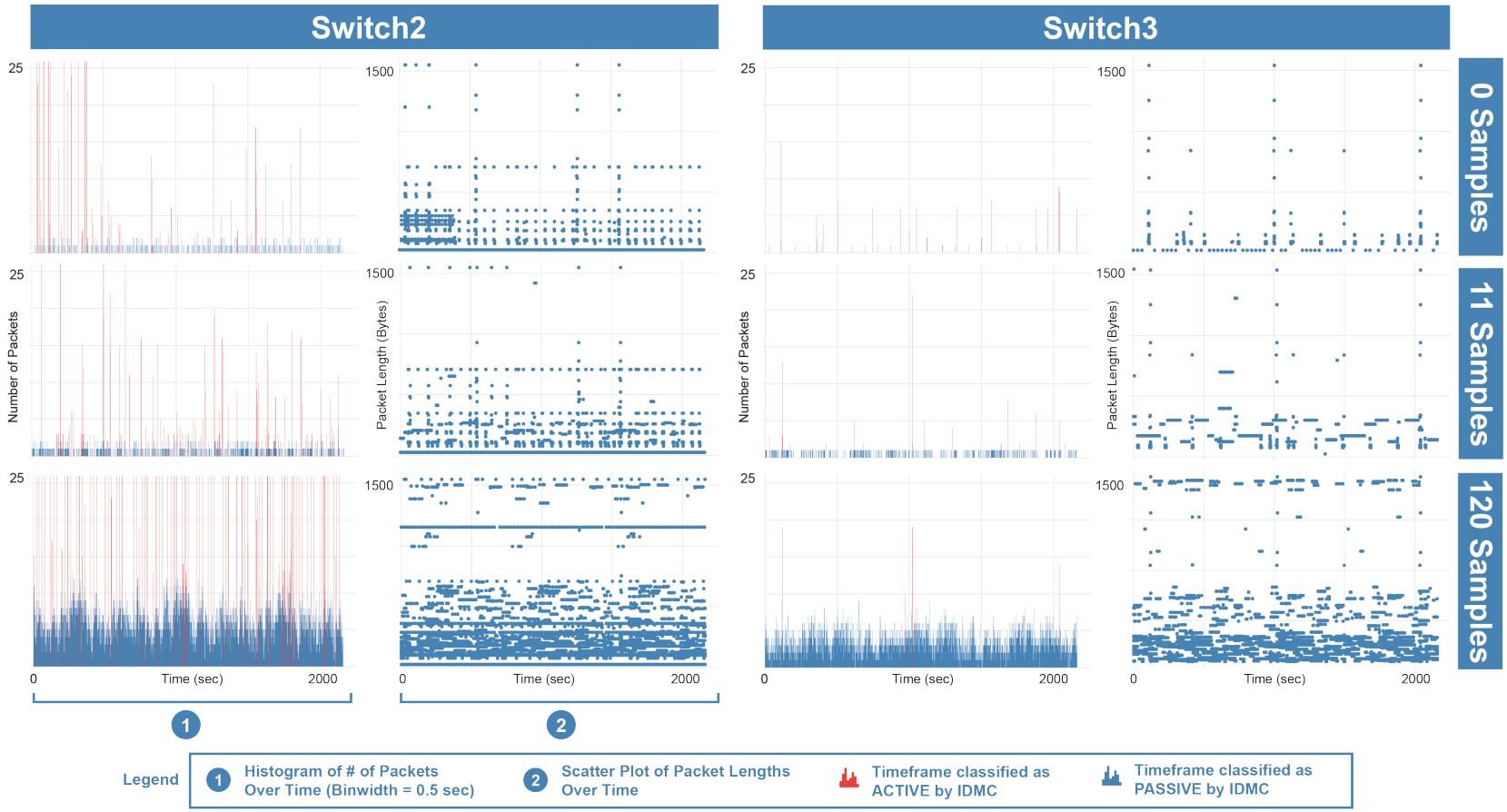


Figure 39: The observed network fingerprints of Switch2 and Switch3 at three different configurations of IoTAMU.

Appendix P. Calculated Pairwise Signature Distances for Experiment 2

Table 24: Summary of PSD for Baseline Network

	C1	CT	C2	C3	LB1	LBT	LB2	S1	ST	S2
CT	0.5125									
C2	8.4300	8.7770								
C3	5.9160	6.1250	8.3440							
LB1	3.3320	3.4650	10.0900	4.5920						
LBT	3.3240	3.4570	10.0800	4.5780	0.03551					
LB2	6.0080	6.2780	10.5500	7.2430	5.6060	5.5770				
S1	2.7150	2.4740	9.6570	7.6690	5.2990	5.2820	6.1400			
ST	2.7610	2.4810	9.5990	7.4310	5.2870	5.2700	6.3420	0.5359		
S2	7.8750	7.8060	10.1400	7.6650	8.9790	8.9490	8.3700	6.8050	6.4200	
S3	6.0140	6.2820	10.9500	7.9760	5.7130	5.6890	1.1790	6.1450	6.4320	9.2480

Table 25: Summary of PSD for Network with 10 Spoofed Samples

	C1	CT	C2	C3	LB1	LBT	LB2	S1	ST	S2
CT	1.3500									
C2	5.8080	4.6470								
C3	6.7230	6.4950	5.6450							
LB1	7.9850	8.7280	10.4800	9.2910						
LBT	5.0160	5.8870	8.4000	5.7650	5.8400					
LB2	8.7990	8.9540	9.8810	9.4830	9.8300	7.1370				
S1	2.5150	3.0410	5.8300	4.5880	7.3130	3.2980	8.3760			
ST	2.3210	3.1800	6.8650	5.8620	8.2900	3.9450	8.9850	1.7110		
S2	2.8860	3.8560	7.5380	6.0950	8.0160	3.4890	8.9250	2.0080	0.8952	
S3	7.8920	8.4590	9.9400	8.6770	8.9640	6.9780	9.8690	7.3530	7.5340	7.8240

Table 26: Summary of PSD for Network with 120 Spoofed Samples

	C1	**CT**	**C2**	**C3**	**LB1**	**LBT**	**LB2**	**S1**	**ST**	**S2**
CT	3.1310									
C2	8.3870	8.5010								
C3	9.1560	9.4600	9.7760							
LB1	5.6570	7.9310	7.8790	8.9660						
LBT	6.5740	8.6110	10.4800	10.9900	4.0900					
LB2	7.5980	8.0430	10.6700	10.3900	6.1910	6.0620				
S1	4.0100	5.3930	7.6980	7.9770	4.5550	5.3680	7.4440			
ST	3.5890	5.6720	8.3390	8.4870	3.5380	4.3660	6.7690	1.5850		
S2	7.5550	9.3270	8.2920	9.3020	5.0380	8.9340	8.5690	7.5320	6.8360	
S3	9.6980	10.0400	10.8000	11.2200	8.3490	9.3680	9.2250	6.8700	7.2290	9.1850

Appendix Q. Network Congestion Analysis Code

```
# Graph the network latency, number of packets, and throughput
# perform ANOVA on network latency times for increasing number of
    samples spoofed

library(ggplot2)
library(dplyr)
library(broom)
library(gtable)
library(grid) # low-level grid functions are required

setwd("C:\\Users\\Youngjun Park\\Desktop\\IoTMU\\Captures5")

#helper function to retrieve packet length information from preprocessed
    csv files
get_info <- function(treatment){
  setwd(paste("C:\\Users\\Youngjun Park\\Desktop\\IoTMU\\congestion2\\",
     treatment, sep=""))
  devices.df <- NULL
  devicelist = c("Camera1", "CameraTest", "Camera2", "Camera3", "
    Lightbulb1","LightbulbTest", "Lightbulb2", "Switch1", "SwitchTest",
    "Switch2", "Switch3")
  #combine data from three trials
  for (device in devicelist){
    devicename <- read.csv(paste(device, ".csv", sep = "_1"))
    tempdevice <- read.csv(paste(device, ".csv", sep = "_2"))
    devicename <- rbind(devicename, tempdevice)
    tempdevice <- read.csv(paste(device, ".csv", sep = "_3"))
    devicename <- rbind(devicename, tempdevice)

    if (is.null(devices.df)){
```

```
26        devices.df <- devicename
27      }
28      else{
29        devices.df <- rbind(devices.df, devicename)
30      }
31    }
32    return(devices.df)
33 }
34 #calculate throughput in kbps
35 t_base <- get_info("Base")
36 t_base <- 8 * sum(t_base$Length)/90/1000
37 t_10 <- get_info("10")
38 t_10 <- 8 * sum(t_10$Length)/90/1000
39 t_20 <- get_info("20")
40 t_20 <- 8 * sum(t_20$Length)/90/1000
41 t_30 <- get_info("30")
42 t_30 <-8 * sum(t_30$Length)/90/1000
43 t_40 <- get_info("40")
44 t_40 <-8 * sum(t_40$Length)/90/1000
45 t_50 <- get_info("50")
46 t_50 <-8 * sum(t_50$Length)/90/1000
47 t_60 <- get_info("60")
48 t_60 <-8 * sum(t_60$Length)/90/1000
49 t_70 <- get_info("70")
50 t_70 <-8 * sum(t_70$Length)/90/1000
51 t_80 <- get_info("80")
52 t_80 <-8 * sum(t_80$Length)/90/1000
53 t_90 <- get_info("90")
54 t_90 <-8 * sum(t_90$Length)/90/1000
55 t_100 <- get_info("100")
56 t_100 <-8 * sum(t_100$Length)/90/1000
57 t_110 <- get_info("110")
```

```
58 t_110 <-8 * sum(t_110$Length)/90/1000
59 t_120 <- get_info("120")
60 t_120 <-8 * sum(t_120$Length)/90/1000
61
62 #read network latency files and create a dataframe with columns network
       latency time, treatment, throughput
63 setwd("C:\\Users\\Youngjun Park\\Desktop\\IoTMU\\congestion2\\latency")
64 s1 <- cbind(c(read.csv("congestion0_0.txt", header = FALSE)[-1,], read.
       csv("congestion0_1.txt", header = FALSE)[-1,], read.csv("congestion0
       _2.txt", header = FALSE)[-1,]), 0, t_base)
65 s2 <- cbind(c(read.csv("congestion10_0.txt", header = FALSE)[-1,], read.
       csv("congestion10_1.txt", header = FALSE)[-1,], read.csv("
       congestion10_2.txt", header = FALSE)[-1,]), 10, t_10)
66 s3 <- cbind(c(read.csv("congestion20_0.txt", header = FALSE)[-1,], read.
       csv("congestion20_1.txt", header = FALSE)[-1,], read.csv("
       congestion20_2.txt", header = FALSE)[-1,]), 20, t_20)
67 s4 <- cbind(c(read.csv("congestion30_0.txt", header = FALSE)[-1,], read.
       csv("congestion30_1.txt", header = FALSE)[-1,], read.csv("
       congestion30_2.txt", header = FALSE)[-1,]), 30, t_30)
68 s5 <- cbind(c(read.csv("congestion40_0.txt", header = FALSE)[-1,], read.
       csv("congestion40_1.txt", header = FALSE)[-1,], read.csv("
       congestion40_2.txt", header = FALSE)[-1,]), 40, t_40)
69 s6 <- cbind(c(read.csv("congestion50_0.txt", header = FALSE)[-1,], read.
       csv("congestion50_1.txt", header = FALSE)[-1,], read.csv("
       congestion50_2.txt", header = FALSE)[-1,]), 50, t_50)
70 s7 <- cbind(c(read.csv("congestion60_0.txt", header = FALSE)[-1,], read.
       csv("congestion60_1.txt", header = FALSE)[-1,], read.csv("
       congestion60_2.txt", header = FALSE)[-1,]), 60, t_60)
71 s8 <- cbind(c(read.csv("congestion70_0.txt", header = FALSE)[-1,], read.
       csv("congestion70_1.txt", header = FALSE)[-1,], read.csv("
       congestion70_2.txt", header = FALSE)[-1,]), 70, t_70)
```

```
72 s9 <- cbind(c(read.csv("congestion80_0.txt", header = FALSE)[-1,], read.
       csv("congestion80_1.txt", header = FALSE)[-1,], read.csv("
       congestion80_2.txt", header = FALSE)[-1,]), 80, t_80)
73 s10 <- cbind(c(read.csv("congestion90_0.txt", header = FALSE)[-1,], read
       .csv("congestion90_1.txt", header = FALSE)[-1,], read.csv("
       congestion90_2.txt", header = FALSE)[-1,]), 90, t_90)
74 s11 <- cbind(c(read.csv("congestion100_0.txt", header = FALSE)[-1,],
       read.csv("congestion100_1.txt", header = FALSE)[-1,], read.csv("
       congestion100_2.txt", header = FALSE)[-1,]), 100, t_100)
75 s12 <- cbind(c(read.csv("congestion110_0.txt", header = FALSE)[-1,],
       read.csv("congestion110_1.txt", header = FALSE)[-1,], read.csv("
       congestion110_2.txt", header = FALSE)[-1,]), 110, t_110)
76 s13 <- cbind(c(read.csv("congestion120_0.txt", header = FALSE)[-1,],
       read.csv("congestion120_1.txt", header = FALSE)[-1,], read.csv("
       congestion120_2.txt", header = FALSE)[-1,]), 120, t_120)
77
78 #change column names for readability
79 colnames(s1) <- c("Time", "Treatment", "Throughput")
80 colnames(s2) <- c("Time", "Treatment", "Throughput")
81 colnames(s3) <- c("Time", "Treatment", "Throughput")
82 colnames(s4) <- c("Time", "Treatment", "Throughput")
83 colnames(s5) <- c("Time", "Treatment", "Throughput")
84 colnames(s6) <- c("Time", "Treatment", "Throughput")
85 colnames(s7) <- c("Time", "Treatment", "Throughput")
86 colnames(s8) <- c("Time", "Treatment", "Throughput")
87 colnames(s9) <- c("Time", "Treatment", "Throughput")
88 colnames(s10) <- c("Time", "Treatment", "Throughput")
89 colnames(s11) <- c("Time", "Treatment", "Throughput")
90 colnames(s12) <- c("Time", "Treatment", "Throughput")
91 colnames(s13) <- c("Time", "Treatment", "Throughput")
92
93 #combine dataframes into one and perform ANOVA
```

```
94 s.all <- data.frame(rbind(s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11,
       s12, s13))
95 s.all$Treatment <- factor(s.all$Treatment)
96 s.aov <- aov(Time ~ Treatment, s.all)
97 summary(s.aov)
98 qqnorm(s.aov$residuals)
99
100 #turn data into log scale
101 s.all$Time <- log(s.all$Time)
102 s.aov <- aov(Time ~ Treatment, s.all)
103 summary(s.aov)
104
105 #verify Q-Q plot and residuals
106 qqnorm(s.aov$residuals)
107 qqline(s.aov$residuals)
108 s.all$Residuals <- s.aov$residuals
109 #residual plot
110 p <- ggplot(s.all, aes(x = Treatment, y = Residuals)) + geom_point(color
       ="steelblue")
111 p <- p + theme_bw() + theme(panel.grid.minor = element_blank())
112 plot(p)
113
114 #create boxplot of network latency for each treatment
115 p1 <- ggplot(s.all, aes(x = Treatment, y = Time)) + geom_boxplot(color="
       steelblue")
116 p1 <- p1 + theme_bw() + theme(panel.grid.minor = element_blank(), axis.
       title.x=element_blank())
117 p1 <- p1 + labs(y = "Network Latency (sec)", x = "")
118
119 #create line graph of number of packets observed for each treatment
120 Packets = c(881, 1457, 1464, 1947, 2038, 2151, 2731, 2848, 2933, 3234,
       3961, 3729, 4181)
```

```
121 Treatment = c(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120)
122 packets.df <- data.frame(cbind(Packets, Treatment))
123 p2 <- ggplot(packets.df, aes(x = Treatment, y = Packets)) + geom_line(
        color="steelblue")
124 p2 <- p2 + theme_bw() + theme(panel.grid.minor = element_blank(), axis.
        title.x=element_blank())
125 p2 <- p2 + scale_x_continuous(breaks=seq(0,120,10))
126 p2 <- p2 + labs(y = "Number of Packets Observed")
127
128 #create line graph of throughput for each treatment
129 Throughput = c(t_base, t_10, t_20, t_30, t_40, t_50, t_60, t_70, t_80, t
        _90, t_100, t_110, t_120)
130 Treatment = c(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120)
131 tp.df <- data.frame(cbind(Throughput, Treatment))
132 p3 <- ggplot(tp.df, aes(x = Treatment, y = Throughput)) + geom_line(
        color="steelblue")
133 p3 <- p3 + theme_bw() + theme(panel.grid.minor = element_blank())
134 p3 <- p3 + scale_x_continuous(breaks=seq(0,120,10))
135 p3 <- p3 + labs(y = "Throughput (kbps)", x = "Number of Samples Spoofed"
        )
136
137 #stack the three plots together
138 g1 <- ggplotGrob(p1)
139 g2 <- ggplotGrob(p2)
140 g3 <- ggplotGrob(p3)
141 g <- rbind(g1, g2, g3, size="first") # stack the three plots
142 g$widths <- unit.pmax(g1$widths, g2$widths, g3$widths) # use the largest
        widths
143 # center the legend vertically
144 g$layout[grepl("guide", g$layout$name),c("t","b")] <- c(1,nrow(g))
145 grid.newpage()
146 grid.draw(g)
```

Code/latency.R

Bibliography

1. "IDC Forecasts Worldwide Spending on the Internet of Things to Reach $745 Billion in 2019, Led by the Manufacturing, Consumer, Transportation, and Utilities Sectors," 2019, accessed Jan 23, 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS44596319

2. Y. Park, M. Reith, and B. Mullins, "Operational Risk Assessment on Internet of Things: Mitigating Inherent Vulnerabilities," in *ECCWS 2019 18th European Conference on Cyber Warfare and Security*. Academic Conferences and publishing limited, 2019, pp. 346–353.

3. S. M. Beyer, B. E. Mullins, S. R. Graham, and J. M. Bindewald, "Pattern-of-Life Modeling in Smart Homes," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5317–5325, 2018.

4. A. Aragon Jr, "Evaluating Machine Learning Techniques for Smart Home Device Classification," Air Force Institute OF Technology, Tech. Rep., 2019, accessed Jan 31, 2020. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/1074013.pdf

5. "Common Vulnerabilities and Exposures," 2020, accessed Jan 23, 2020. [Online]. Available: https://cve.mitre.org/

6. HP, "HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack," 2014, accessed Jan 23, 2020. [Online]. Available: https://www8.hp.com/us/en/hp-news/press-release.html?id=1744676

7. J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. New Jersey: Pearson, 2017.

8. "Device Security," 2019, accessed Jan 23, 2020. [Online]. Available: https://cloud.google.com/iot/docs/concepts/device-security

9. "AWS IoT Device Defender," 2019, accessed Jan 23, 2020. [Online]. Available: https://aws.amazon.com/iot-device-defender/

10. D. V. Dimitrov, "Medical Internet of Things and Big Data in Healthcare," *Healthcare informatics research*, vol. 22, no. 3, pp. 156–163, 2016.

11. A. Sajid, H. Abbas, and K. Saleem, "Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges," *IEEE Access*, vol. 4, pp. 1375–1384, 2016.

12. P. P. Ray, "A Survey on Internet of Things Architectures," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 3, pp. 291–319, 2018. [Online]. Available: https://doi.org/10.1016/j.jksuci.2016.10.003

13. I. . W. Group, "IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems Local and Metropolitan Area Networks—Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std*, vol. 802, no. 11, pp. 1–3534, Dec 2016.

14. Y. Park, R. Dill, and B. Mullins, "IoTAMU: Protecting Smart Home Networks via Obfuscation and Encryption," in *SECURWARE 2019, The Thirteenth International Conference on Emerging Security Information, Systems and Technologies*, 2019, pp. 101–106.

15. E. Skoudis and T. Liston, *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice Hall Press, 2005.

16. A. Sari and M. Karay, "Comparative Analysis of Wireless Security Protocols: WEP vs WPA," *International Journal of Communications, Network and System Sciences*, vol. 08, no. 12, pp. 483–491, Dec 2015.

17. A. Abdelrahman, H. Khaled, E. Shaaban, and W. S. Elkilani, "WPA-WPA2 PSK Cracking Implementation on Parallel Platforms," in *2018 13th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, 2018, pp. 448–453.

18. O. Nakhila and C. Zou, "Parallel Active Dictionary Attack on IEEE 802.11 Enterprise Networks," in *MILCOM 2016 IEEE Military Communications Conference*. IEEE, 2016, pp. 265–270.

19. "Protocol Specifications," 2019, accessed Jan 23, 2020. [Online]. Available: https://www.bluetooth.com/specifications/protocol-specifications/

20. "What is Zigbee?" 2020, accessed Jan 23, 2020. [Online]. Available: https://www.zigbee.org/what-is-zigbee/

21. "About Z-Wave Technology," 2020, accessed Jan 23, 2020. [Online]. Available: https://z-wavealliance.org/about_z-wave_technology/

22. X. Fan, F. Susan, W. Long, and S. Li, "Security Analysis of Zigbee," 2017, accessed Jan 31, 2020. [Online]. Available: https://pdfs.semanticscholar.org/3d1d/5a51d05cde08b6e52afd5bd7bc325b487a10.pdf

23. A. Lonzetta, P. Cope, J. Campbell, B. Mohd, and T. Hayajneh, "Security Vulnerabilities in Bluetooth Technology as Used in IoT," *Journal of Sensor and Actuator Networks*, vol. 7, no. 3, pp. 1–26, 2018.

24. E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Tech. Rep. 8446, Aug 2018, accessed Jan 23, 2020. [Online]. Available: https://tools.ietf.org/html/rfc8446

25. I. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.

26. "OWASP Internet of Things," 2020, accessed Jan 23, 2020. [Online]. Available: https://owasp.org/www-project-internet-of-things/

27. United States Government Accountability Office, "INTERNET OF THINGS Enhanced Assessments and Guidance Are Needed to Address Security Risks in DOD," Tech. Rep. July, 2017, accessed Jan 23, 2020. [Online]. Available: https://www.gao.gov/assets/690/686203.pdf

28. United States Government Accountability Office, "HIGH-RISK SERIES: Urgent Actions Are Needed to Address Cybersecurity Challenges Facing the Nation." Tech. Rep. September, 2018, accessed Jan 23, 2020. [Online]. Available: https://www.gao.gov/assets/700/694355.pdf

29. U.S. Senate. 116th Congress, "S.734 - Internet of Things Cybersecurity Improvement Act of 2019," 2019, accessed Jan 23, 2020. [Online]. Available: https://www.congress.gov/bill/116th-congress/senate-bill/734/

30. C. Heffner, "Exploiting Surveillance Cameras Like a Hollywood Hacker," Tactical Network Solutions, Tech. Rep. February, 2013, accessed Jan 31, 2020. [Online]. Available: https://media.blackhat.com/us-13/US-13-Heffner-Exploiting-Network-Surveillance-Cameras-Like-A-Hollywood-Hacker-WP.pdf

31. J. Ostrom and A. Sambamoorthy, "DEFCON 17: Advancing Video Application Attacks with Video Interception, Recording, and Replay," 2011, accessed Jan 23, 2020. [Online]. Available: https://www.youtube.com/watch?v=QcsQ6UzMJiU

32. M. Stanislav and T. Beardsley, "HACKING IoT: A Case Study on Baby Monitor Exposures and Vulnerabilities," 2015, accessed Jan 23, 2020. [Online]. Available: https://www.rapid7.com/globalassets/external/docs/Hacking-IoT-A-Case-Study-on-Baby-Monitor-Exposures-and-Vulnerabilities.pdf

33. C. Valasek and C. Miller, "Remote Exploitation of an Unaltered Passenger Vehicle," *Black Hat USA*, pp. 1–91, 2015, accessed Jan 23, 2020. [Online]. Available: http://illmatics.com/RemoteCarHacking.pdf

34. "Fiat Chrysler Recalls 1.4 Million Cars After Jeep Hack," 2015, accessed Jan 23, 2020. [Online]. Available: https://www.bbc.com/news/technology-33650491

35. A. Liptak, "Strava's Fitness Tracker Heat Map Reveals the Location of Military Bases," 2018, accessed Jan 23, 2020. [Online]. Available: https://www.theverge.com/2018/1/28/16942626/strava-fitness-tracker-heat-map-military-base-internet-of-things-geolocation

36. J. S. Atkinson, J. E. Mitchell, M. Rio, and G. Matich, "Your WiFi is Leaking: What Do Your Mobile Apps Gossip About You?" *Future Generation Computer Systems*, vol. 80, pp. 546–557, 2018.

37. M. Miettinen, S. Marchal, I. Hafeez, T. Frassetto, N. Asokan, A. R. Sadeghi, and S. Tarkoma, "Iot Sentinel: Automated Device-Type Identification for Security Enforcement in IoT," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

38. S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "HanGuard: SDN-Driven Protection of Smart Home WiFi Devices from Malicious Mobile Apps," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017, pp. 122–133.

39. C. Doukas, I. Maglogiannis, V. Koufi, F. Malamateniou, and G. Vassilacopoulos, "Enabling data protection through PKI encryption in IoT m-Health devices," in *IEEE 12th International Conference on BioInformatics and BioEngineering*, 2012, pp. 25–29.

40. R. H. Hsu, J. Lee, T. Q. S. Quek, and J. C. Chen, "Reconfigurable Security : Edge Computing-based Framework for IoT," *IEEE Network*, vol. 32, no. 5, pp. 92–99, 2018.

41. Z. Huang, W. Du, and B. Chen, "Deriving Private Information From Randomized Data," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 37–48.

42. "Wireshark," 2020, accessed Jan 23, 2020. [Online]. Available: https://www.wireshark.org/

43. "aircrack-ng," 2020, accessed Jan 23, 2020. [Online]. Available: https://www.aircrack-ng.org/

44. "Python," 2020, accessed Jan 23, 2020. [Online]. Available: https://www.python.org/

45. "Scapy," 2019, accessed Jan 23, 2020. [Online]. Available: https://scapy.net/

46. "The R Project for Statistical Computing," 2020, accessed Jan 23, 2020. [Online]. Available: https://www.r-project.org/

47. "The netfilter.org 'iptables' Project," 2014, accessed Jan 23, 2020. [Online]. Available: https://netfilter.org/projects/iptables/index.html

48. "ebtables," 2020, accessed Jan 23, 2020. [Online]. Available: https://ebtables.netfilter.org/

49. "hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator," 2013, accessed Jan 23, 2020. [Online]. Available: https://w1.fi/hostapd/

50. "tlslite-ng," 2020, accessed Jan 23, 2020. [Online]. Available: https://github.com/tomato42/tlslite-ng

51. "Symmetric Encryption," 2017, accessed Jan 23, 2020. [Online]. Available: https://cryptography.io/en/latest/hazmat/primitives/symmetric-encryption/

52. "OUI Lookup Tool," 2020, accessed Jan 23, 2020. [Online]. Available: https://www.wireshark.org/tools/oui-lookup.html

53. C. Neumann, O. Heen, and S. Onno, "An Empirical Study of Passive 802.11 Device Fingerprinting," in *2012 32nd International Conference on Distributed Computing Systems Workshops*. IEEE, 2012, pp. 593–602.

54. Yang Xiao, "IEEE 802.11n: Enhancements for Higher Throughput in Wireless LANs," *IEEE Wireless Communications*, vol. 12, no. 6, pp. 82–91, Dec 2005.

55. Eng Hwee Ong, J. Kneckt, O. Alanen, Z. Chang, T. Huovinen, and T. Nihtilä, "IEEE 802.11ac: Enhancements for Very High Throughput WLANs," in *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, Sep 2011, pp. 849–853.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Sept 2018 — Mar 2020

4. TITLE AND SUBTITLE

DEVELOPMENT AND EVALUATION OF A SECURITY AGENT FOR INTERNET OF THINGS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Park, Youngjun, 2d Lt, USAF

5d. PROJECT NUMBER

19G437

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-20-M-053

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Joseph A. Misher
Department of Homeland Security
Cyber Physical Division, Federal Protective Service
800 North Capitol Street NW, Washington D.C. 20001
COMM: 202-658-8806 EMAIL: Joseph.misher@hq.dhs.gov

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The proposed security agent, Internet of Things Active Management Unit (IoTAMU), provides confidentiality of IoT networks via the following capabilities: (1) authentication, (2) firewall, (3) encryption, and (4) spoofing. To test the spoofer's effect, an Identical Device Model Classifier (IDMC) is developed, which measures the similarities of the observed network signatures of each pair of devices, and recognize identical model devices. The IDMC performs well in baseline network settings without the spoofer, achieving 100% precision, recall, and specificity at high threshold (SS>0.9). When the spoofer is enabled, none of the identical pairs are identified at high threshold, and upto 66% identical pairs are identified at lower thresholds(SS>0.8, 0.7). Overall, the spoofer is able to sufficiently modify the observed network signatures of each device; the observed differences between each pair increase overall (p-value = 0.01132) at 120 spoofed samples, making it more difficult to identify similar devices. Finally, the experiments in this work show the spoofer has a negligible effect on network congestion.

15. SUBJECT TERMS

IoT, Smart Home, Cybersecurity, Wi-Fi, IoTAMU, Data Confidentiality

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Barry E. Mullins, AFIT/ENG
U	U	U	UU	193	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636, ext 7979; barry.mullins@afit.edu