2024

Internet-of-Things Privacy in WiFi Networks: Side-Channel Leakage and Mitigations

Mnassar Alyami
University of Central Florida

INTERNET-OF-THINGS PRIVACY IN WIFI NETWORKS: SIDE-CHANNEL LEAKAGE AND MITIGATIONS

by

MNASSAR ALYAMI
M.S. Kentucky State University, USA, 2017
B.S. Jazan University , Saudi Arabia, 2012

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2024

Major Professor: Yan Solihin

ABSTRACT

WiFi networks are susceptible to statistical traffic analysis attacks. Despite encryption, the metadata of encrypted traffic, such as packet inter-arrival time and size, remains visible. This visibility allows potential eavesdroppers to infer private information in the Internet of Things (IoT) environment. For example, it allows for the identification of sleep monitors and the inference of whether a user is awake or asleep.

WiFi eavesdropping theoretically enables the identification of IoT devices without the need to join the victim's network. This attack scenario is more realistic and much harder to defend against, thus posing a real threat to user privacy. However, researchers have not thoroughly investigated this type of attack due to the noisy nature of wireless channels and the relatively low accuracy of WiFi sniffers.

Furthermore, many countermeasures proposed in the literature are inefficient in addressing side-channel leakage in WiFi networks. They often burden internet traffic with high data overhead and disrupt the user experience by introducing deliberate delays in packet transmission.

This dissertation investigates privacy leakage resulting from WiFi eavesdropping and proposes efficient defensive techniques. We begin by assessing the practical feasibility of IoT device identification in WiFi networks. We demonstrate how an eavesdropper can fingerprint IoT devices by passively monitoring the wireless channel without joining the network. After exploring this privacy attack, we introduce a traffic spoofing-based defense within the WiFi channel to protect against such threats. Additionally, we propose a more data-efficient obfuscation technique to counter traffic analytics based on packet size without adding unnecessary noise to the traffic.

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my advisors, Dr. Yan Solihin and Dr. Cliff Zou, for their guidance, support, and invaluable insights that have been instrumental in shaping this work. Their expertise and encouragement have been a driving force behind the successful completion of this dissertation.

I extend my thanks to the members of my academic committee, Dr. Damla Turgut and Dr. Basem Assiri, for their time, expertise, and commitment to serving on my dissertation committee.

A heartfelt acknowledgment to my parents, brothers, and sister; their unwavering support has been my source of strength during the ups and downs of this academic journey. To my soulmate, Taqwa, and my adorable children, Rahmah and Abdullah, your joyful presence over the past half-decade has transformed my PhD study into a truly joyful undertaking.

I want to remember and honor the memory of my beloved grandparents, Mnassar and Neamah, and my father-in-law, Mohammed. Their love, wisdom, and encouragement have left an indelible mark on my life and academic pursuits. Though they are no longer with us, their influence continues to inspire me.

I express my gratitude to friends and peers who shared their knowledge, provided valuable insights, and offered help during challenging moments. Their involvement has been pivotal to the success of this study.

Finally, I am appreciative of Jazan University for providing me with the necessary funding to pursue my PhD degree.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

IEEE 802.11 Wireless network (WiFi) is increasingly connected to a wide variety of Internet-of-Things (IoT) devices, including smart locks, baby monitors, blood pressure monitors, voice assistants, etc. Hence, the security and privacy of the WiFi network are increasingly targeted by attackers. Examples of security attacks on WiFi-connected IoT devices and applications include the Mirai malware that caused distributed denial-of-service (DDoS) [1] and worms in smart bulbs that allowed attackers to control all nearby compatible IoT lights [1]. Privacy attacks are just as concerning. WiFi traffic analysis allows an observer to "fingerprint" devices to infer private user activities, for example by monitoring a camera's bitrate, the adversary can infer object movements inside a building [2]. The privacy and security concerns are sometimes related. For example, the adversary's ability to identify IoT devices in the network allows him/her to infer what vulnerabilities are present to exploit.

We focus on privacy concern that arises from the ability of the observer to fingerprinting IoT devices in the WiFi network. In particular, while there is abundant work in this area [3–8], they all rely on an assumption that the attacker is a *network insider*, i.e. the attacker has to either join the WiFi network prior to fingerprinting, or be able to wiretap the Internet-side network link of the WiFi network. The assumption, if true, provides a false sense of security, in that one may conclude that as long as the WiFi network uses good password scheme, device fingerprinting is avoided.

In this dissertation, we explore a hypothesis whether an *out-of-network attacker* can effectively fingerprint IoT devices *without* joining a WiFi network. In contrast to in-network attack, there are many challenges for an out-of-network attacker: the attacker cannot see any plaintext of packet payload due to WiFi data-link layer encryption, the captured traffic is noisy (if mixed with neighboring networks), and very limited information (e.g., no IP address and port information), hence it is not immediately clear if the hypothesis is valid. However, if the hypothesis is true, the implications are serious. First, the attack is applicable to all WiFi networks that the attacker has close

proximity to, instead of only networks with breakable password. Second, the attack does not require any special steps to be performed beforehand; the attacker can just walk or drive to close proximity and start analyzing traffic. Third, the attack is not traceable as it does not leave any footprint detectable by users or forensic examiners. Other implications are discussed in Section 3.5.

Many studies have conducted WiFi traffic analytics with a variety of goals [9, 10], where the traffic can be collected outside the network using off-the-shelf WiFi monitoring devices. However, none of these studies take into consideration the missing rate reported with off-the-shelf network sniffers [11]. Therefore, the practical viability of the attack remains uncertain, as it is impossible for the attacker to build a profile based on partial data traffic. Missing high number of frames may have a significant impact on the features extracted from observed traffic, such as average packet inter-arrival times, standard deviation, etc. For instance, consider a device exhibits smooth traffic patterns. If frames are missed during eavesdropping, it can result in the observed traffic appearing bursty.

To explore the feasibility of fingerprinting attack via out-of-network WiFi eavesdropping, we profile 10 real-world IoT devices working in different states, and report their prediction accuracy results and insights. We show that our best technique can fingerprint the devices and their working modes (idle vs. busy) with 95% accuracy on average. The research result was published in IEEE CCNC in 2022 as "WiFi-based IoT devices profiling attack based on eavesdropping of encrypted wifi traffic".

We then provide a strategy to mitigate the discovered privacy leakage in our paper "MAC-layer traffic shaping defense against WiFi device fingerprinting attacks" published in IEEE ISCC in 2022. In this work, we present a confusion-based obfuscation approach that defeats device fingerprinting attacks by making a pair of devices look indistinguishable. This traffic shaping defense is effective against data-link device profiling attacks without adding any Internet-side overhead or time delay in legitimate traffic.

We also focus on mitigating the side-channel leakage from packet size analytics in a broader con-

text, to defend against both internal and external adversaries (i.e., IP and MAC-level observers). We propose a packet size randomization technique, which could achieve high randomization entropy without adding any noise; thus, much more efficient than noise-based solutions. The research result was concluded in our research paper "Random Segmentation: New Traffic Obfuscation against Packet-Size-Based Side-Channel Attacks", which has been published in Electronics 2023 [12].

This dissertation covers three research papers published by the author. In Chapter 2, we examine the relevant literature and previous studies related to traffic analysis of encrypted traffic and countermeasures. Chapter 3 presents our analysis and evaluation of the IoT device identification based on eavesdropping of encrypted WiFi traffic. A data-link layer traffic shaping defense is introduced in Chapter 4. After addressing the risk from WiFi eavesdropping, we then propose a packet size randomization approach to obfuscate the IoT communication with servers in Chapter 5, followed by an extended version that integrates adaptive functionality in Chapter 6. Finally, we conclude all researches and discuss future work in Chapter 7.

CHAPTER 2: LITERATURE REVIEW

In this chapter, we discuss previous work related to our research.

2.1 IoT Devices Fingerprinting Attacks

Device identification is one of many types of information that can be performed using network traffic classification, and has been a topic of interest from the early stage of the Internet. Studies looking at both WiFi and ethernet traffic showed that traffic classification could accurately identify various information, including IoT devices [4–8] and mobile phone app activities [13].

These prior works assumed traffic as seen by an in-network observer that collects the TCP/IP level packets. Such exposure reveals useful and distinguishable network characteristics of the device. For example, the authors in [5] note a single attribute signature for different IoT devices using the destination port number. Additionally, timing features such as packet interarrival time are shown to be adequate to conduct device fingerprinting using deep learning algorithms [8]. These methodologies are not applicable by an out-of-network adversary as the IP traffic in this context is encapsulated to the upper layer, encrypting all the influential network characteristics such as port number, protocol, cipher suites, etc. Different from existing works, we utilize a different set of features that are easily extractable even for an observer that is not part of the WiFi network.

The closest related work to our proposed WiFi profiling attack is the work conducted by Acar et al. [3], which used traffic analysis (WiFi, Zigbee, and Bluetooth) to identify devices, their states, and user activities, and presented traffic spoofing as a defense method. However, they assumed a rogue access point with *tcpdump* to collect WiFi traces. Thus, these studies [3, 5–8] all assume an *in-network* observer, requiring physical access or extensive knowledge to break in and join the victim's encrypted network. In contrast, our assumption is an *out-of-network* adversary, who simply eavesdrops on WiFi traffic without performing elaborate steps to join or break into the network.

Another research domain closely related to our work focuses on hardware fingerprinting and device category classification. Hardware fingerprinting, as discussed in [9], relies on clock skew measurements to authenticate a device, differentiating it from similar devices. However, this research primarily emphasizes hardware aspects rather than device-specific features, making it less suitable for our specific device identification task.

In contrast, other research efforts attempt to classify devices into broad categories [14] such as power management and sensor-based devices. While this categorization is valuable for understanding device functionalities, it lacks the precision needed to distinguish between devices that may share similar functions and traffic patterns. For example, it may struggle to differentiate between devices like Amazon Echo, Google Home, light bulbs, or smart plugs, all of which can exhibit similar functionality and traffic intensity.

Our research, however, enables a more refined and precise profiling of IoT devices. This heightened granularity enhances our ability to identify distinct devices but also raises concerns about potential invasive data collection and privacy breaches. As we achieve more precise device identification, we must remain mindful of the privacy implications that accompany such advancements.

Previous studies have employed passive WiFi monitoring but were designed with specific objectives that didn't necessitate the complete capture of all transmitted data packets. Their focus was primarily on tasks such as identifying channel saturation [15], which can be achieved using beacon frames. These frames are easily captured as they are broadcast to announce the presence of the Access Point (AP). Another example is Rogue AP detection [10] based on signal strength, where capturing every single packet is not crucial because important information is redundantly present in multiple packets. However, when it comes to device identification, the situation is different. This process relies on analyzing data patterns, including packet timing and size. Any missing packets in this context could potentially distort these patterns, resulting in inaccurate identification. In comparison, we run experiments to validate our sniffer's performance (See Section 3.2.1).

Many studies have been proposed to defend against traffic analysis attacks. These countermeasures

largely focus on website fingerprinting [16–18] or achieving location anonymity [19] and hence are ineffective against device fingerprinting. Traffic reshaping techniques such as traffic morphing and padding [20] reduce classifier accuracy. Still, they will fail with time-based classification as both methods are limited to obfuscate traffic patterns based on packet size. Signal Jamming approaches use antennas to disrupt traffic flow at potential adversary locations by increasing the noise ratio. However, it causes interference and degrades nearby networks' performance and is illegal by law [21]. Thus, our attack remains effective against all the defenses proposed in the studies above.

In brief, different from related works, we explore and demonstrate the more realistic privacy attack to WiFi-based devices relying on out-of-network WiFi traffic monitoring. We implement a practical and accurate proof-of-concept attack assuming a realistic threat model. We report a fast detection of 30 seconds and discuss the implications.

2.2 Traffic Shaping Defenses

Several papers introduced countermeasures against various privacy attacks such as internet traffic tracing [19], app identification [22], and website fingerprinting [23]. Are these countermeasures effective against data-link device fingerprinting (DF) attacks? We center our answer on two aspects: the effectiveness of mitigating the privacy leakage from statistical traffic analysis and the feasibility of implementing the defense in terms of time and bandwidth overhead. That leads us to argue that all the prior defenses are ineffective against our attack as some defenses do not extend to our threat model [19, 22] (i.e., they are tailored towards a different scope), others are expensive to implement [23]. We discuss below related work and their limitations to our attack scenario.

Effective defenses, though with high overheads (i.e., >150% bandwidth and/or latency), were proposed in response to newer website fingerprinting attacks [24–26]. Later, Wang et al. [23] developed a practical solution that is well-appreciated by The Onion Router (Tor) community to mitigate the privacy leakage at lower overheads. They modified the proxy and client's browser to maintain

a buffer that holds burst sequences to be molded with each other to form a non-sensitive pattern at little bandwidth instead of injecting additional dummy packets per burst. Still, their approach requires over 30% overhead in both bandwidth and time. Additionally, Tor uses fixed-length packets to deliver information, unlike IoT traffic that varies in length. Observers can perform the attack by solely leveraging packet length statistics, as reported by [27]. Therefore, efficient countermeasures designed for Tor cannot be extended to IoT networks. However, other solutions that distort only packet length features [22] are ineffective either; packet interarrival time features are sufficient to identify IoT devices using deep learning algorithms [8]. Our obfuscation technique, on the other hand, obscures all statistical features in the traffic and does not increase the internet traffic.

As mentioned by Xiong et al. (2022) [28], IoT networks require special tailoring as current countermeasures have tended to focus on shaping web-level traffic rather than device-level. Unlike web servers, IoT devices are resource-constrained. This led the authors in [28] to design a tunable privacy model for IoT network traffic so users can balance the trade-off between their privacy demands and resource requirements. Nevertheless, their defense is limited to obfuscating event-indicating traffic (e.g., a sensor detects motion), so attackers can still fingerprint IoT devices. In this work, we protect against DF attacks, which implies the immunity against the event-level adversary.

Related defenses [3,14] have made efforts to counter the data-link profiling attack. Their proposed approach involves deploying a device that mimics the MAC addresses of other devices, generating fake IoT traffic to confuse potential attackers. However, this defense strategy exhibits certain limitations that need to be considered. One significant limitation of these prior defenses is their inconsistent protection. They are effective only when the genuine IoT device is offline (i.e., not engaged in any network activity), as multiple devices cannot share the same MAC address concurrently. This restriction makes the defense less reliable when IoT devices remain continuously active. Another noteworthy constraint lies in the potential for the attacker to distinguish the spoofed traffic. This can be accomplished by analyzing signal strength, as the legitimate IoT device and the device spoofing the MAC address may exhibit varying signal strengths. Such a discrepancy in signal strength could potentially expose the ruse, undermining the defense's effectiveness.

2.3 Packet-size Obfuscation

Packet padding has acceptable effectiveness but incurs high data overhead. Authors in [29] reported that several packet padding strategies could thwart the attackers' classification but increased the amount of data sent significantly (>500%). A lightweight solution presented by Pinheiro et al. [30] could reduce the accuracy of IoT device identification to higher than random guessing by 15%. Their mechanism inserts random bytes between 1 and the available space to fill the packet (i.e., to equal MTU). Still, the added noisy data (54%) can lead to undesirable communication overhead.

Packet size randomization has previously been proposed to address the side-channel leakage in Secure Shell (SSH) communications [31–33], widely used for secure remote access and communication. However, their proposed modifications are specific to the SSH protocol. Many IoT devices often rely on lightweight messaging protocols to fulfill the IoT communication requirements [34], such as Hypertext Transfer Protocol Secure (HTTPS), Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Extensible Messaging and Presence Protocol (XMPP), and Data Distribution Service (DDS). Our work proposes a novel use of random segmentation at the transport layer, responsible for passing data received from all the application layer communication protocols mentioned above. In this manner, our defense is less demanding for deployment and suitable for the IoT architecture.

The idea of TCP segmentation was previously proposed to reduce the per-packet overhead on host processors for wired networks [35]. This approach delays segmenting the data into smaller units and sends it as a larger TCP segment to improve efficiency. Unlike this work, our technique makes the segmentation random and, thus, unpredictable to evade patterns on side-channel information that can be used to identify IoT devices.

CHAPTER 3: IOT DEVICES FINGERPRINTING ATTACK BASED ON EAVESDROPPING OF ENCRYPTED WIFI TRAFFIC

This chapter delves into the realistic and challenging scenario of an out-of-network traffic eavesdropper, investigating the privacy implications of encrypted WiFi traffic from popular IoT devices[1].

3.1 Threat Model and Assumptions

As shown in Fig. 3.1, we assume an attacker that passively observes out-of-network WiFi traffic of the victim's WiFi router or AP. To achieve that, the attacker must be physically located within the signal range of the AP. The attacker may be *wardriving* (or warcycling, warwalking, etc.), i.e., searching for WiFi networks from a moving vehicle, while using a listen-only sniffing tool that eavesdrops and collects raw traffic from nearby WiFi networks. The attacker does not have an ability to break into the network or join it. Most WiFi-based IoT devices operate at 2.4 GHz frequency and our study focuses on 2.4GHz, but our findings apply to 5GHz as well.

The goal of the attacker is to infer information of devices in a particular WiFi network, including how many unique devices, which type the devices are (e.g., light bulb, smart TV, laptop, etc.),

[1]The contents of this Chapter are based on our publication to IEEE CCNC 2022 [36]

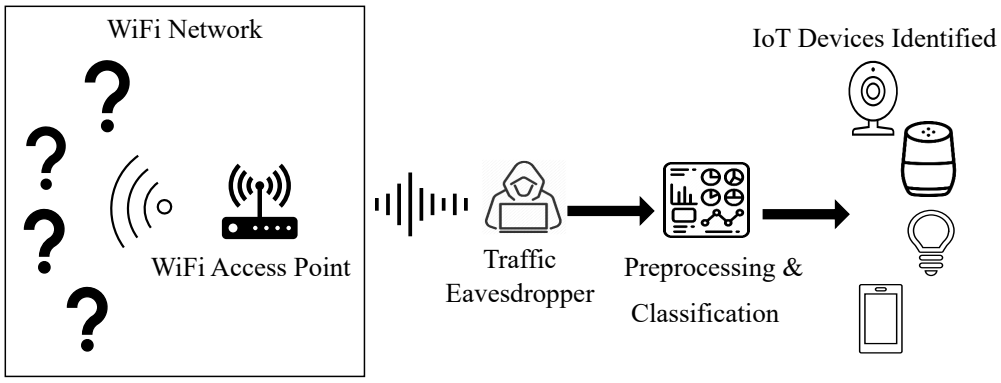


Figure 3.1: Threat Model.

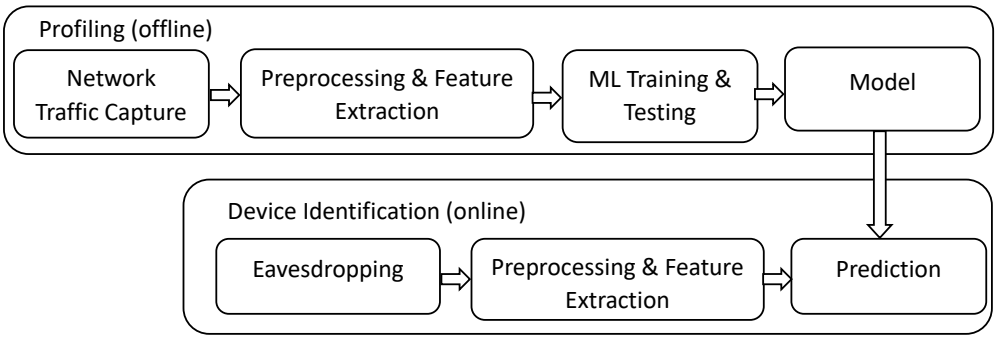


Figure 3.2: IoT device profiling attack system.

and to some extent their operating status (idle or busy). The purpose of the attack is to gather important data that reveals potentially sensitive information. For example, the number of devices may reveal the family size, number of employees or customers in a business, etc. The number and types of devices may reveal socioeconomic status. The type of devices may reveal potential hardware/software vulnerabilities of some IoT devices that could be exploited by the attacker later.

3.2 Capturing of Out-of-Network Encrypted WiFi Traffic

Figure 3.2 shows the system architecture that we assume the attacker uses. It consists of two stages: the training model for profiling (offline phase) and device identification (online phase). In the offline phase, the attacker sets up many IoT devices connected to a WiFi gateway simultaneously; network traffic is collected using sniffing tools and data is labeled for the device name using the MAC address. Then, we pre-process the data, remove noise (e.g., traffic belonging to other WiFi networks, beacon frames and broadcast frames), and extract useful features into a csv file for supervised learning (Section 3.3). Afterward, we train these features through different machine learning (ML) algorithms and retain the highest accuracy model for online inference (i.e., device identification).

In the online phase, the attacker sniffs network traffic of victims' AP for a short time (such as 30 seconds) and stores the trace for pre-processing, like in the offline phase. As we shall explain

in Section 3.2.2, our pre-processing and data cleaning by no means consider prior knowledge of the devices' information. Our methodology applies statistical and standard filtering to clean noise frames that do not represent data patterns. Then, we extract the features out of the pre-processed file using a python script. Finally, we utilize the stored model to predict each device's type and its working activity.

We will now discuss each step in the attack.

3.2.1 Out-of-Network WiFi Traffic Capturing

To capture frames, we first tested out several popular sniffing tools: Airodump-ng[2] and Kismet[3]. Airodump-ng and Kismet sniff raw 802.11 frames. Both are capable single-channel or multi-channel monitoring via frequency hopping between all channels. The main difference between the two is that Airodump-ng dumps the capture into a capture file format (such as pcap[4]), whereas Kismet stores the trace as an SQLite3 database. Airodump-ng's pcap output is in a compatible format to perform packet inspection using a network analyzer like Wireshark[5]. For the hardware, we used an external wireless adapter (Alfa AWUS036ACM) in the monitor mode, as shown in Figure 3.3.a, because most computer built-in WiFi cards are programmed to receive only packets addressed to the machine's interface card or a broadcast.

After analyzing the captured traffic, we observed a significant proportion of captured packets with small sizes, which contradicts the elephant-mouse internet traffic phenomenon [37]. The elephant flows of 1,500 bytes were never seen for all devices, including video packets of smart TV and cameras. Our investigation shows that both tools are limited to catch a limited range of packet sizes up to 472 bytes, which is sufficient for specific signal intelligence applications. For example, authors in [10] suggested rogue AP detection using Kismet by utilizing a few captured packets

[2]http://aircrack-ng.org/doku.php?id=airodump-ng

[3]https://www.kismetwireless.net

[4]https://en.wikipedia.org/wiki/Pcap

[5]https://www.wireshark.org/

(a) Sniffing using Alfa WiFi interface and either Airodump-ng or Kismet software.

(b) Sniffing using MacBook built-in WiFi interface and Airtool software.

Figure 3.3: Two ways of setting up out-of-network capturing.

of any size to identify rogue AP based on the received signal strength because it differs from a legitimate AP.

Due to the limitations of Airodump-ng and Kismet, we evaluate a third tool called Airtool[6]. Airtool is a free WiFi traffic sniffer using Mac's built-in network interface card (NIC) (See Figure 3.3.b). It

[6]https://www.intuitibits.com/products/airtool

Table 3.1: Airtool testing: comparison of passive out-of-network Airtool capture with in-network Wireshark capture.

Packets/Frames Size Range	Wireshark	Airtool	
	#Packets	#Frames	#Data frames
0-19	0	2428	0
20-39	0	5593	199
40-79	2441	0	0
80-159	260	2890	2883
160-319	108	239	239
320-639	173	194	190
640-1279	241	255	255
1280-2559	13574	13846	13846
Total	16797	25445	17612

can be used to passively sniff WiFi traffic and store the traces in a pcap format for further analysis using Wireshark. To verify the completeness of traffic captured by Airtool running on an out-of-network MacBook, we simultaneously run another in-network laptop's Wireshark to record its own incoming/outgoing network traffic to the AP. Then we compared the two traces focusing on the traffic between the second in-network laptop and the AP.

As illustrated in Table 3.1, we found that Airtool collects more frames than packets captured by Wireshark because it captures additional control and management frames at the data-link layer (most frames have sizes between 0 to 39), which do not appear in Wireshark's in-network traffic capturing. Airtool interprets each captured frame as a WiFi data-link layer frame, while in Wireshark, each WiFi frame is interpreted as an Ethernet II frame. Thus, for the same WiFi packet the Wireshark capture has a smaller size than the data-link layer frame interpretation captured by Airtool. This is the reason why 2441 packets in the size range of 40-79 in Wireshark capture all appear in the upper packet size range (80-159) in the Airtool capture. Furthermore, after filtering out all control and management frames in the Airtool capture (shown in the last column), the comparison still holds, and therefore there are no noticeable missing packets by Airtool. Thus, we use Airtool for our evaluation testbed.

3.2.2 Pre-Processing of Captured WiFi Traffic

After the Airtool software captures the encrypted WiFi traffic, the resulting traces in pcap file format are analyzed using Wireshark. More specifically, the raw trace data is pre-processed using the following steps:

1. We extract the bidirectional flows associated with the MAC address of the WiFi router under investigation. This is needed since Airtool could capture WiFi traffic from multiple APs in the neighborhood. Only data frame types are kept because all other control and management MAC-layer frames do not represent the profiling data pattern. To filter out non-data frames,

we use the following display filter in Wireshark:[7]

```
< (wlan.sa = = "Router's MAC" || wlan.da = = "Router's MAC") &&
wlan.fc.type = = 2>
```

2. We export the pcap files as csv for steps 3 and 4.

3. We remove noise frames that were generated by some MACs. These noise frames are easy to filter out since they mostly appeared with a single frame. They are filtered out by keeping only traffic frames that have bi-direction communication traffic (i.e., having both send and receive frames).

4. We replace the MAC addresses with the corresponding device names and their working status to facilitate dataset labeling. This step is only performed in the offline training phase. We skip this step in the online attack phase. The label is used for classification training data and testing verification purposes.

5. We use a Python script to extract and calculate statistical features we will discuss in Section 3.3.5 and finally obtain the needed dataset for both offline training and performance testing.

3.3 Data Processing and Profiling Based on Machine Learning

In this section, we first discuss what data fields we can observe and utilize in out-of-network monitoring. Next, we develop and demonstrate two data processing approaches to generate representative data suitable to feed into machine learning (ML) classification: Time-series and Summary data.

[7](wlan.sa) and (wlan.da) keywords filter by the source and destination MAC address respectively, and (wlan.fc.type = = 2) filters to keep all data type frames by removing WiFi control and management frames.

3.3.1 Observable Data Fields in Out-of-Network Monitoring

For out-of-network monitoring on a secured WiFi network, everything at and above the data-link layer is encrypted by a WiFi protocol (e.g., WPA-PSK). The only observable information is the MAC-layer frame header, plus the frame observation timestamp and signal strength. The MAC-layer frame header has the following useful attributes: source MAC address, destination MAC address, frame type, and frame size.

Initially, we thought that the signal strength might be a good attribute to utilize, which could reflect device hardware property and its distance to the AP (e.g., mobile or stationary). However, from our experiments, we found that the signal strength is affected by many unpredictable factors such as neighboring WiFi networks, and by the reflections, absorption, and deflection of the surrounding objects and rooms, etc. For this reason, we do not consider signal strength in our device profiling.

3.3.2 Preliminary Data Analysis

To provide insights into the IoT traffic, we analyzed a captured traffic seen over 5 minutes for three IoT devices chosen for illustrative purposes: A smart light bulb, a smart plug, and a WiFi-based printer. We changed their working mode in the third minute to monitor their behavioral change. In other words, in the middle of our observation, we turned the light bulb and the plug off, and stopped printing, so the printer would be idle.

The header-based features are either a (a) flow-related or (b) volume-related feature that can be observed within some time window. Flow-based features refer to the frequency and duration of transmission, whereas volume-based features refer to the traffic size in bytes. Although it is limited information, it is sufficient to yield signatures as we can observe unique statistical differences among various types of devices. For example, we can easily notice in Figure 3.4 a clear distinction for the printer in terms of the number of received packets, unlike the bulb and the plug. From a different angle, both the bulb and plug exhibit distinct behavior in terms of packets sizes. Specifically, both devices seem to send most packets at a unique size, 82 bytes for the light bulb and

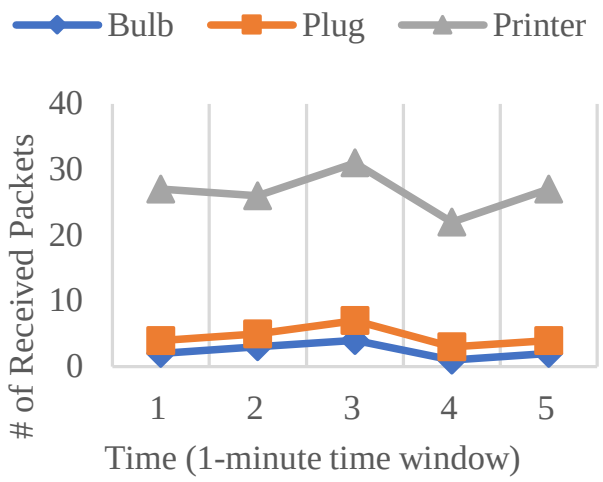


Figure 3.4: Traffic flow of three devices.

92 bytes for the plug. Figure 3.5 shows the word cloud of top packet sizes initiated from the two devices. Hence, we believe the adversary can build signatures with a set of efficient statistics.

The adversary can also infer the event of working status change. We captured a little spike of flow in the data as seen in Figure 3.4 with all devices because the AP initiated a communication to send an off signal to the bulb and plug and to stop the printer. Such behavior is a transient network behavior that can only be observed momentarily from the traffic flow (in part of seconds), not from statistical features based on some time window (+20 seconds). Therefore, it is difficult to infer the operational mode of the devices that are exhibiting a minimal traffic change. On the other hand, the working status of the other devices with higher network capability and memory storage such as Alexa, smart TV, and WiFi cameras, are remarkably observable due to the significant drop in the flow when transitioning to the idle state. For instance, Alexa will receive very few packets when it is idle as it is not receiving voice commands from the user to search the Internet (e.g., checking the weather or deliveries).

3.3.3 Machine Learning Algorithms

We choose several ML models, including Random Forest (RF), Support Vector Machine (SVM), and Naïve Bayes (NB) for learning and inference. RF was reported in [38] to provide superior performance on a network traffic classification problem among 11 ML models. Additionally, we chose SVM and NB due to the two sequence sizes in our time-series dataset. SVM has a good performance on multidimensional datasets as appeared with IoT devices that have intensive traffic. On the other hand, NB excels in relatively small data, suitable for devices that generate an infrequent and small amount of network traffic [39].

In our IoT device classification, we always consider one additional class called 'unknown', which contains all devices that the classifier cannot determine their classes with a predefined confidence.

3.3.4 Device Profiling based on Time-series Data

Time-series data processing is straightforward. We transform the monitored data-link layer trace into a series of three-feature entries. Each monitored data frame is transformed into three numeric values, including the inter-arrival time T (i.e., the time difference between two consecutive packets), direction D (i.e., 0 represents sent and 1 for received packets by an IoT device), and the packet size S. Assume we create a sequence from N frames. Then we can obtain the series $\{T_0, D_0, S_0\}, \{T_1, D_1, S_1\}, \ldots, \{T_N, D_N, S_N\}$.

The main drawback of this approach is that it requires sufficient amounts of packets to create each

(a) Smart light bulb. (b) Smart plug.

Figure 3.5: Word cloud of sent packet sizes from two devices.

data point for machine learning training or classification. Since we deal with a heterogeneous system monitored in a fixed time window, some devices initiate a considerable amount of data communication (such as security cameras), and others generate very few packets (such as smart plugs). For example, if we consider collecting a 100-packet series from the plug and camera, we need over 30 minutes of capturing for the plug while a single second of monitored data is enough for the camera. To balance between the two groups, we adopt a two-level classification strategy starting with a traffic intensity threshold. The first level splits devices into two groups based on traffic density, whether high or low. Then, we accordingly utilize an appropriate sequence size that aligns with the device traffic intensity. The second level calculates the prediction probability using the ML algorithms. If the probability is above a specific threshold, it gives the prediction; otherwise, the instance will be classified as 'unknown' device.

3.3.5 Device Profiling based on Summary Data

In this approach, we profile IoT devices with various traffic features observed over a specific time window. As shown in Figure 3.6, we divide the trace of n seconds into a fixed window size of W seconds, such that we start with a time window $[t_{start} \dots t_{end}]$, and recursively increment both ends by W as long as $t_{end} + W \leq n$. Hence, for n seconds of monitored data, we can obtain $\frac{n}{W}$ data points.

To efficiently generate more sample data points, we use a sliding window $s = \frac{W}{2}$ in which we increment the window $[t_{start} \dots t_{end}]$ by s seconds instead of W seconds, which yields $(\frac{n}{W} \times 2) - 1$ data points. Each device is then identified by its MAC address for each time window W for feature extraction and labeling.

After creating our dataset, we build three classifiers, using the three mentioned ML algorithms (SVM, NB, and RF). In the following, we list all 14 extracted features from monitored packets within each time window:

1. The number of packets sent from the device to AP.

2. The number of packets received by the device from AP.

3. The variance of inter-arrival time.

4. The average number of consecutively sent packets before seeing a received packet.

5. The average number of consecutively received packets before seeing a sent packet.

6. Total number of bytes in sent packets.

7. Total number of bytes in received packets.

8. Number of different sizes in sent packets.

9. Number of different sizes in received packets.

10. Maximum packet size.

11. Mode of sent packet lengths (i.e., the packet size that appeared most in the monitoring window).

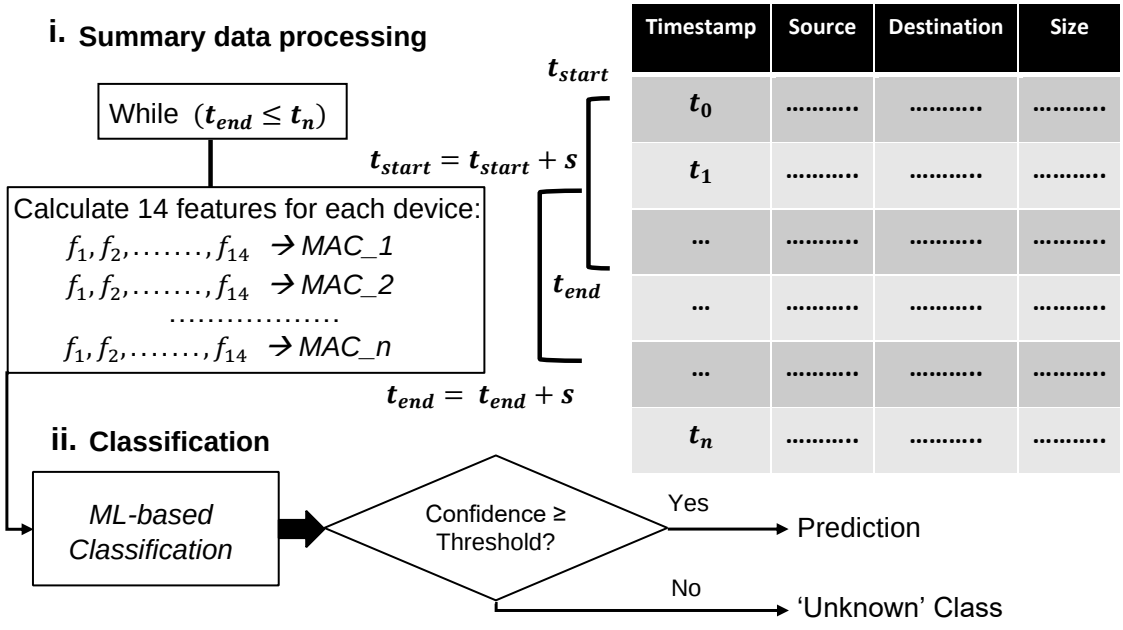


Figure 3.6: Summary data processing and classification.

Table 3.2: IoT dataset capture setup.

<table>
<tr><th>Device</th><th>15 min</th><th>15 min</th><th>30 min</th></tr>
<tr><td>Laptop</td><td colspan="2">Browsing</td><td rowspan="6">Idle</td></tr>
<tr><td>iPhone</td><td colspan="2">Social Media</td></tr>
<tr><td>TV</td><td colspan="2">Internet Television</td></tr>
<tr><td>TV fire stick</td><td colspan="2">Streaming</td></tr>
<tr><td>Amazon Echo</td><td rowspan="2">Receive query/control command regularly</td><td rowspan="2">Play media (e.g., Music).</td></tr>
<tr><td>Google Home</td></tr>
<tr><td>Printer</td><td colspan="2">Printing</td></tr>
<tr><td>Bulb</td><td colspan="2" rowspan="5">ON</td><td rowspan="5">OFF</td></tr>
<tr><td>Plug</td></tr>
<tr><td>Baby Monitor</td></tr>
<tr><td>Doorbell Cam</td></tr>
<tr><td>Camera</td></tr>
</table>

12. Mode of received packet lengths.

13. The variance of sent packet size distribution.

14. The variance of received packet size distribution.

3.4 Evaluation

In this section, we first discuss our testbed to collect the dataset and our evaluation metrics. Then, we present our evaluation results on the testbed trace.

3.4.1 Testbed Setup and Evaluation Metrics

We set up a testbed with 10 different IoT devices and 2 non-IoT (a smartphone and laptop) devices, and allow them to connect to the Internet via a WiFi router. To collect ground truth data, we capture the trace for all devices for a sufficient monitoring time (1 hour). We list in Table 3.2 our devices along with their operational modes for the capture setup of our dataset. These working scenarios are by no means exhaustive of devices that have unlimited functionalities, but they encompass

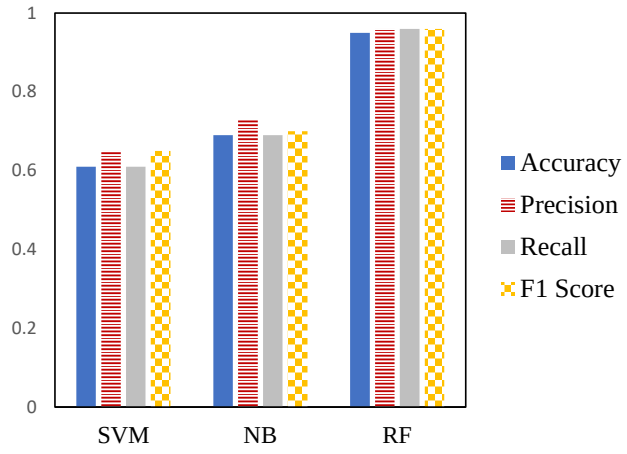


Figure 3.7: Overall performance of summary data-based profiling.

various common usages. Using the captured raw packets, we construct two datasets (time-series and summary data formats as explained in the previous section) and randomly split each dataset's instances into two groups, approximately 75% of the instances for training and 25% for testing.

We evaluate our classification models using the following aspects: Accuracy, Precision, Recall and F1 Score. Let us denote true prediction as T, broken further into true positives TP and true negatives TN. Likewise, false prediction is denoted as F, broken into false positives FP and false negatives FN. Accuracy is measured as $\frac{T}{T+N}$, Precision is measured as $\frac{TP}{TP+FP}$, Recall is measured as $\frac{TP}{TP+FN}$, and F1 is measured as $2 \times \frac{Precision \times Recall}{Precision+Recall}$.

Table 3.3: Comparing the accuracies of ML models using time-series vs. summary data.

		SVM	**NB**	**RF**
Time Series	Non-IoT	0.34	0.25	**0.41**
	IoT	0.57	**0.74**	0.68
Summary Data	Non-IoT	0.51	0.41	**0.94**
	IoT	0.65	0.77	**0.96**

3.4.2 Model Accuracy Results

Table 3.3 shows the accuracies of various ML models with the time-series data vs. summary data. In this testing, we set time window size of 30 seconds. The non-IoT devices include laptop and iPhone in our testing. For all cases, the results show that using the summary data achieves much higher accuracies than using the time-series data. In addition, RF model achieves the best prediction accuracies than the other two ML models.

We think that time-series patterns can be better learned from long-term observed trace more than short-term, which requires significantly longer time observation (e.g., +30 minutes) to perform the attack. However, such a long time of eavesdropping is an unrealistic attack scenario.

As shown in Table 3.3, the summary data profiling yields more accurate results than time-series data, across all models and all types of devices. We think that the summary data approach yields better results because its features are more useful for profiling heterogeneous devices. For instance, a TV sends a regular number of packets per time window, in contrast to Google Home traffic that varies over time. This is not easily captured in the time-series data but is distinguishable from the packet size perspective in the summary data.

Furthermore, the table shows that RF outperforms the other two algorithms in all cases with summary data. We further plot all evaluation metrics on Figure 3.7. The figure confirms RF's superiority: it outperforms SVM and NB in all metrics, and achieves at least 95% in all metrics.

3.4.3 Working Status Detection and Detection Speed

We further investigate if our RF model can detect the status of various devices that have two working modes (busy vs. idle). The accuracy results are presented in Figure 3.8. The figure shows that the classification accuracies are above 90% with a few exceptions: iPhone (idle state) and Amazon Echo (Active and Idle states). The worst accuracy occurs for Amazon Echo, where 28.1% of Amazon Echo in the busy state is incorrectly classified as an Amazon Echo in idle state.

The reason behind this misclassification is that it takes a few seconds to execute some commands (e.g., turning the light on), hence the generated traffic is very small, making the traffic pattern appears similar to the idle state. As explained in Section 3.3.2, we found that some devices like the printer, plug, doorbell, smart bulb, and baby monitor do not have distinct traffic statistics when changing their working states, thus in our experiments, each of these IoT devices has only one model without distinguishing its working state.

We also analyzed the impact of the time window size on the accuracy (See Figure 3.9), by varying it from 20 seconds to 60 seconds. We observed an increase in accuracy by 3% when moving to 30 seconds with RF, but the accuracy is flat from 30 to 60 seconds. Therefore, we adopted the 30-second time window as the default.

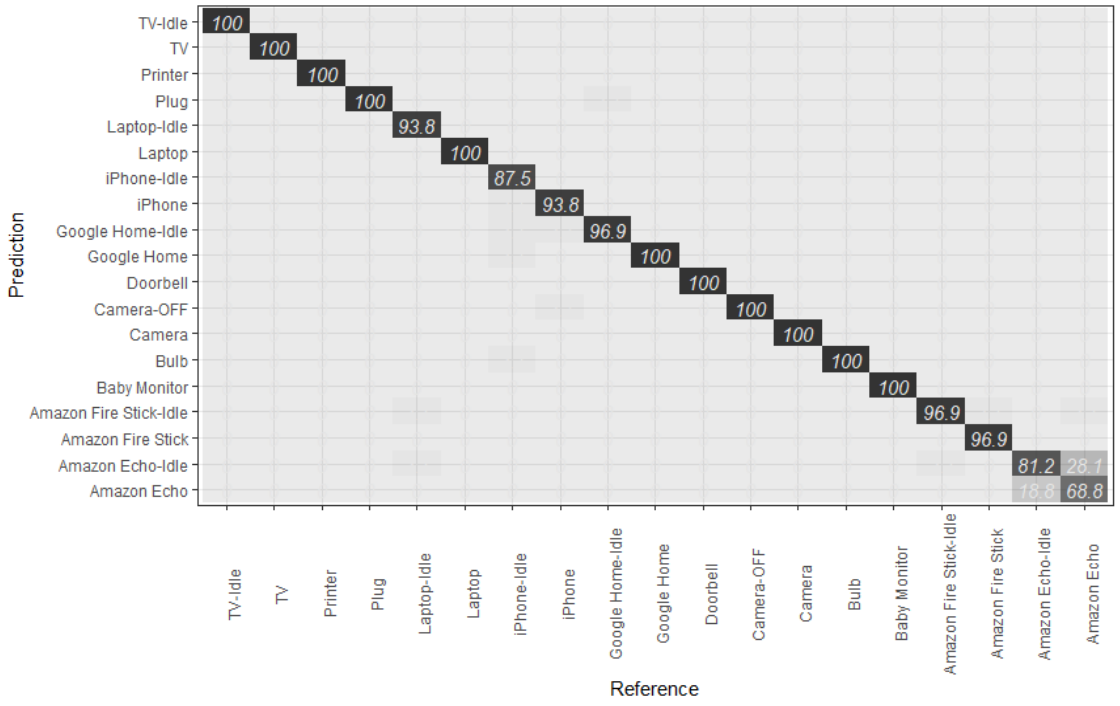


Figure 3.8: Confusion matrix of our IoT devices classification based on summary dataset using RF machine learning algorithm.

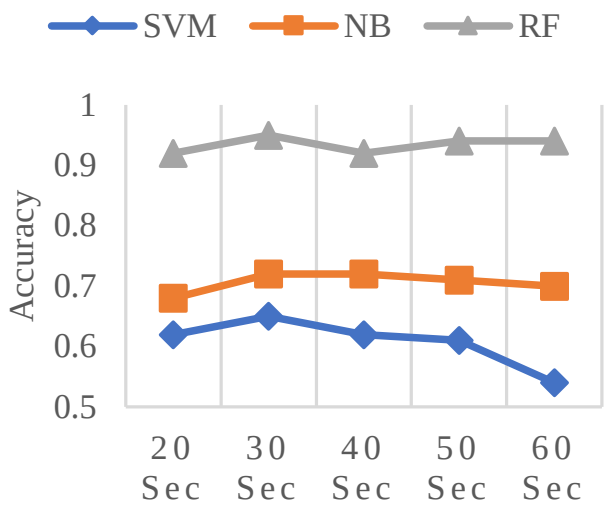


Figure 3.9: Impact of time window size on accuracy.

3.5 Discussion

Our results have validated the hypothesis that an out-of-network attacker can effectively fingerprint IoT devices without joining a WiFi network. Features such as number of packets, inter-arrival time, packet sizes, and their distributions, are sufficient in fingerprinting the types of devices and in most cases, their working modes. The attack is easy to carry out (no need to join or break into a network, and no special equipment is required), does not take a long time to perform (30 seconds provides 95%+ accuracy), and does not leave any detectable footprints.

Implications: The attack raises substantial privacy concerns. First, it enables covert business surveillance. An adversary can drive near the business area to infer the level of economic activity, clients' socioeconomic groups, estimated revenues, revenue trends, or even discover potential vulnerable devices to target in further attacks. In more complex settings, it can provide environmental awareness that can be used to track mobile devices (e.g., cars, drones, phones, etc.). For instance, a swarm of drones can be deployed over a large area to classify and identify signal-emitting devices in the covered region and track their movement and interactions. This may reveal how devices interact with each other, and reveal the geographic movement of each device.

Does out-of-network profiling attack scale? Our proof-of-concept attack results are accurate on our experimental testbed. However, a key question we have not investigated is, can our attack results be extended to the actual open world? We plan further investigation in this direction as future work.

Potential Defenses: A robust defense against device fingerprinting at the *data-link* layer protection is expensive to implement. For example, encrypting MAC header information may conceal device identities hence reduce the attack success probability [40]. However, it does not fully eliminate the attack and the resulting key management and encryption overheads may make it impractical. Other defenses at the *network* and *application* layers are ineffective at the data-link layer [16–18].

Another possible defense is obfuscation via virtual identifiers, e.g., virtual wireless clients (VWC) [41, 42] that generate multiple virtual NICs for each physical NIC. With VWC, a WiFi device's traffic will be dispersed among multiple virtual clients either randomly or using some rules. A large enough number of virtual clients may provide sufficient entropy to prevent an attacker to map different MACs to a single WiFi device for profiling.

CHAPTER 4: MAC-LAYER TRAFFIC SHAPING DEFENSE AGAINST WIFI DEVICE FINGERPRINTING ATTACKS

This chapter introduces a MAC-layer packet injection technique, uniquely confined to the WiFi link between IoT devices and their access points[1].

4.1 Obfuscation via Confusion

Our proposed traffic shaping defense is based on dummy packet injection to make two different devices look very similar. It sends dummy packets in a way that masquerades another device and blends into the original traffic so that the attacker cannot distinguish which of the two devices the observed traffic belongs to. To be specific, let a classifier C that classifies a device D using its traffic T_D. Our defense shapes T_D to $T\prime_D$ to confuse C such that it cannot classify D correctly.

To achieve this, we shape the original traffic of a device x, $T_{D(x)}$, by injecting a *traffic pattern* from another device y, $\mathcal{P}_{D(y)}$, so that we obtain the shaped traffic as:

$$T\prime_{D(x)} = T_{D(x)} + \mathcal{P}_{D(y)}, \tag{4.1}$$

where $\mathcal{P}_{D(y)}$ is a segment of trace recorded from another device y. Likewise, for the device y and its original traffic $T_{D(y)}$, we use device x's traffic pattern to shape it, which means the shaped traffic of device y is:

$$T\prime_{D(y)} = T_{D(y)} + \mathcal{P}_{D(x)}. \tag{4.2}$$

As a result, classifier C cannot distinguish device x from device y because $T\prime_{D(x)} \equiv T\prime_{D(y)}$. This is empirically true as the pattern in a recorded trace $\mathcal{P}_D$ is equivalent to the pattern in the online traffic observed in T_D.

[1]The contents of this Chapter are based on our publication to IEEE ISCC 2022 [43]

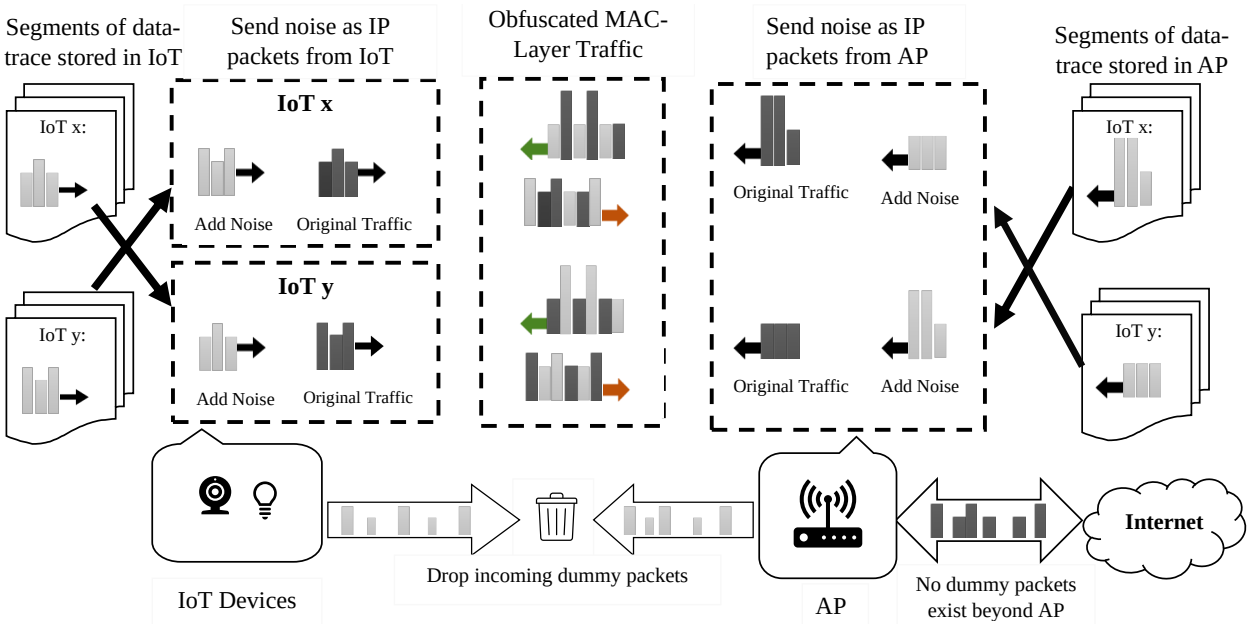


Figure 4.1: WiFi traffic shaping scenario between AP and two IoT devices.

Furthermore, we assume the adversary knows about our defense being performed at the WiFi network; therefore, if we adversarially train $C\prime$ using $T\prime_D$, then $C\prime$ also cannot classify D as $T\prime_{D(x)} \equiv T\prime_{D(y)}$.

In our wireless communication scenario, two objects are involved in packet injection (i.e., AP and a client device). We adopt a one-way injection at each node to obfuscate the bi-directional traffic between the AP and client. Thus, we have a two-way injection in the entire network. In other words, the AP sends noisy traffic to the client, and the client sends noisy traffic to the AP. As a result, the incoming and outgoing traffic is thoroughly obfuscated.

Fig. 5.1 depicts the traffic shaping scenario between the AP and two IoT devices. The system enables the AP and IoT to store packet traces to mask the traffic. On the right hand, the AP injects dummy packets into IoT(x) traffic with sending patterns derived from data traces recorded from IoT(y). The same is performed for the second device (IoT(y)) but using the corresponding packet trace (IoT(x)). On the left hand, each device also uses stored packet traces to mimic the behavior of each other when responding to the AP. All noisy traffic is sent at the network-layer. Hence, all

packets are encrypted at the data-link layer. Therefore, the WiFi traffic is fully obfuscated, and the wireless eavesdropper cannot identify and filter out those dummy packets. Finally, all generated dummy flow is dropped by the AP and IoT devices when they receive them, which means the dummy traffic only exists in the WiFi network, never appears beyond the AP to the Internet. In this way, our proposed approach does not add any bandwidth overhead on the Internet side.

4.1.1 Algorithm

A pseudo-code for the main loop of our traffic shaping defense is shown in Algorithm 1. Both the AP and client run the same loop but using the corresponding data trace (i.e., incoming or outgoing). As mentioned before, the data trace is recorded offline from a device to provide the basis for the mimicry process.

The input of data trace is stored in a sequence $Seq\{\}$, which contains a set of segments S. Each individual segment holds a series of packets $\{Pkt_0^n\}$ captured over a specific time window.

The program, once executed, feeds the injector with a segment of packets S_x. In this manner, the injector uses the two values provided from each packet in the segment (i.e., timestamp and length) to decide each packet's length and insertion schedule (i.e., waiting time).

We randomize S_x using two operations. First, we add a marginal random value $\in \{0.0 \text{ to } 0.09\}$ to each time interval. Second, we run a pseudorandom number generator to get two lists L_{Drop} and $L_{Replace}$ of the same size as S_x. Both list elements $\in [0, 1]$ with a random probability $Prob$ from 0% to 5% to be ones. So, when injecting the dummy packets from S_x, we skip the i^{th} element of S_x when the i^{th} element in $L_{Drop} = 1$. However, if the corresponding element in $L_{Drop} = 0$ and the i^{th} element in $L_{Replace} = 1$, the program will replace the packet from S_x with a random size packet that is less than or equal to the maximum transmission unit (MTU) and send it after a random waiting time, up to 0.09 seconds. Otherwise (i.e., when both elements in L_{Drop} and $L_{Replace} = 0$), the program will inject form S_x, which will be at least 95% of the time. Thus, we slightly randomize the time-based and flow-based features of packet sequences; hence every time

we use the same segment, it looks different. This randomization is necessary to prevent the attacker from conducting frequency analysis to identify dummy packets (i.e., using the same segment at a fixed pattern).

Algorithm 1 Mimicry-Based Traffic Injection

Data: $Seq\{\} \leftarrow$ multiple segments S
while *True* **do**
  $DummyPkt \leftarrow$ Source, Destination, flag = 'Dummy'
  $S_x \leftarrow$ Randomly select a segment from Seq
  $Volume \leftarrow$ length of S_x
  $Prob \leftarrow$ random (low = 0, high = 5)
  L_{Drop} []$\leftarrow$ random (value $\in \{0, 1\}$, size = $Volume$, probability = ($Prob$, value = 1)).
  $L_{Replace}$ []$\leftarrow$ random (value $\in \{0, 1\}$, size = $Volume$, probability = ($Prob$, value = 1)).
  for $i \leftarrow 1$ *to* $Volume$ **do**
    if $L_{Drop}[i] = 1$ **then**
      continue
    else if $L_{Replace}[i] = 1$ **then**
      Waiting time = random (low = 0, high = 0.09)
      send $DummyPkt$ (size = random)
    else
      $Interval \leftarrow S_x : Pkt_i - Pkt_{i-1}$
      r = random (low = 0, high = 0.09)
      Waiting time = $Interval + r$
      send $DummyPkt$ (size = $S_x : Pkt_i$)

4.1.2 Implementation

This section discusses practical considerations involved in implementing our defense and noise generation prototype.

4.1.2.1 Practical Considerations

In creating the dummy packets, we need to design a mechanism that allows IoT devices and the AP to easily identify and discard the noisy frames. To achieve this, we propose putting a flag on

the header of dummy packets. We exploit the reserved/unused bit flag on the IP header to label dummy packets. Since the adversary observes only the data-link layer traffic in an encrypted WiFi network, the IP header is fully encrypted, and hence, the attacker cannot observe the IP header to filter out dummy packets. The tagged packets can be ignored in different ways, such as using the *ip* command *blackhole* in Linux OS[2], which is used to drop packets silently.

4.1.2.2 Noise Generation Prototype

Since we cannot modify IoT devices due to firmware restrictions, we develop a proof-of-concept implementation on two Linux computers; the first is on the AP mode to emulate as an AP and the second one emulates as an IoT device. We developed a Python script with the *scapy* library[3] to generate and send dummy packets between the two computers over a WiFi connection. The packet creation and transmission are implemented using the $IP()$ and $send()$ functions at layer 3. Our source code can be found on GitHub [44].

4.2 Evaluation

We evaluate the performance of our obfuscation technique by simulating the noise injection on a data trace collected over 30 minutes from four WiFi-based IoT devices (Baby monitor, Camera, Light bulb and Smart plug). Note we capture additional data trace from the same devices to be used as background noise. We opted for a 30-minutes sample size due to the observed persistent pattern by all devices, which is also noted in [36]. Fig. 5.2 gives two examples of these steady patterns from two devices (Light bulb and Smart plug). The spikes indicate packet arrivals at different times. The larger the spike, the more packets observed during a 1-sec time window. We can notice that both devices exhibit a consistent pattern (regular number of packets per second), with higher traffic volume when the device is turned ON/OFF. For example, the light bulb on the

[2]https://man.archlinux.org/man/ip-rule.8

[3]https://pypi.org/project/scapy/

left char mostly has a frequent number of packets/sec of 2.5 and increases above 5 when changing its working status. Besides, the time difference between packet streams is remarkably constant. Therefore, we believe that our data has a sufficient mix of testing signatures to prove the concept of our defense.

To construct our dataset, we adopted the summary-data format described in [36]; For every device (i.e., every unique MAC address)[4], we calculate 14 features $\{f_1, f_2, \ldots, f_{14}\}$, as outlined in Section 3.3.5, observed over a 30-seconds time window. For example, to obtain the first datapoint, we calculate f during the time window $W_1 = \{0, \ldots, 30\}$. Then, we slide the time window forward by 30 seconds to obtain the second datapoint within $W_2 = \{30, \ldots, 60\}$ and repeat the process until the end of the trace. (We refer readers to more detailed descriptions in [36]). Note that we created two datasets: the first represents the undefended traffic, and the second is constructed after shaping the traffic using our proposed technique.

For evaluation, we randomly split our datasets into 75% for training and 25% for testing. We evaluate our defense against the best classifier (using Random Forest) reported in [36]. Below, we discuss our evaluation metrics and compare our results with random guessing as a baseline.

4.2.1 Evaluation Metrics

We evaluate our defense using the same standard metrics used for evaluating our prior attack: Accuracy, Precision, Recall and F1 Score. The detailed calculation of these evaluation metrics is given in the Appendix.

Our goal is to confuse the classifier from predicting the true device. Hence, we show the indistinguishability of the four devices under investigation through a lower score in all metrics compared to our attack, ideally close to random guessing[5] (50%).

[4]The basic assumption here is that a device will not change its MAC address during this observing time window, which is true for current IoT devices.

[5]Random guessing achieves $1/k$ accuracy where k is the number of classes in the model. Since we confuse the attacker between two devices at a time, then $k = 2$.

Additionally, we evaluate the overhead of our defense by calculating the percentage of added dummy data, which we define as:

$$Overhead = \frac{D}{R} \tag{4.3}$$

where D is the total amount of dummy packets, and R is the total amount of real packets.

Generally, inserting extra dummy packets affects the internet bandwidth, but as we mentioned earlier, our technique avoids this limitation by dropping noise packets at the AP before entering the Internet. However, we define this metric to simplify the analysis of the extra burden laid on the WiFi network or the devices (e.g., power consumption overhead).

4.2.2 Results

Table. 5.3 compares the accuracy, precision, recall and F1 score of two classifiers: the first is trained using the original traffic, which represents the attack performance and the second is trained using the defended/shaped traffic (i.e., After injecting dummy packets). We can observe the reduction of all metrics from 100% to 54% on average by our defense.

Furthermore, Table. 4.2 shows how the classifier confuses and misclassifies one device for another in our testbed (Bulb vs. Plug and Baby Monitor vs. Camera). The model could not learn from the "mix" as we inject from the bulb to the plug and vice versa. Likewise, we pair the traffic of the baby monitor and camera. As a result, a potential attack classifier will have difficulty learning from data created in this fashion.

The same Table (Table. 4.2) also shows the percentage of dummy packets to the real packets as 106%. This is expected as the WiFi network traffic doubles when we program each device to duplicate another device's traffic.

Table 4.1: Classification Accuracy, Precision, Recall and F1 Score of two datasets.

Device	**No Defense (%)**				**Defended Traffic (%)**			
	Accuracy	*Precision*	*Recall*	*F1*	*Accuracy*	*Precision*	*Recall*	*F1*
Baby Monitor	100	100	100	100	60	56	60	58
Bulb	100	100	100	100	50	50	50	50
Camera	100	100	100	100	53	57	53	55
Plug	100	100	100	100	50	50	50	50

Table 4.2: Confusion matrix of our IoT devices classification against our defense and Overhead.

		Actual			
		Baby Mon.	Bulb	Camera	Plug
Prediction	Baby Mon.	**0.6**	0.0	0.47	0.0
	Bulb	0.0	**0.5**	0.0	0.5
	Camera	0.4	0.0	**0.53**	0.0
	Plug	0.0	0.5	0.0	**0.5**
Balanced Accuracy		54%			
Added Packets		106%			

4.3 Discussion

Privacy leakage mitigation: Our results validate that our traffic-confusion-based obfuscation technique can defeat data-link device fingerprinting attacks. We show the four devices in our experiments become indistinguishable for the classifier, although it was trained using the obfuscated traffic. The learned model resulted in 54% accuracy compared to 50% using random guessing. This confusion is expected as signatures in both training and testing data are very similar for each device.

The side-channel leakage from encrypted traffic analysis is not only a privacy leakage but also a critical information leakage to vulnerability scanners. Mapping the high fidelity classification information to known vulnerable devices [1] provides a potential attack plan for when the adversary wishes to penetrate the environment. This preparation phase can provide insight to the adversary on what tools and penetration assets to establish a foothold in the target environment. Hence, this

defense may also provide *security* protection as privacy and security threats are sometimes related.

Overhead: Our method provides privacy without incurring internet bandwidth overhead. Moreover, we rely on pre-recorded traces to generate packet sequences to enforce confusion against inference attacks, which carries a small computational cost. However, since our defense requires additional packets to obfuscate traffic patterns, that would introduce an inevitable power consumption overhead.

CHAPTER 5: RANDOM SEGMENTATION: NEW TRAFFIC OBFUSCATION AGAINST PACKET-SIZE BASED SIDE-CHANNEL ATTACKS

This chapter introduces a data-efficient defensive approach that randomizes packet sizes without introducing noise[1].

5.1 Noise-Free Randomization

We defend against traffic analysis attacks by enabling IoT applications to control their packets, where it is not typically controllable. The application layer delivers a message of byte stream to the transport layer, which appends its header information (e.g., port number) and passes the data to the network layer as a segment. The segment size is determined by the Maximum Segment Size (MSS) and specific situations summarized below [45]:

- If the application message $\geq$ MSS, the TCP protocol sends the data in full segments equal to MSS for transmission and holds any portion of data surpassing MSS in an incomplete segment accumulating more bytes.
- If the message < MSS (i.e., there is still space in the segment) and a previously sent packet have not been acknowledged yet, the protocol waits for some time (+/- 200 ms [46]) to accumulate more bytes. This time delay allows for collecting more data to optimize network usage.
- If no additional data arrives within the timer period, the protocol dispatches the available data for transmission.

[1]The contents of this Chapter are based on our publication to MDPI Electronics 2023 [12]

The network layer, which is responsible for sending the data over the network, receives the segments and breaks them into as few full packets as possible by the MTU restriction. Indeed, MSS value at the transport layer depends on the underlying network MTU to ensure TCP segments can be properly encapsulated within network packets. MSS parameters are exchanged during the TCP handshake phase, and the network interface card's device driver provides the TCP/IP stack with the MTU value.

With our defense, the packet size is indistinguishable due to the randomness. That is, it consistently breaks received messages into arbitrarily sized segments. Suppose an IoT application sends a message m of n bytes. As stated earlier, the system, by default, will transmit m in a single segment if n = MSS or pass the small message (n < MSS) as it is after the timeout period (+/- 200 ms). If n > MSS, m will be divided into $\lceil n/\text{MSS} \rceil$ segments, all equal MSS except the last segment if n is not an exact multiple of MSS (i.e., $n\%\text{MSS} > 0$), then the last segment will hold the remaining bytes ($n\%$MSS). For example, assuming MSS is 1500, an application message of 3500 bytes will be transmitted in 3 segments (1500, 1500, 500). Unlike this dynamic, our defense will divide m into a random number of segments, and each segment's length will also be random. As a result, the data pattern from length features is not predictable.

Fig. 5.1 depicts two communication scenarios between a cloud server and an IoT device; the top represents the regular/undefended traffic, and the bottom shows the shaped/defended one. In both examples, we assume the server sends the same message two times. We here focus on the incoming traffic from the server side to demonstrate the concept of our defense. However, we expect both endpoints (i.e., server and client IoT) to implement our defense when communicating with each other. Thus, the bidirectional traffic is fully obfuscated.

In the first scenario, the server with no defense sends data in a typical packetized flow, leaking exploitable signatures for fingerprinting attacks. In contrast, the second defended scenario shows randomization occurs at the transport layer. Specifically, our technique passes random-sized segments to the network layer. Consequently, the traffic pattern generated by our shaping technique is

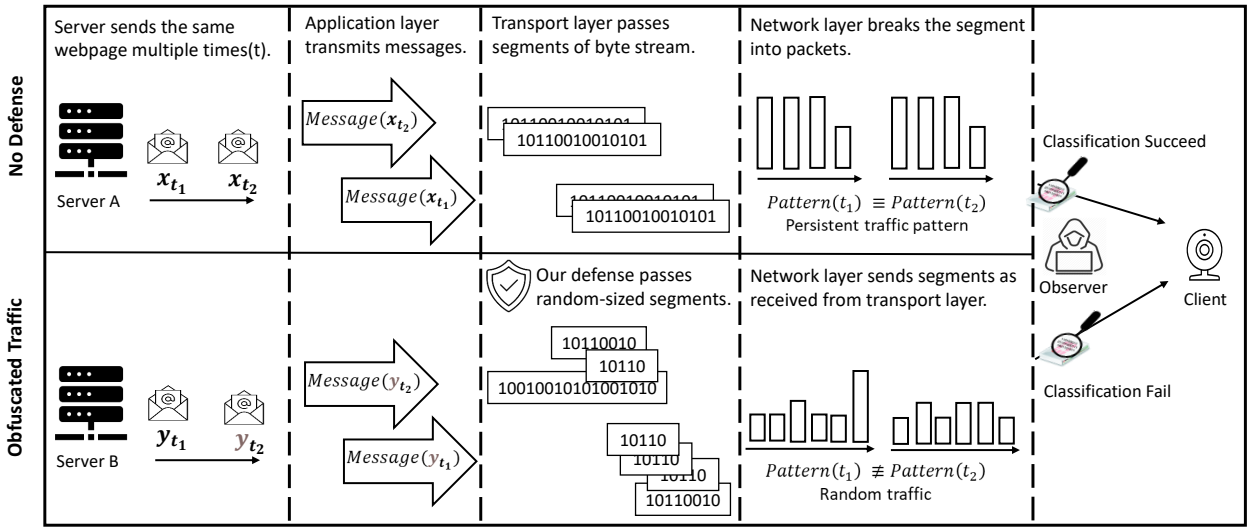


Figure 5.1: Two communication scenarios between a cloud server and a client IoT device. The top shows the original traffic without any defense, and the bottom depicts the obfuscated traffic after utilizing our proposed defense.

challenging to classify.

5.1.1 Algorithm

Algorithm 2 shows the pseudo-code processes of the proposed defense. We assume cloud servers and IoT devices run our program when sending packets.

The program stores the received message from the application layer in a byte array $Data$. Provided that the message is long enough to perform random segmentation (such that the length of $Data \geq$ the minimum segment size Min), our algorithm randomly decides whether to split the message but with a specific probability threshold $Prob$, such that $0 \leq Prob \leq 1$. If yes, the main loop loads a random chunk of Data into an individual segment Seg for transmission and loads another random chunk from $Data$ in the next iteration process until the array becomes empty. The size of each segment is determined randomly by $RandLen$, but does not surpass MTU to avoid fragmentation. However, the upper and lower bound of $RandLen$ (i.e., Min and Max) is adjustable to suit the

Algorithm 2 Random Segmentation of Application Messages

Data: $Data[]$ ← Byte array holding the application message,
Min, Max ← Select the minimum and maximum segment sizes,
$Prob$ ← Select the segmentation probability
if *length of* $Data \geq Min$ ***and*** $random.random() \leq Prob$ **then**
    $start \leftarrow 0$
    $end \leftarrow$ length of $Data - 1$
    while $start \leq end$ **do**
        $RandLen \leftarrow$ random(Min, Max)
        if $start$ + $RandLen \geq end$ **then**
            $index \leftarrow end$
        else
            $index \leftarrow start$ + $RandLen$
            $Seg \leftarrow Data[start : index]$
            send Seg
            $start \leftarrow index++$
else
    $Data$ are left to be handled by the operating system without segmentation.

device traffic pattern. For example, for a device that sends light traffic with a maximum of 300 bytes in length, the upper bound of $RandLen$ should be less than 300 to achieve randomness. Otherwise, the program will send the whole payload in $Data$ if $RandLen$ exceeds the message's size.

5.1.2 Multi-level Segmentation

Due to the significant differences in the packet size range of many IoT devices operating in different modes, choosing the appropriate range for length randomization (i.e., Min and Max) is challenging. For example, our preliminary analysis of our camera traffic shows that 93% of the packets are below 150 bytes when the camera is idle. On the other hand, when the camera becomes active, 58% of packets are above 1000 bytes. Hence, utilizing one range to mask all functional scenarios, such as splitting all segments into chunks between 100 and 150 bytes, will obfuscate the whole traffic but lead to excessive segmentation and increase the overhead.

Given the limitation of the one-level segmentation, it is necessary to make our defense adapt to the change in traffic volume. Thus, we adopt a multi-level segmentation to enable our algorithm to use a suitable range based on traffic intensity. For example, we use three levels for the high-bandwidth devices; Level 1 splits messages ≤ 200 bytes into random chunks between 20 and 40. Other messages above 200 and ≤ 500 are randomized using random lengths between 100 and 300 in level 2. Larger streams in level 3 that are above 500 are obfuscated using a random size between 500 and 1000. Obviously, that creates non-overlapping bands where the observer can recognize the corresponding band, but it is still insufficient to perform the attack due to the randomness within each band.

5.1.3 Practical Considerations

To change the standard packet sizing enforced by underlying protocols, which are controlled by the operating system, we initially considered changing MTU. Dynamic modification of MTU value will directly impact the packet size, as packets that exceed MTU will be divided to fit within the newer limit. However, MTU is a system-wide parameter, and such modification will not affect our program but the entire system, which is undesirable. In addition, smaller MTUs increase packet fragmentation, leading to adverse consequences for the system's performance. Assembling fragments at the destination adds extra burden and reduces the overall efficiency. Further delay can also occur when a fragment is missed or corrupted. In this case, the receiver cannot read partial data, and the whole data frame must be retransmitted. On the other hand, our approach does not fragment packets but divides TCP segments into separate IP packets. Hence, we avoid the drawbacks associated with packet fragmentation.

The main challenge in applying our idea is that applications do not have direct packet abstraction to control packet lengths. We could workaround this technical obstacle by modifying two parameters on a per-socket basis, thus, not affecting other programs.

First, we disable Nagle's algorithm [46] using the TCP_NODELAY option. Nagle's algorithm

Table 5.1: Our experimental setup of the system parameters.

Parameter	Low-Bandwidth Devices		High Bandwidth Devices		
	Bulb	Plug	Doorbell	Camera	
$Prob$	0.8		0.8		
Min	5		L1= 20	L2= 100	L3= 500
Max	20		L1= 40	L2= 300	L3= 1000

[1] L1, L2, and L3 refer to the three levels of segmentation introduced in Section 5.1.2.

avoids sending small TCP segments by introducing a time delay to collect more data so that it sends full segments. Since we aim to send data in predetermined sizes, this data aggregation contradicts our purpose and needs to be disabled.

Second, in the case of intensive traffic, we limit the amount of data that can be pushed out of a socket at the initial stage of the communication. TCP begins with a small congestion window to assess the network condition and find the optimal window size. As we turn off Nagle's algorithm, many small packets can be sent immediately. Consequently, the operating system overrides our program and accumulates the data into larger packets to improve efficiency, rendering our defense ineffective. To avoid this problem, we decrease the send socket buffer to less than the receive buffer.

As stated, our traffic shaping technique masks packet length without adding noisy data. Thus, the data rate remains unchanged (i.e., the amount of transmitted data is the same). Therefore, we added another module to select the amount of covered traffic as needed. In our approach, we inject a certain amount of traffic to make one device similar to another in terms of data rate.

5.1.4 Implementation

We developed a server-client implementation using socket programming in Python. We use TCP protocol to send packets due to its reliability and expect our obfuscation methodology to apply to UDP as well. Our source code is publicly available at GitHub [47].

We assume the defender has access to the targeted IoT devices and servers to install our program as a patch used by a hook that intercepts every send command from the application layer. That is, it will replace the standard send code with our patch to randomize packet size.

5.2 Evaluation

In this section, we first evaluate our defense against traffic classification of IoT devices based on packet length. We compare the performance of our noise-free randomization with an analogous noise-based mechanism that uses random packet padding [30]. Second, we quantify the impact of our technique on communication performance through real-world experiments. Below, we discuss each aspect and present our results.

5.2.1 Effectiveness

To evaluate the randomness in the packet length by our technique, we developed a program to simulate our defense on a WiFi trace of four IoT devices: Doorbell, camera, light bulb, and smart plug. We captured the encrypted traffic for one hour on different operating modes (e.g., ON-OFF).

Our program reads the pcap file of each device and produces the obfuscated traffic to test our defense against DF attacks. Table. 5.1 outlines our configuration for the adjustable system parameters introduced in Section 4.1.1 and Section 5.1.2. The probability was manually chosen with the goal of minimizing overhead (i.e., reducing the number of segmented packets) while maintaining a lower accuracy. The initial value of 0.6 resulted in a high accuracy (¿ 80). Consequently, we increased the probability to 0.7, and the accuracy remained consistently high. Subsequently, when we further elevated the probability to 0.8, a reduction in accuracy was achieved. We use the same defense parameters for each group of devices categorized based on the traffic intensity because different parameters will likely create new patterns to distinguish the devices.

Furthermore, we run another simulation to obfuscate our captured trace using random padding in-

troduced in [30] to compare the effectiveness of a traditional noise-based solution with our defense.

The attacker's profiling classifier is trained using the training data of the original trace. We evaluate the performance of attack without defense by testing the classifier using the testing data of the original trace. Likewise, we evaluate the defense by first generating the modified trace using our defense program based on the original trace, then training the attack classifier on the training data of the modified trace and then testing the classifier using the test data in the modified trace.

We use Random Forest for our classification due to its outperformance on similar IoT device identification attacks compared with several other ML algorithms [27, 36]. We randomly divided our dataset into 70% for training and 30% for testing and quantified the performance of our classifier using the following metrics: Accuracy, Precision, Recall, and F1 Score. The specific computation of these metrics can be found in the Appendix.

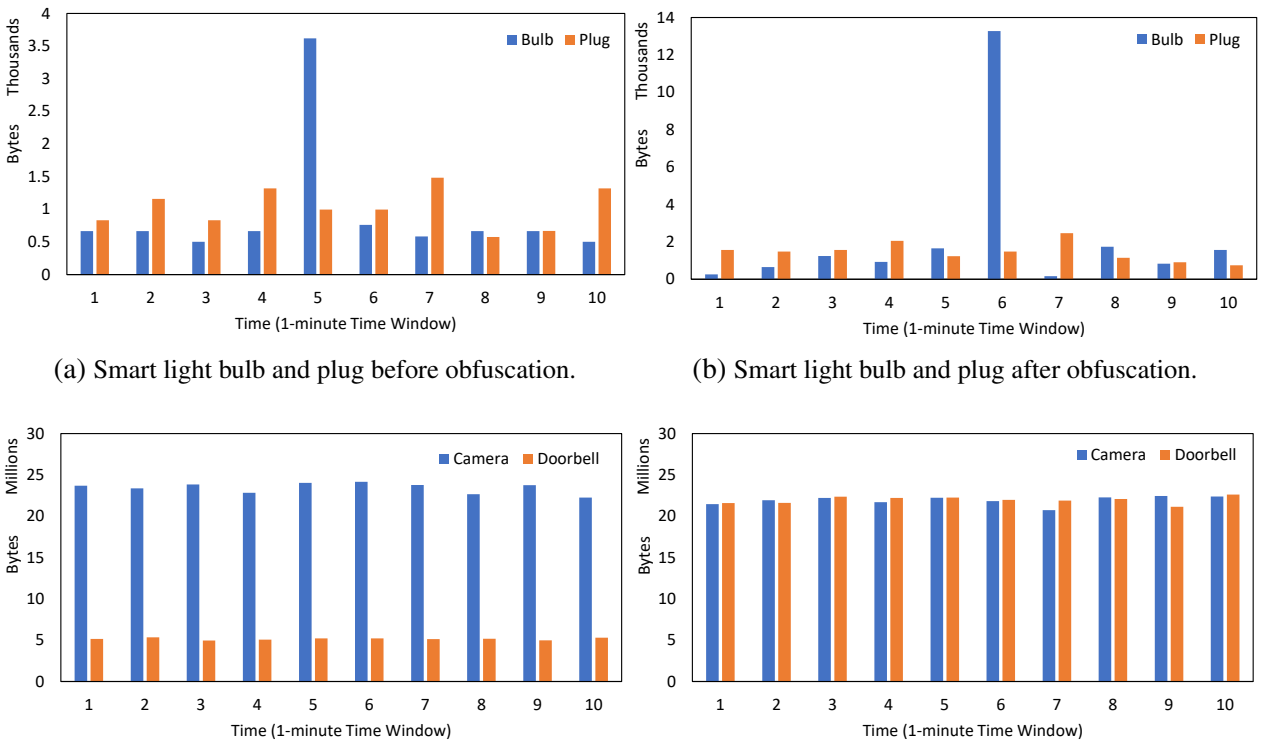
Note that we show the indistinguishability of devices by applying our technique to training and testing data; the closer the accuracy to random guessing[2] (50%), the more effective our defense is in confusing the classifier.

We evaluate against an attacker who exploits the packet size and direction only. Thus, we construct our dataset using packet sizes in a binary format to represent directions; positive size represents incoming packets, and negative size for outgoing ones. Similar to [36, 43], we break the trace into sequences observed within a 30-seconds time window for classification.

5.2.1.1 Preliminary Data Analysis

Fig. 5.2 presents a sample traffic flow observed over a period of 10 minutes before and after implementing our defense. Before obfuscating the traffic, we observed variable traffic patterns in the light bulb and smart plug. Specifically, the bulb's traffic was higher than the plug's in 7 instances, lower in 2 instances, and similar in one case (Fig. 5.2.(a)). This inconsistency in traffic

[2]Random guessing attains an accuracy rate of $1/k$ where k is the number of labels/devices. As we confuse the attacker between two devices of similar traffic intensity, then $k = 2$.



(a) Smart light bulb and plug before obfuscation.

(b) Smart light bulb and plug after obfuscation.

(c) Camera and doorbell before obfuscation.

(d) Camera and doorbell after obfuscation.

Figure 5.2: Traffic flow of four IoT devices over 10 minutes.

poses a challenge for reliable device profiling based on data volume-related features. The impact of this variability is evident in the spike in bulb traffic (Fig. 5.2.(a), traffic period 5), which was misclassified as camera traffic.

On the contrary, the camera and doorbell exhibit stable traffic patterns while operating in a fixed working mode, such as recording videos (Fig. 5.2.(c)). However, the camera's higher resolution results in significantly larger traffic compared to the doorbell. Consequently, to account for the variance in data volume between the camera and doorbell, we introduced covered bytes to the doorbell to compensate for the variance in data volume (Fig. 5.2.(d)). Dummy/covered packets can be labeled and discarded at the receiving side. We assume the defender can leverage the header field of the Traffic Flow Confidentiality (TFC) mechanism [48]. This mechanism provides a tool

Table 5.2: Injected covered bytes to hide data rate features.

Device	Covered Bytes (%)
Bulb	0
Plug	0
Camera	0.7
Doorbell	340

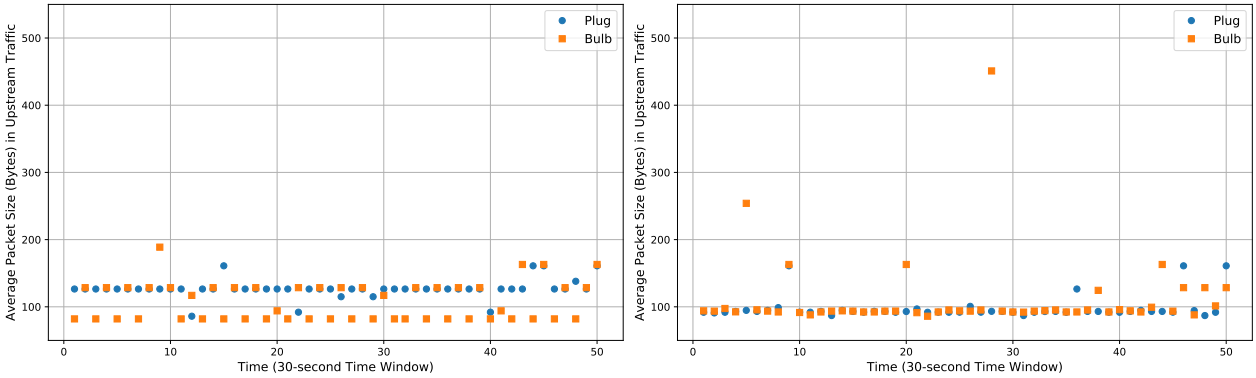
to inject dummy packets using a wrapped header field that is encrypted and cannot be observed by network observers. The injection statistics are summarized in Table. 5.2, revealing that no dummy packets were injected for the bulb and plug since the data rate does not serve as a suitable representative for those two devices. Conversely, the doorbell necessitates a greater incorporation of covered bytes to achieve parity with the camera, in order to introduce confusion within the classifier's discrimination between the two devices. Additionally, a marginal proportion of covered bytes (0.7%) was introduced to the camera to mask its disparities from the doorbell, particularly during periods of inactivity.

It is important to highlight that we incorporated a 20% time delay[3] in our simulation to align it with the findings in Section 4.2.2. As a result, the overall traffic attributes were affected. For instance, the event surge observed in the light bulb's traffic before applying our defense (Fig. 5.2.(a), traffic period 5) can be seen in the subsequent observation time window in the defended traffic (Fig. 5.2.(b), traffic period 6).

Fig. 5.3 illustrates the impact of our approach on packet size. We chose the bulb and plug for demonstration as they rely entirely on random segmentation for packet size obfuscation (i.e., no covered bytes were injected). Before implementing our defense, we can notice in Fig. 5.3.(a) a steady average size (nearly 130 bytes)[4] sent by the plug versus fluctuating value by the light bulb. We can observe similar behavior in the return traffic represented by negative values in Fig. 5.3.(c).

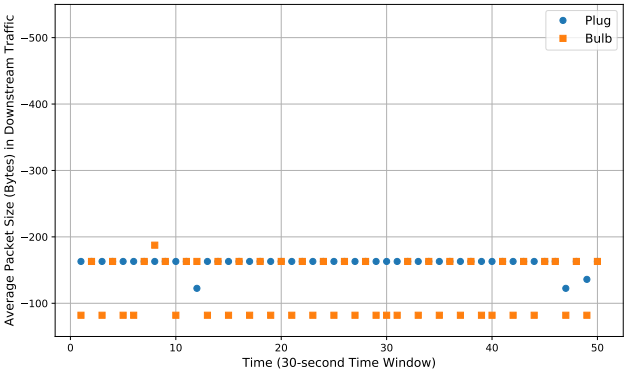
[3]Refer to Equation 5.2 for the specific computation of time delay.

[4]After subtracting the frame header, 130 bytes means there are 48 (i.e., 130-82) bytes in the payload for the segmentation.

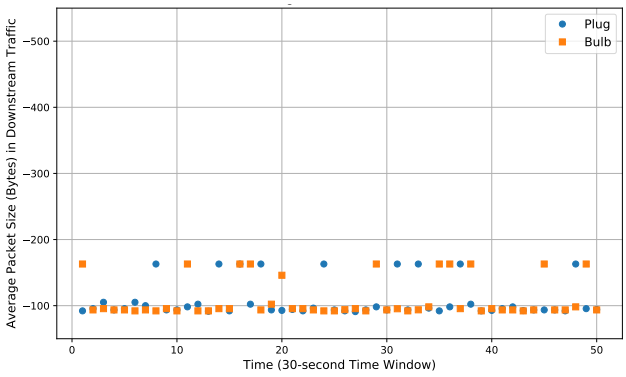


(a) Outgoing traffic before obfuscation.

(b) Outgoing traffic after obfuscation.



(c) Incoming traffic before obfuscation.



(d) Incoming traffic after obfuscation.

Figure 5.3: Average packet size before and after obfuscating the bidirectional traffic of two devices over time.

After obfuscating the traffic (Fig. 5.3.(b) and Fig. 5.3.(d)), the described range by each device starts to overlap, introducing uncertainty in the learning process for device classification.

5.2.2 Efficiency

In this section, we evaluate the byte overhead B and the time taken T by our algorithm. We calculate B as:

$$B = \frac{D_b - W_b}{W_b} \tag{5.1}$$

where D_b[5] is the total amount of bytes transferred when implementing our defense, and W_b is the total amount of bytes transferred without implementing our defense.

For the second aspect (T), we set up a remote virtual server and let our local machine send a large file of 10 MB, with and without our defense. Thus, we calculate the added latency by our methodology compared with the standard/undefended transmission scenario. We define T as:

$$T = \frac{D_t - W_t}{W_t} \tag{5.2}$$

where D_t is the time span to send the file when implementing our defense, and W_t is the time span to send the same file without implementing our defense.

Furthermore, we implemented two randomization levels to analyze the impact of obfuscation intensity (i.e., range of random values) on T. The wider the range of random length, the more packets are required to carry the payload. Consequently, more packets might take longer time to deliver. For instance, sending a large array of data using random-sized packets ranging from 100 to MTU will result in significantly more packets than using a range of larger lengths between 1200 and MTU. For this experiment, we define two randomization levels ($Rand_{(low)}$ and $Rand_{(high)}$) upper bound by a common maximum length of 1400 bytes, whereas the lower bound of each level varies as follows: $Rand_{(low)} = 1200$ and $Rand_{(high)} = 100$. We run ten sets of experiments and report the average result in the following section.

5.2.3 Results

As shown in Table. 5.3, all the classification metrics used to evaluate the randomness in the shaped traffic are close to the baseline of random guessing. The values in the last column are bolded to represent the best performing results. The result demonstrates the reduction of classification accuracy from 98% by the attack to 63% by our defense. Also, our technique achieves better

[5]As we utilize the covered bytes presented in Table. 5.2 solely for concealing data rate features, we deduct them from this calculation in the context of packet-size obfuscation.

Table 5.3: Classification Accuracy, Precision, Recall and F1 Score of two defenses.

Metric	No Obfuscation (%)	Random Padding [30] (%)	Random Segmentation (%)
Accuracy	98	71	**63**
Precision	98	77	**67**
Recall	98	71	**63**
F1	98	71	**63**

Table 5.4: Byte overhead B by two defenses*.

Device	**No obfuscation**	**Random Padding** [30]		**Random Segmentation**	
	W_b (MB)	D_b (MB)	B (%)	D_b (MB)	B (%)
Bulb	0.0348	0.1936	456	0.1091	**214**
Plug	0.0287	0.1626	467	0.0887	**209**
Camera	549.2	766.7	40	589	**7**
Doorbell	155.3	317.6	105	168	**8**
Total	704.6	1084.7	54	757.2	**7**

obfuscation (lower accuracy) than random padding. The attack accuracy under our defense is 8% lower.

Moreover, Table. 5.4 compares the byte overhead B by our defense with random padding. The bolded values in the last column represent the most favorable outcomes. Our defense incurs significantly lower overhead for all devices, which saves nearly 47% of total overhead[6].

Last, we report the latency results from our large file transmission experiments between our client machine and remote server. As shown in Fig. 5.4, our defense comes with an average time overhead of 20.5%. Compared with random padding, our technique underperforms by only 0.7%, as [30] reported 19.8% latency. The same figure (Fig. 5.4) also shows a stable T regardless of whether we perform low or high splitting (using $Rand_{(low)}$ and $Rand_{(high)}$). Although B has increased by 4.5% due to the intensive splitting using $Rand_{(high)}$, more splitting seems not to introduce

[6]From Table. 5.4, B by Random Padding is 54% more than Random Segmentation which achieves 7%, resulting in a saving of 47% (54-7).

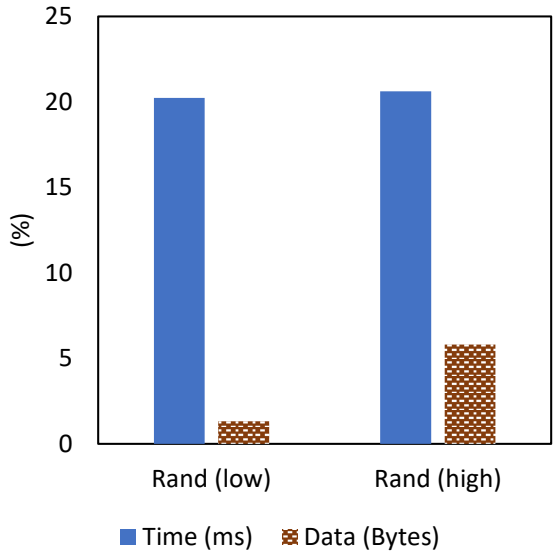


Figure 5.4: Time and byte overhead using two obfuscation levels.

noticeable latency.

5.3 Discussion

Effectiveness: Our results validate that our approach can disrupt packet length features to protect against traffic analysis attacks. By simulating our defense on a trace from real IoT devices, we show that the obfuscated traffic resulted in a baseless classifier comparable to random guessing. This is because our defense sends data packets in random sizes, which prevents the classifier from learning length-based fingerprints for profiling.

Timing characteristics, such as interarrival time, is not considered by our technique. However, timing leakage has been addressed by delaying data packets to obscure the related patterns [49]. Indeed, packet delay can be incorporated with our defense to conceal both timing and length features.

Efficiency: Our randomization technique is noise-free, hence more data-efficient than other noise-based approaches like packet padding. However, there is a case where header overhead by our

Table 5.5: Send buffer size impact on transmission time using two obfuscation levels.

Buffer Size	Time Overhead (%)	
	$Rand_{(low)}$	$Rand_{(high)}$
2^{15} Bytes	132.53	128.67
2^{16} Bytes	26.1	26.78

defense is higher than padding. If the size of the transmitted flow is relatively small compared to the packet headers (54 bytes), then every split by our approach adds more data than the payload itself. For example, if we have a device that sends a small packet of 100 bytes per second. Then, it becomes expensive to split the packet into multiple chunks. Because the header overhead from generating an additional packet becomes 54% (54/100). Nevertheless, such small traffic would be marginal to the total bandwidth in the network.

In terms of time overhead, our countermeasure incurs reasonable latency (20.5%), which many IoT devices can tolerate, such as sleep monitors and smart plug [50]. However, high-bandwidth devices like cameras may experience degradation due to the need for greater bandwidth to support video streaming.

One interesting insight we observe in Fig. 5.4 is that intensive obfuscation (i.e., a higher degree of randomness) does not increase the latency. Although the number of packets is higher by our mechanism, that does not add noticeable latency due to the immediate transmission enabled by the TCP_NODELAY option. The factor that led to the increase in transmission time was the limited send buffer, which puts the socket on hold from pushing more data until the buffer is empty. It is evident from Table. 5.5 that smaller buffer sizes increase the time overhead significantly.

Note that we are not claiming that the deactivation of Nagle's algorithm is an efficient solution. Sending small packets can lead to additional header and processing overhead. However, turning off Nagle's algorithm has been introduced in prior works as an effective technique with no adverse effect on performance, such as preventing many deadlock situations [45]. Similarly, our proof-of-concept implementation on consumer-grade laptops shows that our defense can mitigate privacy

leakage but may introduce some latency, as the sender needs to send data stream in a larger number of packets. With that being said, further research is needed to investigate the impact of our technique on devices with limited processing capacity and storage, similar to IoT devices. We plan further investigation in this direction as future work.

Compatibility and Deployment Challenges: While our current implementation showcases the feasibility of the approach through a client-server setup using TCP sockets, we fully acknowledge that IoT environments present unique challenges. For instance, adapting our proposed technique for communication with generic browsers in server roles may necessitate updates on server-side software. Hence, there could be technical issues to accommodate all existing devices, which need further investigation in future work.

Vulnerability Analysis: It is vital to address potential vulnerabilities and ensure the robustness of our approach. However, our technique only enables IoT applications to change packet length without any other modification of the entire IoT device communication. For example, it does not affect WiFi encryption, IPsec or SSL implementation, etc. Hence, we see no immediate and evident vulnerabilities in our proposed method.

Adversarial Attack: We assume the attacker knows how our defense works and, hence, can try to merge the length of consecutive packets to overcome our defense. However, there is no basis for the attacker to retrieve packet size patterns. If the attacker combines all consecutive packets, the attacker will merge packets that were not initially split because it is customary to observe a series of small-sized packets, such as mouse flow, when there is no defense implemented. In addition, our mechanism performs random segmentation for randomly selected messages. Hence, there is no fixed rule for our splitting to perform adversarial de-splitting with high accuracy.

CHAPTER 6: ADAPTIVE SEGMENTATION: A TRADEOFF BETWEEN PACKET-SIZE OBFUSCATION AND PERFORMANCE

6.1 Adaptive Control

As discussed in Chapter 5, our randomization approach introduces latency due to the increase in packet count and data overhead caused by the introduction of additional packet headers. However, in dynamic networks, fluctuations in network utilization can lead to prolonged queue delays, negatively impacting network performance. Therefore, a robust defense system should incorporate an "adaptive defense" feature—adaptively adjusting its configurations according to the network condition to balance privacy protection and network performance. Particularly, we employ optimization to maximize privacy protection (increasing the randomness in packet size) while considering the impact of message splitting on system performance (reducing latency and overhead).

Indeed, the fundamental concept of "adaptive defense" has found application in various domains, including real-world epidemic disease control, the five-tier terrorism alert system, and others [51]. The key challenge lies in translating this foundational principle into the design of an effective defense system within the realm of traffic obfuscation. In this chapter, we introduce a specific adaptive defense system that estimates defense strength and its impact on performance. The adaptive parameter adjustment involves straightforward calculations and optimization, ensuring minimal computational overhead.

6.1.1 Background

Random segmentation involves intercepting application messages and splitting them into random chunks at the transport layer. The quantity of messages subject to segmentation is determined by a predetermined splitting percentage P. This variable serves as a control mechanism, specifying the proportion of messages to be segmented—such as 10% or 50%.

Notably, as P escalates, the degree of packet size randomization, quantified as R, increases. We assess the randomness in packet size, denoted as R, using the same metric proposed in a comparable network obfuscation system [52]. Specifically, we employ Shannon entropy to measure the effectiveness of masking through randomization. Shannon entropy, an information-theoretic metric, evaluates the uncertainty inherent in a random variable. Given a random variable X with values in the finite set $\{x_1, x_2, \ldots, x_M\}$ and l_i denoting the likelihood of $X = x_i$, Shannon entropy is expressed as follows:

$$R(X) = -\sum_{i=1}^{M} l_i \log(l_i)$$

where:

- $R(X)$ is the entropy of the random variable X,
- M is the number of possible outcomes in the discrete random variable X,
- x_i represents each possible outcome, and
- $l(x_i)$ is the likelihood/probability of the occurrence of x_i.

However, the increase in randomness is likely to lead to higher time overhead T, stemming from the construction of additional packets and data overhead introduced by extra packet headers. To validate this relationship, we utilized a program to emulate an IoT application sending an array of data (≈ 976 KB) to a cloud server, where each message in the array has a length of 1000 bytes. The program sent the same array using various P values, ranging from 0 to 1, and calculated the elapsed time for each P. We conducted five sets of experiments, calculating the average time overhead T as described in Equation 5.2.

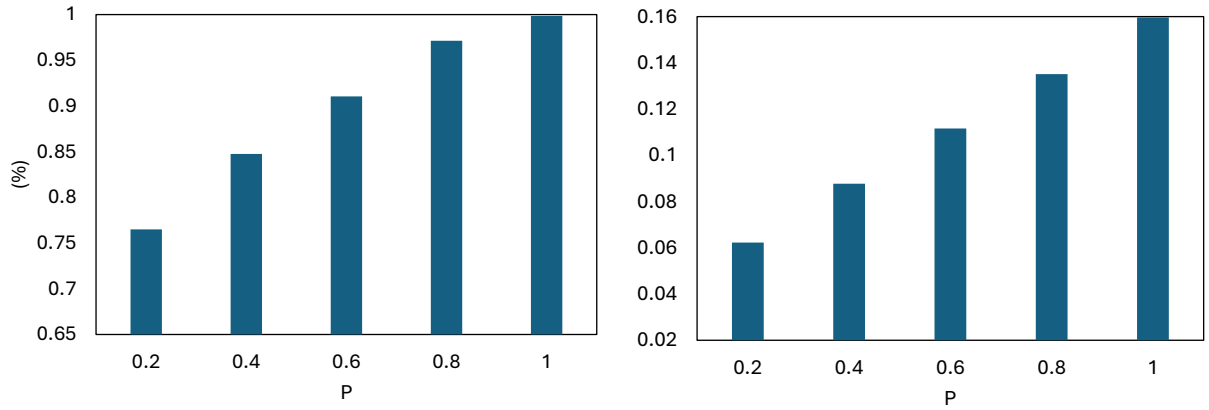
Figure 6.1 illustrates the interrelationship between R and T. On the x-axis, 0.2 indicates a defense split of 20% of the total packets, while 1 signifies a defense split involving all messages (100%). R was calculated by simulating our defense on the data traces collected from our IoT devices. We

normalized R on the y-axis by each device's maximum, aggregating the performance of all devices into a single chart for comparable results with T. The latter is calculated based on our client-server experiment described above.

In Figure 6.1, a clear linear upward trend is observed for both R and T with the increase in P; therefore, it is reasonable to assume they linearly increase with P for the sake of simulation. This assumption is made because the linear or exponential increase will have the same effect on the reward and penalty scheme of our optimization, given that R is the only dimension we aim to maximize, and T is the only dimension we aim to minimize.

6.1.2 Obfuscation Optimization Problem

At a conceptual level, our goal is to randomize the packet sizes of a device, thereby thwarting potential classifiers from identifying patterns that facilitate DF. The model determines the optimal P for maximizing R while considering its impact on T. Hence, the objective function defines a score S, combining the defense benefit measured by traffic randomness R, with a penalty term associated with the increase in latency T. Thus, our goal is to maximize the objective function,



(a) R. (b) T.

Figure 6.1: Packet size randomness R versus latency T under different defense splitting percentage P.

defined as:

$$\text{Maximize} \quad S_P = w \cdot R - (1 - w) \cdot T$$

where w is a weight constant that predetermines the trade-off between R and T before running the adaptive optimization. It could be predetermined by a field expert to express how much we value privacy protection compared with the network performance degradation.

Based on the analysis reported in [53], the latency shows a slight increase with the rise in network usage u. However, when the usage reaches 60%, the latency starts to increase rapidly. Hence, we set our defense to ensure maximum protection when u is below 60%; otherwise, our optimization model is triggered to find the optimal parameter P_{optimal} for the desired tradeoff.

$$P = \begin{cases} 1 & \text{if } u < 0.6 \\ P_{\text{optimal}} & \text{otherwise} \end{cases}$$

6.2 Evaluation

We evaluate the performance of the adaptive obfuscation system using the data traces collected from four IoT devices. We deploy our defense across a range of u from 0 to 100%, calculating the corresponding P for each setting.

6.2.1 Results

As shown in Figure 6.2, our optimization is triggered when u exceeds the threshold (i.e., $u \geq 0.6$), returning the optimal solution for the maximum score S. We demonstrate the effectiveness of our approach using different weights, $w_{\text{high}} = 0.8$, $w_{\text{moderate}} = 0.6$, and $w_{\text{low}} = 0.4$. It is important to note that the selection of these weights is for demonstrative purposes, and the scope of our work does not encompass providing specific guidelines on how to choose the optimal weight. The

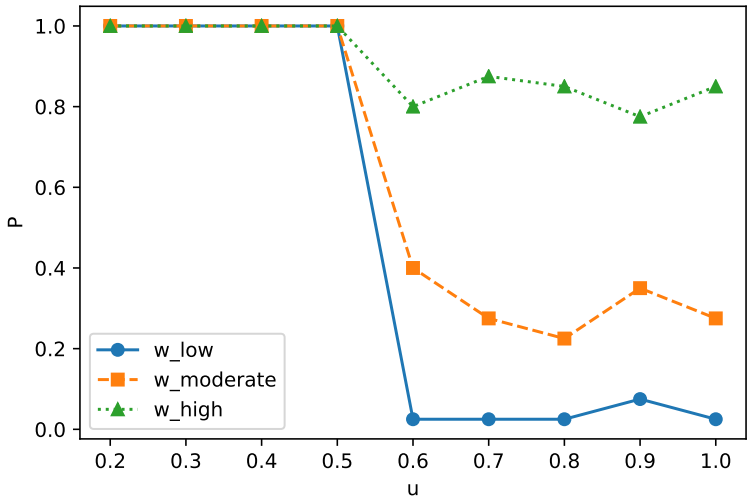


Figure 6.2: Performance of our adaptive defense system using different weights w.

intention is to highlight the flexibility of the defense mechanism rather than prescribe a definitive weight value.

To ensure high protection, a user can adjust w similarly to w_{high} to place more emphasis on privacy than performance. Alternatively, w_{moderate} provides lower obfuscation for reduced overhead, or one can opt for w_{low} for weaker protection but improved network performance.

6.3 Discussion

Effectiveness: Our adaptive segmentation approach provides a systematic and effective means of optimizing privacy with controllable costs. By considering both entropy and latency, the method offers a balanced solution that enhances the overall efficiency of communication networks. Unlike previous defenses that prioritize traffic obfuscation at any cost, our approach is designed to give users the flexibility to balance protection and performance according to their preferences.

Limitation: It is important to acknowledge a potential limitation in our adaptive defense mechanism. The system dynamically adjusts its parameter in response to the congestion level to optimize performance and reduce overhead. While this adaptiveness is designed to enhance efficiency un-

der normal network conditions, we recognize a potential vulnerability in the form of a Distributed Denial of Service (DDoS) attack. An attacker could deliberately flood the network to trigger our adaptive defense, causing it to lower the protection by reducing the number of splitting operations. This raises a concern about the potential exploitation of the adaptiveness feature under malicious conditions. Future research could explore mechanisms to enhance the resilience of the defense against such deliberate attempts to manipulate its adaptive behavior.

CHAPTER 7: CONCLUSION AND FUTURE WORK

7.1 Conclusion

This dissertation presents a new privacy attack that an out-of-network eavesdropper can use to identify various IoT devices and, in most cases, infer their working modes, within a 30 second time window. In Chapter 3, we demonstrated this attack's feasibility on a testbed of 10 IoT and 2 non-IoT devices with a high accuracy of 95%.

The existing countermeasures against network traffic analysis attacks at the network-layer are not a viable option against data-link DF attacks. One reason is that they would negatively impact the internet condition with significant noisy traffic. Besides, prior studies present lightweight solutions in bandwidth but at a higher latency. By adding noise traffic only on the WiFi link, we present a zero-overhead defense in internet bandwidth against WiFi-based traffic analysis attacks (See Chapter 4). In addition, our technique injects dummy packets without posing a deliberate delay on the original traffic of a device. We present a traffic obfuscation mechanism in which a pair of devices become indistinguishable. Each device sends dummy packets to mimic the other, so it generates two patterns simultaneously to hamper the fingerprinting attack. Our method significantly reduces the classification accuracy of the presented privacy attack to be close to random guessing.

Last, we have shown how random segmentation can obfuscate packet size patterns without injecting noisy data. The proposed approach in Chapter 5 enables network devices to send application messages through random-sized segments and pass them to the network layer for immediate transmission. Consequently, the observed traffic at and above the network layer is randomized, defending against both in-network and out-network observers (i.e., IP and MAC level observation). The technique has been tested on a client machine connected to a remote server, and the results demonstrate the effectiveness of our defense with a reasonable time overhead ($< 21\%$).

7.2 Future Work

For future work, our objective is to integrate network traffic obfuscation with machine learning techniques, enhancing both defense effectiveness and efficiency. The primary goal is to optimize obfuscation quality while minimizing performance degradation. A promising approach involves implementing reinforcement learning, a technique renowned for making optimal decisions based on experiences. For example, in a gaming scenario, one can observe the agent's performance improvement over time, as this form of learning involves the agent interacting with the environment to maximize the received reward.

With this in mind, we can design a learning environment for our agent where it is rewarded for effectively confusing the classifier and penalized for any decline in system performance. Specifically, the agent can manage generative adversarial network parameters to produce adversarial examples that obfuscate network traffic with minimal overhead.

It is essential to demonstrate how we could confuse the attacker's classifier, as we cannot predict the classifier the attacker might develop. To address this issue, we can develop a dynamic feature extraction unit that utilizes deep learning models to extract relevant features from network traffic data. These features should capture patterns in network traffic, as the model adapts to evolving network traffic characteristics. Hence, adversarial examples effective in confusing our classifier may also be effective for any classifier the attacker uses due to the 'transferability' of adversarial examples. Transferability, in the context of adversarial attacks, means that an adversarial example crafted to deceive one model may also be effective in deceiving another, even if the other model is different and trained independently. This phenomenon is notable in adversarial attacks and has been observed across various machine learning models.

LIST OF REFERENCES

[1] Y. Wan, K. Xu, G. Xue, and F. Wang, "Iotargos: A multi-layer security monitoring system for internet-of-things in smart homes," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 874–883.

[2] J. Li, Z. Li, G. Tyson, and G. Xie, "Your privilege gives your privacy away: An analysis of a home security camera service," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 387–396.

[3] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[4] L. Yu, B. Luo, J. Ma, Z. Zhou, and Q. Liu, "You are what you broadcast: Identification of mobile and {IoT} devices from (public) wifi," in *29th USENIX security symposium (USENIX security 20)*, 2020, pp. 55–72.

[5] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.

[6] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All things considered: an analysis of iot devices on home networks," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1169–1185.

[7] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "Iot devices recognition through network traffic analysis," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 5187–5192.

[8] S. Aneja, N. Aneja, and M. S. Islam, "Iot device fingerprint using deep learning," in *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*. IEEE, 2018, pp. 174–179.

[9] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[10] B. Chandavarkar *et al.*, "Detecting rogue access points using kismet," in *2015 International Conference on Communications and Signal Processing (ICCSP)*. IEEE, 2015, pp. 0172–0175.

[11] P. Serrano, M. Zink, and J. Kurose, "Assessing the fidelity of cots 802.11 sniffers," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 1089–1097.

[12] M. Alyami, A. Alghamdi, M. A. Alkhowaiter, C. Zou, and Y. Solihin, "Random segmentation: New traffic obfuscation against packet-size-based side-channel attacks," *Electronics*, vol. 12, no. 18, p. 3816, 2023.

[13] J. S. Atkinson, "Your wifi is leaking: inferring private user information despite encryption," Ph.D. dissertation, UCL (University College London), 2015.

[14] Q. Zhu, C. Yang, Y. Zheng, J. Ma, H. Li, J. Zhang, and J. Shao, "Smart home: Keeping privacy based on air-padding," *IET Information Security*, vol. 15, no. 2, pp. 156–168, 2021.

[15] L. Molina, A. Blanc, N. Montavont, and L. Simić, "Identifying channel´ saturation in wi-fi networks via passive monitoring of ieee 802.11 beacon jitter," in *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*, 2017, pp. 63–70.

[16] A. Abusnaina, R. Jang, A. Khormali, D. Nyang, and D. Mohaisen, "Dfd: Adversarial learning-based approach to defend against website fingerprinting," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2459–2468.

[17] S. Feghhi and D. J. Leith, "An efficient web traffic defence against timing-analysis attacks," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 2, pp. 525–540, 2018.

[18] W. De la Cadena, A. Mitseva, J. Hiller, J. Pennekamp, S. Reuter, J. Filter, T. Engel, K. Wehrle, and A. Panchenko, "Trafficsliver: Fighting website fingerprinting attacks with traffic splitting," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1971–1985.

[19] J. Liu, C. Zhang, and Y. Fang, "Epic: A differential privacy framework to defend smart homes against internet traffic analysis," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1206–1217, 2018.

[20] F. Zhang, W. He, and X. Liu, "Defending against traffic analysis in wireless networks through traffic reshaping," in *2011 31st International Conference on Distributed Computing Systems*. IEEE, 2011, pp. 593–602.

[21] "Jammer enforcement," https://www.fcc.gov/general/jammer-enforcement/, accessed: 2020-11-24.

[22] L. Chaddad, A. Chehab, I. H. Elhajj, and A. Kayssi, "Optimal packet camouflage against traffic analysis," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 3, pp. 1–23, 2021.

[23] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1375–1390.

[24] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 332–346.

[25] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014, pp. 121–130.

[26] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 143–157.

[27] A. J. Pinheiro, J. d. M. Bezerra, C. A. Burgardt, and D. R. Campelo, "Identifying iot devices and events based on packet length from encrypted traffic," *Computer Communications*, vol. 144, pp. 8–17, 2019.

[28] S. Xiong, A. D. Sarwate, and N. B. Mandayam, "Network traffic shaping for enhancing privacy in iot systems," *IEEE/ACM Transactions on Networking*, pp. 1–16, 2022.

[29] S. Kennedy, H. Li, C. Wang, H. Liu, B. Wang, and W. Sun, "I can hear your alexa: Voice command fingerprinting on smart home speakers," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019, pp. 232–240.

[30] A. J. Pinheiro, J. M. Bezerra, and D. R. Campelo, "Packet padding for improving privacy in consumer iot," in *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2018, pp. 00 925–00 929.

[31] M. R. Albrecht, K. G. Paterson, and G. J. Watson, "Plaintext recovery attacks against ssh," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 16–26.

[32] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh," in *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[33] M. Bellare, T. Kohno, and C. Namprempre, "Authenticated encryption in ssh: provably fixing the ssh binary packet protocol," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 1–11.

[34] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–29, 2019.

[35] H. Bilic, Y. Birk, I. Chirashnya, and Z. Machulsky, "Deferred segmentation for wire-speed transmission of large tcp frames over standard gbe networks," in *HOT 9 Interconnects. Symposium on High Performance Interconnects*. IEEE, 2001, pp. 81–85.

[36] M. Alyami, I. Alharbi, C. Zou, Y. Solihin, and K. Ackerman, "Wifi-based iot devices profiling attack based on eavesdropping of encrypted wifi traffic," in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, 2022, pp. 385–392.

[37] K.-C. Lan and J. Heidemann, "On the correlation of internet flow characteristics," Citeseer, Tech. Rep., 2003.

[38] S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Identification of user application by an external eavesdropper using machine learning analysis on network traffic," in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2019, pp. 1–6.

[39] F. Osisanwo, J. Akinsola, O. Awodele, J. Hinmikaiye, O. Olakanmi, and J. Akinjobi, "Supervised machine learning algorithms: classification and comparison," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 48, no. 3, pp. 128–138, 2017.

[40] L. Khermosh, Z. Haramaty, and J. Mandin, "Implementing ieee 802.1 ae and 802.1 af security in epon (1gepon and 10gepon) networks," Mar. 12 2013, uS Patent 8,397,064.

[41] O. Nakhila and C. Zou, "Parallel active dictionary attack on ieee 802.11 enterprise networks," in *MILCOM 2016-2016 IEEE Military Communications Conference*. IEEE, 2016, pp. 265–270.

[42] ——, "Circumvent traffic shaping using virtual wireless clients in ieee 802.11 wireless local area network," in *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 2017, pp. 52–56.

[43] M. Alyami, M. Alkhowaiter, M. A. Ghanim, C. Zou, and Y. Solihin, "Mac-layer traffic shaping defense against wifi device fingerprinting attacks," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–7.

[44] M. Alyami, "Mimicry-based-traffic-injection," GitHub, 2022. [Online]. Available: https://github.com/MnassarAlyami/Mimicry-Based-Traffic-Injection.git

[45] K. Moldeklev and P. Gunningberg, "How a large atm mtu causes deadlocks in tcp data transfers," *IEEE/ACM Transactions On Networking*, vol. 3, no. 4, pp. 409–422, 1995.

[46] J. Nagle, "Rfc0896: Congestion control in ip/tcp internetworks," 1984.

[47] M. Alyami, "Random-tcp-segmentation," GitHub, 2023. [Online]. Available: https://github.com/MnassarAlyami/Random-TCP-Segmentation.git

[48] C. Kiraly, G. Bianchi, F. Formisano, S. Teofili, and R. L. Cigno, "Traffic masking in ipsec: architecture and implementation," in *2007 16th IST Mobile and Wireless Communications Summit*. IEEE, 2007, pp. 1–5.

[49] N. Prates, A. Vergütz, R. T. Macedo, A. Santos, and M. Nogueira, "A defense mechanism for timing-based side-channel attacks on iot traffic," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[50] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *arXiv preprint arXiv:1708.05044*, 2017.

[51] C. C. Zou, N. Duffield, D. Towsley, and W. Gong, "Adaptive defense against various network attacks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1877–1888, 2006.

[52] L. Chaddad, A. Chehab, and A. Kayssi, "Opriv: Optimizing privacy protection for network traffic," *Journal of Sensor and Actuator Networks*, vol. 10, no. 3, p. 38, 2021.

[53] R. Underwood, J. Anderson, and A. Apon, "Measuring network latency variation impacts to high performance computing application performance," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 68–79.