Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment

Rahul Anand Sharma, Elahe Soltanaghaei, Anthony Rowe, Vyas Sekar

Carnegie Mellon University

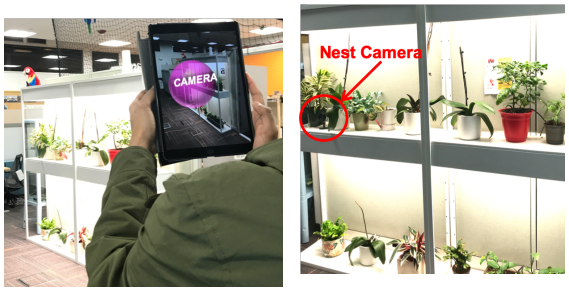
Abstract

Hidden IoT devices are increasingly being used to snoop on users in hotel rooms or AirBnBs. We envision empowering users entering such unfamiliar environments to *identify and locate* (e.g., hidden camera behind plants) *diverse* hidden devices (e.g., cameras, microphones, speakers) using only their personal handhelds. What makes this challenging is the limited network visibility and physical access that a user has in such unfamiliar environments, coupled with the lack of specialized equipment. This paper presents Lumos, a system that runs on commodity user devices (e.g., phone, laptop) and enables users to identify and locate WiFi-connected hidden IoT devices and visualize their presence using an augmented reality interface. Lumos addresses key challenges in: (1) identifying diverse devices using only coarse-grained wireless layer features, without IP/DNS layer information and without knowledge of the WiFi channel assignments of the hidden devices; and (2) locating the identified IoT devices with respect to the user using only phone sensors and wireless signal strength measurements. We evaluated Lumos across 44 different IoT devices spanning various types, models, and brands across six different environments. Our results show that Lumos can identify hidden devices with 95% accuracy and locate them with a median error of 1.5m within 30 minutes in a two-bedroom, 1000 sq. ft. apartment.

1 Introduction

Imagine a user walking into an unfamiliar environment such as a hotel room or AirBnB. Nowadays, the user has to be wary of wireless Internet-of-Things (IoT) devices being used to spy on them. These devices could be installed by the owner or by a previous guest. This threat is not just hypothetical; there are numerous reported incidents where IoT surveillance devices were used in AirBnBs [1–3, 5, 9, 11, 16], cruise ships [6], and motels [10]. A 2019 survey of 2,000 American travelers revealed that 58% were worried that their host had installed hidden surveillance equipment, and 11% of respondents had actually found a hidden camera in some past rental [13].

Ideally, we want to empower users so that as they enter an unfamiliar space, they can run an app on their personal handheld (e.g., phone or tablet). This app would report a list of *detected* and *identified* devices and their corresponding locations. "Detect," here, means knowing that there is some device (i.e., binary notification), "identify" entails knowing what type of device it is (e.g., type=camera), and "localize" entails knowing the device's location in the physical space (e.g., behind the plants). While cameras in particular are imminent privacy threats, in general we want to detect/identify and localize *diverse* hidden IoT devices, as these could also be potential threats for tracking users (e.g., [21, 25, 33, 66]).



(a) Overlaying AR object **(b) True location of the camera**

Figure 1: A snapshot of Lumos identifying a device and visualizing the location with ARKit

This problem is challenging due to two practical factors. First, users have limited visibility and control inside such an unfamiliar environment with their little knowledge of the devices and their wireless configurations; e.g., they cannot tap into network interfaces at wireless access points or instrument the environment. Second, users typically only have personal (commodity) handhelds and do not carry expensive hardware or specialized sensing equipment [15, 41]. Given our requirements and these constraints, existing methods are not sufficient for our context (see Table 1). For example, today's "spy-tech" solutions rely on manual and thorough scanning of the environment [4, 7, 8, 15, 18]. Other efforts focus exclusively on camera-specific effects (e.g., motion or light triggering) and do not generalize to other, more diverse hidden IoT devices [26, 51]. Similarly, network-based device fingerprinting solutions [45, 48, 53] rely on privileged access to the host network and fail in the presence of limited network visibility. Finally, many of these solutions cannot localize devices, and/or would need separate instrumentation of the environment [37, 59].

Approach	Compatible with			Localization Ability
	Personal Hand-helds	Limited Network Access	Diverse Devices	
Bug Finder [4, 15]	✗	✓	✗	✓
Camera Detector [7, 8, 18]	✓	✓	✗	✓
mmWave Sensing (E-Eye) [41]	✗	✓	✓	✗
Network Traffic at Router [45, 48, 53]	✓	✗	✓	✗
Camera Detection w 802.11 Packets [26, 42]	✓	✓	✗	✗
Lumos	✓	✓	✓	✓

Table 1: Comparing existing approaches vs. Lumos

This paper presents Lumos, a system that enables a user to identify and locate IoT devices in an unfamiliar environment using a commodity personal device. As a starting point, we focus on 802.11 WiFi connected devices, which represent a significant fraction of the IoT device market today [58]. At a high level, Lumos sniffs and collects encrypted wireless packets over the air (aka 802.11) to detect and identify the hidden devices. It then predicts the location of each identified device with respect to the user as they walk around the perimeter of the space. Our design makes three contributions:

Identifying diverse devices with limited features: Prior work associates IoT devices with signatures using higher-layer information IP, DNS, port numbers, and NTP protocols (e.g., [45, 53]). However, due to limited network visibility, we can only observe 802.11 headers with coarse attributes. To address these issues, we design a systematic machine learning (ML) framework, which considers a broad observable feature set, rather than handcrafted features [45, 53], both temporally and across packet header attributes. To tackle device diversity, we use multiple timescales in feature engineering to extract device-specific attributes. This allows us to generalize across a large set of device types from different vendors and with different hardware settings.

Data acquisition with limited knowledge: Even within a single protocol like 802.11, there is a large set of channels that the hidden IoT devices may use. In an unfamiliar setting, we have no knowledge of when, on what channels, and for how long each device is transmitting. Prior spectrum sensing approaches [50] and naive strategies for sequentially sampling the various channels are slow and miss capturing devices. Lumos addresses this challenge with a novel reformulation of the spectrum sensing problem to learn a coarse transmission pattern of each device over time and uses this to inform the channel sensing strategy.

Infrastructure-free device localization: Classical wireless localization systems rely on knowledge of the floor plan or spatial geometry, or require anchor points (e.g., [29, 54, 59]), which are infeasible in our problem setting. Lumos addresses these challenges by leveraging mobile phone sensors and the correlation of the user's motion with variations in signal strength. By requesting the user to take a short walk around the perimeter of the space, we can estimate the location of the IoT device from sparse measurements.

We implemented Lumos on two platforms, a MacBook and an iPhone, and combined it with an augmented reality (AR) feature that overlays the device type on the estimated device location relative to the user (Figure 1). This provides users with a virtual world view of the physical space. We prototyped Lumos using a laptop (2018 MacBook Pro) and an Intel RealSense Tracking Camera T265. The T265 acts in place of the visual inertial odometry (VIO) provided by augmented reality frameworks like AR Kit/Core [20, 32] on mobile phones. Since promiscuous WiFi access is currently disabled on mobile phones, we implemented Lumos as an iOS app running on an iPhone paired with a Raspberry Pi (Rpi) over Bluetooth.

We evaluate Lumos in six different environments and across a wide spectrum of IoT device types, for a total of 44 devices. Our evaluation shows that we can accurately identify device types by 95% in under 30 minutes, and the devices are then localized with a median localization accuracy of 1.5m with only one walk around the perimeter of each space of around 1000 sq. ft. We have released our code on `https://bit.ly/lumos-code` and have uploaded a demo of our system at `https://youtu.be/QwMXiyn-e28`.

2 Problem Setting, Threat Model, and Scope

Our work deals with an *attacker* who has placed *IoT devices* to spy on *users* in an *unfamiliar environment* such as an AirBnB or hotel room. Figure 2 shows an overview of the key actors and resources. Our setting consists of two actors: an *Attacker* and a *User*. An attacker is either the host or a previous guest who wants to use IoT devices to spy on a user/guest (in an AirBnB or a hotel room) who has entered this unfamiliar environment. The user wants to identify and localize these hidden IoT devices.

These two actors interact with three key resources: *Physical Environment*, *IoT Devices*, and the *Wireless Network*. In our setting, the *Environment* could be a single room in a hotel or a complex multi-room setup in an AirBnB. *IoT Devices* could be of various types, such as cameras, speakers, plugs, vacuum cleaners, and more. We focus on devices that communicate over WiFi, as this is the most prevalent method of wireless communication. These devices are connected to the Internet via an 802.11 *Wireless Network* controlled by the attacker.

Attacker Capabilities: Next, we formulate the adversary's capabilities and constraints.

- **Physical Environment:** The attacker has complete control of the environment ahead of time to modify the environment and to install and hide IoT devices.
- **IoT Devices:** The attacker purchases and places off-the-shelf wireless IoT devices to spy on the User. They can also control various device settings such as resolution, sensitivity, etc., through device APIs. Similar to prior

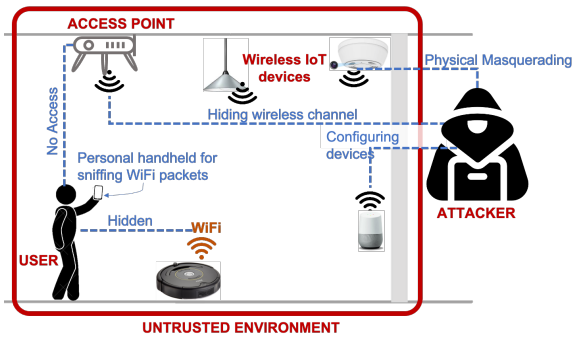


Figure 2: System model with the user, the attacker, and hidden IoT devices in an unfamiliar environment

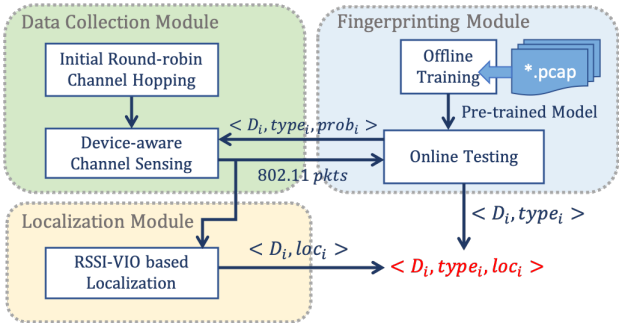


Figure 3: System overview with three main modules

work [26, 42, 45, 48, 53], we assume that an attacker does not alter the fundamental behavior of these devices, such as hacking the firmware, changing the network protocol, or changing wireless transmission behavior. However, the attacker can physically masquerade devices; e.g., a camera hidden inside a thermostat [5] or a smart electric plug that doubles up as a camera [14]).

- **Wireless Network:** The attacker has complete access to the 802.11 wireless network and access point. They can take a variety of measures to hide the IoT devices. For instance, they can use a separate WiFi network for the IoT devices and provide the user access to a separate guest network. Furthermore, they can assign devices to different 802.11 wireless channels, enable encryption (e.g., WPA2/WPA3), and hide the SSID of the network(s) the IoT devices are connected to.

User Capabilities and Constraints: We assume the user has access to a personal device such as a mobile phone, tablet, or laptop, and no other equipment. We assume that they can enable monitor/promiscuous mode on the personal device for wireless packet sniffing.

- **Physical Environment:** The users have access to the physical space to search and walk around, but they can not instrument new hardware/equipment in the physical environment.
- **IoT Devices:** The user does not have any knowledge of hidden IoT devices. They don't know how many devices are in this unfamiliar environment, what types of devices are installed, the access point and wireless channel(s) they are using, or where they are located.
- **Wireless Network:** The user has limited access to the wireless network; e.g., given access to a *guest* network which could be different from the network(s) that IoT devices are operating on. They can still sniff encrypted broadcast WiFi 802.11 packets (across all channels) over the air.

3 System overview

We envision that a user enters an unfamiliar space and runs the Lumos app on their phone or laptop to identify hidden IoT devices. The Lumos app can run in the background, while the phone is sitting in a corner collecting raw wireless (i.e., 802.11) packets. At any point, the user can request a report, and Lumos will provide the list of identified devices so far. Each identified device is depicted using an augmented reality frontend to assist the user in finding the hidden IoT devices.

Lumos consists of three main modules:

- **Device Fingerprinting Module:** There are two main challenges we need to address. First, unlike prior work, we only have access to MAC-layer information based on 802.11 headers. Second, we need to handle a diverse set of devices with different transmission rates. Section 4 explains how we address these challenges by developing a systematic machine learning approach.
- **Data Collection Module:** For fingerprinting to work well, we need a sufficient number of packets from all devices. However, sniffing the packets transmitted by each hidden IoT device requires knowing their associated wireless channels. Unfortunately, this information is not available in a limited access environment; e.g., the IoT devices can operate on a different network than the guest wireless network. We design a *device-aware channel sensing* approach, explained in Section 5, that learns the traffic pattern of each device overtime to decide when, and for how long, to sense each wireless channel.
- **Localization Module:** At first glance, this seems similar to classical wireless localization [59]. Unfortunately, we cannot directly use these as they require infrastructure instrumentation [29, 59, 64, 65], prior knowledge of the floor-plan [22, 27, 39, 46, 60], or fine-grained channel measurements [24, 37, 44, 54, 56, 62]. To address these limitations, Lumos fuses signal strength measurements that are available in 802.11 packets with VIO (Visual Inertial Odometry) traces available in mobile phones by asking the user to take a short walk around the perimeter of the space. Section 6 elaborates on how Lumos locates devices from sparse measurements.

End to End View: Figure 3 shows an end-to-end view of Lumos. First, we use an offline training phase in Lumos' fingerprinting module for common IoT devices. When a user enters a new unfamiliar space, Lumos runs a client agent (e.g., on the phone) which sniffs the ongoing 802.11 traffic. The associated packets to each device are then inspected through

the *fingerprinting module* to identify these devices. Since the user has no information about the wireless channel on which these devices are operating, Lumos uses a *device-aware channel sensing mechanism* to decide what channel to sniff, when, and for how long. Lastly, Lumos uses an *RSSI-VIO based localization technique* to estimate the coarse location of each identified device with respect to the user by requesting the user to walk once around the perimeter of the space.

4 Device Fingerprinting Module

Previous efforts have identified network layer features from IP, DNS, and NTP packets to be highly correlated with the IoT device type [45, 48, 53]. However, they assume privileged access to the router and network layer headers to obtain this information. In an unfamiliar environment with limited access, encrypted wireless 802.11 headers are the only coarse attributes available to the user's personal device. This is even more problematic when dealing with a diverse set of IoT devices with different transmission behaviors and communication protocols.

To address these issues, we design a systematic machine learning framework that extracts the effective features for each device type by considering the broadest feature set temporally and across the available packet header attributes. In addition, our proposed framework automatically tunes the timescale of the aggregate features (e.g., mean, max, std, etc.) based on the transmission rate of each device. As a simplified starting point, we first start with a single channel scenario, where all IoT devices are operating on the same channel. In Section 5, we relax this assumption and generalize our proposed algorithm across multiple unknown wireless channels and will explain how to integrate the classification module with the data acquisition for the multi-channel operation of IoT devices.

Lumos' classification engine receives the wireless 802.11 packets transmitted to or from all available IoT devices as the input. Then, it groups the collected packets based on their MAC addresses and predicts the *device type* for each MAC address by using a systematic feature engineering and classification method. Next, we explain how Lumos extracts relevant attributes from 802.11 packets, and how these attributes are aggregated over time to account for the diversity of devices. Finally, we explain Lumos' classifier.

4.1 Feature Engineering

We begin by discussing the available 802.11 layer features we can use for fingerprinting and how we can select them.

Available Features: Figure 5 shows a sample 802.11 wireless packet. The packet contains metadata attributes, such as packet inter-arrival times and packet sizes, which can serve as the basis for defining features for fingerprinting.

Some prior efforts have handcrafted features for device fingerprinting (e.g., [61]). Many of these features, such as packet length, could still be extracted at the 802.11 layer. If we take a look at Figure 4, we can see that this feature is still useful to distinguish between various IoT devices. We also have access to more classes of features such as packet *subtype* which are

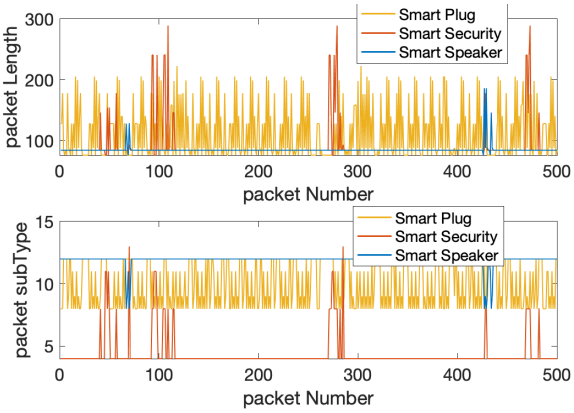


Figure 4: The important features are device-specific, due to the diversity of IoT devices and their heterogeneous traffic patterns

specific to the 802.11 protocol. For example, the *subtype* attribute is used in Nest doorbells for informing the access point that the device is going into sleep mode, while this attribute is used differently in Nest cameras. Handcrafting these features is a challenge given the high heterogeneity of IoT devices.

Hence, instead of defining fixed handcrafted features, we automatically extract relevant attributes per device. To this end, we start by collecting all possible 802.11 packet headers and then extract all attributes from each packet. This results in a total of 125 (max) attributes. However, some of these attributes have the same value across different devices (e.g., AP-specific attributes) and do not carry any useful information. We discard these attributes to simplify the processing and prevent over-fitting. In our model, after this pruning step, 52 out of 125 attributes remain.

Multi-Time Resolution Aggregation: Given the set of attributes, we then construct the feature set by considering different temporal aggregations of each attribute. Specifically, we define a sliding window of time and apply different aggregate functions on each raw attribute. These aggregate functions include mean, standard deviation, median, max and min, sum, entropy, histogram (normalized frequency count of each bin), and the number of unique values in a given time window.[1]

A key challenge, however, is that using a fixed size of time window does not generalize well across IoT devices with varying packet transmission rates. On one end, a very small aggregation window is prone to noise, while on the other end, a very large aggregation window will dilute the variations, which is a classical bias-variance tradeoff in ML. Ideally, we want a small aggregation window for high rate transmission devices, but a large aggregation window for a low rate transmitting device. To achieve this goal, we design a multiple timescales scheme to pick a time window suitable for each device's transmission pattern. As shown in Figure 5, we define a set of time windows with different lengths at a given time

[1] Some previous work defines aggregation window in terms of the number of packets. However, this is not extendable to diverse devices, especially if they do not transmit often [48].

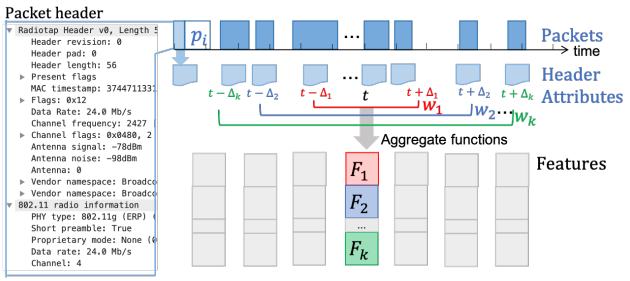


Figure 5: Lumos uses multiple time resolutions to define features for capturing device behaviors

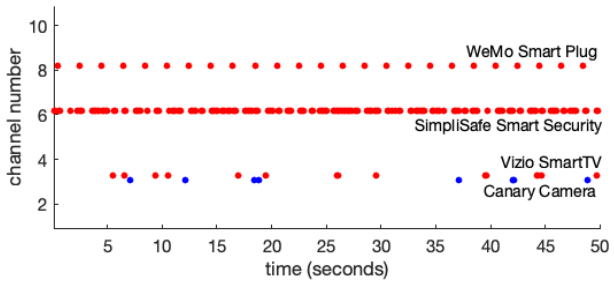


Figure 6: An example of devices spread across multiple channels. Lumos uses a channel sensing strategy to minimize the time needed to ensure it logs a sufficient number of packets for each device.

t and then apply each aggregate function on all defined time windows. The feature vector at time t is the concatenation of all aggregate functions applied at all time windows.

Feature Post-Processing: After computing the features for each time window, we apply two post-processing steps to prepare the data for ML training. First, to handle the diversity of feature value ranges, which can adversely affect the training,[2] we standardize the features [17] while maintaining the distribution of values. Second, we remove correlated features to avoid over-fitting in the training phase. Specifically, we use two feature reduction techniques: (1) Selecting the top ten features that have the highest mutual information score [57], and (2) We compute the cross-correlation of features, and for those features that have a higher than 95% correlation score, we only keep one of the correlated features and drop the others.

4.2 Model Training and Inference

Training: After post-processing the features, we train different kind of ML classifiers and evaluate their performance on a separate held-out validation set. We picked XGBoost as our classifier, as it had the highest accuracy and is also fairly robust to high dimensional data. We trained our classifier on the final set of features and define the following device types as the classes: Smart Camera, Speaker, TV, Plug, Security Systems, Vacuum, Kitchen Appliances, Bulb, and Doorbell. There are typically two types of classifiers that can be trained for such a problem: multi-class and one-vs-rest. A multi-class classifier learns a single classifier for all the classes, while one-vs-rest learns one classifier per class. Due to the high diversity of IoT devices, we select a one-vs-rest classifier. The intuition is that some IoT devices transmit much more frequently than others, which leads to an extreme class imbalance, both during training and testing. While multi-class classifiers are prone to be biased towards the majority class, the one-vs-rest classifier can independently learn each device. In addition, the relevant and informative features for each device type could be different, so picking a set of globally relevant features for all devices is a sub-optimal choice. Instead, we define the one-vs-rest classifier to learn important features on a per device basis. As such, we train a binary classifier per class.

[2]For example, a packet size could change from 64 to 1000, but a packet type only takes discrete finite values of either 0 or 1.

Inference: During inference, we sniff packets on a channel and group them by their MAC addresses. For a device, let us denote P as the set of sniffed packets for that device. Then, we define the center of the time window as t, which corresponds to packet arrival times and computes the feature vector F_t based on the algorithm explained in Algorithm 1. Next, Lumos applies all the K classifiers corresponding to each one-vs-rest device type to F_t and computes the probability of predictions as

$$L_{t,k} = M_k(F_{t,i}), \forall\, k = 1:K \quad (1)$$

where M_k is the one-vs-rest classifier for device type k and $L_{t,k}$ is the probability of predicting the type of device as k at time t. We select the final label of the feature vector F_t as G_t

$$G_t = argmax_k L_{t,k} \quad (2)$$

Lumos then performs a majority voting for G_ts in a given scanning period to assign a single label to the device. The same process is repeated for the sniffed packets of other available devices (every unique MAC address).

5 Device-Aware Channel Sensing

In the previous section, we presented the fingerprinting module under a simplified assumption where all the devices operate on a *single known channel*. In practice, however, we need Lumos to work in an environment where the IoT devices are possibly on different wireless networks (shown in Figure 6) spread over 30 channels across 2.4 and 5GHz WiFi frequency ranges. Thus, we need a mechanism to monitor various channels and "hop" across them in order to collect wireless data from all IoT devices for the fingerprinting step. However, note that we have no knowledge of what channel, when, where, and for how long each device is transmitting.

This problem, at a high level, is similar to the spectrum sensing [50, 63] idea in wireless networks where the goal is to sense as many packets as possible across wireless spectrum within a given time budget. In spectrum sensing, however, the objective is to maximize the total number of received packets across different wireless channels. However, our problem is different—we need to capture a sufficient number of packets from *each* active device to identify its device type.

In the rest of this section, we first start with a hindsight optimal formulation which assumes that we know the traffic

behavior of each device ahead of time. While this assumption is not practical, it allows us to formally define the problem before we relax this assumption.

Hindsight-Optimal Problem Formulation: We consider a setting where we chunk time into epochs, and in each epoch, our channel sniffer can sense at most one channel.[3] Suppose we have a total time budget of T epochs and C channels and M devices assigned to various channels. Our goal is to determine a sensing schedule to cover as many devices as possible. For any given time epoch, let $sense_{j,t}$ ($j \in [1,C]; t \in [1,T]$) be a binary decision variable denoting if channel j should be sensed at time t.

Note that in order to accurately fingerprint IoT devices, we need to collect a *sufficient number of packets* from each IoT device, so that the ML models have accurate features. Let *NumThresh* denote the *sensing threshold* determined based on the requirements of the classification engine to correctly identify the types of devices. Let num_i denote the actual number of packets sensed from a device i given our choice of $\{sense_{j,t}\}$. This depends on how active the device is. To this end, we assume that we know the activity matrix (a constant input) $A_{i,j,t}$ denoting if device i is active on channel j at time t. Let $covered_i$ be an indicator binary variable denoting if device i has a sufficient number of packets; i.e., $num_i \geq NumThresh$.

Formally, the hindsight-optimal problem formulation can be written as an Integer Linear Program (ILP) as follows:

$$\forall j,t: \quad \sum_j sense_{j,t} <= 1 \tag{3}$$

$$\forall i: \quad num_i = \sum_{j,t} A_{i,j,t} \times sense_{j,t} \tag{4}$$

$$covered_i = \begin{cases} 1, & \text{if } num_i >= NumThresh \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$\forall j \in [1,C], t \in [1,T]: sense_{j,t} \in \{0,1\} \tag{6}$$

$$\forall i \in [1,M]: covered_i \in \{0,1\} \tag{7}$$

$$\forall i \in [1,M]: num_i \in Integer \tag{8}$$

Here, Eq 3 captures that we can sense at most one channel in any given time epoch. Eq 4 captures the total number of packets sensed per device, and Eq 5 captures that each device is successfully sensed if we have more than *NumThresh* packets. The last three equations simply capture the constraints on the variables.

This hindsight optimization problem can be solved using an ILP solver [19, 47], *if the activity matrix is known*; i.e., if we already know when each device transmits a packet on its assigned channel. In our setting, however, the activity matrix is unknown and we need to solve it without this information. Next, we explain how Lumos predicts the activity matrix of each device based on the coarse collected data.

Prior Work on Spectrum Sensing and Limitations: One approach to solve the above challenge of the missing activity matrix is to view this as a multi-armed bandit based problem, as done in SpecInsight [50]. Specifically, to estimate the next channel number and the time of hopping, they formulate the problem as a multi-armed bandit game [23] and use the ε-*greedy strategy* [55] as the solution, wherein it picks a random channel with probability ε and picks the channel with maximum reward with probability $(1-\varepsilon)$. The choice of ε controls how much to rely on the learned information and is set to 0.1. To define the reward function, SpecInsight uses an indication of how close we are to receiving a packet from the active devices. So, for a device d at time t, the reward function is defined as

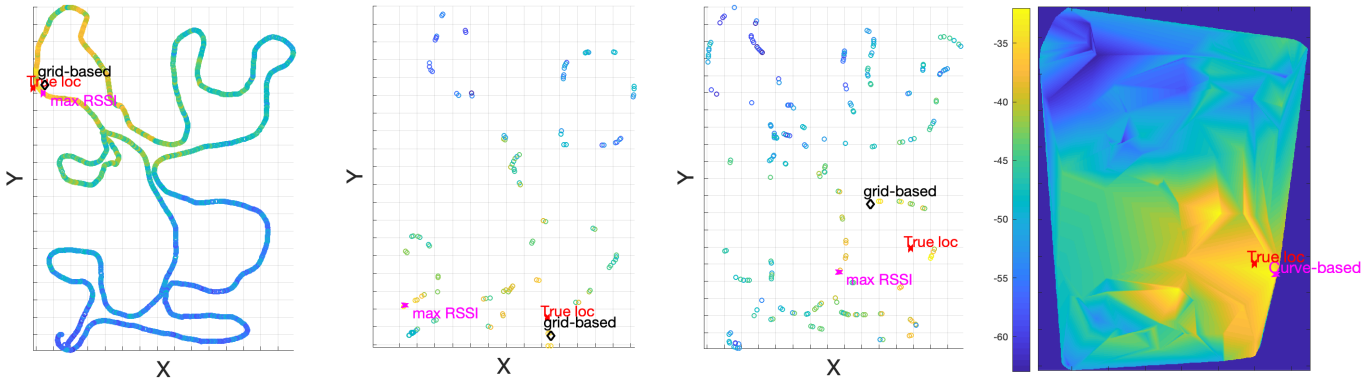
$$R_d(t) = max(1 - \frac{T + \mu_i * \lceil (t-T)/\mu_i \rceil - t}{\mu_i})$$

where T is the last time a packet was observed, and μ_i represents the mean packet inter-arrival time for the device. This reward function assumes that the next packet will arrive at time $T+\mu$, $T+2\mu$, and so on. At time t, the next packet is expected to arrive at time $T + \mu * \lceil (t-T)/\mu \rceil$. The value of ε controls how much to rely on mean packet inter-arrival estimates. This approach has several issues that make it ill-suited for our problem. First, the proposed reward function tries to capture all packets from every device. A high transmission rate device has lower packet inter-arrival times, and as a result, high reward value. This results in missing packets from a low transmission rate device as it is still trying to collect every packet from a high transmission rate device. Second, it calculates the mean inter-arrival time from the previously captured packets. However, some packets transmitted by a device may be missed while sniffing in another channel, resulting in inaccurate estimation of averaged inter-arrival time. This penalty is huge for low transmission devices, as we now need to wait even longer to capture sufficient packets. For example, if a device transmits packets at time t = 1, 3, 5, 7 . . . seconds and we captured packets at time 1 and 7 seconds, our estimate of mean packet inter-arrival time would be 6 seconds instead of the actual 2 seconds. To capture the same number of packets, it would take 3x longer. Moreover, there are more than 30 possible wireless channels, but a majority of them might not be active in the vicinity of a user, so it ends up wasting a lot of time sensing traffic on inactive wireless channels.

Our Approach: We address these shortcomings as follows. First, to avoid wasting time sensing inactive channels, Lumos performs a quick round robin iteration across all wireless channels to discover the active channels. We can discover the active channels based on whether we sense any beacon frames. The key insight is that the presence of an IoT device in a channel corresponds to the presence of an active access point to communicate with, which is periodically transmitting beacon frames. Therefore, a simple round robin channel hopping is sufficient to find the subset of active channels.

Next, to make our scheme unbiased towards low transmission rate devices, we modify the problem formulation to make reward 0 for a device if we have sensed enough packets from that device. It enables Lumos to handle IoT devices with diverse transmission behaviors. For high transmission rate devices, we can sense a sufficient number of packets very quickly and its reward is reduced to 0 so that Lumos can now

[3] More powerful SDR hardware [30] can sense in parallel but is not a commodity handheld solution.



(a) Nest Camera (high traffic) **(b) WiZ Smart Bulb 5 (low traffic)** **(c) WiZ Smart Bulb 2 (irregular traffic)** **(d) WiZ Smart Bulb 2 (curve fitting output)**

Figure 7: While the correlation of maximum measured RSSI and VIO is sufficient to accurately locate high transmission devices, our curve-fitting approach is more robust for IoT devices with irregular and low rates of traffic

focus on capturing packets from low transmission devices.

To address the issue of incorrect packet arrival time estimates, Lumos learns the inter-arrival time from a coarse estimate of its device type. It uses the classification engine to determine the device type using a small number of packets collected up to that time instant. Since this prediction of device type is based on very few packets, the classifier is prone to errors. Our empirical studies show that the correct device type is usually within the top three predictions. For each device, we make a prediction of its device type and fetch the corresponding mean inter-arrival times of the top three predictions directly from the training data as shown in Algorithm 2. At a first glance, the formulation might look circular, as we are trying to develop channel sensing to capture data for device fingerprinting while at the same time using fingerprinting for our channel sensing scheme. This is possible because even fewer packets from a device are good enough for coarse fingerprinting (correct device type in top three predictions). Currently, we pick a single packet inter-arrival time estimate for each device type, but this could easily be modified to pick multiple inter-arrival time estimates.

Since in our scenario we are using a coarse classifier and its predictions might be inaccurate, we take the maximum of the above reward function to maximize the number of packets captured under uncertainty. After switching to a channel, Lumos spends a fixed time (10 seconds) on that channel. We set R_d for a device to be 0 when we have sensed more than N packets (e.g., 50 packets) from a device. In addition, the reward for a channel c is defined as

$$R_c(t) = max(R_d), \quad \text{if } R_d \text{ exists}$$
$$= RAND(0,1), \quad \text{otherwise}$$

where

$R_d(t) = 0$, if sensed more than N packets from device d

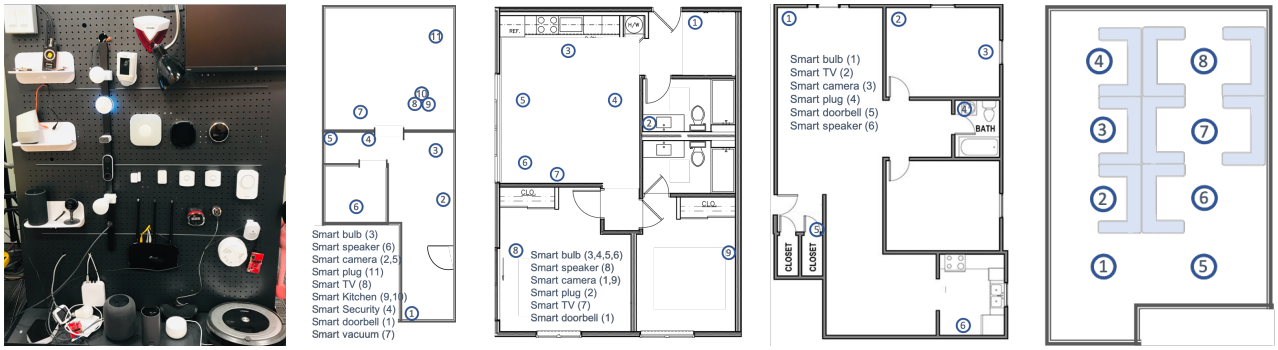
$$= max(1 - \frac{T + \mu_i * \lceil (t-T)/\mu_i \rceil - t}{\mu_i}), \quad \text{otherwise}$$

6 Localization

Recall that in addition to detecting and identifying hidden IoT devices, we need to locate the detected devices to provide some situational awareness to the user. RF-based source localization is an extremely well-studied problem with a number of techniques described below. The main contribution of our work is to recognize the potential of using highly accurate relative tracking of mobile phones through recent advances in Visual Inertial Odometry (VIO) in combination with sparse RSSI sampling. Due to the logarithmic relationship between RSSI and distance, it is possible to accurately localize a source when the receiver is in close proximity. This simple but effective technique allows us to rapidly estimate and visualize the position of RF sources in augmented reality.

Intuitively, we combine the RSSI measurements with the relative user position determined by VIO on smartphones. RSSI provides a coarse estimate of distance between each IoT device and the user's phone, while VIO determines the change in position and orientation of the user over time by integrating the IMU readings with camera frames. At a high level, we can take multiple measurements of the signal strengths from different distances to the devices by asking the user to walk around the environment. While this is a sparse set of measurements, it is sufficiently spatially diverse to locate hidden IoT devices within 1 meter of accuracy.

As the user walks in the space, Lumos collects the (x,y,RSSI) samples, where x and y are relative to the initial point where the user starts walking. Figure 7 demonstrates the measured samples for three different IoT devices as the user walks around a two-bedroom house (the true location of each device is marked in red, and the point colors represent the RSSI values). We can see that as the user walks closer to each device, the RSSI values corresponding to those data points increase and then reduce as she walks away from the device (shown in Figure 7). Lumos leverages the spatial measurements of RSSI values and their variations to estimate the location of each device.



(a) Training Setup **(b) Test Office** **(c) Test Apartment 1** **(d) Test Apartment 2** **(e) Test Lab Space**

Figure 8: Our mini IoT testbed and floor plans of the different experimental setups

A naive approach for estimating the location of IoT devices is to to pick the (x,y,RSSI) sample with the highest RSSI value, which represents the closest the user got to the device. However, this approach only works if the user walks within a very close proximity of the device. Grid-based optimization [35] is another technique that is widely used for parameter estimation with sparse samples. In this case, Lumos forms a grid in the user's walking region (with cells of 0.5m by 0.5m) and calculates the average observed RSSI values for each grid. The center of the grid with the maximum average RSSI is then selected as the device location. We can see in Figure 7 that the grid-based localization outperforms the maximum RSSI approach in lower rate devices such as WiZ Smart Light Bulbs. However, it is still sensitive to noisy measurements, especially for devices with irregular transmission rates (WiZ2 vs. WiZ5 in Figure 7).

To further refine location, Lumos takes the sparse samples and fits a surface of the form v = f(x,y) to the scattered data in the vectors of (x,y,RSSI). For this, Lumos forms 1 by 1 centimeter grids of (xq,yq) as the query points and interpolates 3D triangulation-based linear surfaces [40] at each point. Then the regional maximum of the surface is extracted as the estimated device location. A sample of the interpolated surface is shown in Figure 7 for the WiZ2 Smart Light Bulb, and we can see that this interpolation technique outperforms the other two methods. As such, Lumos combines the (x,y,RSSI) data points for each device to locate them relative to the user's location. While this technique cannot estimate the absolute location of devices in the physical space, this relative localization is sufficient to overlay a virtual object relative to the position and orientation of the user's phone.

7 Evaluation

We implemented Lumos in multiple unfamiliar environments and diverse hidden devices. Our main results are:

- Lumos can accurately identify diverse devices with 95% accuracy in under 30 minutes. This is comparable to techniques using more fine-grained features from higher network layers at the router, such as IP, DNS, etc.
- Lumos' channel sensing outperforms baselines such as random, round robin, and state-of-art spectrum sensing techniques.
- Our localization system can locate devices within 1.5m with a single random walk through the space.
- Lumos can identify previously unseen devices of the same type from different vendors and is robust across typical changes in device settings.

Category	Devices
Camera	Nest, Canary, Ring, Blink, EZVIZ, TP-Link KC100, TP-Link KC120, D-Link, Geeni, NightOwl, HIDVCAM, OVEHEL, LookCam, MiniSpy, AlphaTech
Microphones	Google Home, Amazon Echo, SONOS, Amazon Show, Apple HomePod, Lenovo Smart Clock
Doorbell	Nest Doorbell, Kangaroo, Ring
Security	Simplisafe, ADT, Ring
TV	Vizio, Panasonic, TCL
Plug	Amazon, Wemo, TP-Link, Jinvoo Smart Plug, Gosund Smart Power Strip, TP-Link Power Strip
Kitchen	Anova Cooker, iKettle
Bulb	WiZ1, WiZ2, WiZ3, WiZ4
Vacuum	Roomba & Deebot

Table 2: Devices used as candidate hidden devices

7.1 Implementation and Experimental Setup

Prototype: Lumos needs to sniff 802.11 packets over the air, which is currently disabled on mobile phones without special permission from the manufacturer. There are some device-specific workarounds [12] for WiFi sniffing using rooted Android devices, which shows there is no fundamental hardware/software limitation in providing such capability and this functionality could be unlocked given enough justification. Alternatively, we develop two proof-of-concept implementations of Lumos using commodity hardware:

- Using a MacBook Pro(2018) to sniff 802.11 wireless traffic and an Intel RealSense Camera T265 for capturing the

VIO traces. This roughly approximates the performance of ARKit/Core available on smartphones.

- Using a combination of an iOS device and a Rpi, Lumos runs as an application on the phone and uses Bluetooth to communicate with Rpi. The Rpi is used to sniff 802.11 wireless traffic (since wireless sniffing is currently disabled on mobile phones), and the phone is used to capture VIO trace using ARKit. Our mobile application is developed with Unity and can be compiled for iOS or Android. since we only rely on Visual Inertial Odometry module in smartphones for localization, Lumos can run even in older smartphones with basic camera and IMU sensors.

For the experiments below, we use the first setup.

Devices: Table 2 shows the devices we use for evaluation. We chose these to cover the major types of IoT devices available through retailers such as Amazon or Target. We include many cameras and microphones, as they collect the most privacy-sensitive data about a user. In addition, each category has multiple devices of the same type to avoid over-fitting to a particular vendor. We also included multiple devices from the same vendor to highlight the operational differences of different types of devices.

Environments: For testing, we deployed a variable number of devices in four different physical spaces (a total of six setups) as shown in Figure 8. Our first deployment was in an office space wherein a total of eleven IoT devices were deployed in various locations and they all were connected to a single access point. To test the performance of our system in new settings with different background traffic, we deployed a subset of devices in two separate houses with different floor plans, shown in Figure 8. We also performed a more comprehensive evaluation in a lab space with various device setups, each of which included a different density of devices at all eight locations. These setups include Low with one device from each category at each of the eight locations, Medium with two devices from each category, and High with all devices across the eight different locations shown in Figure 8e.

In all of these experiments, the IoT devices operated in a normal steady-state mode; the occupants used the same WiFi access point without any instructions, and the background traffic from non-IoT devices existed. In addition, the traffic from Apartment 1 and 2 included IoT devices that were unseen in the training dataset. For example, in Apartment 1, there were instances of smart TVs and smart light bulbs that do not exist in the training set.

For localization, we asked users to walk once around the perimeter of the experimental spaces with no other instructions and each user picked the direction or walking route freely. We repeated this process 5 times in each experimental location for more thorough evaluations.

Baselines for Comparison: To the best of our knowledge, there is no prior work that presents an end-to-end combination of device fingerprinting, cross-channel data acquisition, and localization. As such, we use baselines for individual modules.

- *Fingerprinting:* We use the device classification technique proposed by Sivanathan et.al. [53]. Note that this technique uses raw and fine-grained information from higher layers (TCP, DNS), while Lumos only uses (encrypted) 802.11 packets. So, the goal is to show that even without accessing these fine-grained features, Lumos is capable of achieving similar performance.
- *Channel Sensing:* We compare Lumos to SpecInsight [50], the closest related work developed for spectrum sensing.
- *Localization:* We compare our proposed surface-fitting based localization technique against maximized RSSI and grid-based techniques presented in Section 6.

Environment	# Devices	Device Identification	Localization
Office	11	95.64%	1.6 m
Apartment 1	10	95%	1.4 m
Apartment 2	6	98.05%	1.5 m
Lab Space-Low	9	95%	1.6 m
Lab Space-Medium	18	95.02%	1.6 m
Lab Space-High	44	96%	1.5 m

Table 3: End-to-end results

7.2 End-to-End Performance

We run end-to-end evaluations of Lumos in four separate settings. We ran our system for 30 minutes of scan time (27 minutes of wireless sniffing followed by 3 minutes of walking) and we report the final device identification accuracy and localization error. In the first three buildings shown in Figures 8b-8d, we used the pre-deployed IoT devices and their original setups as more natural evaluations. For the lab space setup (Figure 8e), we ran more sensitivity experiments; e.g., deploying at least one, two, or three devices from each category in each of the eight locations. The results are reported in Table 3. There was only one access point in all four settings, and all the devices were in the same channel. However, the channel was unknown to the user. (We evaluated the sensitivity to multiple active channels in subsequent evaluations.)

As we can see in Table 3, Lumos can identify the type of devices with an accuracy of 95% to 98%. Figure 9 shows the localization performance of Lumos across the four testing environments. We can see that Lumos achieves a median localization accuracy of 1.5m, which is sufficient to roughly locate the IoT devices in the space. Determining a sufficient level of accuracy for detecting a device is subjective; however, 1-2 meters provide users with a reasonable starting point. Figure 1 shows a 1 meter transparent sphere floating above a shelf with a localized IoT device using Apple's ARKit for tracking. The origin for the ARKit session was manually aligned with the origin from our T265 tracking system to simulate the expected final AR performance. One would imagine as new localization technologies like UWB gain traction, the expected accuracy would dramatically increase.

7.3 Device Fingerprinting Sensitivity Analysis

Device-Based Confusion Matrix: Figure 10a and Figure 10b compare Lumos' fingerprinting performance with the baseline method across different device types. Among all

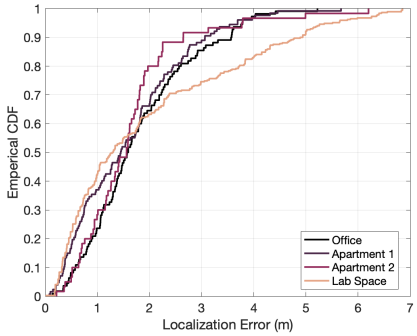
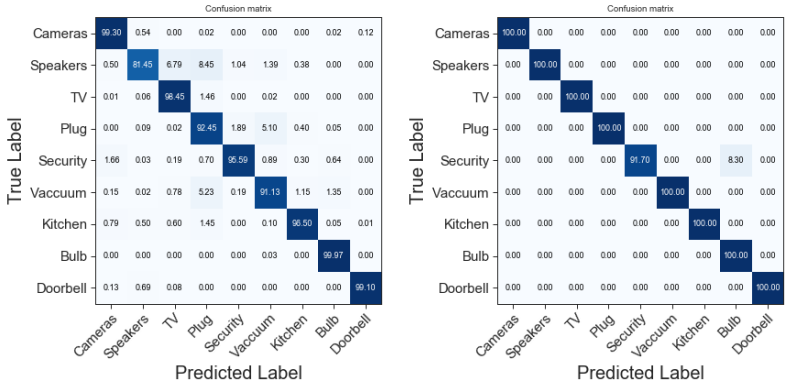


Figure 9: Lumos' localization module can achieve a median accuracy of 1.5m across three different test environments



(a) Lumos performance **(b) Baseline performance [53]**

Figure 10: Lumos can accurately identify the types of different IoT devices with accuracy comparable to alternatives that assume full network access

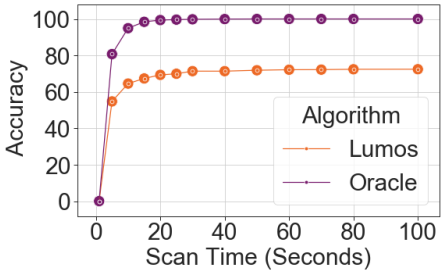


Figure 11: Lumos receives at least one packet from each IoT device at 80 seconds and can accurately detect the device types by 92%

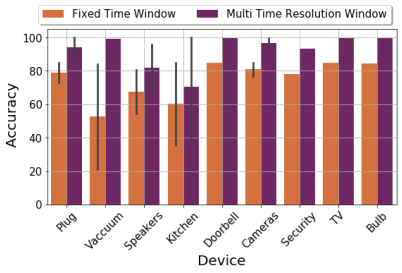


Figure 12: Multi-time resolution aggregation improves Lumos's performance from 72.34% to 90.22%

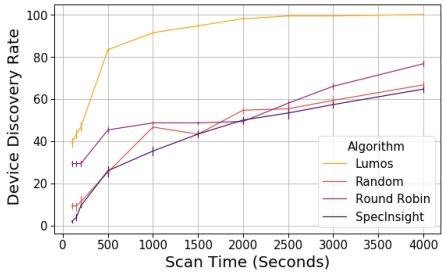


Figure 13: Device-aware channel sensing outperforms the baselines by discovering 2x more devices in a fixed scan time.

the device types, we can see that Lumos achieves very high accuracy in detecting cameras and lower accuracy in detecting smart light bulbs. This is mainly due to the packet transmission rate. Cameras usually have a very high data transmission rate as they are continuously streaming video frames. In addition, the format of transmitted data (image or video) is unique enough to create a distinct signature. On the other hand, smart plugs have very infrequent data transmission, providing little information for fingerprinting. Compared to the baseline, Lumos has comparable but slightly lower accuracy than prior work, which has access to more fine-grained features from higher network layers (e.g., flow duration, port numbers, and DNS data). This validates our hypothesis that 802.11 packets contain enough information for fingerprinting IoT devices.

It should be noted that some of the IoT devices in Apartments 1 and 2 (such as smart TVs and smart light bulbs) are previously unseen instances; i.e., the ML models are not trained for that specific device. However, we can see that Lumos can still accurately identify these devices based on their common traffic behavior to their device type.

Impact of Classifiers: To evaluate the impact of classifiers on the performance of the system, we calculated the raw classification accuracy for different classifiers before applying majority voting. Table 4 shows the classification accuracy for the top four accurate classifiers, and we can see that XGBoost performs best across different device types by achieving an average accuracy of 95.26%. The automatic feature selection in XGBoost and its robustness to high dimensional data are the main factors that make this classifier suitable for our use case.

Classifier	Accuracy
Extra Trees	83.88
Random Forest	83.79
AdaBoost	93.20
XGBoost	95.26

Table 4: Comparison of different classifiers

Impact of Multi-Time Resolution Aggregation: Next, we evaluate the utility of multi-time resolution features in fingerprinting. We train a new classifier using a fixed aggregate time window of 20 seconds, compared with the full system including multiple timescales between 1 second

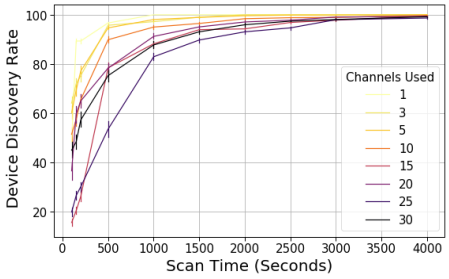


Figure 14: Lumos can discover all active IoT devices in 16 to 50 minutes depending on the number of active channels

and 20 seconds. As shown in Figure 12, using multi-time resolution aggregation improves the average classification accuracy from 72.3% to 90.2%.

Impact of Scan Time on Fingerprinting: Next, we analyze how the classification module performs for different scan times. To isolate the impact of the fingerprinting module from channel hopping, here we assume that all devices are in a single channel which is also known to the user.

To put our results in context, we consider a hypothetical oracle that is able to identify a device as soon as it observes a single packet. As shown in Figure 11, the oracle achieves 100% accuracy at 80 sec; i.e., it has received at least one packet from all IoT devices. This is a lower bound on the time to classify, since we need to observe at least one packet.

In this respect, we see that with only 40 seconds worth of capture, Lumos can achieve 92% accuracy. Across all devices, the average number of packets received at 80 seconds is 20 packets, with a minimum and maximum of 0 and 40 packets, respectively. The accuracy improves as we collect data over more time, but Lumos can accurately classify most devices within a shorter interval. It should be noted that this is under the assumption of a known and single channel and does not account for the channel sensing scan time to find the active devices, which will be further evaluated in the next section.

7.4 Channel Sensing Sensitivity Analysis

Impact of Channel Sensing Scan Time on Device Discovery: To decouple the impact of classification performance from channel sensing and to provide a fair comparison with prior work [50], we define a notion of *Device Discovery Rate* for this evaluation. We mark a device as *discovered* if we have seen more than N packets from that device. Our analysis shows that an average of 50 packets is sufficient to identify different IoT devices, so we set $NumThresh = 50$ packets. For a fixed number of channels (in this case, five channels), we randomly assign each device (for a total of 15 devices) to a channel and compare Lumos' device-aware channel sensing algorithm with random, round robin, and SpecInsight [50]. Figure 13 shows that Lumos substantially outperforms the baselines and discovers 92% of devices in 15 minutes and 100% of devices by 40 minutes.

Intuitively, the baselines ignore the diversity of device types and their traffic patterns, while Lumos predicts the next packet arrival time from each device. SpecInsight wastes time capturing packets from a high transmission device, thus missing the lower transmission devices. We note that the minimum required scan time to discover all devices depends on the number of access points or active channels, which we evaluate next.

Impact of the Number of Active Channels: As discussed above, the scan time needed is a function of the number of active channels and active devices. To study this, we vary the number of active channels (C) from 1 to 30 and the number of active devices (M) from 1 to 15. In each iteration, we randomly select M devices and C channels and randomly assign the selected devices to selected channels. For each scan time and C value, we perform ten independent channel assignment iterations and vary the number of devices. Figure 14 shows that to achieve a fixed device discovery rate, the scan time increases as we increase the number of channels. (The error bars show the variations for the different number of devices and across different iterations.) With only one active channel, Lumos only requires 16 minutes to discover all devices. (This time is mainly used in the initial learning phase.) Even in a complicated scenario with devices spread across 20 channels, Lumos can achieve 80% discovery rate in around 15 minutes and 100% discovery rate by 50 minutes. This is well beyond the worst-case scenario we envision, as there are usually 1–5 access points in a regular indoor environment. Even in these hypothetical cases, the user can leave their smartphone or laptop running Lumos for a longer time to discover all hidden devices.

7.5 Localization Sensitivity Analysis

The user's walking pattern can significantly affect the localization performance. On one hand, RSSI is more accurate at close ranges, so the closer the user walks by the devices, the more accurate the RSSI measurements we can obtain. On the other hand, VIO systems suffer from accumulated error as the user walks away from the starting point. To evaluate the impact of the user's walking speed on localization performance, we collected two sets of data from each test location: one when the user walks the perimeter of the room with a normal to fast speed, and one when the user walks very slowly. We observe slightly lower accuracy in faster speeds (from a median accuracy of 1.4m with lower speed to 1.6m at higher speed). The reason is that the received packets are more sparse in space when the user walks faster. However, the majority of IoT devices have a higher transmission rate than the typical walking speed. So, Lumos can localize IoT devices irrespective of the user's walking speed.

We also compared the three localization methods explained in Section 6 and noticed negligible difference across the three houses. The surface-fitting based localization is expected to outperform the baselines for low transmission rate devices. However, all three methods perform similarly if the device has a frequent and regular transmission pattern. In our current testbed, we have a mix of both low and high transmission devices, so the average accuracy remains constant.

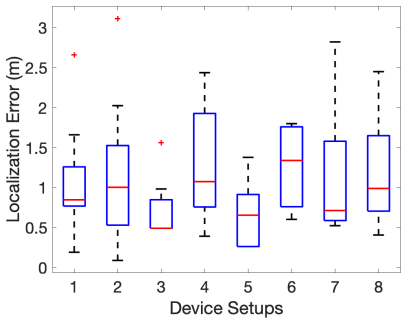


Figure 15: Lumos maintains its localization performance in different device setups

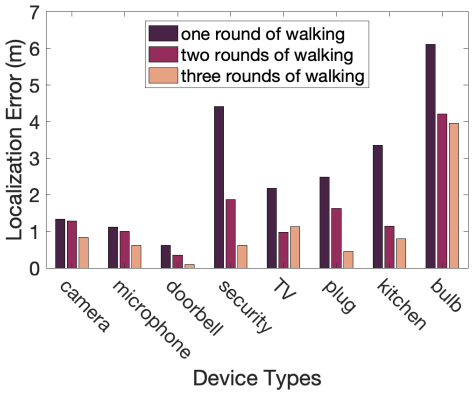


Figure 16: Longer walks can improve the localization performance especially for low data rate devices, but only one round of walking is sufficient for cameras and microphones with frequent transmission patterns

Impact of Device Setups: While the location of devices in the room does not affect the fingerprinting, it could impact localization accuracy depending on how close the user ends up walking around them. To study the effect of device setups, we select a subset of eight devices with semi-uniform packet transmission rates to isolate the effect of the traffic pattern and locate them in eight different locations in the lab space (shown in Figure 8e). For each of these setups, the user performs the same walking pattern around the perimeters of the room. Figure 15 shows the localization accuracy in each of these eight locations. The median localization accuracy in all of these setups falls between 0.5m to 1.5m, which shows the robustness of the localization. However, we still see that the range of errors varies across setups, which is mainly due to the relative distance between the actual location of the devices and the walking pattern. For example, in both setups 4 and 8, the devices are located at the corner of the room, where the user's walking route may fall further from the devices. This adds more noise to the interpolation and localization process. Nevertheless, the 1.5m average accuracy of Lumos still provides fairly accurate zone-based localization. For example, if we divide the entire lab space into eight zones of 2 by 2 meters, each corresponding to one cubicle, Lumos achieves 93% accuracy in correctly estimating the zone of each device among all eight setups.

Impact of Walking Duration: Another factor affecting the localization performance is the amount of exploration the user is inclined to do. This particularly impacts devices with low transmission rates, as longer exploration provides more samples to correct the curve fitting process. To evaluate this factor, we installed 16 devices (at least one from each device type) in Locations 1, 5, and 7 of the lab space, shown in Figure 8e. For each of these setups, we performed three experiments with 1 round, 2 rounds, and 3 rounds of walking around the perimeters of the space. Figure 16 shows that the user can improve the localization performance by taking longer exploration routes. This effect is more significant for devices with lower transmission rates, such as security devices, smart bulbs, or kitchen appliances. Note that cameras and microphones stream data with high transmission rates, so one round of walking is sufficient to estimate their locations.

7.6 Other Sensitivity Analysis

Effect of Feature Reduction Technique: It is a common technique to reduce the feature set by selecting a subset of features from the original dataset as it helps to prevent over-fitting on the training dataset. Also, many of the machine learning models don't perform well in the presence of correlated or duplicated features. So, as explained in Section 4.1, we try two different reduction techniques: (1) selecting the top ten features that have the highest mutual information score, and (2) dropping the features with a high cross correlation score (more than 95%) and keeping only one of those features. Table 5 demonstrates that removing correlated features is more effective in improving the classification performance, while the top ten features do not cover all the important features for differentiating device types.

Feature Reduction	Accuracy
Use All	90.67
Top 10	88.08
Drop Correlated	95.03

Table 5: Removing correlated features increases the average accuracy to 95%

Traffic Type	Accuracy
Incoming	89.73
Outgoing	83.18
Bidirectional	95.03

Table 6: Combining incoming and outgoing traffic improves the classification performance

Effect of Changing Device Settings: Many IoT devices have various configurations or settings such as image, audio or video quality, motion sensitivity, or microphone on/off. Such setting modifications could affect the traffic pattern and hence the signature of each device type. We evaluated the impact of device settings by selecting a subset of IoT devices (in this case, all 15 cameras in our testbed). We trained Lumos with the default setting of the cameras and tested the fingerprinting performance under three different setting modes: High (max resolution, max sensitivity, microphone

on), Low (min resolution, min sensitivity, microphone off), and Medium (min resolution, max sensitivity, microphone on). As we can see in Figure 17, the fingerprinting accuracy slightly drops when we evaluate Lumos on cameras with modified settings. However, Lumos can still generalize well across different cameras from different vendors and settings.

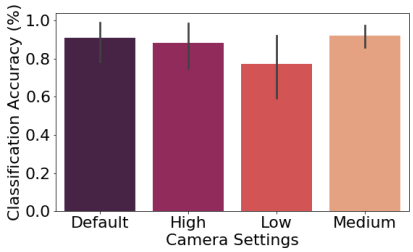


Figure 17: Classification performance of cameras for various device parameters

Effect of Traffic Direction: Another factor that could affect the performance of classifier is the type of traffic. When sniffing 802.11 packets, we can capture both incoming traffic from IoT devices and outgoing traffic to the devices. We compare the classification performance for each of these cases as well as considering both directions. In the case of bidirectional traffic, we compute separate features for incoming and outgoing traffic and then concatenate those before passing them on to the classifier. As we can see in Table 6, combining both incoming and outgoing traffic results in higher accuracy. Since each IoT device has a distinct incoming and outgoing behavior, combining two directions of data allows Lumos to capture more fine-grained information from each IoT device.

8 Discussion

Evading Lumos: Our goal with Lumos is primarily to empower users to gain awareness about potential surveillance by typical attackers in unfamiliar environments. That said, we acknowledge some avenues for more powerful attackers to evade Lumos. Since the ML step uses the MAC address to classify devices, the attacker can evade Lumos by performing frequent MAC address randomization. The adversary can also modify hidden device behavior to vary packet sizes or inter-arrival time, via custom hardware or modifying the firmware. Similarly, an attacker could evade localization by randomly changing their transmit power. Note, however, that these assume more expert attackers outside of our threat model of a typical host/guest in an AirBnB-like setting. Finally, since we can only detect/localize WiFi transmitting devices, an attacker can evade detection by avoiding wireless transmission; e.g., storing data locally, using wired connections, or other wireless communication protocols.

Unprofiled Devices: Lumos can identify devices as long as similar device types were seen in the training corpus. Recall that in some of our setups, Lumos did uncover previously unseen hidden devices. As a preliminary experiment to evaluate how Lumos performs for unprofiled devices, we did a leave-one-out experiment. We trained Lumos on all but one of the cameras we use for testing. For 13 out of 15 unprofiled cameras (Blink, TP-Link KC100, Canary, etc.), Lumos achieves median accuracy of 98%. (On inspecting the misclassifications, we find that the Nest camera has a unique behavior. It is the only camera in our dataset that uses the 802.11 subtype header field for different message exchanges.) This is promising as it suggests Lumos can potentially generalize across different device brands and models, as long as it has seen at least one device with similar behavior in the training phase. We plan to explore this further in future work.

Other Wireless Technologies: While our approach conceptually extends to other wireless protocols (including 5G), the actual performance may vary due to different channel allocation/resource management algorithms that could impact relevant features.

End-to-End Prototype on a Phone: Lumos needs to sniff 802.11 packets over the air, which is currently disabled on mobile phones without special permission from the manufacturer. There is no fundamental hardware/software limitation in providing such capability, and this functionality could be unlocked given enough justification. There has been also some workaround rooted Android phones for enabling WiFi sniffing. In this paper, we provide two alternative proof of concept prototypes that circumvent this limitation by either pairing a smartphone with a Rpi for WiFi sniffing, or pairing a laptop with an Intel RealSense Camera for VIO tracking. Another alternative setup could use a combination of a laptop for WiFi sniffing and a phone for VIO tracking, which are both readily available to most users. However, a user has to carry both the laptop and mobile phone for localization, which may be less convenient than carrying a small Rpi.

9 Related Work

We discussed a number of closest related efforts inline. Here, we focus on other related work.

Device Fingerprinting Methods: The topic of device fingerprinting has seen many solutions over the past decade in network analytics [43,49], IoT space [36,45,48,53], or at the hardware level, especially for surveillance cameras [26,42]. These solutions can be grouped into various categories depending on the kinds of features they are using for device fingerprinting:

- *Device identification using encrypted wired packets*: Many solutions [28, 43, 49] have been proposed for device identification using encrypted wired packets, and are typically run at ISPs. As such, these techniques are not applicable for our scenario, as a user in an AirBnB or a hotel wouldn't have access to these set of features. However, some of the features related to packet timing and size could also be extracted at the 802.11 layer.
- *Device identification using decrypted wired packets*: Much of the existing effort for device classification in the IoT space focuses on network traffic at the router [36,45,48,53]. These systems usually build ML models using full-stack packet information across the link, network, transport, and application layers or have full

access to the wireless router, which is not possible in our scenario due to limited access provided to the user.

- *Device identification using encrypted wireless packets (802.11 layer)*: There are a few techniques proposed for device identification at the 802.11 layer, but they are tailored towards detecting hidden cameras. Most of these methods use a stimulating or probing approach to trigger the hidden cameras; e.g., by altering the light level [42], shining light toward the camera to detect the reflected light from the lens [7, 8, 18], or walking to activate motion sensors [52]. While these techniques are shown to be effective for detecting cameras, they don't extend to other IoT devices which can still pose a serious privacy risk, e.g., [66].

Unlike these solutions, Lumos uses a passive approach by only relying on encrypted wireless 802.11 packets without access to the router and also detects a broader spectrum of hidden devices.

Using Hardware Properties for Detection: Some hidden detectors are based on semiconductor-specific properties such as a harmonic signature that can be sensed by transmitting an RF signal. Non-Linear Junction Detector (NJLD) [15] Low-end bug finders [4] are popular examples.However, the transceiver should be in close proximity to the hidden device. A recent work [41] also develops a portable 24GHz millimeter-wave (mmWave) probe to detect active electronics by observing their response in mmWave signal reflections. Unfortunately, all of these systems take significant time to scan an entire space. Moreover, they merely detect the presence of devices and do not identify the device types.

Device Localization: In general, wireless localization schemes map signal measurements into geometric parameters (distance or angle) to localize a target device with respect to reference points [31, 38, 54], or fingerprint the received signal strength at all possible locations [29, 65]. The key challenge with the first class of efforts is that commercial WiFi chipsets do not provide fine-grained data to calculate the distance or angle between the transmitter and receiver. While there is some work on estimating Angle of Arrival (AoA) or Time of Flight (ToF) [34], these techniques only apply for 802.11n (with High-Throughput and OFDM), while many IoT devices still use legacy WiFi protocols. With respect to the second class of work, it requires significant effort to characterize the environment and pre-label landmarks, which is not feasible in our problem setting.

10 Conclusions

Given the recent spate of abuse of IoT devices to spy on unsuspecting users, there is an imminent need for a low-cost approach to help users to detect, identify, and localize IoT devices when they enter an unfamiliar environment. What makes this problem uniquely challenging is the combination of the limited wireless access, the lack of sophisticated hardware, the diversity of potential snooping devices, and the inability to instrument the environment. In designing Lumos, we tackle these challenges and present a practical proof-of-concept realization that can be overlaid on a user-friendly AR interface to inform users of the risks lurking in an unknown environment with high accuracy and in near real-time. We plan to open source Lumos to help users gain awareness in unfamiliar settings and to inspire further innovation. As future work, we plan to extend Lumos to run on more mobile devices and support a broader spectrum of wireless protocols.

11 Acknowledgement

We thank our shepherd, Sascha Fahl, for his help with the final version of this paper, as well as the anonymous reviewers for their detailed comments. This work was supported in part by NSF award CNS-1564009. This work was also supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA

References

[1] 4 ways to tell if there are hidden cameras in your airbnb. https://www.huffpost.com/entry/airbnb-hidden-cameras-how-to-find_l_5cad177de4b01bf96007085c?guccounter=1.

[2] Airbnb Has a Hidden-Camera Problem . https://www.theatlantic.com/technology/archive/2019/03/what-happens-when-you-find-cameras-your-airbnb/585007/.

[3] Airbnb rules about security cameras. https://www.airbnb.com/help/article/887/what-are-airbnbs-rules-about-security-cameras-and-other-recording-devices-in-listings .

[4] Bug detector and hidden camera finder. https://www.spygadgets.com/counter-surveillance/.

[5] A camera is watching you in your airbnb:and, you consented to it. http://jeffreybigham.com/blog/2019/who-is-watching-you-in-your-airbnb.html.

[6] Couple says they found hidden camera pointing at their bed in carnival cruise room. https://www:insideedition:com/ couple-says-they-found-hidden-camera-pointingtheir- bed-carnival-cruise-room-47948.

[7] Glint finder - camera detector. https://play.google.com/store/apps/details?id=com.workshop512.glintfinder.

[8] Hidden camera detector. https://apps.apple.com/us/app/hidden-camera-detector/id532882360.

[9] How to scan your airbnb for hidden cameras. https://slate.com/technology/2019/04/how-to-scan-airbnb-hidden-camera-apps.html.

[10] Hundreds of south korean motel guests were secretly filmed and live-streamed online. https://www:cnn:com/2019/03/20/asia/southkorea- hotel-spy-cam-intl/index:html.

[11] Is your airbnb host spying on you with a hidden camera? https://www.businessinsider.com/airbnb-host-spying-hidden-camera-how-to-find-trick-2019-11#then-turn-it-off-by-either-unplugging-it-or-using-the-power-button-2.

[12] Kali nethunter. https://www.kali.org/docs/nethunter/.

[13] More than 1 in 10 airbnb guests have found hidden cameras: Survey. https://www.inman.com/2019/06/07/more-than-1-in-10-airbnb-guest-have-found-cameras-in-rentals-survey/#:~:text=A%20recent%20survey%20shows%20that,have%20actually%20found%20surveillance%20equipment.

[14] Noir: Discreet security camera and usb charger. https://www.indiegogo.com/projects/noir-discreet-security-camera-and-usb-charger#/ .

[15] Orion hx deluxe non-linear junction detector. https://reiusa.net/nljd/orion-hx-deluxe-nljd/.

[16] Should we be searching for hidden spy cameras in airbnbs and hotels? https://www.cnn.com/travel/article/hidden-spy-cam-airbnb-scli-intl/index.html.

[17] sklearn.preprocessing.StandardScaler — scikit-learn 0.22.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[18] Spy hidden camera detector. https://apps.apple.com/us/app/spy-hidden-camera-detector/id925967783?mt=8.

[19] Erling D Andersen and Knud D Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High performance optimization*, pages 197–232. Springer, 2000.

[20] Apple. Arkit. https://developer.apple.com/arkit/, 2019.

[21] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044*, 2017.

[22] Paramvir Bahl and Venkata N Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 2, pages 775–784. Ieee, 2000.

[23] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[24] Liqiong Chang, Xiaojiang Chen, Yu Wang, Dingyi Fang, Ju Wang, Tianzhang Xing, and Zhanyong Tang. Fitloc: Fine-grained and low-cost device-free localization for multiple targets over various areas. *IEEE/ACM Transactions on Networking (TON)*, 25(4):1994–2007, 2017.

[25] Yuxin Chen, Huiying Li, Shan-Yuan Teng, Steven Nagels, Zhijing Li, Pedro Lopes, Ben Y Zhao, and Haitao Zheng. Wearable microphone jamming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[26] Yushi Cheng, Xiaoyu Ji, Tianyang Lu, and Wenyuan Xu. Dewicam: Detecting hidden wireless cameras via smartphones. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 1–13. ACM, 2018.

[27] Kevin Chetty, Graeme E Smith, and Karl Woodbridge. Through-the-wall sensing of personnel using passive bistatic wifi radar at standoff distances. *IEEE Transactions on Geoscience and Remote Sensing*, 50(4):1218–1226, 2012.

[28] Kevin P Dyer, Scott E Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[29] Rizanne Elbakly and Moustafa Youssef. A robust zero-calibration rf-based localization system for realistic environments. In *Sensing, Communication, and Networking (SECON), 2016 13th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[30] Matt Ettus and Martin Braun. The universal software radio peripheral (usrp) family of low-cost sdrs. *Opportunistic spectrum sharing and white space access: The practical reality*, pages 3–23, 2015.

[31] Jon Gjengset, Jie Xiong, Graeme McPhillips, and Kyle Jamieson. Phaser: Enabling phased array signal processing on commodity wifi access points. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 153–164. ACM, 2014.

[32] Google. Arcore. https://developers.google.com/ar/, 2019.

[33] Stacey Gray. Always on: privacy implications of microphone-enabled devices. In *Future of privacy forum*, 2016.

[34] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. In *ACM SIGCOMM Computer Communication Review*, pages 159–170. ACM, 2010.

[35] The-Nhan Ho, Nicolas Marilleau, Laurent Philippe, Hong-Quang Nguyen, and Jean-Daniel Zucker. A grid-based multistage algorithm for parameter simulation-optimization of complex system. In *The 2013 RIVF International Conference on Computing & Communication Technologies-Research, Innovation, and Vision for Future (RIVF)*, pages 221–226. IEEE, 2013.

[36] Danny Yuxing Huang, Noah Apthorpe, Gunes Acar, Frank Li, and Nick Feamster. Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale. *arXiv preprint arXiv:1909.09848*, 2019.

[37] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. Spotfi: Decimeter level localization using wifi. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 269–282, 2015.

[38] Manikanta Kotaru, Kiran Joshi, Dinesh Bharadia, and Sachin Katti. Spotfi: Decimeter level localization using wifi. In *ACM SIGCOMM Computer Communication Review*, pages 269–282. ACM, 2015.

[39] Sivanand Krishnan, Pankaj Sharma, Zhang Guoping, and Ong Hwee Woon. A uwb based localization system for indoor robot navigation. In *Ultra-Wideband, 2007. ICUWB 2007. IEEE International Conference on*, pages 77–82. IEEE, 2007.

[40] Na Li, Jiquan Yang, Aiqing Guo, Yijian Liu, and Hai Liu. Triangulation reconstruction for 3d surface based on information model. *Cybernetics and Information Technologies*, 16(5):27–33, 2016.

[41] Zhengxiong Li, Zhuolin Yang, Chen Song, Changzhi Li, Zhengyu Peng, and Wenyao Xu. E-eye: Hidden electronics recognition through mmwave nonlinear effects. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 68–81, 2018.

[42] Tian Liu, Ziyu Liu, Jun Huang, Rui Tan, and Zhen Tan. Detecting wireless spy cameras via stimulating and probing. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 243–255. ACM, 2018.

[43] Eman Maali, David Boyle, and Hamed Haddadi. Towards identifying iot traffic anomalies on the home gateway. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 735–736, 2020.

[44] Alex T Mariakakis, Souvik Sen, Jeongkeun Lee, and Kyu-Han Kim. Sail: Single access point-based indoor localization. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 315–328. ACM, 2014.

[45] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE, 2017.

[46] Lionel M Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P Patil. Landmarc: indoor location sensing using active rfid. *Wireless networks*, 10(6):701–710, 2004.

[47] Gurobi Optimization. Inc.,"gurobi optimizer reference manual," 2015, 2014.

[48] Jorge Ortiz, Catherine Crawford, and Franck Le. Devicemien: network device behavior modeling for identifying unknown iot devices. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, pages 106–117. ACM, 2019.

[49] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. A haystack full of needles: Scalable detection of iot devices in the wild. In *Proceedings of the ACM Internet Measurement Conference*, pages 87–100, 2020.

[50] Lixin Shi, Paramvir Bahl, and Dina Katabi. Beyond sensing: Multi-ghz realtime spectrum analytics. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 159–172, 2015.

[51] Akash Deep Singh, Luis Garcia, Joseph Noor, and Mani Srivastava. I always feel like somebody's sensing me! a framework to detect, identify, and localize clandestine wireless sensors. *arXiv preprint arXiv:2005.03068*, 2020.

[52] Akash Deep Singh, Luis Garcia, Joseph Noor, and Mani Srivastava. I always feel like somebody's sensing me! a framework to detect, identify, and localize clandestine wireless sensors. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[53] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. Classifying iot devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, 2018.

[54] Elahe Soltanaghaei, Avinash Kalyanaraman, and Kamin Whitehouse. Multipath triangulation: Decimeter-level wifi localization and orientation with a single unaided receiver. In *Proceedings of the 16th annual international conference on mobile systems, applications, and services*, pages 376–388, 2018.

[55] Richard S Sutton and Andrew G Barto. Reinforcement learning: an introduction cambridge. *MA: MIT Press.[Google Scholar]*, 1998.

[56] Deepak Vasisht, Swarun Kumar, and Dina Katabi. Decimeter-level localization with a single wifi access point. In *NSDI*, pages 165–178, 2016.

[57] Jorge R Vergara and Pablo A Estévez. A review of feature selection methods based on mutual information. *Neural computing and applications*, 24(1):175–186, 2014.

[58] Very. Bluetooth vs. wi-fi for iot: Which is better? https://www.verypossible.com/insights/bluetooth-vs.-wi-fi-for-iot-which-is-better.

[59] Jiang Xiao, Zimu Zhou, Youwen Yi, and Lionel M Ni. A survey on wireless indoor localization from the device perspective. *ACM Computing Surveys (CSUR)*, 49(2):1–31, 2016.

[60] Jun Xu, Maode Ma, and Choi Look Law. Position estimation using uwb tdoa measurements. In *Ultra-wideband, the 2006 IEEE 2006 International Conference on*, pages 605–610. IEEE, 2006.

[61] Junhua Yan and Jasleen Kaur. Feature selection for website fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2018(4):200–219, 2018.

[62] Moustafa Youssef, Adel Youssef, Chuck Rieger, Udaya Shankar, and Ashok Agrawala. Pinpoint: An asynchronous time-based location determination system. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 165–176. ACM, 2006.

[63] Tevfik Yucek and Huseyin Arslan. A survey of spectrum sensing algorithms for cognitive radio applications. *IEEE communications surveys & tutorials*, 11(1):116–130, 2009.

[64] Gergely V Zàruba, Manfred Huber, FA Kamangar, and Imrich Chlamtac. Indoor location tracking using rssi readings from a single wi-fi access point. *Wireless networks*, 13(2):221–235, 2007.

[65] Dian Zhang, Yunhuai Liu, Xiaonan Guo, Min Gao, and Lionel M Ni. On distinguishing the multiple radio paths in rss-based ranging. In *INFOCOM, 2012 Proceedings IEEE*, pages 2201–2209. IEEE, 2012.

[66] Yanzi Zhu, Zhujun Xiao, Yuxin Chen, Zhijing Li, Max Liu, Ben Y Zhao, and Haitao Zheng. Et tu alexa? when commodity wifi devices turn into adversarial motion sensors. *arXiv preprint arXiv:1810.10109*, 2018.

Appendix A Pseudo-Code for Feature Extraction Algorithm

Algorithm 1 Feature Extraction and Aggregation

procedure FEAT_EXTRACT($packets$,t,Δ)
  $CF_t = \{\}$ ▷ Feature at time t
  $\Delta \leftarrow$ maximum sensing time
  for feat in {packet size, packet time, flags ...} **do**
    $F_t = P_t.feat$
    $CF_t = \{\}$
    for δ in {1,2,3,5,10,.. $\Delta/2$}) **do**
      $G_{t,\delta} = \{F_{t-\delta}, F_{t+\delta}\}$
      for AggFun in {mean, std, hist, sum, ...} **do**
        $A = AggFun(G_{t,\delta})$
        $CF_t = [CF_t, A]$
      end for
    end for
  end for
end procedure

Appendix B Pseudo-Code for Device Aware Channel Sensing Algorithm

Algorithm 2 Device Aware Channel Sensing

if RAND < ε **then** ▷ epsilon greedy
  $c*_t = RAND$(c)
else
  for d in devices **do**
    if Sensed_Enough > N **then**
      $R_d = 0 \quad \forall t$
    else
      $G_{1,2,3} = Device\_Classification(packets_d)$
      $\mu_{1,2,3} = Get\_Interarrival(G_{1,2,3})$
      R_d(t) $= max(1 - \frac{T + \mu_{1,2,3} * \lceil (t-T) \rceil / \mu_{1,2,3} - t}{\mu_{1,2,3}})$
    end if
  end for
  for c in channels **do**
    if devices sensed on that channel **then**
      $R_c(t) = max(R_d)$
    else
      $R_c(t) = RAND$ [0,1]
    end if
  end for
end if
$c*_t \leftarrow argmax(R\_c)$

Appendix C Impact of Device Density

We deployed three subsets of devices in the lab test environment and compared our classification performance. These three subsets were Low (with 1 device from each category), Medium (2 devices from each category), and High (all devices). As we can see from Figure 18, device density doesn't

have any significant impact on our classification performance. This is mainly because the device inference happens for each device independently. So, the only parameter that changes is the volume of background traffic as the number of active devices increases. However, this factor does not affect fingerprinting performance except in very rare cases when the network is at the highest capacity, resulting in some packet loss.

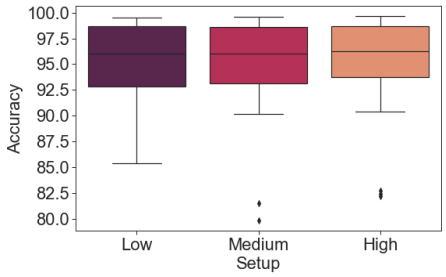


Figure 18: Impact of device density on classification performance