IoTScanner: Detecting Privacy Threats in IoT Neighborhoods

Sandra Siby
sandra_ds
@sutd.edu.sg

Rajib Ranjan Maiti
rajib_maiti
@sutd.edu.sg

Nils Ole Tippenhauer
nils_tippenhauer
@sutd.edu.sg

Singapore University of Technology and Design (SUTD)
8 Somapah Road
487372 Singapore

ABSTRACT

In the context of the emerging Internet of Things (IoT), a proliferation of wireless connectivity can be expected. That ubiquitous wireless communication will be hard to centrally manage and control, and can be expected to be opaque to end users. As a result, owners and users of physical space are threatened to lose control over their digital environments.

In this work, we propose the idea of an IoTScanner. The IoTScanner integrates a range of radios to allow local reconnaissance of existing wireless infrastructure and participating nodes. It enumerates such devices, identifies connection patterns, and provides valuable insights for technical support and home users alike. Using our IoTScanner, we investigate metrics that could be used to classify devices and identify privacy threats in an IoT neighborhood.

1. INTRODUCTION

In the context of the emerging Internet of Things (IoT), a proliferation of wireless connectivity can be expected, with predictions reaching as many as 500 smart devices per household in 2022 [9]. Heterogeneous smart devices that require transparent Internet connectivity need to be integrated into a common infrastructure. In particular, communication standards such as Zigbee, Bluetooth, Bluetooth Low Energy, and WiFi are expected to provide such infrastructure either as mesh networks, or traditional single-hop access points. Although wireless communications have many benefits in terms of usability, flexibility, and accessibility, there are also security concerns [21]. Among those concerns are privacy and in general, controllability. With this growth in the size and complexity of wireless networks, appropriate tools are required to better understand the environment's wireless traffic.

In this work, we propose the idea of an IoTScanner. The IoTScanner is a system that allows for passive, real-time monitoring of an existing wireless infrastructure. It classifies and identifies devices that are communicating using the infrastructure and traffic patterns among the participating devices. It will supply this information in a structured manner so as to provide valuable insights for technical support and home users alike.

While related work usually focuses on detecting either the infrastructure, or eavesdropping on traffic from a specific node, we focus on providing a general overview of operations in all observed networks. We do not assume prior knowledge of used SSIDs, preshared passwords, or similar. In addition to this, the IoTScanner operates passively, without active probing or other transmission. Our emphasis is also on providing real-time analysis and visualization of the scanned network, unlike some related work that focuses on offline analysis.

We note that we do not consider physical layer effects such as collisions and packet loss, goodput, or interference between networks in this work. We leverage recent developments in the area of cheap software-defined radio modules to handle the physical layer of the wireless traffic to provide the link layer traffic. As most wireless communication standards nowadays by default encrypt the link layer payload, we consider only link layer traffic for analysis in this work (without considering higher layer traffic).

We summarize our contributions as follows:

- We identify the issue of *opaqueness* in IoTs: the integration of heterogeneous IoT devices will lead to a plethora of wireless standard and networks operated in parallel.
- We propose an abstract system design that allows to seamlessly integrate a range of radio devices with a scanning server which provides a RESTful API.
- We present an implementation of the proposed design, and demonstrate its effectiveness through a set of experiments.
- We discuss how our system can be potentially used to identify privacy threats in an IoT environment.

The structure of this work is as follows: In Section 2, we briefly summarize the relevant background for our work. We propose our system design for the IoTScanner in Section 3. Our implementation is described in Section 4. In Section 5, we run some experiments to evaluate our implementation

IoTPTS'17, April 02 2017, Abu Dhabi, United Arab Emirates

ISBN 978-1-4503-4969-7/17/04...$15.00

DOI: http://dx.doi.org/10.1145/3055245.3055253

in an environment that consists of WiFi devices. In addition, we discuss the feasibility of the IoTScanner to perform traffic classification and device identification in this section. Related work and comparison of features with existing tools is summarized in Section 6. We conclude the paper and discuss possible future work in Section 7.

2. BACKGROUND

2.1 Internet of Things

The Internet of Things (IoT) refers to a network of physical devices that have the capability of gathering and transferring data from the environment, and interacting with remote computers over Internet [1, 10]. IoT devices can be anything from cellphones to smart lamps, as long as they have the ability to transfer data over the Internet, which essentially means they are assigned some public IP addresses.

2.2 Passive Monitoring

Passive monitoring is a technique for observing the traffic in a network only by listening to signals that are already available in the network, without injecting any extra signal. Unlike active monitoring, passive monitors do not probe the devices they are observing [14]. In the context of wireless network, the technique is more suitable as it only requires a traffic interceptor to be physically present in the environment, without requiring any wire tapping or so. Devices called sniffers or monitors are placed in the network to intercept frames transmitted by devices in their vicinity. Radios that can be used to sniff different standards (WiFi, Bluetooth, Zigbee) are available on the market and can be used for passive monitoring.

2.3 System Model

In our system model, we assume that there are one or more wireless networks in an IoT scenario in which the scanner is placed. For example, in a smart home, the electronic gadgets like smart lighting can make use of Zigbee communication, personalized health monitoring can use Bluetooth, Internet access can happen via WiFi communication. The scenario can have one or more IoT devices, but the devices need not be active (i.e., sending or receiving some information) at all times. In the basic operation mode, the scanner needs no prior knowledge of the infrastructure and network configuration in the IoT environment, i.e, no information is required about which device is used for what purpose, or what kind of encryption is used by them, if any. We assume that the user of our IoTScanner system does not have any control over the devices, or know if they are present or where exactly they are located. Hence, there is a chance that IoT devices that are present in the environment (but not actively participating in the network) are not detected by the scanner.

2.4 Attacker Model

We assume a limited attacker who is *honest but curious*. For example, the attacker might access a webcam set up in a room to spy on persons in the room, but the attacker will not set up a dedicated device for this (in particular, the device will not use some non-standard wireless communication). By obtaining the images from the camera over a certain time period, the attacker violates the privacy of the user. How the attacker obtained access to the camera does not matter for our analysis. As we do not assume that we have control over the network(s) in the environment, we will not be able to detect the camera activity on the Network Layer or at a gateway or similar.

3. A PASSIVE ANALYSIS FRAMEWORK

In the following, we provide details on our proposed passive analysis framework. We start with a concise problem statement, summarize the intuition behind our design and then provide additional details on individual components.

3.1 Problem Statement

In this work, we address the following problem: *"In an IoT environment, which devices are present, and communicating via wireless communications?"* That information can then be used to address questions such as: *"Are the IoT devices in my environment used by an attacker to violate my privacy?"* Ideally, such a goal should be reached passively, without actively interfering with the environment (which could be a public place such as an airport, hotel, or similar).

This problem cannot be solved by normal end-users easily; it requires specific software, hardware, and technical understanding of wireless protocols. We refer to this challenge as *wireless opaqueness* for the end users: they are agnostic to the way networking works, which links exists, and how data flows in the neighborhood.

3.2 Intuition

To address the problem stated above, we now propose the *IoTScanner*. The aim of the IoTScanner is to provide real-time, passive monitoring of an existing wireless infrastructure that potentially constitute an IoT environment. The scanner will identify active devices that are communicating using that infrastructure, and attempt to categorize IoT traffic depending on features such as the volume of traffic observed.

In particular, we use to following constraints to achieve passive monitoring. The IoTScanner:

- will not associate with any access point present,
- will not perform active probing or fingerprinting, and
- will not decrypt the observed network traffic.

The IoTScanner only observes the network traffic at the Link layer, and then analyzes this traffic using frame header information. A more offensive active scanner would not need to follow those constraints. The long term goal is to turn the IoTScanner into a convenient hand-held device. In the context of this paper, we are using a Raspberry Pi 3 [18] as platform, together with devices such as Android tablets for visualization via a web application.

3.3 IoTScanner: System Architecture

The main features provided by our IoTScanner are wireless signal interception, analysis of the captured packets and storage of the results. Finally, results will be accessible via APIs to be visualized through a web-based system.

Figure 1 shows the overall architecture of our proposed passive analysis framework. The IoTScanner has four main functional modules: traffic interceptor (captures wireless signals), traffic analyzer (analyzes MAC frames), data storage (stores processed frames and results), traffic visualizer (displays the status of network).

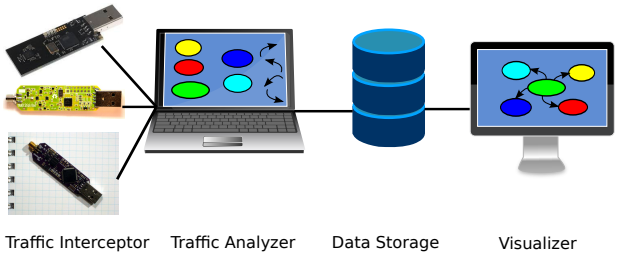

Figure 1: Overview of the IoTScanner architecture, which consists of four modules: traffic interceptor, traffic analyzer, data storage, and traffic visualization.

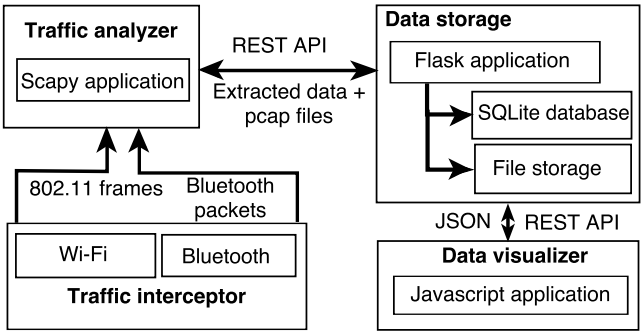

Figure 2: Overview of IoTScanner implementation.

3.4 Traffic Interceptor

The traffic interceptor module provides flexible low level access to the wireless medium. In the context of IoT, widely used protocols are 802.11/WiFi, Bluetooth, Bluetooth Low Energy (BLE), Zigbee, and Z-Wave; each being prevalent in particular application area(s). For example, the devices accessing Internet primarily use WiFi, wearable/on-body devices (e.g., blood pressure monitor, smart watch) use Bluetooth or BLE to connect to their parent devices, smart home appliances (e.g., smart meter) use Zigbee, and electronic appliances (e.g., AC, fridge, fan) use Z-Wave protocols [1]. However, there is no clear line of separation on what device can use which protocol; it primarily depends on the devices' usage or energy consumption behavior. Therefore, it is important to have interception capabilities leveraging multiple radios, either one radio for each protocol, or software defined radios to cope up with the variety of protocols available in the IoT spectrum. If multiple channels of the same standard should be reliably monitored, it might even be necessary to use multiple radios of the same kind, each on its own channel.

3.5 Traffic Analyzer

The traffic analyzer scrutinizes each Link layer frame, captured by the traffic interceptor. It extracts relevant information from the frame header such as the source and destination addresses, frame type and sub-type, SSIDs present, to be used for targeted analytics. We note that the frames need to be treated on a protocol by protocol basis, because parsing BLE frames is significantly different from parsing WiFi or Zigbee frames. In addition, the analyzer records some additional useful information such as the channel number on which the interceptor is capturing traffic, size of the frame captured (in bytes) and the timestamp at which the frame is captured. It sends the extracted frame information to the data storage module on a per frame basis, where actual analytics is performed.

3.6 Data Storage

The data storage module, acting as a simple server, provides a stable storage unit primarily for storing processed information about individual frames. The storage can be done by various means, however we prefer to employ a simple and light weight database system. In addition to storing the information, the system provides a simple interface for querying the database to carry out analytics. For example, the data storage interface can be realized using a set of RESTful APIs that can easily accessed over Internet, and be used by third-party applications. The APIs are provided for the following functions: storage and retrieval of frame information, generalized categorical queries on the DB, storage and retrieval of analysis results.

3.7 Traffic Visualizer

The traffic visualizer provides a visual representation of the observed IoT environment during a particular time window. An IoT environment can be viewed from different perspectives, starting from device-to-device connection to device or link specific information to the underlying activities being performed. In this paper, we consider a mix of all these perspectives and provide a high level overview of the IoT environment. We display a graphical view of the underlying network along with some supporting description of the devices and the associated links, if any. We do not infer any application specific information, e.g., if the IoT is used for maintaining room temperature by controlling AC, and instead focus on creating an inventory of the devices present and the amount of traffic generated by each of those devices. Because the devices need not be sending or receiving frames at all times, the traffic visualizer needs to automatically and periodically update to reflect any changes (in terms of both the devices and links) in the IoT environment under surveillance. Note that the visualizer only displays information present in the database currently.

4. SYSTEM IMPLEMENTATION

In this section, we present our implementation of the four proposed IoTScanner modules (traffic interceptor, traffic analyzer, data storage, and traffic visualizer). An overview of our implementation is shown in Figure 1. We use different wireless interfaces for the traffic interceptor, Scapy (a python library) for the traffic analyzer, SQLite for the data storage, and Javascript for the data visualization. The data storage uses RESTful APIs to communicate with the analyzer and the visualizer modules. The interceptor and analyzer components run on a Raspberry Pi 3 device, the data storage can be run on the same device or a server, and the visualization is displayed on an Android Tablet.

4.1 Traffic Interceptor

The traffic interceptor module can be implemented by a number of radio interfaces or by using software defined radios. We decide to use the following radio interfaces that are commercially available, and connectable via USB: the TP-

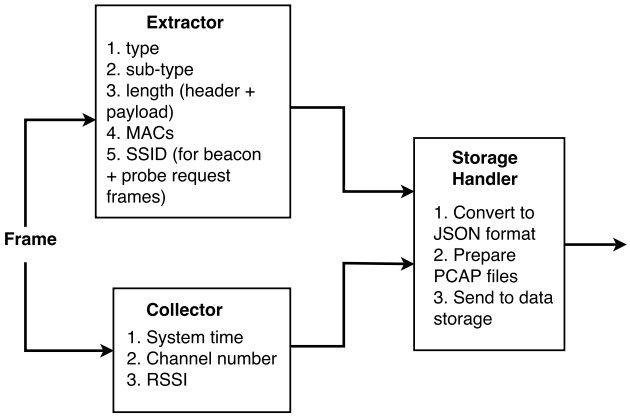

Figure 3: Traffic Analyzer module of our IoTScanner, for WiFi frames.

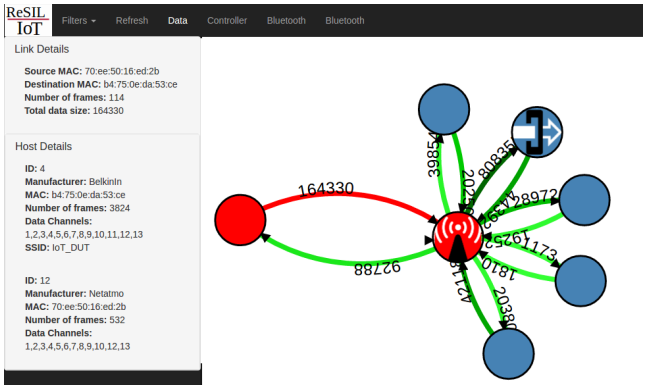

Figure 4: Example screenshot of our visualization app: IoT scenario represented using a graph structure. Selecting a node (red circle) in the graph displays the details about the underlying device.

Link TL-WN722N 802.11n wireless adapter (for WiFi), the Ubertooth One (for Bluetooth LE), and Atmel-RZUSBstick (for Zigbee). Due to space limitations in this paper, we will only discuss the WiFi setup in detail.

The TP-Link TL-WN722N adapter [22] can be easily configured to operate in monitor mode and capture WiFi frames with configurable channel hopping (over 13 channels). The adapter can also support certain active attacks such as deauthentication attacks, a useful feature in case we extend the functionality of the IoTScanner to include traffic decryption. We perform sequential channel hopping to obtain an overview of the traffic on all channels. The interface dwells on a particular channel for a certain period of time before hopping to the next channel. The dwell time can be configured by the user, otherwise takes a default constant value.

Since we dwell on a single channel at a time, frames on channels the interceptor is not listening on will not be captured. Hence, we capture a subset of the overall traffic.

4.2 Traffic Analyzer

The traffic analyzer module consists of three sub-modules: extractor, collector, and storage handler, as shown in Figure 3. The extractor and the collector sub-modules are implemented using Scapy [3], a python library for packet capture and analysis. Every frame captured by the traffic interceptor is an input to both the extractor and collector sub-modules. Since the aim of the IoTScanner is to provide a quick overview of the IoT environment, only relevant pieces of information are extracted from every captured frame. This is performed *online* (without buffering) so as to find quick answers to only those questions that we have assumed to be sufficient to provide an overview of the environment. The extractor sub-module extracts the following from the respective frames:

- In WiFi frames (type, sub-type, length, MAC address and SSID)
- In BLE frames (type, length, MAC address type (public or random), MAC address, node local name)
- In Zigbee frames (type, length, PAN ID, addresses)

The collector sub-module gathers information such as the system time during frame capture, the channel number on which the frame is captured, and the RSSI (for potential device localization). Both of these sub-modules supply the captured information to the storage handler.

The storage handler sub-module sends the collected and extracted information to the data storage module. It sends the frame information (in JSON format) to the data storage module via HTTP POST method. The module also has the option of storing the frames in a PCAP file and periodically sending the files to the data storage for further analysis of the overall traffic (which might be more computationally intensive). It is worth mentioning that high level frame analysis is not possible at the traffic analyzer since this requires aggregated information from multiple frames, and hence such analysis is performed at the data storage server end.

4.3 Data Storage

The data storage module of the IoTScanner provides a database server, to be accessible via a set of APIs, to store and retrieve the extracted/collected frame information. Two modules, the traffic analyzer and the traffic visualizer, interact with the storage module using these APIs over the network, potentially the Internet. We have implemented this module as a web server which interacts with a light-weight database. The APIs in the server are developed using Flask (a Python web framework), which helps to easily build the RESTful APIs for our purpose. We use SQLite to build a light-weight relational database system for our data storage module.

4.4 Traffic Visualizer

The visualizer is compatible with any hand-held devices, such as smart phones, tablets, etc. in addition to the desktop browsers. The visualizer displays the IoT environment in a number of ways (e.g., summary text, connectivity graph, bipartite relation, etc.) to make it suitable for the user to understand different aspects of the underlying network. By default, it displays a network graph accompanied by a brief summary text.

Figure 4 shows a sample network graph obtained during one of our experiments. The colored circles represent the nodes in the network and the arrows between a pair of nodes indicate that the pair exchanged at least one frame. We

Figure 5: IoTScanner with visualization on a handheld device.

identify the access points from beacon and probe request frames and internet gateways using simple heuristics. The access points and gateways have icons to identify them in the visualizer. Our overall implementation using the Raspberry Pi as interceptor (along with the tablet as visualizer) is shown in Figure 5.

5. WIFI EXPERIMENTS

All our experiments use the scanner in our IoT testbed [19]. In this section, we discuss the experiments using WiFi enabled IoT devices. Results of experiments with BLE and Zigbee are available at [20].

5.1 IoTScanner configuration

The IoTScanner, while intercepting WiFi traffic, uses two input parameters,

- Dwell Time (T_d): period of time (in seconds) that the traffic interceptor listens on a channel before moving to next channel ($T_d \in 5, 10, 20, 30(default), 40, 50, 60$).
- Hops T_h: number of channel hops performed by the traffic interceptor ($T_h \in 1, 6, 13(default), 26, 65, 130$).

These parameters account for the amount of time the IoTScanner is exposed to an IoT environment; for example, $T_h = 13$ and $T_d = 5s$ implies that the traffic interceptor scans for $13 \times 5 = 65s$, and the analysis will be performed only on the traffic captured during this period.

5.2 Evaluation metrics

Following are the common metrics for our experiments:

- *nodes*: the total number of active devices in the observed environment (including access points). A device is considered to be active if it is observed to have sent/received at least one frame.
- *links*: the number of unique pairs of nodes that have exchanged at least one frame (excluding broadcast and multicast frames).
- *SSIDs*: the number of access points seen in the environment.
- *Frames*: the total number of frames (sent or received) per device; these are further classified by type into *dFrames* (data), *mFrames* (management) and *cFrames* (control).
- *Bytes*: the aggregated number of bytes (sent and received) per device; these are further classified by type into *dBytes* (data), *mBytes* (management) and *cBytes* (control).

5.3 Experiment Settings

In our controlled experiments, we use six IoT devices: one Nest Cam security camera, one Netatmo camera (with face recognition feature), one TP Link security camera (of relatively lower resolution compared to the other two cameras), one Amazon Echo wireless speaker, one desktop with wireless adapter (to perform general web surfing), and one WiFi access point. We conduct 10 experiments each, in a *high-load* (being default setting) and a *low-load* setting.

High-load. This is the default setting for our experiments, and in this setting, all three cameras (focusing on the same area) are actively streaming video via Internet to a mobile device located outside the test environment. The Amazon Echo loudspeaker is streaming audio songs continuously during the experiments. The desktop with wireless adapter is used to browse web pages intermittently.

Low-load. In this setting, all the devices are present but none of them are actively used. For example, the IP cameras are switched on, but the live video is not accessed. The Amazon Echo is kept on but is not playing any music.

Understanding the Network Structure. First, we verify if IoTScanner can identify the nodes (with their MAC addresses) and the links among them from the captured traffic, hence determining the underlying network structure. We observe that our IoTScanner can correctly capture traffic from all the six devices in each of our experiments. The scanner identifies the devices that sent out broadcast frames to advertise their presence in the network, and the wireless channels on which each of these devices sent/receive traffic. We experiment with various values (as noted earlier) of the two input parameters, dwell time (T_d) and hops (T_h). We observe that lower values of these parameters result in the scanner being unable to capture all the devices during the observation window. After multiple rounds of experiments, we conclude that $T_d = 30$ and $T_h = 13$ are optimal values to quickly capture a sufficient amount of traffic for the traffic classification analytics we want to perform. Finally, we use beacon and probe request frames to identify access points in the network, and simple heuristics (amount of data and destination MAC) to identify the Internet gateway.

5.4 Per-node Traffic Classification

We perform simple analytics on the captured traffic to classify IoT devices. The mapping between the device labels we use and their actual name is shown in Table 1.

Frames, mFrames, cFrames, and dFrames. First, we find the total amount of traffic associated with each device in the network, along with the type (management, control and data) in the high-load setting. We determine the traffic in terms of bytes (Figure 6A) and frames (Figure 6B). A frame is associated with a device if its MAC address is found in the frame either as the source or destination address.

Label	Device Name
Dev-1	Desktop
Dev-2	Netatmo camera
Dev-3	TP-Link camera
Dev-4	Access Point
Dev-5	Gateway
Dev-6	Amazon Echo
Dev-7	Nest Cam camera

Table 1: Device labels and the corresponding devices used in WiFi experiments.

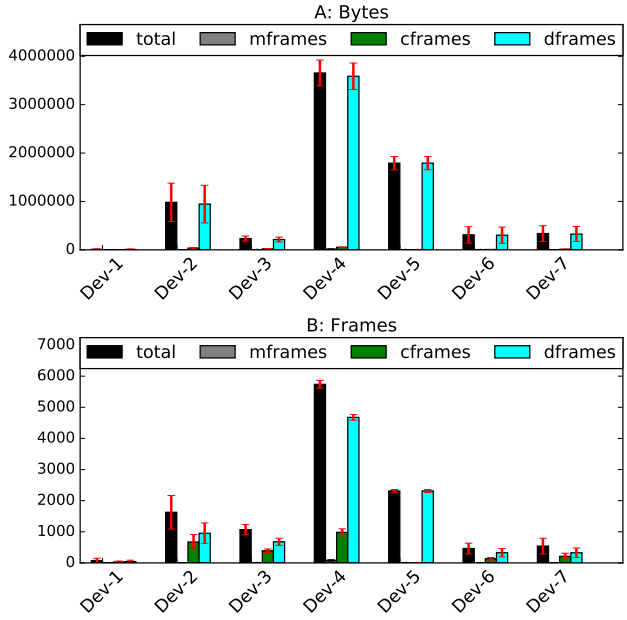

Figure 6: Traffic per device (subplot (A) in Bytes, and (B) in Frames) in high-load setting.

Interestingly, it can be seen that the traffic amount and its type can classify the devices at a high level. For example, the highest amount of traffic, in terms of both bytes and frames, is associated with Dev-4 which is the access point (see Table 1) that connects to all other devices present. Dev-5, which has no control and management frames, is the gateway device connected through Ethernet to the access point. The lowest amount of traffic is seen in Dev-1, the desktop, and it is used for occasional browsing during the experiments. The rest of the devices (Dev-2, Dev-3, Dev-6 and Dev-7) are associated with high traffic as they are either IP cameras or the Amazon Echo performing continuous streaming. In each of the devices, it can be seen that the data traffic (in bytes) dominates the control or management traffic, and is almost equal to the total amount of traffic of the corresponding devices. However, the difference between the number of control and data frames is not as high. We observe that the acknowledgement frames contribute to the large number of control frames in this case.

We explore the traffic volume in low-load settings (results shown in Figure 7). It can be seen that almost all the devices have control traffic comparable to data traffic (in bytes), as opposed to high-load settings where data traffic dominates

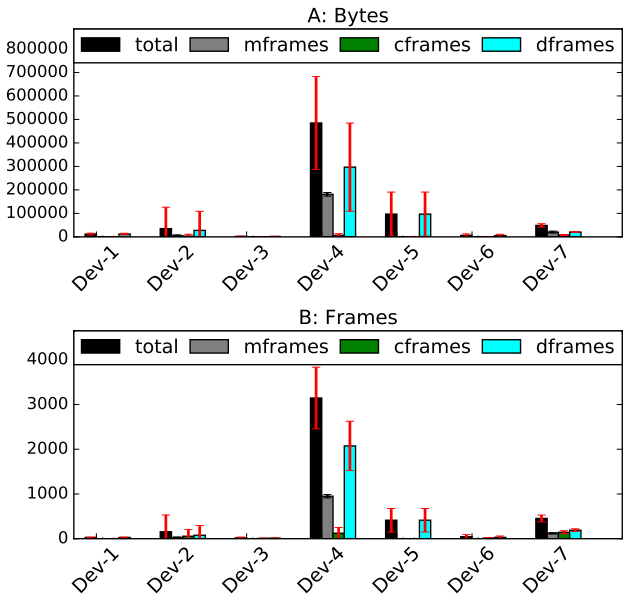

Figure 7: Traffic per device (subplot (A) in Bytes, and (B) in Frames) in low-load setting.

the control traffic. In fact, standard deviation of traffic volume is quite significant in this setting in all the devices, perhaps because the devices send their status information more at times. Thus, an analysis of the traffic amount and its composition can potentially be used to learn if an IoT setting generates a high volume of data traffic.

Sent and Received Volume. Since the overall traffic mainly consists of data frames, we investigate the amount of data traffic (in terms of bytes and number of frames) sent and received by each device (Figure 8 shows results in high-load settings). The highest amount of traffic (either sent or received) is observed in *Dev-4*, which is the access point. *Dev-5* (the gateway) has about three times higher received traffic (in Bytes) as compared to sent. Note that the traffic towards the Internet is the received traffic for the gateway, and traffic coming into the local network accounts for sent for it. Thus, the result is consistent with the ground truth, as there are three cameras sending video traffic. We also notice that the cameras (*Dev-2*, *Dev-3* and *Dev-7*) have a high amount of sent traffic, as expected. However, the amount of sent traffic varies significantly among the cameras. The received traffic is higher than the sent traffic for Dev-6, which is the Amazon Echo continuously streaming and playing audio songs from its server. Our experiments indicate that active IoT devices can be identified by analysis of sent and received traffic volumes in high-load settings.

We also investigate traffic flow in the low-load settings (results shown in Figure 9). Surprisingly, it can be seen that cameras do not necessarily produce a higher amount of sent traffic compared to received traffic (e.g., Dev-2). The Amazon Echo sends and receives almost equal amount of data in this setting. As expected, the gateway still receives more data than it sends, probably because these IoT devices continue to update their status to their associated cloud servers.

Sent-to-Received Ratio. We explore the possibility of identifying the devices by computing the ratio of sent to

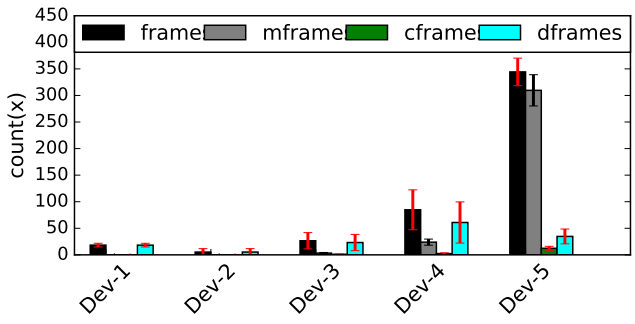

Figure 8: Number of frames and their types for each participating device in high-load setting.

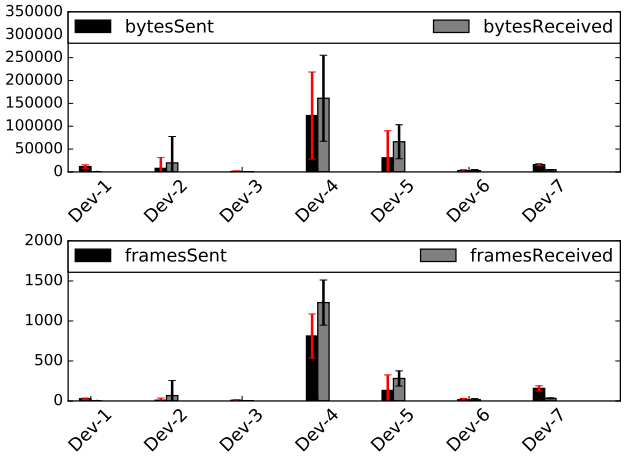

Figure 9: Number of frames and their types for each participating device in low-load setting.

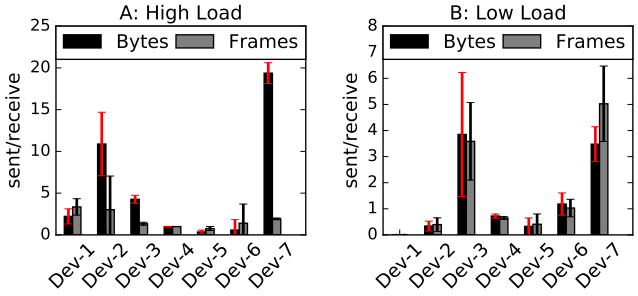

Figure 10: Variation of the ratio of sent and received traffic, per device basis in both high- and low-load settings.

received traffic (in terms of both bytes and frames). We consider only data traffic for this analysis, and ignore management and control frames. Figures 10A and 10B show the sent-to-received ratios in high-load and low-load settings respectively. In the high-load setting, the IP cameras (*Dev-2*, *Dev-3* and *Dev-7*) have a ratio greater than 4 for traffic in bytes and greater than 1.5 for traffic in number of frames. This indicates that an IP camera that actively streams video traffic may potentially be identified when the ratio is greater than 1.0. Also, the ratio in bytes is greater than the ratio in frames for the cameras, implying that, per frame, a larger amount of data is originated from the cameras. The desktop with adapter (*Dev-1*) has a lower ratio (>1.0) than the cameras but higher than the access point (≈ 1.0) and gateway (<1.0). The ratio of frames is much higher than the ratio in bytes, which indicates that the desktop sends more number of frames of smaller size. The access point can be clearly identified as it has a ratio close to 1.0 for both bytes and frames. The gateway (*Dev-5*) has a ratio less than 1.0, which indicates higher received traffic. Finally, the Amazon Echo (*Dev-6*) shows a ratio less than 1.0; as it continuously downloads audio traffic from the Internet.

In the low load setting, the sent-to-receive ratio does not look promising as a metric to classify the devices. The ratio does not behave in the same manner for all the IP cameras - *Dev-2* has a ratio less than 1.0, while *Dev-3* and *Dev-7* have a higher amount of sent traffic. The Amazon Echo has a ratio higher than 1.0 in this setting. Our experiments show that an analysis of the ratio alone in low-load settings may not be good enough to identify IoT devices.

6. RELATED WORK

WiFi monitoring. In [13], Kotz and Essien used syslog messages, SNMP polling and tcpdump packet captures to characterize WLAN usage on a college campus over a period of 77 days. Henderson et al. [11] built upon the work of [13] by capturing traces, including VoIP traces, from a larger set of access points and users. In these works, the packets were captured by associating with access points and the trace analysis was done offline.

Davis developed a passive monitoring framework in [8] to measure resource usage on 802.11b networks, and used it to analyze various setups involving video streaming. Further work on resource usage during streaming was done in [17] and [16]. Yeo et al. implemented a wireless monitoring system in [23], using multiple sniffers that produced a merged, synchronized trace which could be used for Link layer traffic characterization and network diagnosis. They also discussed the possibility of using anomalies in Link layer traffic for security monitoring. The challenges posed by analyzing traces from multiple sniffers was further explored in [15] and [5]. In [15], the authors introduced a finite state machine to infer missing packets from a distributed system of sniffers. In [5], the authors focused on large scale monitoring by utilizing 150 monitors to capture 802.11 frames. LiveNet [4] used multiple sniffers to monitor sensor network deployments by reconstructing routing behavior and network load from captured traces. In [4], the authors proposed algorithms for route inference and topology reconstruction among nodes in a network and provided visualization of the network topology and data transfer. Chhetri and Zheng introduced the WiserAnalyzer in [6]—a passive monitoring tool to capture wireless traces and infer relationships in the network.

A real-time passive monitoring framework was developed by Benmoshe et al. [2] and deployed on a university campus. Details such as number of clients, channel, error rate etc. were stored in a database and a map of active devices was built, assuming prior knowledge of the network setup.

Kismet [12] is one of the most widely used real-time, passive sniffing tools. Kismet only provides basic analysis of traffic (enumerating networks, hosts and amount of data), higher level analysis is not available. In addition to this,

Table 2: Proposed IoTScanner Features vs Related Works. ●= supported.

Related Work	Passive	Real-time	Aut. Analys.	WiFi	Bluetooth	Zigbee	API	Visualization
Proposed work	●	●	●	●	●	●	●	●
Wireshark [7]	●	●		●	●	●		
Kismet [12]	●	●		●	●	●		●
Benmoshe et al. [2]	●	●		●				●
Chhetri et al. [6]	●			●				
Yang et al. [5]	●	●	●	●				●
Chen et al. [4]	●		●	●				●

Kismet does not have a detailed visualization tool. Some tools have been built on top of Kismet, mainly for the purpose of visualizing the node locations (using GPS plugins) but several of them are no longer maintained and use outdated libraries. Wireshark [7] is protocol analyzer for pcap traces. Wireshark does not have the provision for automated analysis or visualization of the observed network. An overview of the features present in the IoTScanner and other tools is shown in Table 2.

7. CONCLUSION

In this paper, we proposed the IoTScanner, a system that can passively scan and analyze an IoT environment. We described the architecture and implementation details of our system. The IoTScanner currently scans WiFi, Bluetooth Low Energy and Zigbee traffic. It can analyze the traffic to provide an overview of the devices that are active in the environment and the communication among them. In this work, we present results of experiments to evaluate the performance of the IoTScanner in WiFi environments.

We introduced a simple ratio analysis of sent-to-received traffic that can classify WiFi enabled devices in an active IoT environment, which can potentially contribute to identification of privacy threats.

8. REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[2] B. Benmoshe, E. Berliner, A. Dvir, and A. Gorodischer. A joint framework of passive monitoring system for complex wireless networks. In *Proc. of IEEE Consumer Communications and Networking Conference (CCNC)*, 2011.

[3] Biondi, Philippe. Scapy. http://www.secdev.org/projects/scapy.

[4] B.-R. Chen, G. Peterson, G. Mainland, and M. Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. In *Proc. of IEEE International Conference on Distributed Computing in Sensor Systems*, 2008.

[5] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: Solving the puzzle of enterprise 802.11 analysis. In *Proc. of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2006.

[6] A. Chhetri and R. Zheng. WiserAnalyzer: A passive monitoring framework for WLANs. In *Proc. of IEEE International Conference on Mobile Ad-hoc and Sensor Networks (MSN)*, 2009.

[7] G. Combs et al. Wireshark. http://www.wireshark.org.

[8] M. Davis. A wireless traffic probe for radio resource management and QoS provisioning in IEEE 802.11 WLANs. In *Proc. of the ACM symposium on Modeling, analysis and simulation of wireless and mobile systems*, 2004.

[9] Gartner. http://www.gartner.com/newsroom/id/283971, 2015.

[10] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[11] T. Henderson, D. Kotz, and I. Abyzov. The changing usage of a mature campus-wide wireless network. *Computer Networks*, 52(14):2690–2712, 2008.

[12] M. Kershaw. Kismet. http://www.kismetwireless.net.

[13] D. Kotz and K. Essien. Analysis of a campus-wide wireless network. *Wireless Networks*, 11(1-2):115–133, 2005.

[14] L.Cottrell. Passive vs. Active Monitoring. https://www.slac.stanford.edu/comp/net/wan-mon/passive-vs-active.html.

[15] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Analyzing the mac-level behavior of wireless networks in the wild. *SIGCOMM Comput. Commun. Rev.*, pages 75–86, 2006.

[16] M. Narbutt and M. Davis. Experimental investigation on VoIP performance and the resource utilization in 802.11 b WLANs. In *Proc. of IEEE Conference on Local Computer Networks*, 2006.

[17] M. Narbutt and M. Davis. Gauging VoIP call quality from 802.11 WLAN resource usage. In *Proc. of IEEE Symposium on on World of Wireless, Mobile and Multimedia Networks*, 2006.

[18] Raspberry Pi Foundation. Rapsberry Pi 3 Model B. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[19] S. Siboni, A. Shabtai, Y. Elovici, N. O. Tippenhauer, and J. Lee. Advanced Security Testbed Framework for Wearable IoT Devices. *ACM Transactions on Internet Technology (TOIT)*, 2016.

[20] S. Siby, R. R. Maiti, and N. O. Tippenhauer. IoTScanner: Detecting and classifying privacy threats in IoT neighborhoods. arXiv preprint arXiv:1701.05007, Jan. 2017.

[21] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini. Security, privacy and trust in Internet of Things: The road ahead. *Computer Networks*, 76:146–164, 2015.

[22] TP-Link. TP-Link TL-WN722N. http://www.tp-link.com/en/products/details/cat-11_TL-WN722N.html.

[23] J. Yeo, M. Youssef, and A. Agrawala. A framework for wireless lan monitoring and its applications. In *Proc. of the ACM Workshop on Wireless Security (WiSe)*, 2004.