IoTShark: Monitoring and Analyzing IoT Traffic

Sahil Gandhi, Max Wang, Daniel Achee
University of California, Los Angeles
{sahilmgandhi,yingbowang, dpachee}@ucla.edu

Abstract

IoT devices continue to flood stores and homes at alarming rates with no indication of slowing down. They provide helpful functions from controlling lights, blinds, networks and more, but at a great cost. Both the devices and the services are controlled by the same company (often times) and thus users have no idea what exactly is being transmitted. There are tools to set up man-in-the-middle attacks and sniff the network, but they require some non-negligible time to set up and require extensive knowledge about computer security. We present IoTShark as a bootstrapping tool to quickly let users hook onto an IoT device on their network and monitor its activity. They can store these "traces" as csv files for long term use and can also quickly view graphs and statistics of their traces in a GUI to observe if any extra or anomalous network traffic is being generated by the devices.

1 Introduction

In recent years there has been a proliferation of voice assistants in people's homes. These useful IoT devices provide a convenient voice interface for users, but also present some privacy concerns as they are an always-on listening device. Companies selling these products, such as Amazon, claim that the devices are constantly listening for a keyword, such as "Alexa", and only after hearing this keyword, record and process what users say. Therefore, anything a user says before a keyword is issued and after their voice transaction has occurred is not recorded and is private. Unfortunately, there is no easy way for a user to verify that the device is upholding these public statements and the privacy agreement. Companies do admit that what a user says following this keyword is used for targeting advertisements and customizing content. This creates a conflict of interest as there is a financial incentive for companies to collect more voice data on their users and abuse their agreements.

These devices are relatively new and have emerged due to advances in natural language processing. These advances have allowed for fairly accurate keyword detection at the edge device in real time without the need to send the data to the cloud. With the increased accuracy of natural language processing and the ability to extract more semantic meaning from phrases, these devices are able to provide services that make them attractive to consumers and advertising agencies alike.

1.1 Background and Motivation

There are many players in the market, with the most popular being Amazon Alexa, Google Home Assistant, Apple Siri, and Microsoft Cortana. There is no standardization amongst voice assistants, with each using different protocols and traffic patterns. These devices typically encrypt the traffic. This encryption is a double edged sword as it both protects a user from attackers sniffing their data, but also prevents the user from auditing the behavior of the device.

Given that voice assistants appear to be only expanding in popularity and are projected to be a 7.8 billion dollar market by 2023 [6], it is imperative that users, even those that are not technically sophisticated, can easily audit and understand the behavior of their devices.

2 Related Work

Given that the data being transmitted across IoT devices is encrypted (end-to-end) it is quite difficult to discern what exactly is being transmitted, and even complicated man-in-the-middle style attacks can't be used since the data payload format and security cert /authentication format are unknown to an auditor. Despite these hindrances, there has been some work in the field by bloggers and researchers to capture the information being transmitted and infer some patterns.

Blog posts such as [3] and [5] go into great detail for setting up man-in-the-middle style proxies to capture all data being sent across the network by an IoT device (a Google Home and an Amazon Echo respectively for these blogs) and use an application such as Wireshark to sniff packets and determine

whether they originated from an IoT device. We use and build off of these techniques to capture data from our own IoT devices during our experiments and for creating our deliverables mentioned below. However, these blog posts are not always easy to follow without extensive computer science knowledge and are often times outdated. We aim to create a plug and play tool that takes this complexity away from the user.

In academia, there has been some work on this topic but often it is trying to change the existing IoT model to allow the user to see unencrypted traffic. An example of this would be TLS-RaR, which would allow a user to audit a device given the manufacturer consents and modifies their devices to support TLS-RaR [7]. We take a more cynical approach and don't believe that manufacturers are going to spend any money to make their devices more transparent because of the conflict of interest mentioned before. Therefore, we want to develop a tool to infer as much as possible given the traffic will be primarily encrypted.

Although we weren't aware of this tool during our preliminary research and development, we have recently discovered a tool called IoT Inspector [4]. This tool has similar goals as IoTShark, which we believe demonstrates the importance of our work. Although our tools are similar, users of IoT Inspector must agree to send their collected traces to the research group at Princeton who developed it. However, we strive to create a tool where a user's data is truly their own and not shared to any third parties. Furthermore, IoT Inspector is not compatible with the latest MacOS Catalina, prohibiting many users like ourselves from leveraging it.

Lastly, as more modules ("skills" in the Alexa world) are created, our smart assistants become ever smarter and are able to do more, including such sensitive tasks like writing emails, calling other individuals, and more. The following paper, [2], determines the relative ease with which a hacker can mimic a user and send off malicious requests. Even someone without malicious intents can take advantage of the information and skills of a voice assistant and breach a user's privacy. Since 2014, when the previous paper was published, Google, Amazon and others have taken the initiative to alert users after their device was accessed (and send a transcript /recording of the voice data) and protect some sensitive information. However the casual user is not aware of the repercussions of purchasing and using the device.

We hope that our monitoring tool can serve as this "wake up" call that the smart assistant is a very versatile device that should be treated with the same level of privacy and scrutiny that we treat our laptops /phones. Monitoring what it is doing should be easy enough for the common non-technical person to set up on their own.

3 System Overview

IoTShark consists of three main components:

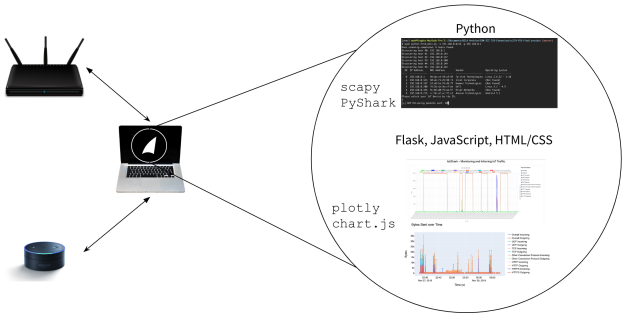

Figure 1: Overview of the system. The laptop running IoTShark acts as a man-in-the-middle for the IoT device and the router. As it captures the traffic, it presents the user with some useful information and graphics.

1. A Man-in-the-Middle attack that directs all traffic between the IoT device and the gateway router through the user's computer
2. A sniffer that sniffs on this traffic and produces a csv file of all the captured data.
3. Dynamic and static graphs that portray the captured traces as well as statistics generated from the csv data

3.1 Host Discovery and Man-in-the-Middle Attack

In a normal setup, the host laptop is connected to the same Local Area Network (LAN) as the target IoT device (either wirelessly via Wi-Fi or by wired connection to the router). The tool has to know the IP address of the device and the gateway router before performing the Man-in-the-Middle attack to the target device. It offers three methods to start gathering this information when the tool is executed:

1. The user directly gives the two IP addresses as command line arguments.
2. The user gives the subnet address, subnet mask and the gateway router's IP address.
3. The user gives nothing.

ARP spoofing immediately starts in Case 1, while the tool first scans all IP addresses in the user's entire subnet to discover all connected devices in Case 2 and 3. It scans the user-provided subnet (Case 2) or a set of pre-defined subnets common in home Wi-Fi routers (Case 3). `nmap` is used to perform the address and port scanning. The tool tries to set up TCP connections with SYN bit set to 1000 common ports of each IP address, discover connected devices and their open ports, and deduce device's hardware and software characteristics based on its TCP response patterns. `nmap` reports the

network-layer IP address, link-layer MAC address, the hardware vendor and the running Operating System of each device. A list of discovered hosts with this information is provided to the user in a command-line interface. The user would then be able to recognize and select the target IoT device to start ARP spoofing.

To achieve the Man-in-the-Middle attack, essentially the user's laptop identifies itself as the target IoT device to the gateway router and as the gateway router to the IoT device. In the TCP-IP network stack, the network-layer IP packets are wrapped inside link-layer data frames for local data transmission in hardware, with a MAC address in the frame header that uniquely identifies each network interface. The Address Resolution Protocol (ARP) in the link-layer handles discovering hosts' MAC addresses and translation between an IP address and a MAC address. Thus to achieve the goal, our tool repeatedly sends link-layer ARP Announcement frames to the two devices with faked IP address-to-MAC address translation. First, it sends out ARP Probes over broadcast Ethernet frames to get MAC addresses of the IoT device and the gateway router in their responses. It then periodically sends ARP Announcements to the IoT device with a mapping from the gateway router's IP address to the laptop's MAC address, and to the gateway router with a mapping from the IoT device's IP address to the laptop's MAC address. All packets to be delivered to the two hosts would then be delivered to the laptop that enables traffic analysis. Also, it's necessary to make sure IP packet forwarding is enabled in the user laptop's firewall settings, so these diverted packets will be automatically forwarded to the correct host specified in the IP header and normal two-way communication would not be disturbed. These ARP Announcements are sent every 2 seconds while the tool is running until it receives a Keyboard Intercept signal from the user. In this case, it announces the correct IP-MAC address translation to the two devices, terminates the Man-in-the-Middle attack and exits the program. The ARP spoofing is done by the `scapy` network packet manipulation framework. We conducted the experiments in macOS Catalina and IP packet forwarding is explicitly enabled by a `setctl` command that sets BSD kernel state.

3.2 Sniffing the Traffic

After the ARP spoofing has started, all incoming and outgoing network packets going through the user's computer are sniffed using the `pyshark` framework. Running in a separate thread, the sniffer first filters out relevant IP packets that have the IoT device's IP address to be in either the "source" or "destination" header. It also detects the packet to be either incoming traffic to the IoT device or outgoing traffic from the IoT device by checking the same two header fields. Then each packet is classified by their application-layer and transport-layer protocols: HTTP/HTTPS/Other for the former and TCP/UDP/Other for the latter. Once the sniffer collects a certain amount of packages, currently five, it writes the packets' metadata to a csv file. This metadata includes:

1. The timestamp of the packet
2. The incoming or outgoing bytes of the packet
3. The source and destination IP address of the packet
4. The source and destination port of the packet
5. The application-layer and transport-layer protocol of the packet

Additionally, we gather some overall statistics of all sniffed packets before the user exits a Man-in-the-Middle attack session. These statistics are saved in another csv file and presented in a web page as described in the following sections. This data includes:

1. The total number of incoming and outgoing bytes transmitted over this session
2. The aggregated number of incoming and outgoing bytes for each application-layer protocol (HTTP, HTTPS, Other) and transport-layer protocol (TCP, UDP, Other)
3. The aggregated number of incoming and outgoing bytes for each port on the source hosts and destination hosts
4. The aggregated number of incoming and outgoing bytes for each IP address
5. The ISP name of every remote host given its IP address

3.3 Graph Generation and Statistics

After the sniffing session has accrued enough captured packets to write to the csv file, our tool spawns a thread that runs a `Flask` web server which the user can access by visiting `localhost:5000` on the browser of their choice. This web server reads from the csv and in real time as packets are being captured, plots them to a graph. This graph was implemented using the client-side JavaScript library `Chart.js`. This graph updates every second and can be toggled to show any subset of the following:

1. Incoming bytes
2. Outgoing bytes
3. Incoming TCP bytes
4. Outgoing TCP bytes
5. Incoming UDP btyes
6. Outgoing UDP btyes
7. Incoming other transport protocols btyes

8. Outgoing other transport protocols btyes
9. Incoming HTTP btyes
10. Outgoing HTTP btyes
11. Incoming HTTPS btyes
12. Outgoing HTTPS btyes
13. User talking to device

All of them except for the last are extracted from the captured packet headers and aggregated over the last second. If the user is talking to the device is determined by the user pressing the `SPACE` bar during the capture. The intention of this is to make it easier to denote on the graph when the user triggers the voice assistant for the purposes of experiments. In addition to the graph, the web page contains the total bytes for the session for each of the above categories, which also updates in real time.

In order to analyze csv files of previous sessions, our tool can be ran in a mode where you supply the path to the csv and the tool will generate a static graph of the session traffic. This static graph was implemented with the library `plotly` and similarly can be toggled to show all of the same categories as the real time graph. Additionally their are a variety of statistics also displayed, with a subset being shown in Figure 4. The most useful statistic is the ISP map, which shows which company owns each of the IP addresses that the device connected to during the session.

4 Evaluation

We use IoTShark to capture and evaluate traces from two devices: a third-generation Amazon Echo Dot and a Google Home Mini. According to Amazon and Google's device manual, firmware and software updates are automatically downloaded and installed when the devices are connected. Thus the two devices used for testing should be running the latest firmware and our findings can be generalized for all devices on the market of the same kind. Both devices contain a physical "microphone-off" button that is supposed to turn off the microphone and thus not capture any voice input. We captured and analyzed five traces for each device in the following situations:

1. Five minute trace with the mic off and no user talking
2. Five minute trace with the mic off and user talking
3. Five minute trace with the mic on and no user talking
4. Five minute trace with the mic on and user talking
5. Thirty minute trace with the mic on and user normally using the device

For experiment 2 and 4, the user reads a specially-designed, 163-word paragraph in whole or in part. This bag of sentences is set up to contain a set of keywords as diverse as possible. It includes geographical locations, upcoming holidays, event names on the user's calendar, recent news headlines, and names of large technology companies. One sentence among them contains the wake word for the tested voice assistant device and asks about the local weather in the upcoming week. This design aims to test the voice assistant's behavior under a large range of topics, where it may be sensitive to some but not others. The entire paragraph is attached in Appendix A.

4.1 Amazon Echo Dot

The five traces for Amazon Echo Dot are gathered first as shown in Figure 2 for the four 5-minute traces and Figure 3 for the 30-minute trace. The grey dots on top of each graph denote the moments when the user is talking (either to the voice assistant or not); the orange dots denote a subset of these moments when the user is talking to the voice assistant started with the wake word "Alexa".

4.1.1 5-Minute Traces

First, we have the following key observations from the 5-minute traces in Figure 2.

First, there are occasional traffic spikes when the physical microphone is turned off, where Trace 1 (top-left) shows three such spikes for incoming traffic of 20 KB or more each. Trace 2 (bottom-left) shows one spike for outgoing traffic of about 25 KB. Further analysis of the traffic dump shows about half of the transmissions are HTTP traffic in plaintext and the other half are TLS-encrypted HTTPS traffic, both using TCP with random ports on multiple remote servers managed by Amazon. Almost all the non-encrypted traffic are HTTP request-response pairs to `http://spectrum.s3.amazonaws.com/kindle-wifi/wifistub.html`. The link points to a short web page that is titled "Kindle Reachability Page" and only contains a UUID-like string in the body. We guess that this is a heartbeat message from the Echo Dot used for periodic checking of its connections to the backend services. These messages were sent frequently during the experiment probably due to a lossy Internet connection, as the device was placed far away from the Wi-Fi router across multiple walls and may have experienced a long Round-Trip Time of the request-response. The other half of the traffic in Trace 1 is completely TLS-encrypted. We attempted to subvert this by having our Man-in-the-Middle attack present self-signed TLS certificates to fool the Echo Dot, but the Dot simply drops all packets in these connections. Thus, it's impossible to decrypt the content of these encrypted data packets and accurately speculating about the encrypted traffic is difficult.

Second, the Echo Dot is transmitting data as long as the

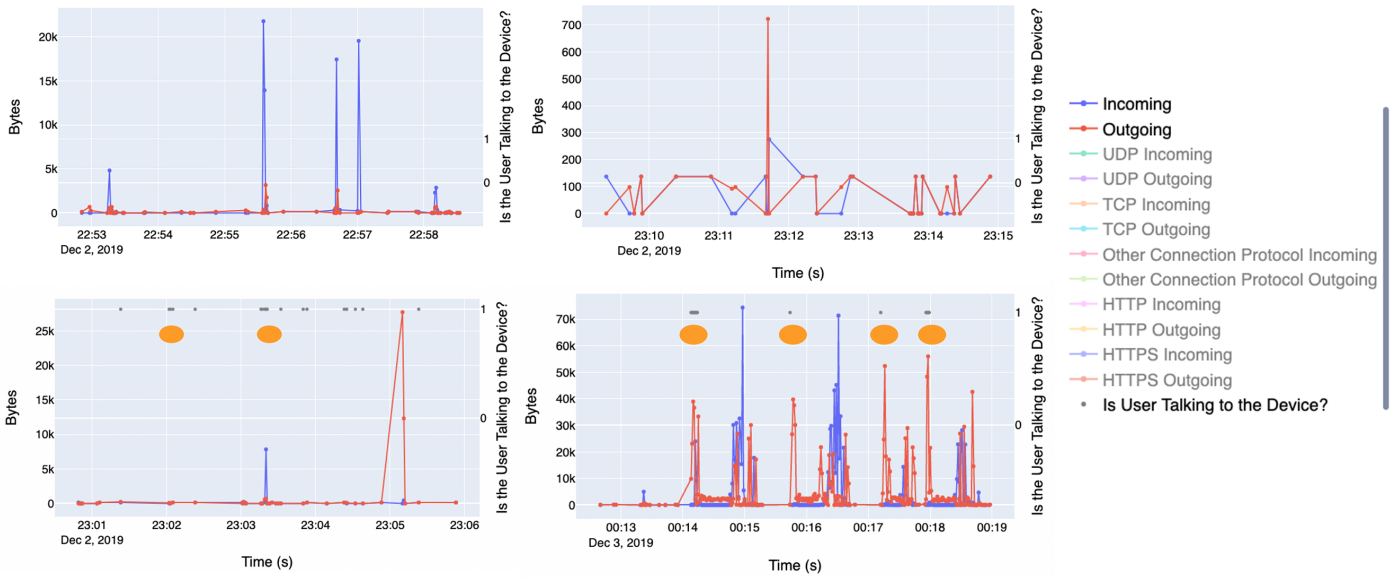

Figure 2: 5-minute traces of Amazon Echo Dot: Top-left: microphone off and no user talking. Bottom-left: microphone off and user talking. Top-right: microphone on and no user talking. Bottom-right: microphone on and user talking.

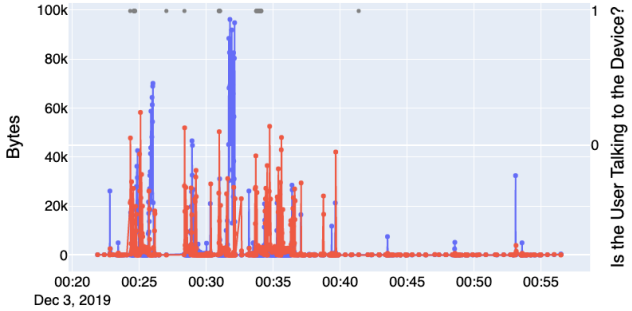

Figure 3: 30 minute trace of Amazon Echo Dot with the mic on and talking. Grey spots are the moments that user speaks the wake word and talks with the device.

```
'isp_map': {'13.225.149.99': 'Amazon.com, Inc.',
            '13.33.228.197': 'Amazon Technologies Inc.',
            '13.35.103.193': 'Amazon Technologies Inc.',
            '23.20.6.188': 'Amazon.com, Inc.',
            '3.213.216.138': 'Amazon Technologies Inc.',
            '3.230.66.170': 'Amazon Technologies Inc.',
            '52.216.108.59': 'Amazon.com, Inc.',
            '52.216.136.43': 'Amazon.com, Inc.',
            ..... (33 IP addresses in total)
'num_global_connections': 11954,
'num_local_connections': 122,
'num_total_connections': 12076,
'total_bytes': 6538813,
'total_incoming_bytes': 3836343,
'total_outgoing_bytes': 2702470,
```

Figure 4: Some chosen statistics of the 30 minute Echo Dot trace.

device's physical microphone is turned on, even if the user is not talking. This conclusion draws from the graph of Trace 3 (top-right): data packets have been sent out continuously and the rate of outgoing traffic stays at slightly above 100 bytes per five packets sent. These transmissions are all TCP traffic and contain both non-encrypted heartbeat messages and TLS-encrypted packets in our experiment.

Third, in the normal case where the microphone is turned on and the user is asking questions, the Echo Dot keeps transmitting and receiving data after each "question-answer session" between the user and the device. As seen from Trace 4 (bottom-right), there's a spike of incoming and outgoing traffic each time the user speaks the wake word and asks the voice assistant a question and the device provides an answer. This traffic is considered reasonable as the device sends relevant text or audio data to the related back-end services for natural language understanding or information lookup. What is concerning is the larger spike of data exchange after a brief pause when the communication session is finished and before the user asks the next question. This spike of traffic follows every question-answer session in our test, and transmits and receives almost twice the data as was exchange during the session. All conversation-related network traffic is TLS-encrypted, so unfortunately we are not able to further inspect the packets' contents. We guess that it may be some data prefetching based on prediction of the user's future interaction with the device given the user's previous behavior.

4.1.2 30-Minute Trace

For the first half of this 30-minute experiment, the user asks the device regular questions, such as weather forecast and calendar events, and performs common tasks, such as setting up alarm clocks. The user remains silent during the second half. Besides the traffic pattern shown in Figure 3, we will now focus on analyzing the end-to-end traffic quantitatively. Some end-to-end statistics are shown in Figure 4.

First, the Echo Dot has communicated with a total of 33 remote hosts over this 30 minutes. Using a Network Information Center (NIC) database lookup of their IP addresses, we determined all the endpoints belonged to Amazon Inc. More than 98.99% of the connections are global connections where the device connects to one of those remote back-end servers. Meanwhile, the remaining connections are local, with the Echo Dot communicating with a device in the same user's home Wi-Fi network. These devices can be the user's laptop, smartphone, tablet, etc. and we leave further investigation as future work. We guess it's likely that the device is connecting to the user's iPhone in this experiment, because the iPhone has the Alexa app installed and has paired with the Echo Dot. Thus, the IoT device is able to pull in user's personal data such as calendar events and emails from the iPhone for quicker responses. The Echo Dot has generated a total of about 3.8 MB incoming traffic and 2.7 MB outgoing traffic in this experiment.

4.2 Google Home Mini

4.2.1 5-Minute Traces

The five Google Home Mini traces paint a radically different picture than what we observed in the Amazon Echo Dot. Here even with the microphone off, a physical off-switch, (top left, and bottom left in figure 5) the traces show that the network is still being constantly used. Fortunately, they do not change regardless of the user talking, but it is concerning there is still so much traffic (far more than the minute heartbeat messages that the Alexa was doing). Upon a deeper dive, we saw 2 types of traffic requests: an unencrypted multi-cast DNS request to 224.0.0.251 that effectively acted as the heartbeat message and then encrypted connections to other local devices on the network such as other laptops and phones. We are not too sure what these encrypted connections are sending, whether it is just "reachability" of the Home Mini from other devices for sound casting purposes, or something else entirely. The Mini does stop sending packets at the 3 min 30 seconds mark for about a minute before resuming its "normal" activity.

What is even more peculiar though is that when the microphone is turned on, (top right and bottom right in figure 5) the traces show that the network traffic does not change from when the microphones are off. This is understandable if the user is not talking, but when a query is asked, the Home Mini does not request data from a Google server but is still able to answer! We suspect that it may either be piggy backing off some of the local connections that it does with other devices on the LAN or through some pre-fetching/caching capability. The question we asked was not super uncommon, so it is possible that it may have the answer stored locally (perhaps carried via one of the encrypted local connections).

4.2.2 30-Minute Trace

Finally, we also captured a 30 minute trace where we talked casually in front of the Home Mini and asked it to do various tasks (i.e. stream music). Figures 6 and 7 portray the trace and some statistics respectively. First of all, we continue to see the local connections and the periodic mDNS connections running throughput the entire 30 minutes. We can also see here that when we asked it to stream music from Spotify or Youtube Music, there is a flurry of incoming packets that carry the music. However these do not hit Spotify servers or even the Google servers but rather a CDN, Fastly. We also asked it a barrage of questions, some more random than others, i.e. "Who was the sixth viking king of Norway?", and "What is DECA?, along with followups like "Tell me more". During these two sessions, shown in the black curly brackets in Figure 6, we finally see more traffic being generated! The answers to these questions were not cached, and it we also see the Home Mini hit three different Google IP addresses to answer these questions, but also *maintain* these connections to transfer more data for other potential questions (again the time span of the black curly brackets). The local connections to devices on the LAN remain during this time however. In total a little over 1.5 Mb of data was sent to the Home Mini, but during the same time it also sent out 415 Kb of data to local connections as well as the Fastly CDN and the Google IP addresses. What exactly is being sent is unknown since it was mostly encrypted except for the mDNS requests.

5 Future Work

While IoTShark can currently monitor and save the traffic of any IoT device (and really any device so long as the correct IP address is provided) there are still a few things that we wish to work on for future iterations.

An interesting paper in this space aims to classify events in different IoT streams, in particular focusing on classifying when a Nest Camera is live streaming data vs motion detecting, when an Alexa is transmitting data back to the server [1]. We intended to do this classification but quickly realized that we would need to collect vast amounts of data from sources that are not just the three of us. However as a future task, we intend to build off of the work from this paper to detect and classify IoT traffic, initially for smart assistants and later for other devices. Consequently this is also the paper that inspired IoTInspector, a similar tool created and maintained by Princeton University.

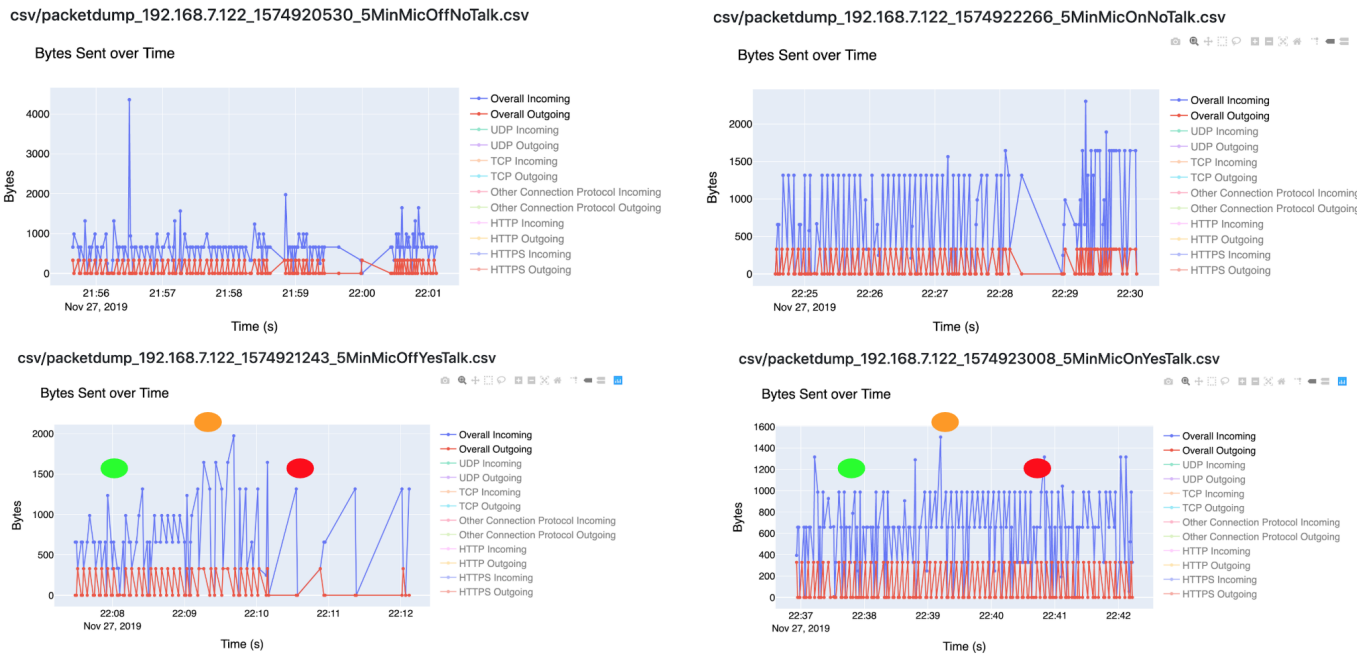

Figure 5: 5-minute traces of Google Home Mini: Top-left: microphone off and no user talking. Bottom-left: microphone off and user talking. Top-right: microphone on and no user talking. Bottom-right: microphone on and user talking.

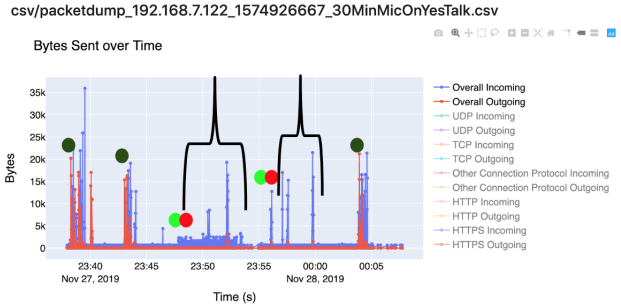

Figure 6: 30 minute trace with the mic on and talking. Dark green spots are music streaming requests, green and red are start/stop talking.

```
'isp_map': {'104.154.127.47': 'Google LLC',
            '151.101.52.246': 'Fastly',
            '35.186.224.44': 'Google LLC',
            '35.186.224.53': 'Google LLC',
            '8.8.8.8': 'Level 3'},
'num_global_connections': 6568,
'num_local_connections': 2460,
'num_total_connections': 9028,
'total_bytes': 1973534,
'total_incoming_bytes': 1558260,
'total_outgoing_bytes': 415274,
```

Figure 7: Some chosen statistics of the 30 minute Home Mini trace.

We also want to include an anomaly detection feature, where given some traces of "defined behavior" as ground truth, other traces can be compared for abnormal traffic patterns. These may be caused by changes in the services of the IoT devices or say in the example of huge traffic being generated when no user is in a home (perhaps someone has broken into the house!). Similar to the trace classification problem though, this requires immense amounts of data to detect what is "normal" and therefore what is "abnormal".

Finally, due to the necessity of physical hardware for this project, we were only able to gather traces from Amazon Alexa and Google Home. However, if given the resources, we also wanted to gather traces from Apple Siri and Microsoft Cortana devices (along with other non-smart-speaker IoT devices) to compare the traces and detect if any funny business is occurring here. At the time of writing this paper, we could find no blog post or research paper tracing the two other smart assistants (Siri and Cortana).

6 Conclusion

IoT devices continue to rise in popularity, but remain as vendor controlled as ever. While we cannot see the data *inside* the transported packets, we *can* capture the raw traffic and make inferences based on it. Unfortunately, understanding the nuances of how to setup up a computer, router, and other properties are time consuming for most computer scientists with limited experience in this domain, and beyond the scope of the common individual. We believe that everyone should have the ability to see who their devices are talking to and how often, and neither Wireshark nor Ettercap can easily produce this information. IoTShark is our solution to this issue and allows individuals to quickly hook onto and capture traffic information from IoT devices, analyze the data statically, and visualize it in network graphs. We hope that other researchers can use this tool to quickly bootstrap their IoT work, and the average individual can look into what their device is doing at a high level! The code and raw data is available at https://github.com/sahilmgandhi/IotShark.

References

[1] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic. *CoRR*, abs/1705.06805, 2017.

[2] Wenrui Diao, Xiangyu Liu, Zhe Zhou, and Kehuan Zhang. Your voice assistant is mine: How to abuse speakers to steal information and control your phone. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '14, pages 63–74, New York, NY, USA, 2014. ACM.

[3] Amir Ghadiry. Is my google home spying on me?, Apr 2017.

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

[5] Maik Morgenstern and Maik Morgenstern. Careless whisper: Does amazon echo send data in silent mode?, Jul 2018.

[6] Market Reports. Global voice assistant market analysis & forecast 2017 to 2023, Jul 2017.

[7] Judson Wilson, Riad Wahby, Henry Corrigan-Gibbs, Dan Boneh, Philip Levis, and Keith Winstein. Trust but verify: Auditing the secure internet of things. In *Proceedings of the 15th MobiSys*, pages 464–474. ACM, 2017.

A Paragraph Spoken to Device

It is a dark and stormy night. My friends and I just came back from the Yosemite National Park, where the quick brown fox jumps over the lazy dog. Next week is Thanksgiving. Black Friday in 2019 is coming as well. It's a good time to do something exciting, such as taking a Computer Security class or a Programming Language class at UCLA. By the way, the first Airbus A380 jumbo jet is retiring. We like flying in that plane.

WAKE_WORD, what is the weather like in Los Angeles on Thanksgiving?

Anyways, we have Boeing 787 Dreamliners for cross-continental flights. The Web and Mobile System class with Ravi is amazing. We should upgrade the commercial laundry machine during the Black Friday sale. The bright and sunny weather is coming back and a trip to Joshua Tree National Park awaits. Well, I just saw a slow cat crashed into a new Android robot. There are some other robots made by Apple and Amazon too.