WiSpider

Authors: Anish Singhani, Thomas Horton King, Ethan Oh
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

Abstract—**TODO: With the increasing ability to build tiny wireless-connected IoT devices, there is massive potential for hostile actors to infringe on individual, corporate, and governmental privacy using hidden wireless devices such as microphones and cameras. We want to create a device that can detect and localize these hidden wireless devices. We will also exploit wireless protocols to obtain some information about the hidden device, such as manufacturer or device purpose. We will create a visual user-interface to display where any detected devices are located, relative to the user. Given the pervasiveness of WiFi for communications, our prototype/MVP will primarily focus on sniffing WiFi packets.**

Index Terms—**privacy, wifi localization, internet of things, wifi security, 802.11**

1 INTRODUCTION

With the rapid expansion and adoption of wireless IoT (Internet-of-Things) devices, it has become easier for hostile actors are to spy on and infringe the privacy of individuals and corporations. In Surveillance devices can take the form of cameras, microphones, or presence detectors, which are either (i) miniaturized to the point of being unnoticeable to the human eye or (ii) disguised as harmless devices, such as wall plugs or light bulbs. Existing technologies to address this problem either focus on detecting magnetic signatures specifically emitted by cameras, which are unable to detect other kinds of surveillance devices, or are designed as 'wands' that measure the Received Signal Strength (RSS) at a certain bandwidth, which can be fooled by intermittently transmitting devices or low transmit power IoT devices.

In this report, we propose WiSpider, a platform to detect and localize these hidden, hostile wireless devices non-cooperatively. Specifically, we focus on sensing devices operating on the WiFi band, which makes up a large part of the IoT market [3]. To address the shortfalls of previous implementations, we will passively sniff device addresses from the air, then force channels between our product and target devices by exploiting the 802.11 WiFi protocol. We will then extract the channel information over a series of user movements and measurements, and aggregate those measurements into an AR (Augmented Reality) interface which shows the user where we suspect the hidden devices to be. To target the majority of off-the-shelf IoT spyware, we intend our product to be used by both individual and corporate users.

2 USE-CASE REQUIREMENTS

2.1 Constraints

Sensing WiFi: While IoT devices span a diverse range of wireless frequencies and modalities, including LoRa and BLE (Bluetooth Low-Energy), we restrict our implementation to specifically sensing WiFi devices for this proof of concept, because of the ease-of-use and wide availability for non-advanced attackers.
Indoors: Our system will be designed and tested solely indoors. This is where the majority of privacy-infringing devices are located, and indoor environments are inherently more challenging for wireless sensing due to obstructions and dense multipath, so the system could be generalized to outdoors.
No moving devices: We are focused solely on sensing devices permanently or semi-permanently planted in the environment - not computer, phones, or laptops, which may change position with their user.

2.2 Metrics

Weight and Size: We require WiSpider to *weigh less than 10 pounds* and *fit within 1 cubic foot.* We envision our device being easy for user to carry, to move between rooms as they are scanning, and to pack and take with them if they are going to a different place. To this end, our device needs to be both lightweight and small.
Cost: We require WiSpider to *cost less than $150.* This places us in-line with most currently available commercial solutions and puts WiSpider in the price range of both private and corporate users.
Detection Rate: We require a *device detection rate of 90%*, where the detection rate is defined as how often we are able to detect a device that is located within our minimum range. If a device's address is observed or sniffed, it would count as being detected.
Scan time: We require WiSpider to *finish scanning within 5 minutes.* This enables the end-user to detect devices in a relatively short period of time.
Lateral accuracy: We require WiSpider to *localize a device within 1m from its actual location.* The user can then easily search manually within the detection zone.
Range: We require a *minimum detection range* of 10m, meaning that devices up to 10 m away can be detected by WiSpider. This is to allow a user to scan an entire room easily and with little movement (i.e. without needing to manually sweep across every wall).
Resolution: WiSpider requires a *spatial resolution of 0.5m*, measuring how well the product is able to distinguish between co-located devices. Wireless devices may be

very closely located to each-other: in order to provide the user with information about devices that could be hiding near another signature, we need to be able to distinguish closely located devices from each other.

2.3 Features

Detect and locate hidden devices: The main feature of WiSpider is to detect and locate hidden devices. The device will sniff WiFi packets to detect devices, and exploit Polite Wifi[1] behavior to gather Time of Flight and RSS data. These data will be used to localize devices.
Non-cooperative: It is likely that a user will want to scan for hidden IoT devices in an unknown environment. WiSpider is able to operate in such situations where the user is not connected to the same network that the IoT devices are connected to.
Visualization: WiSpider must be able to show the user where any detected devices are located in the scanned space.

We acknowledge there is potential for this technology itself to infringe on the privacy of network users by giving their location to an unpriveledged third party. However, this method has already been developed as a vector for attackers to use in other projetcs [2]. Additionally, we believe that the restriction that WiSpider will only localize immobile devices will make our product much more appealing in the toolkit of a defender than an attacker.

3 ARCHITECTURE

3.1 Overall System Diagram

Our architecture has 3 major components: a hardware component, using WiFi chips and a microcontroller; a signals component, processing those measurements using an MLE (Maximum-Likelihood Estimation) localization algorithm; and a software component, creating the AR interface and integrating all of WiSpider's devices.

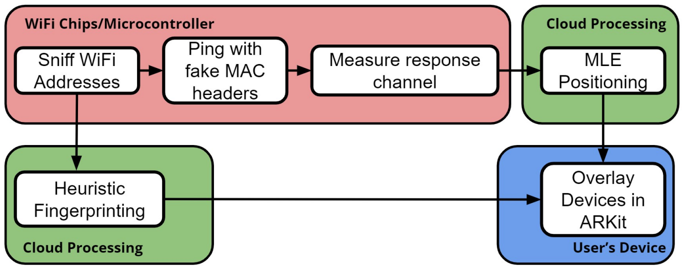


Figure 1: The system diagram of WiSpider, showing the overall pipeline and where each component runs.

The operating steps of WiSpider are as follows: (i) Sniff WiFi packets and addresses passively from the surrounding area (ii) Ping those addresses to non-cooperatively create a channel between WiSpider and the target devices using the "Polite WiFi" exploit described in [1]. (iii) Convert measurements into 2d positioning characteristics (range, angle) and aggregate across user movements to localize where the devices are. (iv) Visualize to the user where the devices are located in the space, and any information we can glean from their unencrypted packet headers (i.e. manufacturer, data rate). This overall pipeline is shown in Figure 1.

3.2 Hardware Diagram

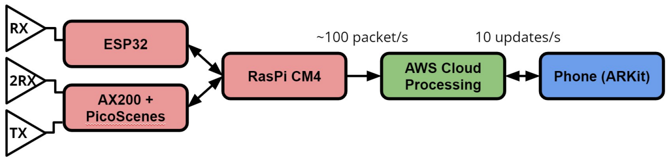


Figure 2: WiSpider operates 2 different WiFi chips simultaneously to extract different measurements of the channel.

Due to the limitations of individual WiFi chips, which will be further described in Section 6, we elect to use two different WiFi chips to sense different aspects of the channel, as shown in Figure 2. At a high level, one will be used for range measurement and another will be used for angle measurement. The diagram also shows the rate at which we expect data transfer between each platform.

3.3 Localization Diagram

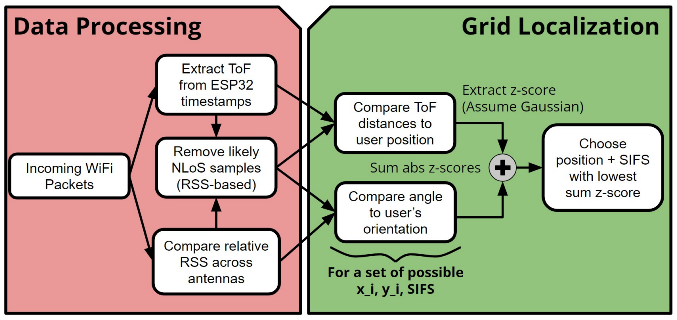

Data Processing
Grid Localization
Incoming WiFi Packets
Extract ToF from ESP32 timestamps
Remove likely NLoS samples (RSS-based)
Compare relative RSS across antennas
Compare ToF distances to user position
Compare angle to user's orientation
Extract z-score (Assume Gaussian)
Sum abs z-scores
Choose position + SIFS with lowest sum z-score
For a set of possible x_i, y_i, SIFS

Figure 3: WiSpider extracts information from the WiFi channel to localize the target in both angle and range, summing the z-score of the 2 estimates to combine them.

Figure 3 shows the outline of how WiSpider handles the measurements it receives. In short, it scans a set of possible target (x, y) locations and calculates how likely the target device is to land there based on WiSpider's measurements. It then combines those two estimates over thousands of measurements per each device, and uses the most likely location as our position estimate.

4 DESIGN REQUIREMENTS

Physical: As stated in the use-case requirement, WiSpider needs to be both lightweight and small, and our design will weigh less than 10 pounds and fit within 1 cubic foot.
Sniffing rate: WiSpider will sniff and identify access points for 1 minute, and will use fake beacon packets to discover devices connected to each access point for another minute.
Injection rate: 100 fake packets per second will be sent.
ToF measurement rate: Since half of the fake packets will be fake beacon packets to prevent the device from going to sleep, there will be 50 ToF (Time-of-Flight) measurements per second.
ToF accuracy: It is hard to test the ToF accuracy in real setting. Therefore, in a test setting with clear line of sight, we expect that the distance calculated from measured ToF is off from the actual distance of the device by no more than 2 meters. Testing method is further discussed in section 7.
Distinguish devices: WiSpider will distinguish different devices by obtaining each device's unique identifiers, such as its MAC address, bandwidth, or transmission rate. This should work even if the devices are physically close together.
Locate infrequently transmitting devices: Even if a device transmits infrequently, if the device is detected at least once, WiSpider will be able to locate it.
Self-localization accuracy: Our self-localization error would be within 0.3 meters such that tracking devices can be done to an accuracy of 1 meter across a large number of measurements.
AR accuracy: In order to make it very clear for a user where a device is located, the AR visualization error will be within 0.3 meters of the actual calculated location. The AR visualization will also be smooth (even if there is some error or offset, there should be minimal jitter) so that the user has a comfortable experience looking at it.

5 DESIGN TRADE STUDIES

In reaching our current design plan, we explored a variety of dimensions of the design space, both in terms of design specifications and implementation details.

5.1 End-User Interface

Since our use case is primarily based on allowing users to easily locate hidden devices, we explored different options on how to show users where the detected devices are located. Most existing solutions [4] are in the form of a device (a wand or handheld antenna) that will beep or show a display when brought near strong RF (Radio-Frequency) emitters. We note that these implementations have major downsides, including needing to sweep the entire room at a very high granularity, potential interference from other devices, and the inability to distinguish multiple devices in the same area.

We also explored doing a hybrid solution, where the user would be able to see directional and signal strength information, as well as decoded information distinguishing multiple devices by MAC address. This would make it easier for the user to distinguish devices without requiring the system to have a self-localization stack (since the user would be able to mentally determine where the devices are, based on the results they see). While this reduces complexity and cost, it also places a significant cognitive load on the user, especially if there are many wireless devices (most of which might be benign) in the room.

We finally settled on using an augmented reality visualization to show the user exactly where the devices are. This catches the best of all worlds - the user can walk around for a while before looking at the data; it can distinguish devices from each other (and filter out trusted devices); and give the user a relatively accurate idea of where the hidden devices might be.

5.2 Wireless Detection: SDR vs Dedicated Receiver

We initially investigated the use of a Software-Defined Radio (SDR) for detecting wireless devices. There are Wi-Fi baseband implementations [6] that can run on an SDR. These implementations can provide us raw IQ-level data (such as phase-difference between antennas, and extremely-precise time-of-flight) along with the decoded 802.11 packets. This would give us the highest quality of data, but an SDR with such capabilities would cost $2-4K minimum, and require an external power source in order to function (along with a lot of specialized software). While this may be appropriate for certain use-cases (including military and RF spectrum enforcement), our end-users are mostly individual users and businesses, for whom a several-thousand-dollar piece of hardware would be unreasonable.

As an alternative, we decided to use hardware containing dedicated Wi-Fi receiver chips. This meets our end-user requirements more effectively, since it is much lower in cost (¡$50) and complexity (Wi-Fi cards can be slotted into any standard laptop, and there are even standalone Wi-Fi microcontrollers such as the ESP32). These will give us a lower precision of data - we do not get any raw IQ data, only time-of-flight based on the internal clock. We do get some limited multi-antenna phase data through the CSI (Channel State Information) which is normally used for MIMO (multi-in-multi-out) and beamforming, but it is not quite as powerful as we would get with an SDR. However, based on a review of similar work [2] [8], we believe that it is still possible to do sufficiently-accurate localization with only off-the-shelf Wi-Fi hardware.

5.3 Wi-Fi Interface Hardware

We explored several different options for Wi-Fi hardware/software stacks. Because we want as much raw data as possible (time-of-flight, signal strength across multiple antennas, phase shift) we are limited to stacks which expose such data to a user-level application (as writing modded firmware for a Wi-Fi card is out of scope for our work).

5.3.1 AX200 + PicoScenes

The Intel AX200 series of Wi-Fi cards has a relatively good firmware-level support for extracting raw CSI and timing information, thanks to its support for modern MIMO and time-of-flight protocols. It is also supported by PicoScenes, which allows us to easily configure the card and programatically extract most of the raw CSI data that we care about. The AX200 supports two antennas, allowing us to extract phase-difference across antennas a half-wavelength apart, and/or see the variation in power across antennas. The major downsides are (1) this requires a user to install the card inside their laptop and run an antenna cable out the back of the laptop and (2) the free version of PicoScenes doesn't support high-precision timestamping, so we cannot do time-of-flight analysis.

5.3.2 IWL5300 + Linux CSI Tool

Arguably the most popular stack for Wi-Fi research (used by thousands of papers), especially before PicoScenes; the IWL5300 supports 802.11n Wi-Fi and three distinct antennas. The latter feature is particularly useful for working with directional antennas, as we could use one omnidirectional and two directional antennas to get a very strong idea of the orientation towards a device. The major downside with the Linux CSI Tool is that there is no support for 802.11a, which is required for the Polite WiFi attack that we rely on for time-of-flight ranging.

5.3.3 QCA9500 + Atheros CSI Tool

The Atheros CSI tool has (with the Atheros QCA card) a very similar feature set but slightly higher resolution. It however suffers from the same other issues as the Intel CSI tool, including the lack of 802.11a support.

5.3.4 ESP32

The ESP32 is a commonly-used microcontroller which has a very low overhead (hence we can do clock-cycle-accurate timestamping in software), well-documented open-source libraries, and allows us fairly raw access to the 802.11 WiFi stack. Since it can run custom code, we can use the serial port to relay results back to a computer over USB. It has been used in past ToF-based research such as Wi-Peep. The major downside with the ESP is that it only supports a single antenna, so we cannot do phase or CSI measurements. It also doesn't support high-precision measurement of frame injection times, so we must use a second Wi-Fi device to inject frame and then measure the reception time of the ping, and the response frames from the ESP.

We eventually settled on using a combination of AX200 + PicoScenes (for sniffing, injection, and phase information) and ESP32 (for time-of-flight), interfaced together via a USB serial port.

5.4 Self-Localization

We explored a few different options for self-localization (tracking the antenna itself, within the room). We looked at various LIDAR and depth camera options, including RPLIDAR and Intel Realsense. However, these would very quickly bust our cost requirements - a depth camera like a RealSense D455 alone costs ¿$400 and a T265 (their flagship odometry camera) costs around $300. Additionally, the long-term accuracy of the T265 is not very high, it can have drift of several meters after a few minutes of movement. It is generally meant to be used when fused with other data sources such as a 3D scan (obtained with a LIDAR or depth camera). This fusion would add additional cost and software complexity to our system.

Instead, we settle on using a mobile device as our camera. Because of augmented-reality and 3D scanning applications, high-end mobile phones come with very well-calibrated cameras (sometimes even depth cameras) and IMUs; and augmented reality stacks such as ARKit will handle 3D scanning and loop closure under-the-hood. This means that we can develop an augmented reality app using one of these tools under-the-hood, and it will give us accurate, low-drift localization. This also has the additional benefit of allowing us to use it for visualization - instead of having two separate origin points (one for our antenna reference frame and one for our visualization reference frame), both will be relative to the AR scene so there will be no added error in the visualization.

6 SYSTEM IMPLEMENTATION

The overall system works in 4 steps.

6.1 Device Detection

We will use Scapy[7] to passively sniff packets and discover devices, the method of which is inspired by Wi-Peep.[2] During this phase, WiSpider sniffs and identifies access points and their SSIDs (Service Set Identifiers). For each access point identified, we inject a beacon frame that appears to be coming from the access point, with the TIM (Traffic Indication Map) bitmap set to all 1's. This will make the devices on that network send a response packet to the access point; we can sniff these packets to detect all the devices connected to that access point.

6.2 Device Pinging

Once we have a list of devices, we use the Polite Wifi mechanism to ping the devices. Specifically, we create a fake packet with null data frame, and send them to the devices; the devices respond with an ACK, even though the packet itself has an invalid source address and no data. This is because the maximum time to send an acknowledgment, the SIFS (Short-Interframe Space) time, required by the 802.11 protocol is too short for the device to actually validate the address before responding. To prevent the devices from going to sleep, we alternate these null packets with a beacon frame with TIM bitmap set to 1 which will force all devices connected to each AP to respond. Using the device responses, we can measure ToF, CSI (Channel State Information), and RSS (Received Signal Strength) data to use for localization. The measurement of RSS data will be done with the PicoScenes framework, using Intel's AX200 WiFi card, while the ToF measurements will be done by counting the clock cycles between receiving the outgoing packet and the acknowledge packet, using the ESP32 microcontroller. The ESP32 is not capable of the one-way CSI measurement the AX200 is, while the AX200 is not capable of the lightweight clock measurement the ESP32 is.

6.3 Device Location Mapping

We will localize the target devices in the range and angle domains separately, then combine those measurements to find the true location. In order to speed up processing, instead of using gradient descent or other large-processing methods, we formulate our problem as a highly parallelizable grid search. We define a set of (x_i, y_i) that spans the entire space scanned by the user. First, let us define how we extract the location estimate from the measurement ToF.

$$\begin{aligned} d_i = min_{SIFS}\{|\sum_{u=1}^{u=U} \sqrt{(x_i - x_u)^2 + (y_i - y_u)^2} \\ + \; c \cdot (SIFS - \delta_u)|\} \end{aligned} \tag{1}$$

where d_i is the sum of absolute distances between point i and each ToF measurement, u is the sample index over U total samples, (x_u, y_u) is our estimate of the user's location, c is the speed of light, $SIFS$ is optimized over the range $(6\mu s, 16\mu s)$ to account for the unknown SIFS of the device, and δ_u is the measured ToF.

Next, let us define how WiSpider will extract the angle. Using PDoA (Phase Difference of Arrival), we can compare the phase across the two RX antennas of the AX200 to determine the AoA of the WiFi signal. However, with only two antennas, this will be fairly imprecise, so we will get a finer estimate by comparing the RSS of a directional and non-directional antenna.

$$\theta_1 = sin^{-1}(\frac{\lambda * (\Delta\phi - \phi_c)}{2 * pi * l}) \tag{2}$$

where θ_1 is the rough angle estimate, λ is the wavelength of WiFi, l is the distance between antennas, and $\Delta\phi$ is the measured phase difference across antennas minus a constant ϕ_c, which we will experimentally measure, to account for different antenna lengths.

$$min_{\theta_2=-5^\circ:5^\circ}\{|\frac{RSS_{d,u}}{RSS_{o,u}} - Rx(\theta_1 + \theta_2)|\}$$
$$\theta_{f,u} = \theta_1 + \theta_{2,min}$$

Where $\theta_{f,u}$ is our final angle estimate for each sample u, RSS_d is our directional antenna's RSS, RSS_o is the omnidirectional antenna's RSS, and $Rx(\theta)$ is a pre-measured transfer function between the of the actual radiation pattern comparison between the two antennas. This function will essentially search the nearby angles from θ_1 to see which angle most closely matches the radiation pattern we observe. This may output multiple angles due to the symmetry of antenna radiation patterns, but this is accounted for in our next aggregation step.

$$\Phi_i = min_{\theta_{f,u}}\{\sum_{u=1}^{u=U} tan2^{-1}(\frac{y_i - y_u}{x_i - x_u}) + \Phi_u - \theta_{f,u}\} \tag{3}$$

Where Phi_i is the sum angle error, $tan2$ is a tangent function that will give a full 2π output, and Φ_u is the measured user orientation.

We aggregate both measurements together via calculated z-scores with σ's derived from our unit tests and choose the point (x_i, y_i) with the minimum z_i as our estimated device location.

$$z_i = \frac{\Phi_i}{\sigma_\theta} + \frac{d_i}{\sigma_d} \tag{4}$$

6.4 Self-Localization + AR Visualization

Our system needs to know its own location (the location of its antennas) at all times, in order to implement the aforementioned device mapping algorithms. It is also very sensitive to drift, since otherwise the accuracy of our measurements will deteriorate over time (i.e. the measurements taken at the start of a 5-minute scan vs at the end of the 5 minutes). We will use a mobile phone's augmented reality stack (ARKit) on top of which we will develop our frontend application. The ARKit stack includes a fairly well-calibrated camera model as well as the ability to do loop-closure by matching against earlier scans, so it is unlikely to drift significantly while walking around a room. Since we can get the device's location and orientation from the AR app, we will stream this data back to our server to be fused together (by matching timestamps) with the measurements taken using the Wi-Fi frontend.

Once the scan has been taken and we have a confident measurement as to the location of a given Wi-Fi device, we will visualize it in the same AR interface. This will be done by showing an overlay on the user's phone, in the form of a grid at the floor-level. For locations where devices are detected, we will show a 3D box around the device. Looking directly at the 3D box will pop up a textual display of some info about the device (MAC address, manufacturer if known, datarate, etc.). We will also have a list of devices and MAC addresses shown on a separate floating panel in the AR world, from which the user can filter out known-trusted devices (like their own phones and laptops).

7 TEST & VALIDATION

We will conduct a unit test on individual sub-systems to check if they meet the design requirements, followed by an integration test of the use-case requirements. We will use several IoT devices specified in Table 2, along with our personal devices to test these requirements. Specifically, the tests will be done for non-moving devices that use 2.4 GHz, and will check if they meet use-case requirements and design requirements specified in Sections 2 and 4.

7.1 Unit-Tests for Device Sniffing

The first test will be done with a single known device, and detecting whether our device can detect that device with the mechanism explained in section 6.1. Then, we will put several devices - phones, tablets, smart cameras, smart plugs, smart lights - in the same network and determine if WiSpider can detect all of them. Then, we will put these devices under two different access points and test if WiSpider can detect all of them.

7.2 Unit-Tests for Device Pinging

First unit test will consist of pinging a single known device, and seeing whether we are successfully able to get a response using the Polite WiFi mechanism. Then, we will have it connected to a known access point, and use the method explained in section 6.2 to consistently get response from pings for 2 minutes without the device going to sleep. Then, we will test it on 5 devices on a single access point, and test if the devices continuously respond to the pings. Lastly, we will test the devices on two different access points on whether they consistently respond to the pings.

7.3 Unit-Tests for ToF

We need to test for ToF, as small inaccuracies in ToF measurement can lead to large error in localization. We do this by collecting ToF against one device at known location. While SIFS is different across devices and therefore is unknown, we can figure it out by having the device and WiSpider right next to each other, with distance of 0m. With the SIFS figured out, we will move the known device with a constant velocity, and compare the measured time against the expected time. Once we verify that the measurements are accurate, we will test it with multiple devices.

7.4 Unit-Tests for Localization & AR

To unit-test our localization performance, we will set up a test AR app which datalogs the user's position. We will then follow a known track (measured using a measuring-tape)

follow known track, check deviation of measurements from track. place AR objects at known locations relative to origin, visually verify that they match the location

8 PROJECT MANAGEMENT

8.1 Schedule

The schedule is shown in a Gantt chart in Fig. 4. Most of the parts in the first half are individual and therefore parallelizable. This is intended to make sure that we're able to test each parts, which we allocate a week for. After a week of testing the initial version on a laptop, we allocate the latter half to integrating the individual parts into a full system, and a week of testing those. We allocate a week of slack, and also reserve spring break as empty, so that team members who feel ahead or on schedule can take a break accordingly, and those who are behind can use the break to catch up.

8.2 Team Member Responsibilities

Thomas will be in charge of the signal processing and localization algorithm; specifically, he will deal with tracking and filtering algorithm, multipath and noise handling, and device localization using the ToF and RSSI data collected. Ethan will be in charge of the software side, dealing with packet sniffing, device detection and identification, measurement using PicoScenes, and software pipeline during integration. Anish will be in charge of hardware, sensing, AR localization, and visualization - he will work with antenna and sensing, as well as the AR visualization with which the user will interact and the localization in said AR.

8.3 Bill of Materials and Budget

Our Bill of Materials for WiSpider is as listed in Table 1. This includes the WiFi network card and antenna used for sniffing and injecting packets, ESP32 to accurately measure ToF, and the associated cables and connectors. We expect the end-user would already own a mobile phone and a laptop, which they can run our software on (hence we do not include these in the system cost). The total cost for WiSpider is \$118.28, which is well within our \$150 target.

We also have a second bill-of-materials for an IoT testbed, listed in Table 2. Our testbed includes a router and several devices (both cameras and benign IoT devices such as plugs and lights) which we will use to test our system's detection and localization capabilities. Our testbed comes to a cost of \$234.89, which is reasonable given how many distinct devices we are testing (including hidden surveillance cameras).

Our total cost comes to \$353.17 which is well within our \$600 cap, giving us enough buffer for unexpected additions,

Table 1: Bill of materials for WiSpider

Description	Model #	Manufacturer	Quantity	Cost @	Total
Intel AX200 Wi-Fi card	AX200	Intel	1	$31.81	$31.81
Antenna Coax Extender	5m RP-SMA 2-pack	Bingfu	1	$8.99	$8.99
ESP32 Microcontroller	ESP32-S2 Saola 1R	Espressif	1	$14.50	$14.50
Micro-USB Cable	Micro-USB to USB-A	Generic	1	$3.00	$3.00
2.4 GHz Directional Antenna	High Gain Yagi	TECHTOO	2	$29.99	$59.98
Phone	Varies	Varies	1	Owned by User	$0.00
Laptop	Varies	Varies	1	Owned by User	$0.00
					$118.28

Table 2: Bill of materials for IoT testbed

Description	Model #	Manufacturer	Quantity	Cost @	Total
WiFi Router	Archer A54	TP-Link	1	$34.99	$34.99
Indoor Security Camera	IPC-TA22CP-G	IMOU	1	$29.99	$29.99
Security Camera	Speed 23T	WGV	1	$24.99	$24.99
Spy Camera	B0837S522C	ALPHA TECH	1	$49.99	$49.99
Spy Camera	B0B8NHZ2XJ	HDCOO	1	$35.96	$35.96
Smart plug	B5081	Govee	1	$28.99	$28.99
Smart light	KL110	TP-Link	1	$9.99	$9.99
Motion Sensor Switch	MFA05	Lesim	1	$19.99	$19.99
					$234.89

mechanical parts/3D printing, and spare parts in case we damage anything while testing.

8.4 Risk Mitigation Plans

We mitigate our risks by thoroughly unit testing individual components before integration, and following it with an integration test. Additionally, we allocate enough time and slack in our schedule to have a buffer in case something gets delayed.

9 RELATED WORK

Polite WiFi[1] is a behavior in which a WiFi device will respond to any frame with an ACK, if the destination address matches its own MAC address. This can create many opportunities as well as threats; for instance, while it can be used to make WiFi sensing more convenient, one can also use this behavior to identify and localize many devices that they should not be able to, or drain a target device's battery by forcing it to continuously acknowledge the pings.

Wi-Peep[2] is one such example of how this behavior might be used maliciously. Using off-the-shelf WiFi modules and a cheap drone, they were able to detect devices on a network they're not connected to, measure ToF using Polite WiFi behavior, and localize devices from outside a building.

Lumos[8] is a similar work aimed at identifying and locating hidden devices with their phone. Lumos differs in the approach they take, in that they fingerprint the devices with a machine learning approach, and use RSSI and VIO for localization.

PicoScenes[6] is an OSS framework for WiFi CSI and metadata collection. It supports CSI measurement from commercial off-the-shelf NICs and SDRs, while also allowing for packet injection. It also is possible to use MATLAB and python with its CSI format, as well as creating plugins.

The Intel Open Wi-Fi RTT Dataset[5] is an open-source dataset of time-of-flight Wi-Fi measurements between various client devices and access-points in an office environment. This dataset has been used to develop localization algorithms based on Wi-Fi measurements, however we do note that the dataset was captured using an API for co-operative Wi-Fi localization (the access points support a dedicated method of getting time-of-flight measurements, as opposed to the non-cooperative approach we intend to take in our work).

10 SUMMARY

WiSpider is a platform that detects and localizes hidden wireless devices in a non-cooperative environment. Using the device, the user will be able to easily locate where hidden devices are, by looking at the AR interface. Additionally, the device will be easy to carry around and low-cost. Upcoming challenges include accurately capturing ToF data, accurately localizing devices with collected data, and integrating all the parts together.

References

[1] Ali Abedi and Omid Abari. “WiFi Says ”Hi!” Back to Strangers!” In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. HotNets ’20. Virtual Event, USA: Association for Computing Machinery, 2020, 132–138. ISBN: 9781450381451. DOI: 10.1145/3422604.3425951. URL: https://doi.org/10.1145/3422604.3425951.

[2] Ali Abedi and Deepak Vasisht. “Non-Cooperative Wi-Fi Localization amp; Its Privacy Implications”. In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. MobiCom ’22. Sydney, NSW, Australia: Association for Computing Machinery, 2022, 570–582. ISBN: 9781450391818. DOI: 10.1145/3495243.3560530. URL: https://doi.org/10.1145/3495243.3560530.

[3] Jacob Arellano. “Bluetooth vs. Wi-Fi for IoT: Which is Better?” In: (July 9, 2019). URL: https://www.verytechnology.com/iot-insights/bluetooth-vs-wi-fi-for-iot-which-is-better.

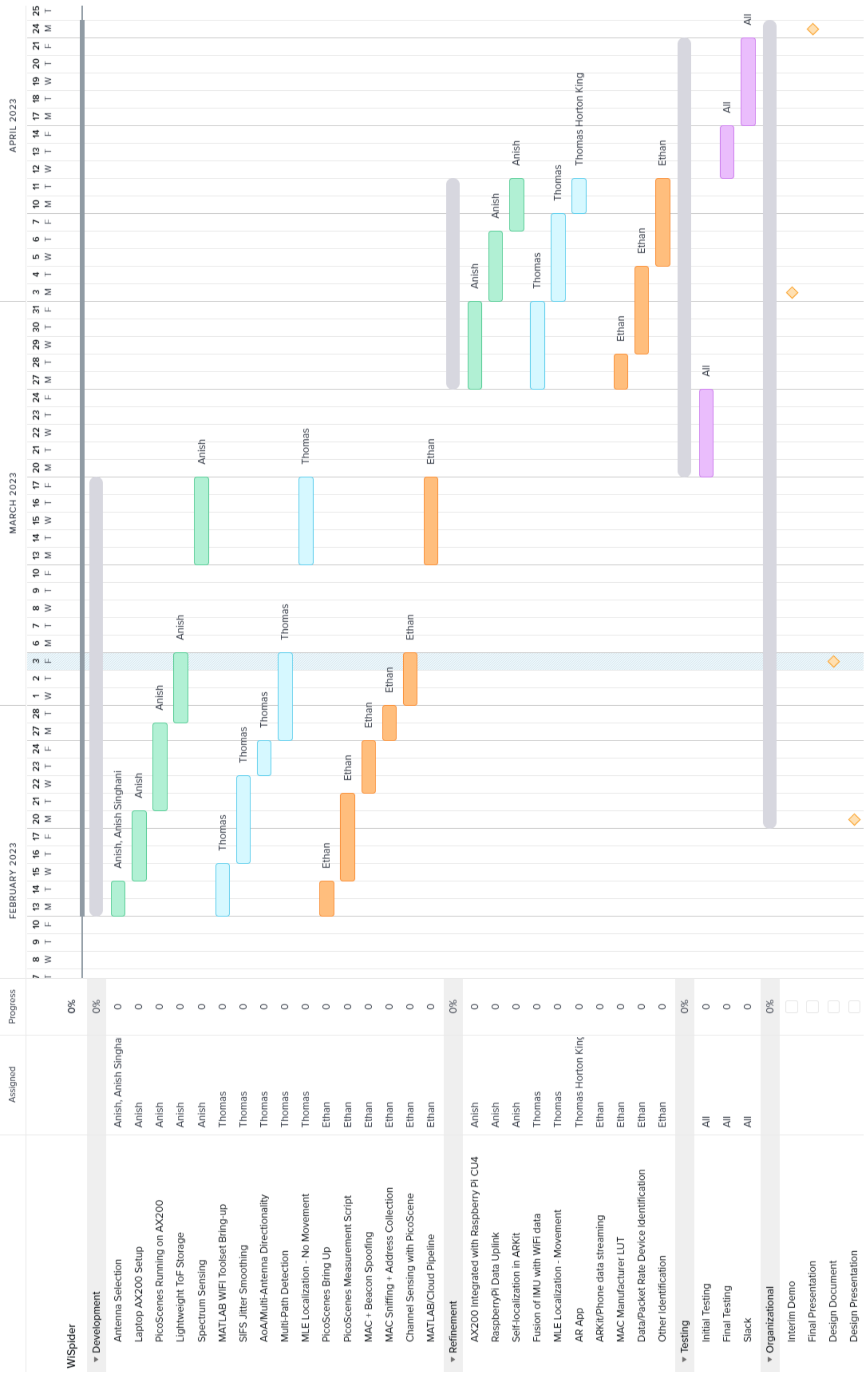
[4] Brick House Security. “Concealable wand detects hidden RF and wireless signals”. In: (2019 [Online]). URL: https://www.brickhousesecurity.com/counter-surveillance/rf-wand/.

[5] Nir Dvorecki et al. *Intel Open Wi-Fi RTT Dataset*. 2020. DOI: 10.21227/h5c2-5439. URL: https://dx.doi.org/10.21227/h5c2-5439.

[6] Zhiping Jiang et al. “Eliminating the Barriers: Demystify Wi-Fi Baseband Design And Introduce PicoScenes Wi-Fi Sensing Platform”. In: *CoRR* abs/2010.10233 (2020). arXiv: 2010.10233. URL: https://arxiv.org/abs/2010.10233.

[7] Philippe Biondi. “Scapy”. In: (2008). URL: https://scapy.net/.

[8] Rahul Anand Sharma et al. “Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment”. In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1095–1112. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/sharma-rahul.



	Assigned	Progress
WiSpider		0%
Development		0%
Antenna Selection	Anish, Anish Singha	0
Laptop AX200 Setup	Anish	0
PicoScenes Running on AX200	Anish	0
Lightweight ToF Storage	Anish	0
Spectrum Sensing	Anish	0
MATLAB WiFi Toolset Bring-up	Thomas	0
SIFS Jitter Smoothing	Thomas	0
AoA/Multi-Antenna Directionality	Thomas	0
Multi-Path Detection	Thomas	0
MLE Localization - No Movement	Thomas	0
PicoScenes Bring Up	Ethan	0
PicoScenes Measurement Script	Ethan	0
MAC + Beacon Spoofing	Ethan	0
MAC Sniffing + Address Collection	Ethan	0
Channel Sensing with PicoScene	Ethan	0
MATLAB/Cloud Pipeline	Ethan	0
Refinement		0%
AX200 Integrated with Raspberry Pi CU4	Anish	0
RaspberryPi Data Uplink	Anish	0
Self-localization in ARKit	Anish	0
Fusion of IMU with WiFi data	Thomas	0
MLE Localization - Movement	Thomas	0
AR App	Thomas Horton King	0
ARKit/Phone data streaming	Ethan	0
MAC Manufacturer LUT	Ethan	0
Data/Packet Rate Device Identification	Ethan	0
Other Identification	Ethan	0
Testing		0%
Initial Testing	All	0
Final Testing	All	0
Slack	All	0
Organizational		0%
Interim Demo		☐
Final Presentation		☐
Design Document		☐
Design Presentation		☐

Figure 4: Gantt Chart