Integrating LoRaWAN and TockOS for Secure, Low-power Geolocation Solutions

CS Distinguished Major Program Thesis
University of Virginia

Author	Yancheng (Ethan) Zhou
Research Advisor	Brad Campbell
Second Reader	Kun Qian
Submitted on	April 19, 2024

Contents

1 Abstract

Growing demand for asset tracking is fueling the need for IoT applications that support low-power, long-range communication and accurate geolocation capabilities. Seeed Studio's Wio-WM1110 Module, which features a LoRa transceiver for extensive, low-power network coverage, GNSS and Wi-Fi scanning for location tracking, and multiple sensors, becomes a promising platform for developing multi-functional geolocation systems. However, executing multiple applications concurrently on a single device presented significant stability challenges. Furthermore, the high complexity of the legacy code for this module makes it vulnerable to errors that can crash the application. Tock OS, designed for running multiple concurrent applications in low-power and low-memory environments, offers robust solutions to these challenges. It supports independent updates for individual applications and its secure architecture maintains the system operational even when errors occur within the network stack. By integrating the Wio-WM1110 Module with Tock OS, this project also expanded Tock's support for LoRa and LoRaWAN functionalities and created the first Wi-Fi scanning application on the platform. By incorporating code from Seeed Studio's library, I activated Wi-Fi scanning for indoor geolocation and successfully joined The Things Network (TTN) to transmit the scan results online. Then I downloaded recent results from TTN and used the Google Geolocation API [1] to convert the Wi-Fi access points into geolocation data. When visualized on maps, data from three locations at or near UVA consistently revealed high precision in coordinates and range. Through empirical tradeoff analysis, I found that using four Wi-Fi access points balanced power consumption and geolocation accuracy. Moving forward, I could use the application to track critical assets in UVA, or in any indoor location that has TTN connectivity.

2 Background

2.1 LoRa and LoRaWAN

Long Range (LoRa) technology, supported by chirp spread spectrum (CSS) modulation, is a low-power protocol with exceptional ability to transmit data across long distances [2]. LoRa can cover a range up to three miles in urban areas, and up to ten miles or more in rural areas. Long Range Wide Area Network (LoRaWAN) builds upon LoRa by establishing a protocol for network communication between remote sensors and central servers via gateways. It is perfect for applications requiring low-power, long-range communication among a large number of devices that collect small amounts of data.

2.2 Wio-Wm1110 Module

The Wio-WM1110 Module, at the heart of the WM1110 Dev Kit, is designed for IoT developers to integrate low-power, long-range communication with precise location tracking into their applications. This module is equipped with the LR1110, a Semtech component that has a power-efficient LoRa transceiver and multi-functional scanning technologies [3].

Equipped with Wi-Fi and a GNSS antenna, the LR1110 solution could transition between outdoor and indoor activity automatically and utilize Semtech's LoRa Cloud Geolocation capabilities to determine asset location through cloud-based computations [4]. Moreover, the device has built-in sensors for temperature, humidity, and a 3-axis accelerometer, and it can be powered by both batteries and solar panels. Its rich feature set enables IoT developers to create advanced, multi-functional tracking applications.

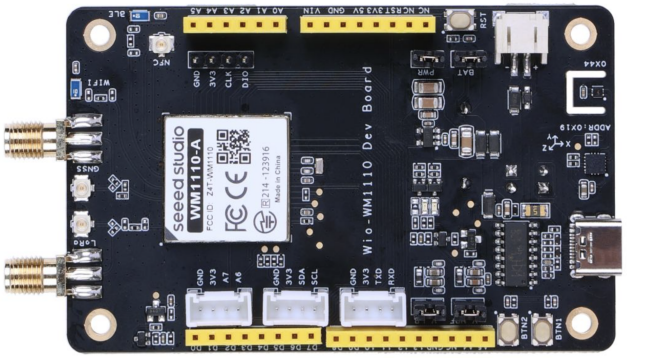

Figure 2.1: The Seeed Studio Wio-WM1110 Development Board [3]

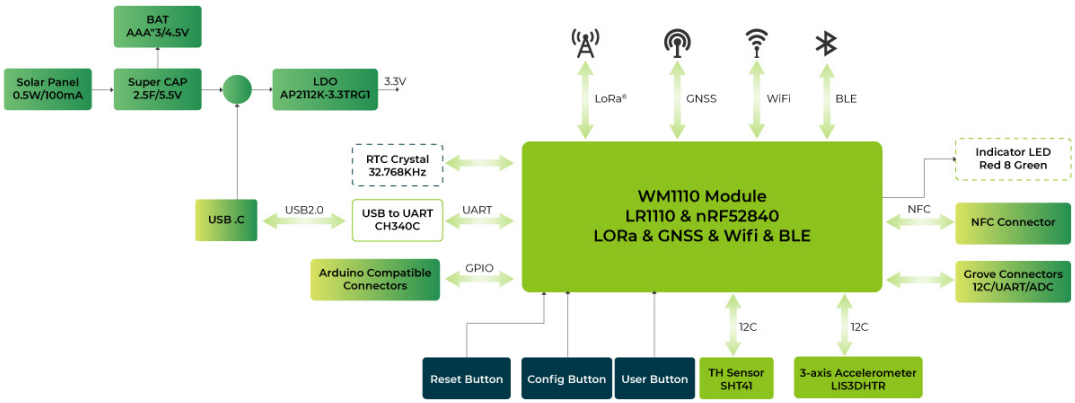

Figure 2.2: Schematic Diagram of Wio-WM1110 Module [3]

2.3 Tock OS

Tock is an embedded operating system designed for running multiple concurrent applications that are mutually distrustful [5]. It is compatible with over 30 development boards based on Cortex-M and RISC-V architectures. Tock OS can execute multiple applications on a single development board and support independent update of any individual application. Moreover, implementing both the kernel and device drivers in Rust, Tock provides high level

of security by ensuring compile-time memory and type safety. It also used memory protection units to protect the kernel from vulnerable device drivers and isolate those drivers from each other, so errors from them will not compromise the entire OS. My research project will help extend Tock's support for LoRa and LoRaWAN, and create the first Wi-Fi scan application executed upon Tock OS.

2.4 The Things Network

The Things Network (TTN) utilizes LoRaWAN to enable low-power devices to communicate with the internet over long distances without the need for cellular or Wi-Fi connectivity [6]. TTN thrives on the collaborative efforts of communities, businesses, and individual developers who expand its reach by setting up LoRaWAN gateways and deploying their IoT applications. TTN's network coverage is already quite strong in large cities and is progressively extending to smaller ones. In the Charlottesville community, seven gateways provide substantial coverage across the city, particularly around the School of Engineering. For my project, this means that the application can transmit data from sensors and Wi-Fi scans to the internet through uplinks from any location that can join TTN.

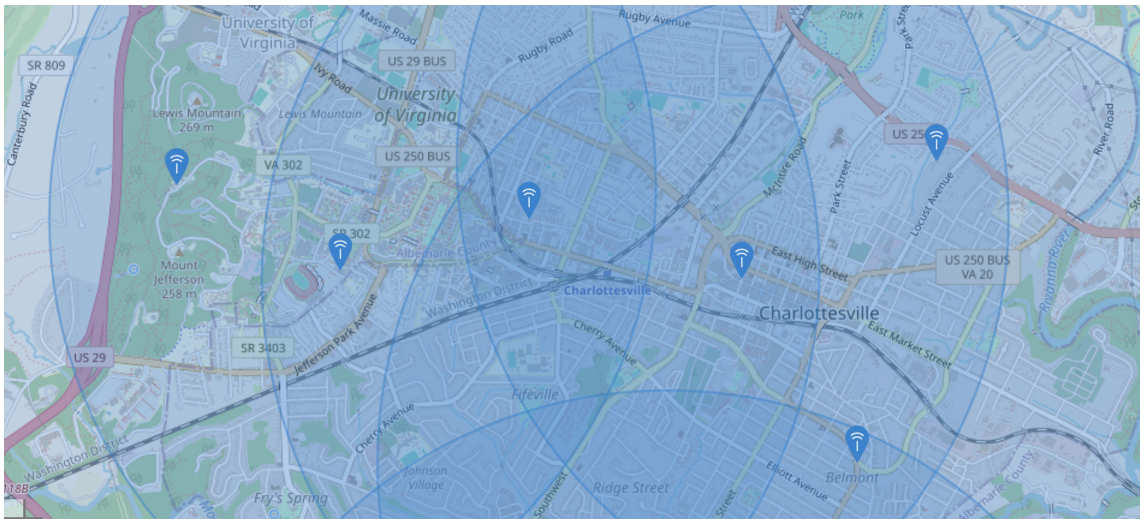

Figure 2.3: TTN Gateways and Coverage in Charlottesville

3 Challenges of Integrating Legacy Code into an Operating System

This project used legacy code from Seeed Studio to build applications over Tock OS on the Wio-WM1110 board. The legacy code repository includes geolocation middleware, ten example apps, hardware abstraction layer (HAL), LoRa basics modem, and LoRa chip configuration. Due to the complexity, I need to fully understand the app's structure and workflow before integrating it with Tock. Moreover, fundamental differences in design choices such as architecture, synchronicity, IRQ mechanisms, and other technical details create significant incompatibilities, complicating the integration process. Figure 3.1 illustrates the entire structure of the application from user space to hardware. Section numbers are annotated on the figure to show which part of the structure the section is discussing.

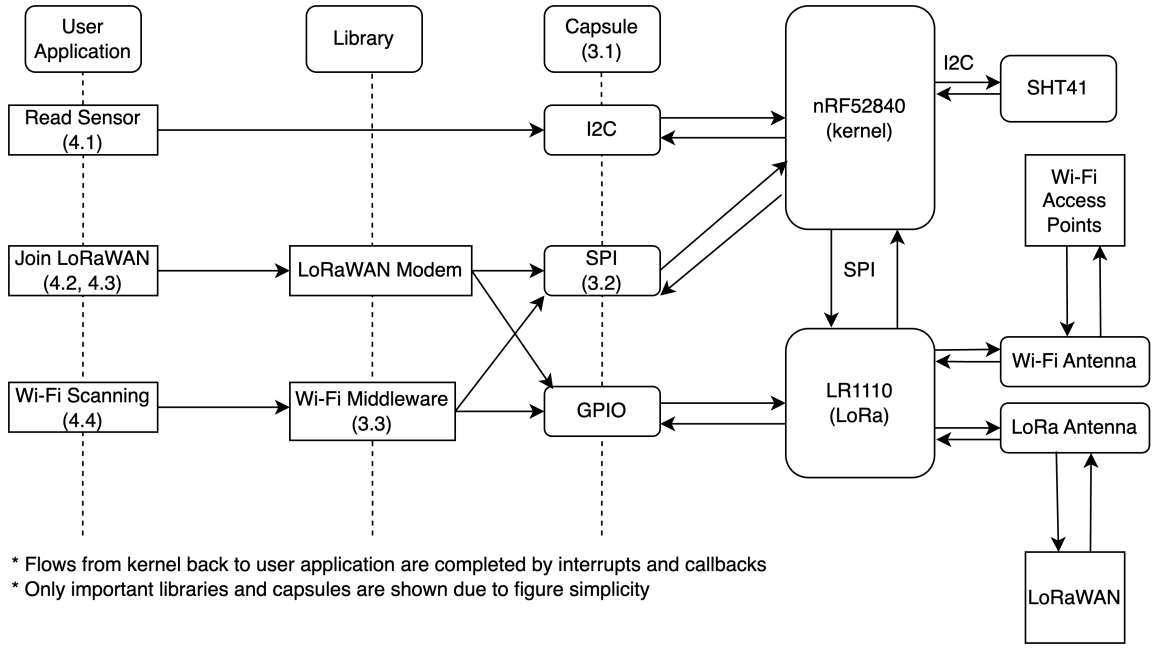

Figure 3.1: Overview of Entire Application Structure

3.1 Difference in Design Choice

The application uses Tock OS as its kernel while utilizing Seeed Studio's Semtech (SMTC) library within user space. To combine them, I created a library consisting of Seeed's code that can be compiled under Tock OS. A primary area of conflict lies within functions used in the hardware abstraction layer (HAL), which encapsulates critical functionalities including SPI, I2C communications, configurations for flash storage, GPIO, MCU, timers, and random number generators. In the HAL, the SMTC library relies heavily on Nordic's nRF series library for kernel-level functions, all of which must be substituted with equivalents from Tock.

Tock's microkernel architecture enhances security by using capsules—drivers written in Rust that are isolated and possess restricted privileges [5]. Consequently, interfacing with temperature and humidity sensors requires crafting a new Tock capsule for I2C communication in the Tock kernel, and then leveraging Tock libraries to facilitate data access in user space.

Beyond structural distinctions, nuances in data transmission methods present additional challenges. The Seeed's code requires manual handling of SPI's Chip Select pin and bit-by-bit setup for commands. Conversely, Tock handles operations like LoRa read/write with a single function call from the Tock library, which enhances convenience and prevents potential bugs. I adapted the Tock function by creating its synchronous equivalent, clearing error status in the context, and consulting the LR1110 datasheet for accurate data transmission to and from the chip.

3.2 Asynchronous and Synchronous Operations

Tock OS is fundamentally asynchronous while Seeed's code is designed synchronous, so combining them requires meticulous approaches to avoid conflicts. Tock is designed to handle multiple tasks concurrently without blocking the main execution flow for I/O operations. This is beneficial in power-constrained environments where processors spend much of their time in low-power modes, waking only in response to events or interrupts. In contrast, a significant portion of Seeed's code adheres to a synchronous model where the code execution blocks until an I/O operation is complete. This straightforward approach for single-threaded applications can lead to inefficiencies by blocking hardware access, causing CPU underutilization and increased latency.

To resolve conflicts between Tock's asynchronous kernel and Seeed's synchronous user space, I made adaptions in both directions in the hardware abstraction layer (HAL) that bridges them. First, in the LoRa physical layer, for operations like SPI read and write for LoRa communication, originally asynchronous Tock functions are made synchronous by adding yield_for() functions to pause execution and wait for read results, which aligns with the user space's synchronous model. In the other way, for functions that initialize GPIO and timers in the HAL, I integrated Tock's functions from the LoRa physical layer to replace Seeed's implementations. I encapsulated the initialization sequences with interrupt callbacks or upcalls, placed at the end of the processes, to maintain the kernel's asynchronous integrity without disruption.

3.3 IRQ Behavior Difference

In conventional operating systems, interrupt requests (IRQ) can occur at any time to disrupt the current process. However, when interrupts occur in Tock OS, they do not directly trigger a callback to pre-empt the user space. Instead, the kernel queues interrupts and processes them at an appropriate time later, without disrupting user space tasks. To manage these pending callbacks, user applications must call the yield() function. This function pauses the current execution flow and transfers control back to the kernel to handle IRQs [5]. Tock's interrupt handling approach creates a non-blocking user space, enhancing system stability and security.

Different mechanisms in handling IRQs could lead to conflicts in synchronization and timing. For instance, the radio planner structure, implemented using Seeed's synchronous code model, manages crucial initialization processes and signal wait times by disabling non-essential IRQs to prevent unwanted behaviors. In contrast, Tock OS's asynchronous approach does not require directly disabling IRQs from user space code. However, this can lead to unexpected IRQs during critical operations within the radio planner layer, altering the original system status.

A specific example of this issue occurs when implementing the Wi-Fi scanning functionality. The scans are initiated after successfully joining the LoRaWAN through TTN, both utilizing the radio planner code. If a Wi-Fi scan is triggered after joining TTN successfully, the RX timeout window may close unexpectedly during this process, resulting in an RX_TIMEOUT IRQ state and causing errors. To mitigate such issues, I removed all

print statements, which can trigger IRQs, to ensure accurate timing for each important IRQ . Also, strategically integrating 'yield()' functions within the kernel code, particularly around sensitive areas, can significantly reduce the occurrence of undesired IRQ behaviors.

4 Development Process and Methods

4.1 Enable SHT41 Sensor

After familiarizing myself with Tock, Tockloader, and Rust through tutorials in the Tock Book, I learned basic commands to flash and listen to the board. I then chose to port SHT41, a sensor capable of measuring temperature and humidity, as my first attempt at interacting with the board through Tock. By referring to the schema and datasheet of Wio-WM1110, I configured I2C-related pins to enable the power supply, serial data (SDA), and serial clock (SCL). Building on Tock's kernel code for the Wio-WM1110, I registered a capsule for the SHT41 based on its datasheet and defined drivers for the temperature and humidity sensor components [7]. Finally, I wrote a user application in C to read sensor data successfully.

4.2 Establish Communication with LR1110

For building applications involving LoRa communication, I integrated code from Seeed Studio's library and adapted it to Tock. I created a new wm1110 library with all related headers and source files from Seeed and a Makefile to compile them by Tock's rules. This enabled the use of Seeed's example application to facilitate SPI communication between the main nRF52840 and the LR1110 chips. Then I implemented the read and write functions within the LR1110's hardware abstraction layer (HAL), substituting the original nRF library functions with LoRa physical layer functions from Tock. According to the LR1110 user manual, the write operation is straightforward, requiring only one function call after preparing the read and write buffers. Conversely, the read operation starts with sending read command bits and requires checking the device's readiness before a subsequent function call retrieves the data [8]. After configuring the SPI-related pins and consulting the LR1110 datasheet, I created the GPIO and SPI system call capsules in the kernel to handle the necessary SPI settings and built successful communication with LR1110.

4.3 Join LoRaWAN and Send Data to TTN

To join The Things Network (TTN), I created an application on TTN to receive uplinks and notifications in the network server console. After registering my device on TTN, it provided device connection parameters including device EUI, join EUI, and App Key. For North America, TTN utilizes the 920-928 MHz frequency range, specifically sub-band 2 on channels 8-15, so I configured the frequency band to US915 and set the correct channels [9]. Following this, I incorporated Seeed Studio's LoRaWAN app to connect to TTN. The application initiates the Semtech (SMTC) modem and enters a loop awaiting callbacks from

the modem. After the first reset callback configures parameters including EUIs, App Key, and region code, the join process starts. Upon successful connection, the join success callback is triggered and sets a timer to activate an alarm in a few seconds. In the alarm function, the application successfully collects sensor data and transmits the payload to TTN through uplinks.

4.4 Enable Wi-Fi Scan and Indoor Geolocation Functionality

To enable Wi-Fi scanning, I integrated the Wi-Fi geolocation application from Seeed's library, which performs a Wi-Fi scan after joining TTN as outlined in 3.3. The application initiates a Wi-Fi middleware and launches a radio planner task to execute the scan at a scheduled time. Upon receiving an IRQ that indicates the completion of the scan, the middleware collects the results and sends them as a payload to TTN. The software stack details for the Wi-Fi scan are illustrated in Figure 4.1.

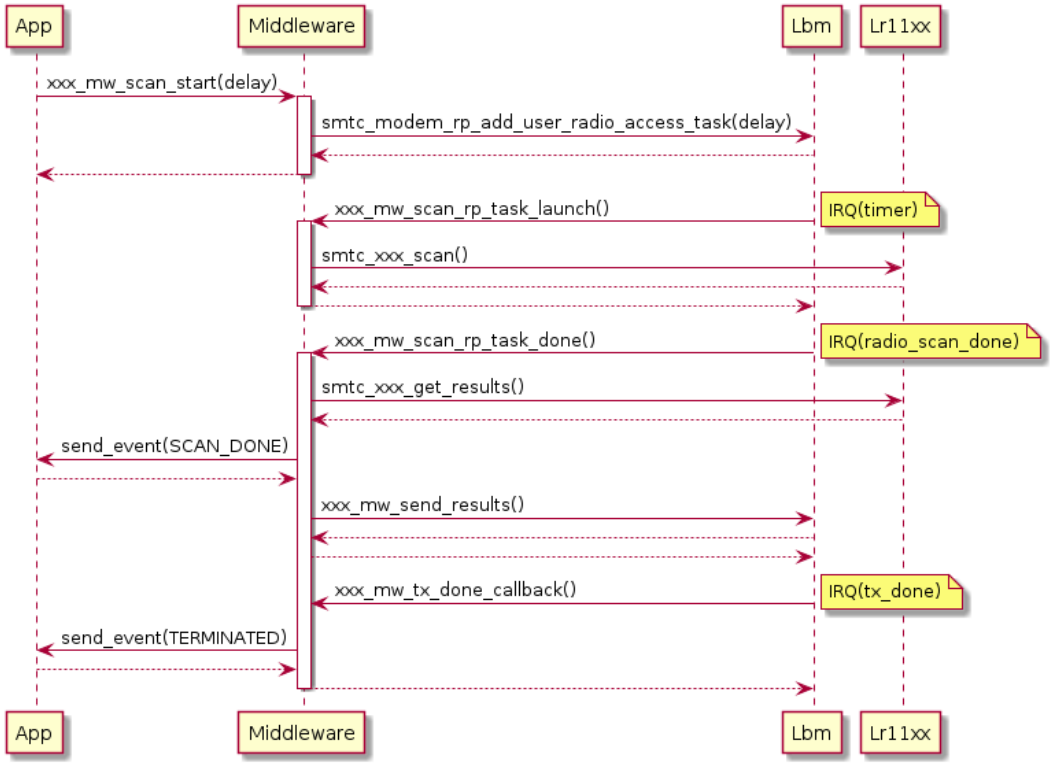

Figure 4.1: Software Stack of Wi-Fi Geolocation Application [10]

Scan results capture the Media Access Control (MAC) address, type (B/G/N), channel, and Received Signal Strength Indicator (RSSI) of up to five visible Wi-Fi access points nearby. If fewer than five Wi-Fi access points are detected, the remaining results are filled with zeros. Then I transmitted these results to TTN in a 36-byte payload via an uplink message. The payload structure comprises an initial byte followed by five entries, each containing seven bytes: one byte for RSSI and six bytes for the MAC address. To capture all possible access points, I conducted multiple scans before performing geolocation calculation. After activating storage integration on TTN, I used the HTTPS GET API to download payloads from the three most recent received LoRaWAN packets containing Wi-Fi scan data

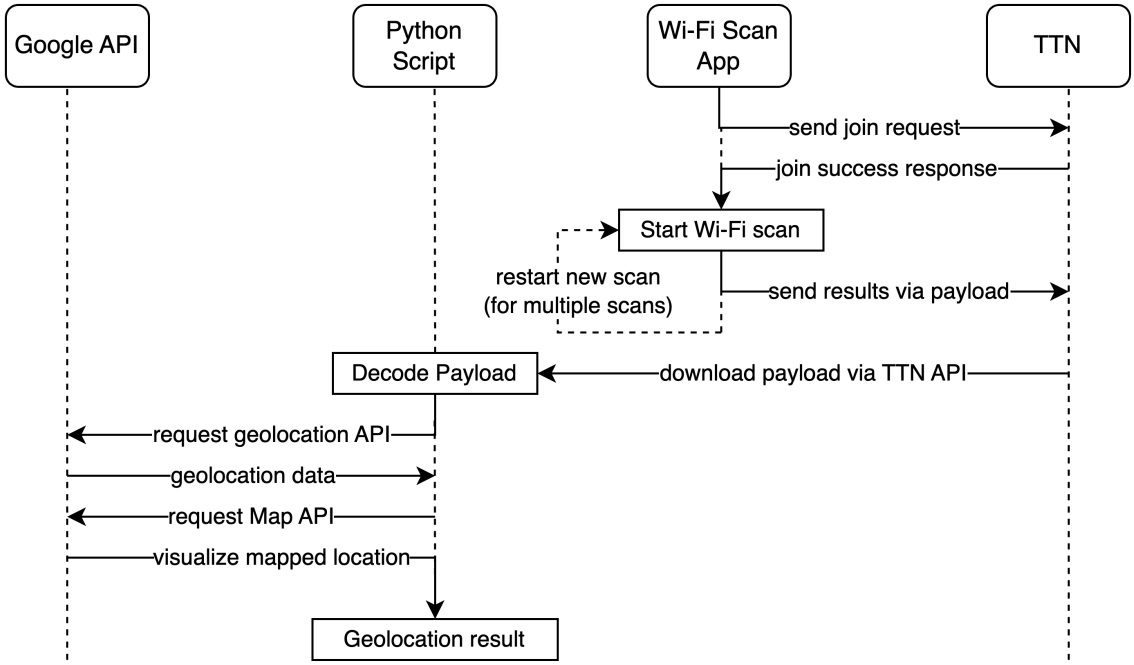

Figure 4.2: Flowchart of Wi-Fi Scan Geolocation Process

at a specific location and processed them with a Python script, which decoded the MAC addresses and RSSIs from the payloads. Subsequently, by integrating the Google Geolocation API, I input the downloaded Wi-Fi access points to obtain actual geolocation data, which includes latitude, longitude, and accuracy range in meters. Finally, I employed the Google Maps JavaScript API [11] to visualize the location and accuracy range on a map to complete the indoor geolocation functionality. Details are illustrated in Figure 4.2.

5 Results

5.1 Assessment of Geolocation Results Across Three UVA Campus Locations

After obtaining data from Google Geolocation API, I used Google Map JavaScript API to visualize the geolocation results. I conducted Wi-Fi scans at three locations within UVA campus covered by LoRaWAN: Olsson Hall in the School of Engineering, New Cabell Hall in the School of Arts and Sciences, and JPA 1819, an off-campus location. At each site, I performed three successful scans to ensure comprehensive coverage of nearby Wi-Fi access points, as the accuracy of the geolocation API improves with the number of different access points included. The Figure 5.1, 5.2, and 5.3 illustrate the geolocation results, which demonstrate high accuracy compared to my physical location during the Wi-Fi scanning

process.

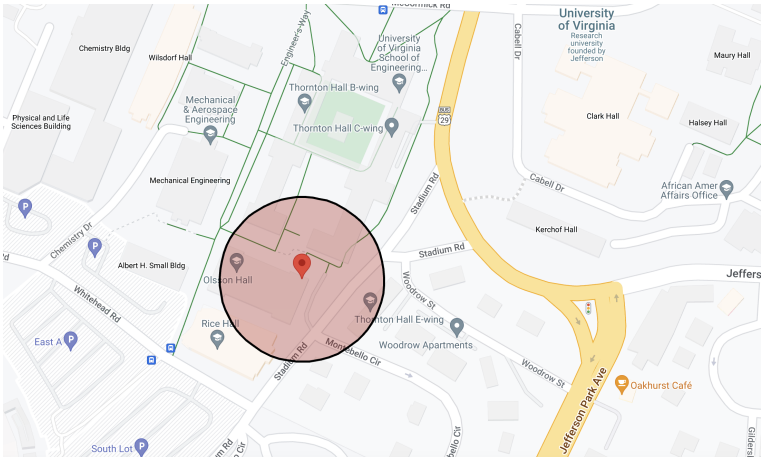

Figure 5.1: Geolocation result from my desk in LINK Lab, Olsson Hall, accurate to within 56.1 meters.

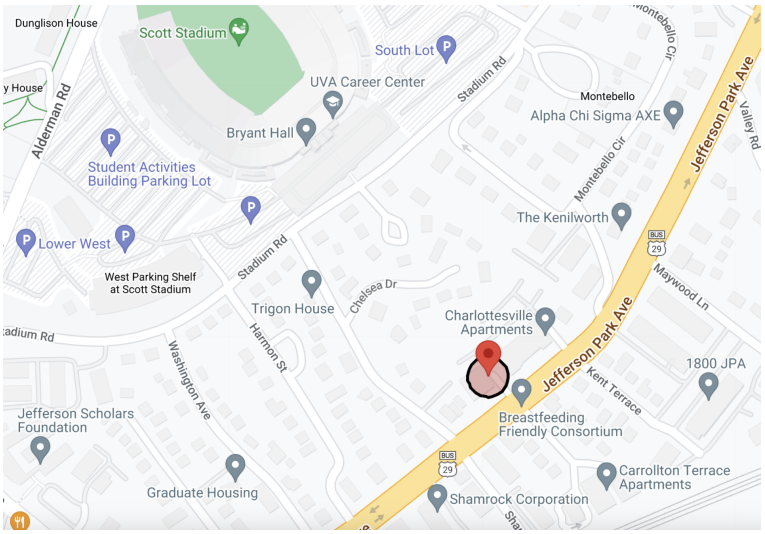

Figure 5.2: Geolocation result from JPA 1819, accurate to within 19.365 meters.

5.2 Effect of Number of WiFi Access Points on Geolocation Accuracy

To ensure geolocation accuracy while minimizing energy consumption from Wi-Fi scans and payload transmission, I analyzed how the geolocation accuracy varies with the number of Wi-Fi access point inputs to the Google Geolocation API. I assessed the accuracy by measuring the accuracy range and coordinate distance error relative to the coordinates from the most accurate result, which was obtained when all Wi-Fi access points were included.

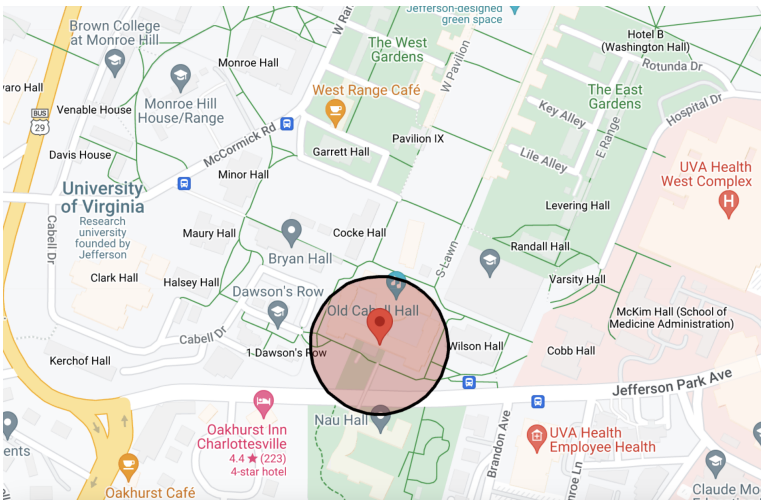

Figure 5.3: Geolocation result from New Cabell Hall, accurate to within 64.1 meters.

The Wi-Fi scan data from JPA 1819 contained six unique Wi-Fi access points, which is the highest number among the three locations. From them, I collected lists of geolocation data by sending the API all possible combinations of Wi-Fi access point inputs, with the number of points used fixed from one to six for each batch. For each number, I retrieved the combinations with maximum and minimum accuracies. According to Figures 5.4 and 5.5, both the maximum coordinate distance error and maximum accuracy range decrease significantly when the number of Wi-Fi access points is increased to four, with further increases yielding diminishing returns. Consequently, utilizing four Wi-Fi access points achieved an optimal balance between energy efficiency and geolocation accuracy.

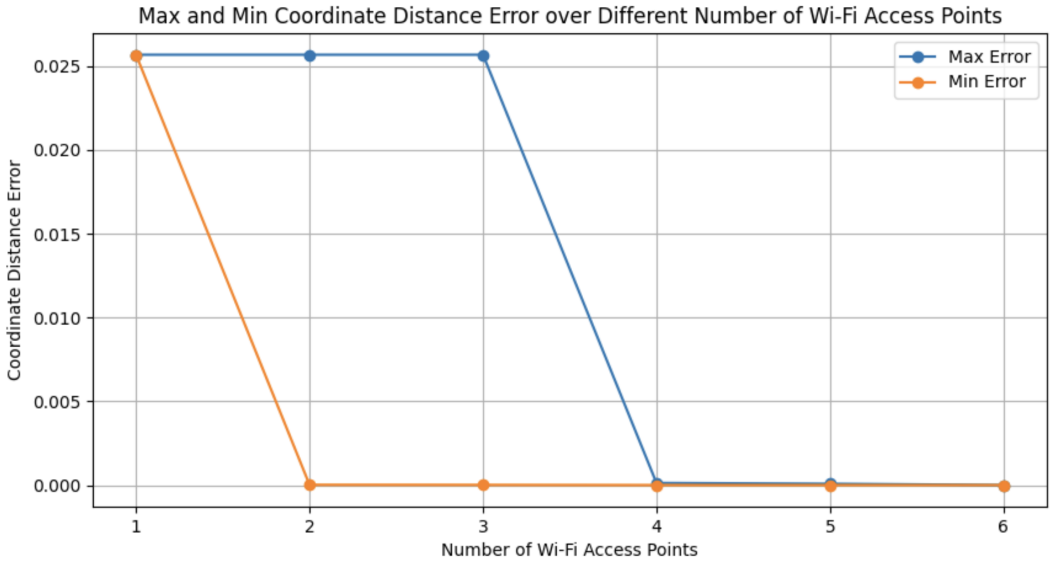

Figure 5.4: Relationship between Coordinate Distance Error and Number of Wi-Fi inputs in JPA 1819

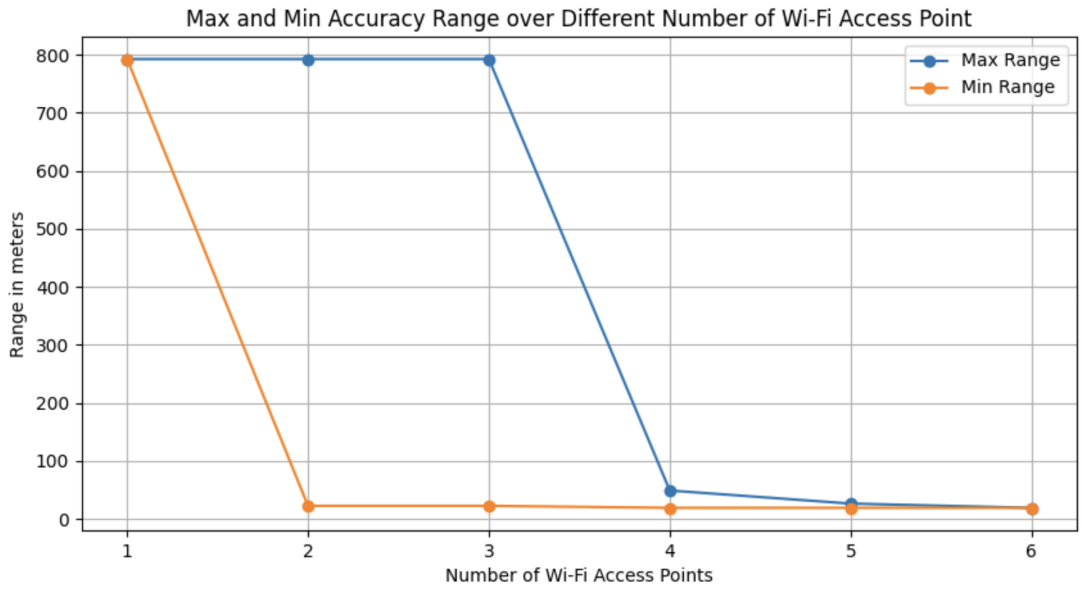

Figure 5.5: Relationship between Geolocation Accuracy Range and Number of Wi-Fi inputs in JPA 1819

6 Future Work and Applications

6.1 Improve Geolocation Methods and Stability

Relying on Wi-Fi scan results processed by the Google Geolocation API, the geolocation methods could be further improved. In experiments in section 5.2, I found that Wi-Fi access point combinations with stronger signal strength and those are distinct from each other result in more accurate geolocation. According to this criteria, we could design an algorithm to selectively filter Wi-Fi access points when the scan yields more than four access points, since four access points are sufficient to track accurate location as indicated by section 5.2. Transmitting only these filtered access points to The Things Network (TTN) would reduce power consumption by minimizing the data sent and potentially improve the quality of input for the Geolocation API.

Moreover, all current Wi-Fi scans and scripts, including API interactions, are conducted manually, which is inconvenient and not user-friendly, especially considering multiple scans are needed for each geolocation. Looking ahead, automating this process through a single program that conducts multiple Wi-Fi scans and executes scripts once data is ready on TTN would significantly simplify operations. The application could set up periodic scans and automatically update the geolocation on a server each time new scan results are received. This would ensure users always have access to the latest location of their tracked item.

6.2 Future Application Scenarios

With geolocation functionality from Wi-Fi scanning, the application could track the location of important assets as long as it's within the LoRaWAN coverage. Even when

the item is lost or moved to another location, we could use the application to perform new Wi-Fi scans and update the location to help track it. Tracking use cases include boxes in a warehouse, expensive power tools in a construction zone, and asset recovery for underutilized medical equipment in hospitals [12]. Compared to common tracking solutions, the device has the potential to evolve into a multi-functional geolocation system. Since Tock supports executing multiple concurrent applications, we could create multiple applications on one device utilizing different functionality of the Wio-WM1110 Module. By analyzing the interaction between data output from sensors and GNSS and Wi-Fi scan results, the system could solve complicated tasks and be applied to diversified environments in the future.

6.3 Explore other Functionality of the Board

By integrating Semtech and Tock's library, the application can expand its functionality from enabled apps on both platforms, utilizing the combined software stack for more potential applications. For example, if we integrate a CO2 sensor into the Wio-WM1110 Module, we could directly install the corresponding application from Tock library and run it concurrently with existing applications.

Currently, the board employs only Wi-Fi scanning for positioning, which is effective in most indoor scenarios due to widespread Wi-Fi signal coverage. However, incorporating outdoor positioning capabilities through GNSS, along with an automatic switch between indoor and outdoor contexts, could fully leverage the board's potential and expand its use cases for asset tracking across diverse environments.

Regarding data transmission, the maximum payload size for LoRa is 256 bytes, yet currently only 36 bytes are utilized for transmitting Wi-Fi scan results to TTN via uplinks. This leaves substantial space available for additional functionalities. For instance, if temperature and humidity sensors are enabled, the board could also serve to remotely monitor environmental conditions and track of temperature and humidity data synchronized with the location of an asset. If the asset is mobile, it could provide insights into how environmental conditions vary across different locations.

Wi-Fi scan results also hold potential for uses beyond geolocation. They play a crucial role in network optimization and planning by analyzing signal strength and coverage in various areas. This data can help optimize Wi-Fi deployment strategies in large facilities, enhancing overall network performance and user experience.

7 Acknowledgement

I am deeply thankful to my research advisor, Professor Brad Campbell, Professor Kun Qian, and members of Professor Campbell's team: Viswajith Govinda Rajan, Victor Sobral, Nabeel Nasir, and the entire group for their invaluable support. Your assistance was necessary as I navigated through the research project and achieved academic growth. As a novice in embedded software development, the comprehensive mentorship and knowledge you all shared made my deep dive into the IoT field not only possible but truly enlightening.

References

[1] "Geolocation API overview | Google for Developers — developers.google.com." https://developers.google.com/maps/documentation/geolocation/overview. [Accessed 19-04-2024].

[2] "LoRa and LoRaWAN: Technical overview | DEVELOPER PORTAL — lora-developers.semtech.com." https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/. [Accessed 18-04-2024].

[3] "Wio-WM1110 Dev Kit, built-in Semtech LR1110 and Nordic nRF52840 — seeedstudio.com." https://www.seeedstudio.com/Wio-WM1110-Dev-Kit-p-5677.html. [Accessed 18-04-2024].

[4] "Semtech LoRa Cloud — loracloud.com." https://www.loracloud.com/documentation/geolocation. [Accessed 18-04-2024].

[5] "Tock Embedded Operating System — tockos.org." https://tockos.org/. [Accessed 18-04-2024].

[6] T. T. Network, "The Things Network — thethingsnetwork.org." https://www.thethingsnetwork.org/. [Accessed 18-04-2024].

[7] S. AG, "SHT41 RH Digital humidity and temperature sensor." https://sensirion.com/products/catalog/SHT41/. [Accessed 18-04-2024].

[8] "LR1110 — semtech.com." https://www.semtech.com/products/wireless-rf/lora-edge/lr1110. [Accessed 18-04-2024].

[9] J. Barani, "LoRaWAN USA frequencies, channels and sub-bands for IoT devices — BARANI — baranidesign.com." https://www.baranidesign.com/faq-articles/2019/4/23/lorawan-usa-frequencies-channels-and-sub-bands-for-iot-devices. [Accessed 18-04-2024].

[10] "Seeed Studio — github.com." https://github.com/Seeed-Studio. [Accessed 18-04-2024].

[11] "Google Maps Platform Documentation | Maps JavaScript API | Google for Developers — developers.google.com." https://developers.google.com/maps/documentation/javascript. [Accessed 19-04-2024].

[12] P. Pachuca, "Indoor Wi-Fi Geolocation with LoRa Edge — blog.semtech.com." https://blog.semtech.com/indoor-wi-fi-geolocation-with-lora-edge. [Accessed 18-04-2024].