# Unmanned Wireless Penetration Testing Device

Ayaan Qayyum*, Omar Hamoudeh*, Gaetano Smith*,
Harris Ransom*, Morriel Kasher*, and Predrag Spasojevic*
*Department of Electrical & Computer Engineering,
Rutgers University, New Brunswick, NJ, USA
Emails: {aaq18, har118, onh5, gls90, mbk94, spasojev}@rutgers.edu

***Abstract*—This paper presents the design and implementation of a modular, remotely operated wireless penetration testing system built from off-the-shelf components. Centered on a Raspberry Pi Zero 2 W and LoRa transceivers, the system enables long-range cybersecurity assessments without requiring proximity to the target network. For proof of concept, the system is deployed on a mobile rover, capable of executing Wi-Fi and Bluetooth reconnaissance and attack commands via LoRa over significant distances. The platform supports real-time command execution and data transmission, including image capture and file transfer, using a custom communication protocol optimized for low-bandwidth environments. The system's modular design allows deployment on aerial platforms like drones and use in remote or infrastructure-limited environments. This work introduces a scalable, open-source framework for secure, real-time red-team operations in otherwise inaccessible environments.**

***Index Terms*—Autonomous Cybersecurity Systems, Portable Offensive Security Tools, LoRa Communication, Penetration Testing, Autonomous Systems**

## I. Introduction

Penetration testing, or "pentesting," is an essential component of cybersecurity that ensures the resilience of wireless networks against external threats. As more organizations adopt wireless technologies for convenience and flexibility, they also become vulnerable to attacks targeting these systems. Traditional pentesting tools often require physical proximity to the target network, which can be a limiting factor when assessing systems in restricted, elevated, or dangerous environments. This presents a critical challenge for security professionals: how can wireless security be tested safely, efficiently, and remotely?

To address this gap, we propose a modular, unmanned penetration testing system that enables long-range wireless assessments by integrating off-the-shelf components, embedded computing, and robust long-range communication protocols. Our system is designed to be lightweight, adaptable, and cost-effective, leveraging a Raspberry Pi Zero 2 W running Kali Linux, LoRa transceivers for control and communication, and a range of penetration testing tools. Mounted on a mobile rover platform, the device can be deployed in environments previously inaccessible to researchers, providing a new layer of flexibility in wireless network assessments.

## II. Background

This project lies at the convergence of embedded systems, wireless communication, and cybersecurity. It aims to provide a viable alternative to traditional, proximity-dependent penetration testing by enabling remote and modular assessments of wireless networks.

### A. Wireless Network Penetration Testing

Wireless pentesting typically involves scanning, analyzing, and attempting to exploit networks through techniques like packet sniffing, de-authentication attacks, and brute-force key cracking [1]. Tools such as aircrack-ng, airodump-ng, and airmon-ng, commonly used on platforms like Kali Linux, allow for these activities to be performed efficiently. However, the requirement of physical proximity for effective testing restricts the ability to audit networks in sensitive or inaccessible areas [1] [2].

### B. LoRa Communication

The LoRa communication protocol is a long-distance protocol built on ALOHA that can send a certain number of bits per second, such as 128 bytes [3]. With back-and-forth communication, there is a significant amount of latency, so communication has to be limited to simple commands and highly optimized codecs for more advanced data, such as images. Our designs rely on the modules implemented by Adafruit for the LoRa communication protocols through the CircuitPython and Python programming languages [4]. Choosing to rely on the pre-implemented modules for the protocol itself led to higher compatibility and rapid iterative development. However, the purpose of the LoRa communication protocol is to provide reliable, long-distance communication that makes it possible for the rover to receive commands [3]. Other options, such as sending files through a custom codec, have been helpful in the delivery and development of this project [5].

### C. Acknowledgements and Packet Loss Mitigation

As the LoRa communication protocol is a bare-bones system, implementations are needed to ensure messages and packets are received and processed. Acknowledgements are a system where a receiver sends back a message indicating it has successfully received the sent message. Our project uses acknowledgements to reduce packet loss. For example, when sending 30 packets that encode an image, once the rover has sent one packet, it waits for the basestation to return an acknowledgement that the packet was received before continuing. This leads to overhead where handshaking must occur every single time a group of packets is to be transmitted. Such a design prevents viable streaming, burst, or one-way communication. Instead of using acknowledgements, a system can use Reed-Solomon codes or other error-correcting codes to reconstruct dropped packets [6]. A typical example is that if an image consists of 30 packets and packets are dropped approximately 15% of the time, five redundant packets are needed to reconstruct the entire picture. Approximately 35 packets must be sent to compensate for the five dropped in this situation. The benefit of Reed-Solomon error correction coding is that handshaking and acknowledgements are no longer necessary as accommodations have been made for dropped packets [4]. This can lead to huge performance and throughput gains when one side must send many packets sequentially without interruption.

However, with Reed-Solomon encodings, each packet is encrypted with the proper encoding scheme necessary to reconstruct the rest of the packets [6]. This leads to higher redundancy and decryption issues. For instance, if only two packets need to be sent but accommodations are made for if one packet is dropped, three packets are sent in total, leading to 33% of the packets sent used for redundancy. Suppose $k$ is the number of redundant packets for error encoding, and $N$ is the total number of packets sent. In that case, the entire message cannot be decrypted until at least $N - k$ packets are successfully received.

One compromise experimented with is to only use Reed-Solomon encoding for data being sent that is already encrypted in some way and requires largely one-way communication for an uninterrupted transmission, such as with image data.

### *D. Modular Architecture for Remote Deployment*

Our system is built around a three-part architecture: the basestation (command interface), the rover (mobile platform), and the target (a simulated Wi-Fi network). The rover, equipped with a Raspberry Pi Zero 2 W, LoRa transceiver, camera, and Wi-Fi adapter, receives commands and performs penetration tests while transmitting data back to the basestation. The architecture is intentionally modular; each component can be swapped or reconfigured for different platforms. The rover platform used in this implementation serves as a scalable base that can easily transition to an aerial drone, enabling testing in high or otherwise unreachable locations.

This flexibility allows the system to be adapted for various operational environments and testing scenarios, establishing a foundation for future research and development in unmanned wireless penetration testing technologies.

### *E. USB Serial Communication*

The Raspberry Pi Zero 2W was chosen over other single-board computers (SBCs) because the device can support USB On-The-Go (OTG) mode [7]. The Zero series can switch to device mode and be controlled by other devices when connected to another device. The micro-USB data port on the Pi Zero 2W is connected to the SoC's USB controller, which makes USB OTG mode possible. Most SBCs wire the USB ports directly in a way that prohibits USB OTG mode [7].

With USB OTG mode, the Pi Zero 2W can emulate a serial port over USB. This makes it possible to interface directly with the Pi Zero 2W via the command line interface just as one would do so over SSH [7]. However, with USB OTG mode, one does not need to set up Wi-Fi on the Pi Zero 2W first before interfacing directly. Waiting for the Pi Zero 2W to connect to the network on boot would typically take a minute or more and be the main bottleneck in the startup sequence. Some Wi-Fi penetration testing modes such as monitor mode require the Wi-Fi chip to be occupied and cannot permit dual SSH and scanning. As we intend to have the system communicate with a basestation entirely using LoRa without first connecting to a Wi-Fi network for interfacing and control, directly accessing and modifying the entire system accelerated our development lifecycle.

## III. Equipment

To build our modular penetration testing platform, we selected various hardware and software components based on their performance, compatibility, and ease of integration. This section outlines the major components of the system, categorized into communication and sensor elements, power and mechanical infrastructure, and the software tools that support operation and data processing.

| **Name** | **Use In Project** |
|---|---|
| Raspberry Pi Zero 2 WH | Main computing unit for processing and control. |
| LoRa Transceiver | Provides wireless communication through SPI on the Raspberry Pi. |
| LoRa Microcontroller | Interfaces with the transceiver for signal handling. |
| SMA Edge Connector | Links the LoRa antenna to the transceiver module. |
| LoRa Antenna (2) | Enables long-distance wireless communication. |
| uFL-SMA Cable Adapter | Connects the antenna to the microcontroller. |
| Camera | Captures images for navigation or monitoring. |
| Netis WF2123 WiFi Adapter | Enables monitor mode for wireless packet capture. |

TABLE I
COMMUNICATION AND SENSOR COMPONENTS

| **Name** | **Use In Project** |
|---|---|
| Rover Kit | Provides the chassis and mechanical components for the rover. |
| Motor HAT | Drives and controls the motors. |
| 4x AA Battery Holder | Supplies power for motors. |
| 2500mAH Lipo | Main power source for the Raspberry Pi and additional components. |
| Boost Converter | Boosts battery voltage to required levels. |
| Powerboost 1000c | Manages efficient power delivery and supports battery charging. |

TABLE II
POWER AND MECHANICAL COMPONENTS

| Software | Use In Project |
|---|---|
| `GitHub` | Store our source code. |
| `Kali Linux` | OS with built-in wireless penetration testing tools used on the rover. |
| `adafruit_rfm9x` | Python module for interfacing with the LoRa RFM95 radio via CircuitPython. |
| `subprocess` | Built-in Python library to run system-level commands and interface with tools. |
| `pyserial` | Serial communication with hardware. |
| `aircrack-ng` | Cracks WPA/WPA2-PSK keys using captured handshake packets. |
| `airodump-ng` | Captures raw 802.11 frames to analyze nearby Wi-Fi networks. |
| `airmon-ng` | Enables monitor mode on wireless interfaces for packet capture. |
| `zlib` | Compression and decompression of data. |
| `png` | PNG image read/write support. |

TABLE III
SOFTWARE TOOLS AND MODULES

## IV. METHODOLOGY

cWe designed our system to be highly modular to scale to suit other applications. Our goals were as follows:

- Design a system that can test wireless networks remotely.
- Use a radio protocol for long-distance reliable communication.
- Minimize size, weight, and power via SWaP-C constraints.
- Implement a modular framework for future expansion.

An overview of the complete system design is shown in Figure 1, illustrating the hardware components across the basestation, rover, and penetration targets, along with their communication flow.

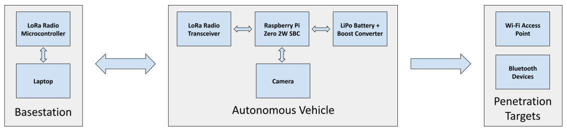

Fig. 1. System architecture showing the relationship between the basestation, autonomous vehicle, and wireless penetration targets.

### A. Communication Architecture

In this project, there are three main aspects of our design: the rover, the basestation, and the target. In this project, we designed the target to be an ESP-32 module with a wireless network that can serve as a network to perform penetration testing regarding security. The basestation consists of a computer with USB serial support connected to an Adafruit Feather LoRa radio. The rover includes a Raspberry Pi Zero 2W with Kali Linux installed. We chose Kali Linux as our operating system due to its extensive community support for wireless penetration testing modules and packages.

Any MacOS, Linux, or Windows computer can connect to the LoRa radio module via USB Serial communication. The Adafruit Feather module runs code that allows it to serve as a passthrough. The local computer running Python has the source code and special function implementations. This design decision was made to minimize compute on the Adafruit Feather module to allow it to just focus on receiving and transmitting LoRa radio packets. Having the Feather reconstruct images for example is too computationally intensive, so the Feather's main goal is to provide the right data to the basestation interface.

### B. Rover

The rover consists of a Raspberry Pi Zero 2W with a LoRa transceiver and Motorkit module. The LoRa transceiver is set up to operate at 915 MHz at max decibel power. The Motorkit module allows the rover to move in its environment. A standard interface lets the basestation send commands that command the rover to move. The only way to let the rover move is through the function calls.

Nearly the entire CLI is implemented on the rover's end. The basestation only sends commands that call the rover's programmed calls for specific uses. For example, one such command is "MOVE", which calls the rover to move. Arguments are needed for a complete function call, so a full command that would instruct the rover to move for 0.1 seconds at max motor power is:

MOVE FORWARD 0.1 10

The specific values of 1 and 10 for the motor movement strength were calibrated to not exceed the limitations of the hardware, as under a certain voltage, the motors do not spin.

### C. Basestation

The basestation connects to any MacOS, Linux or computer that can interface with the radio module through the USB Serial protocol. The AdaFruit Feather LoRa radio module sets up the LoRa communication system with the rover. The AdaFruit Feather is designed to print out what it receives in UTF-8 encoding; corresponding software on the local computer can perform extra processing if necessary. Extra processing includes decrypting and decompressing files and images, and the ability to load command scripts from files. The command scripting language is bare bones with three main features: commands, waiting, and loops. The same commands that are typed into the terminal and sent to the rover can instead be written in the script file. Waiting adds a delay before sending a command to the command queue. The command queue processes commands sequentially in order of reception. Loops add a way to call certain commands repeatedly. The interpreter for the command script is written in Python and is rolled into the logic for the rover, as it is the rover that would catch any errors after receiving whatever is sent in the script.

```
FOR: 3
MOVE FORWARD 0.2 2
WAIT 2
WIFISCAN
WAIT 2
MOVE BACKWARD 0.2 2
```

Listing 1. Example Script of Movement with Wi-Fi Scanning

### *D. Wireless Network Via ESP-32*

An insecure wireless network was created using an ESP-32. An ESP-32 hardware module can act as a Wi-Fi access point and a web server, making it an ideal choice for simulating a vulnerable wireless environment. In this project, the ESP-32 was programmed to broadcast an open Wi-Fi network with no encryption, allowing any nearby device to connect without authentication.

The access point configuration was implemented using the Arduino framework, where the ESP-32 hosted a basic HTTP server that responded to requests on a designated port. This setup mimicked real-world insecure networks often found in public or poorly configured environments, providing a valid test target for penetration tools running on the rover.

This network served as the controlled environment for executing wireless reconnaissance, packet sniffing, and deauthentication attacks. By using the ESP-32 as a customizable and lightweight network emulator, the project could safely and repeatedly test different pentesting strategies without infringing on real-world systems or requiring access to external infrastructure.

### *E. Files and Throughput*

A protocol was developed that was optimized for sending files. Firstly, an internal API was designed for automatically chunking packets to fit within the packet size limitations. This system is used for all communications between the basestation and the rover, allowing the developer to send as many or as few packets as they would like.

### *F. Sending Images*

The image communication protocol aims to transmit and communicate what the rover sees over the LoRa protocol. Firstly, the Raspberry Pi camera takes and saves an image. As the original photograph taken by the Raspberry Pi camera has a resolution of 1080 by 1080 with a bit-depth of 8 and in color, transmitting one compressed image 128 bytes at a time will take over 4,000 packets to be transmitted. Our design and implementation reduced the number of packets sent for this image to just 20. However, a significant reduction in quality was needed for such a result.

Our final transmitted resolution is a square image of 256 by 256 pixels, taken in black and white, with four bits of color depth per pixel. This combination gave the best balance of resolution and color depth. Dithering was used to simulate a higher color depth. The uncompressed image of such a resolution is 64 kilobytes, so transmitting the raw packets will still take around 70 packets. Zlib compression and PNG compression was used to reduce the image size to as low as a single packet for less complex images.

ZLib compression minimizes the file size as it is sent over the communication protocol. With our results, we expect to send such an image, which normally constitutes around 64 kB of data, in an average of 15 packets of information. Although a vast amount of visual fidelity is sacrificed, the trade-off is well worth it for the rapid speed and development of sending images and videos over the communication port. The order of the packets sent was by color bit depth. This solves the problem of the image being reconstructed despite dropped packets. Although we have a 98% success in successfully transmitting packets, we only have a 98%, and not a perfect result. Through the use of hashes and dynamic data, the system tells the basestation if a packet was dropped or received. If a packet was dropped, the decision automatically asks for it to be resent. However, when the packet loss is permanent, we have not implemented a contingency where we have some error control coding. With error control coding, we can see what packets were dropped and how to re-implement them.

### *G. Motor Calibration*

Significant work was taken regarding motor calibration to ensure the system can effectively move and transport itself with a consistent power output. Extensive testing was performed to create a statistical model that roughly estimated the general performance of motors and throttle output based on these numbers to ensure a consistent user experience. For example, the Left motor tends to be stronger than the right motor, so significant calibration had to be based on that. Also, although we did use a vulture stabilizer, the motors had a substantial risk of weakening with the battery over time. However, our circuit provided a clean, consistent output, so there was minimal risk, except when the system ran out of power. Extending the system to work with drones can easily control the drone with computer commands. Although the movement will be relatively jerky and not as seamless as with a regular controller, more can be done to increase and improve the latency aspects of the system. The system is mainly concerned with sending command packets, which tend to be no less or no greater than 50 bytes through, so it is not too much of a concern. All the designs and systems we would want the rover to do are already baked into the Raspberry Pi's system. However, minimizing latency is the most important aspect of the constant feedback loop that comes from wanting to control and see what is going on from the rover in a near real-time situation. This led to some modules being designed in parallel. For example, the code that sends the receiver always runs as early as possible to confirm that the code is operating as expected. Another example is when the motor is commanded to move forward, it will always send a packet and move simultaneously or simultaneously. Significant care was taken to ensure that the threads do not block.

### *H. Commands Implemented*

Many commands were implemented to execute several tasks. Here is the explanation of the significant functions and commands implemented.

### *I. Wi-Fi Penetration Testing*

Wi-Fi penetration testing was a core focus of this project, as it enables the assessment of wireless network vulnerabilities through passive and active techniques. The rover platform, equipped with a Netis WF2123 adapter capable of monitor

| Commands | Purpose |
|---|---|
| MOVE | Send rover in certain directions. |
| HISTORY | Resend the last few commands. |
| ECHO | Return a message a certain number of times. |
| SCREENSHOT | Send a locally stored image. |
| CAMERA | Send a feed from the camera. |

TABLE IV
COMMAND DESCRIPTIONS

mode and packet injection, was able to perform reconnaissance and exploitation on target networks.

Using Kali Linux and tools from the aircrack-ng suite, the system could identify nearby access points, capture 802.11 frames, and perform deauthentication attacks to forcibly disconnect clients. Captured handshake packets were then used in offline dictionary attacks to attempt WPA/WPA2 key recovery.

The modular nature of the rover allowed these attacks to be issued remotely via LoRa communication. For instance, an airodump-ng command could be issued from the basestation to begin network scanning, and results were either displayed locally on the rover or sent back through compressed transmission. This remote testing capability significantly reduces the need for human presence near the network and adds flexibility for assessing networks in restricted or high-risk environments.

### *J. Bluetooth Penetration Testing*

Testing Bluetooth connections allows the rover to interact with nearby Bluetooth-enabled devices and emulate legitimate peripherals. While this aspect of the system is still under development, initial implementations focused on device discovery, spoofing, and scanning using tools such as `bluetoothctl` and `hcitool`.

The rover can identify visible Bluetooth devices with a compatible USB Bluetooth adapter integration, log their MAC addresses, and query device metadata. In future implementations, the system could use spoofing techniques to emulate devices like an Apple TV or a wireless headset, potentially exploiting trust-based pairings or misconfigured devices.

This testing is especially valuable in IoT-heavy environments where Bluetooth Low Energy (BLE) is used for control and communication. As the platform matures, we aim to expand Bluetooth attack capabilities to include man-in-the-middle (MITM) attacks, pairing interception, and fuzzing, further broadening the scope of remote wireless security assessments.

### *K. Tests*

To evaluate and optimize the performance of our unmanned penetration testing system, we conducted a series of experiments focusing on communication reliability, efficiency, and image transfer capabilities. These tests were designed to understand the limitations of the LoRa communication link under different operating conditions and to refine system parameters such as packet size, compression strategy, and operating range.

*1) Range:* The range test was conducted to determine the effective communication distance between the LoRa transceivers on the basestation and the rover. This test involved placing the rover at increasing distances from the basestation in an open outdoor environment, measuring two key metrics: latency and packet loss.

At each distance increment (250, 950, 1750, and 3150 ft), a series of echo commands were sent and round-trip times (RTT) were measured. Packet delivery rates were calculated by sending a fixed number of packets and counting successful acknowledgments. Finding the limit where packets with acknowledgements are dropped was beyond 128 bytes at the farthest distance.

*2) Throughput:* This test aimed to find the optimal balance between packet size and data reliability for transmitting information such as images or command responses over LoRa. Because LoRa has limited bandwidth, choosing the right packet size is crucial to maintaining speed and reliability.

We conducted trials with varying packet sizes (32, 64, 96, 128, 160 bytes), sending a fixed number of packets for each configuration. Throughput was calculated by measuring the total number of successfully received bytes over time. At the same time, we monitored the packet drop rate and latency for each size.

## V. RESULTS

Extensive tests revealed our long-range radio system's performance, throughput, and efficiency. We separated the basestation and the rover at many large distances and recorded data regarding the max achievable payload size at a particular location. At each location, described in Figure 5, we performed three trials of sending a variety of payload sizes, including 4, 16, 32, 48, 64, 80, 96, 112, and 128 bytes. Each payload was sent five times and timed to calculate the throughput. As the calculations for throughput and latency are only based on the payload sizes and not any extra information sent, including handshakes, our throughput measurement also measures goodput. Goodput refers to throughput without redundant details, describing the helpful information sent. It is important to note that although packet sizes larger than 128 bytes can be sent, as our LoRa radio module supports 224 bytes natively, achieving such high payload sizes in practice is tremendously difficult. Packet sizes larger than 128 bytes, especially over long distances, led to significant packet loss and drops, even with acknowledgement protocols in place. The 128-byte limit in our empirical data results from the farthest distance that could send the largest packet size. For example, when data is being sent only 500 feet away, larger payload sizes than 128 bytes can be sent, but to keep all data and arrangements clean, only the maximum packet size achieved from the highest distance was maintained in our figures.

Factors influencing transmission efficiency include redundancy, payload size, and distance. The redundancy factor includes whether acknowledgements are enabled or error correction encodings are. The highest achieved throughput in our testing was 236 bytes per second, approximately 1.88 kbps, completed at a distance of roughly 1,700 feet from the source with a payload size of 128 bytes and a latency of 0.543 seconds per packet.

Figurec2 shows our empirical results of the throughput and latency of the payloads. The results are averaged across all locations.

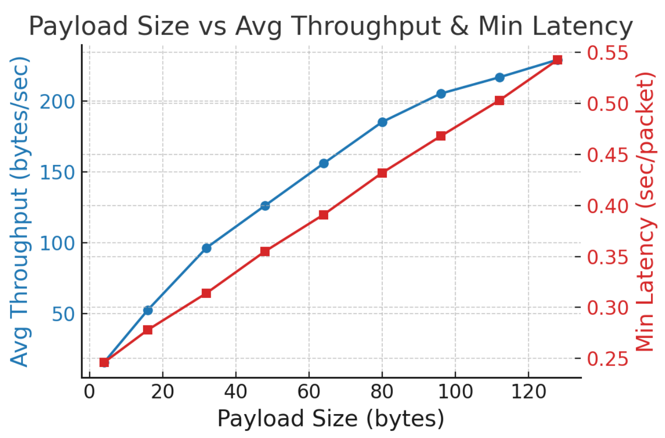

Fig. 2. Payload versus throughput.

Figure 3 describes the latency versus the payload size, fit to a quadratic regression equation to describe how latency scales as a function of payload size. This is averaged across all locations. The best-fit equation under quadratic regression for latency versus payload size is $-2.041 \cdot 10^{-6}x^2 + 0.002475x + 0.2695$. As the quadratic term is extremely close to 0, there is an implication that latency grows steadily with packet size.

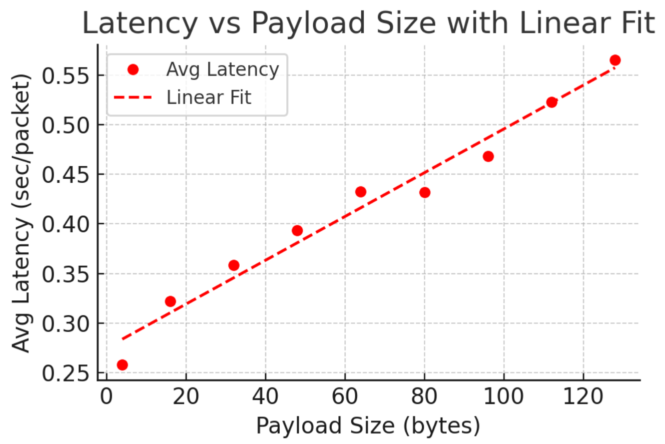

Fig. 3. Latency versus payload size.

Figure 4 describes the best-fit curve for the throughput versus payload size data. The equation that best represents the data is $0.002208x + 0.2748$. As the payload size increases, there are diminishing returns beyond a certain payload size. We had seen this effect in our analysis.

Combining the analysis for the latency and throughput versus packet size directs towards an ideal payload size. In our testing, as calculating goodput as payload size divided by latency, 128 bytes per packet gives the best packet size.

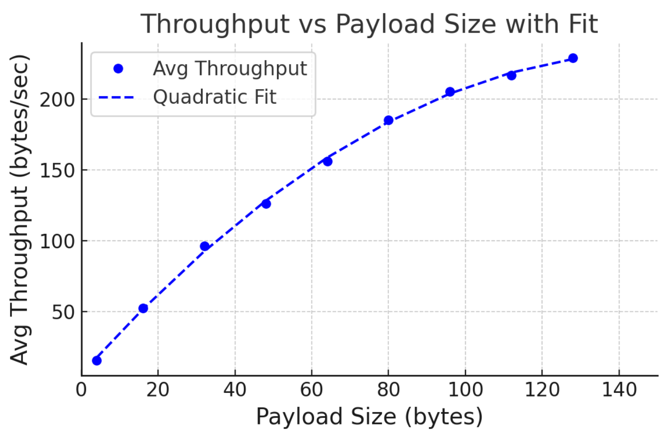

Fig. 4. Throughput versus payload fit to a quadratic curve.

Four distinct locations were tested. First was the ECE parking lot, approximately 250 feet from the source. Next was the Werblin Bus Stop, approximately 950 feet from the source. Then there was the Athletic Parking Lot, approximately 1,750 feet from the source. Then there was the SHI Stadium Parking Lot, approximately 3,150 feet away. Figure 5 describes the physical distances. Distances are approximate due to variability in exact precise GPS coordinates and were calculated using Google Maps.

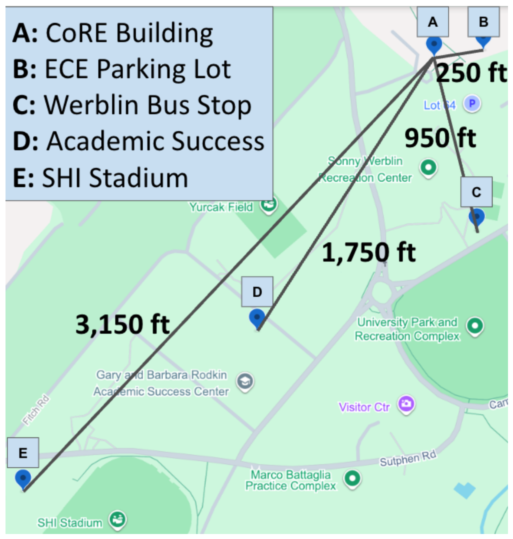

Fig. 5. Map showing distances tested.

Figure 6 describes how efficiency, which is calculated as throughput divided by payload size, decreases roughly exponentially as payload size increases.

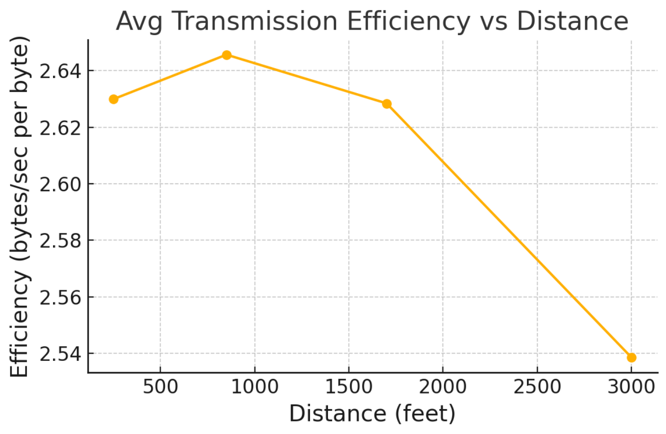

Fig. 6. Average transmission efficiency versus distance. Efficiency is calculated as throughput divided by payload size.

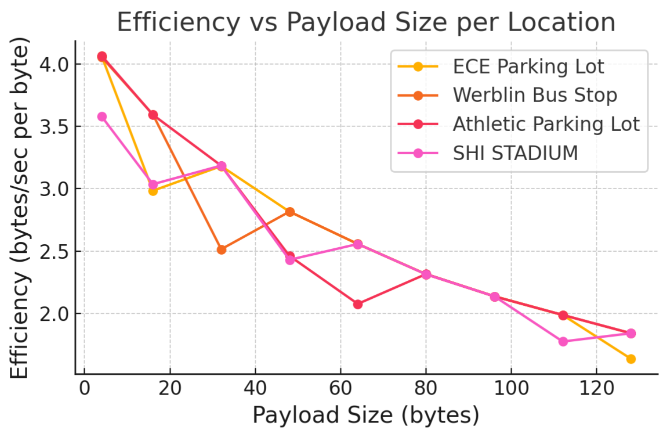

Fig. 7. Efficiency versus payload size per location. Efficiency is calculated as throughput divided by payload size.

## VI. Further Work

As we designed a custom modular scripting language with a robust communication interface, it is straightforward to adapt our technology to control a drone to perform penetration testing or machine learning tasks. Interfacing with the rover's SBC through LoRa commands could run programs on the rover (such as Aircrack in our testing), so any program that can run on the rover can be run remotely if programmed correctly. The radio thread intermediates between the basestation's commands and the rover's internal functions. With this flexibility, it is straightforward to add machine learning capabilities to perform inference on the device itself, then only send the results over the protocol to the transceiver basestation. As our system is set up to automatically chunk and process command-line outputs with flexibility for handling internal functions on the system, such as calling a USB camera or microphone, extending this platform in these manners is straightforward.

One further example of such implementations can help in search and rescue operations. In emergencies, traditional networks such as cellular or Wi-Fi become unavailable due to a lack of power. As time is of the essence, a drone remotely controlled with vision or audio capabilities can scan or listen for those needing help. The rover or drone with the SBC can recognize a survivor and send back the coordinates for further investigation by a person, or send a rudimentary image or short audio sample.

Aside from search and rescue operations, our technology can help penetrate test cellular networks. As the LoRa communication radio interface is abstracted from the penetration testing systems through internal API calls, extending our systems to work with cellular networks is straightforward. This project did not focus on cellular networks due to time and cost constraints. One can imagine that remotely accessing cellular networks for penetration, testing, or compliance safety can lead to benefits when it is impractical to send one person up to solve a particular problem or for maintenance, which may be too expensive by traditional means.

## VII. Conclusion

We have presented a proof-of-concept unmanned wireless penetration testing system for remote red-team cybersecurity audits. Our compact and modular embedded system can perform Wi-Fi and Bluetooth penetration tests while receiving commands from over a mile away. Our system demonstrated real-time remote control and penetration testing. Some features demonstrated include cracking an insecure network's Wi-Fi password or hijacking available devices' Bluetooth connections. With the design of a custom Unix-like scripting language, it is fully operable over the secure and long-range LoRa radio protocol. This prototype aims to lay the foundation for future drone-based open-source wireless auditing tools that can be deployed discreetly and efficiently.

## References

[1] N. V. Barker, "Development of a drone-mounted wireless attack platform," Thesis, Eastern Kentucky University, 2020, theses and Dissertations. 3224. [Online]. Available: https://encompass.eku.edu/etd/3224

[2] P. M. A. Khan, B. R. Babu, V. S, and L. R. K, "LoRa Based Surveillance Rover with Integration of Hybrid Communication Technologies for Versatile Applications," in *2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN)*. Vellore, India: IEEE, 2023.

[3] S. Fahmida, V. P. Modekurthy, D. Ismail, A. Jain, and A. Saifullah, "Real-time communication over lora networks," in *2022 IEEE/ACM Seventh International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2022, pp. 14–27.

[4] Adafruit Industries, *LoRa and LoRaWAN Radio for Raspberry Pi*, January 2025, accessed: 2025-04-20. [Online]. Available: https://learn.adafruit.com/lora-and-lorawan-radio-for-raspberry-pi

[5] H. Gao, Z. Huang, X. Zhang, and L. Huang, "Design of lora communication protocol for image transmission," in *2023 8th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 2023, pp. 2142–2146.

[6] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[7] Raspberry Pi Ltd., *Raspberry Pi Hardware Documentation*, 2025, accessed: 2025-04-20. [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html