BLE Hide & Seek for COMP3210

Dan Roberts, *Student, University of Southampton, dr1n22*

***Abstract*—This project demonstrates a Bluetooth Low Energy (BLE) proximity-based lighting control system framed as a game of 'Hide & Seek'. The system provides feedback to the user by dynamically changing the colour of a smart bulb from red (near) to blue (far) using Matter-over-Thread controlled by a HomeAssistant instance acting as a Thread border router, and receiving proximity updates via a CoAP server from a BLE beacon. The system explores the integration of modern IoT protocols (BLE, CoAP, Matter) in a real-time interactive application, and provides responsive feedback to the user to guide them to the BLE beacon.**

I. Introduction

THIS project implements a 'Hide & Seek' game by changing the colour of a smart LED bulb between 'hot' (red) and 'cold' (blue) to guide the user to a hidden beacon. The system determines proximity using Bluetooth Low Energy (BLE) signal strength between a user and the beacon, and dynamically changes the colour of a smart bulb based on this proximity.

Initially, this project aimed to use the Philips Hue system with Bluetooth for lighting control and feedback. However, this approach proved unfeasible due to proprietary restrictions and a locked-down environment in Hue's BLE stack, which prevents low-level access or control with external systems. As a result, the design of the system switched to a more open-source approach using HomeAssistant (HA) which provides a more flexible environment for smart home control, with support for many IoT protocols, including the primarily-IPv6 protocol Matter [1].

II. System Design

The system uses a variety of different hardware, software, media types and protocols to achieve the overall goal. A full network layout diagram can be seen in Figure 1, and physical devices are shown in Appendix A.

A. Hardware and Software

- BLE Beacon — Raspberry Pi Pico W: A microcontroller uses BLE to determine proximity based on Received Signal Strength Indicator (RSSI) between it and the user's mobile phone. It uses lightweight MicroPython libraries to handle BLE advertising and scanning, and CoAP client requests.
- CoAP Server — Raspberry Pi 4: Running Raspberry Pi OS, a server handles CoAP requests from the beacon, primarily `PUT` requests of the latest RSSI. It calculates a normalised proximity distance and linearly interpolates a colour between blue (far) and red (near) based on this normalised distance. It makes RESTful HTTP requests to a HA server to control the smart bulb.
- HomeAssistant Server — Raspberry Pi 4: Acts as a Thread border router to bridge between IP networks and Thread (IEEE 802.15.4) networks. The Raspberry Pi itself is not capable of communicating over 802.15.4, so requires a Radio Co-Processor (RCP) in the form of an nRF52840 dongle connected by USB. The server runs HomeAssistant OS and hosts a Matter server to interface with Matter devices such as a smart bulb.
- NanoLeaf smart bulb: A colour LED bulb that supports the unified Matter protocol over the Thread layer (rather than Wi-Fi). It is controlled by the HA server via the RCP dongle.

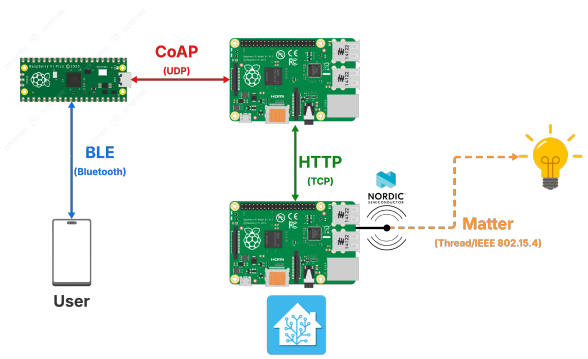


Fig. 1. Layout of final network. CoAP uses UDP over IP. BLE uses the same 2.4GHz frequency as Wi-Fi, but does not use IP. Standard HTTP is used between CoAP and HomeAssistant servers, and Matter-over-Thread is used to control the bulb.

III. Implementation

A. BLE and RSSI

Proximity between the hidden Pico beacon and the user is based on RSSI. Before proximity can be estimated, the beacon must first advertise itself and does so using a custom Generic Attribute Profile (GATT) service/characteristic. During the advertising process, the beacon blinks an onboard LED three times in a cycle to show that it is waiting for a user to connect. The user can use any BLE app (or native support) on their mobile device to connect to the beacon. Once connected, the beacon slowly blinks the LED to indicate that it has paired with the user. Once paired, the user can disconnect from the beacon, after which the LED will remain on, indicating that the beacon is ready to be hidden.

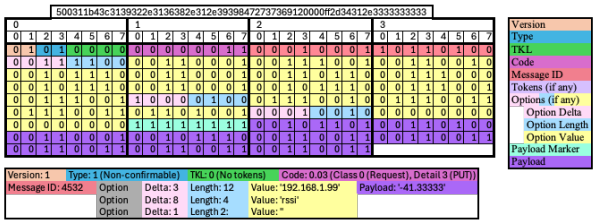


Fig. 2. CoAP Packet received by CoAP server. The CoAP operation (GET/PUT/POST) and the URI are visible in the code and options. The payload corresponds to an RSSI of -41.3dBm, indicating the user was close to the beacon when the packet was sent.

While hidden, the beacon scans for advertisement packets from the user and calculates RSSI. RSSI is a simple and widely available metric for estimating proximity between two devices; however, it is highly susceptible to rapid fluctuations due to interference and environmental noise, even when both devices remain stationary. To mitigate this, the system applies a two-phase smoothing process. First, a short moving average (window size 3) filters out brief spikes. This is followed by a scalar Kalman filter, which models measurement noise and expected variation in signal strength [2]. This combined approach balances noise suppression with responsiveness, enabling smoother and more stable proximity estimates without introducing excessive latency.

B. CoAP Client and Server

After calculating the RSSI, the beacon sends a binary-coded CoAP `PUT` request to the CoAP server using the `microcoapy`[1] library. This library is a minimal and lightweight implementation of a CoAP client that is built for microcontrollers such as the Pico W. To prevent overwhelming the CoAP server (and the downstream HTTP HomeAssistant server), `PUT` requests are sent once every 500ms.

CoAP was chosen over standard HTTP for its lightweight nature and suitability for IoT projects. It also avoids the heavy overhead of HTTP's TCP connection by using UDP.

The CoAP server is hosted on a Pi inside of a Docker container listening for requests. The use of a Docker container was two-fold; firstly, it allows the implementation to be easily ported to other systems in the event of a wider deployment of this demonstration, and secondly (and primarily), it isolates the CoAP server from other pre-existing processes hosted on the Pi server (such as a media and file server).

Upon receiving a request from the beacon, the server extracts the RSSI from the CoAP packet (shown in Figure 2) and calculates a normalised proximity in the range $[0, 1]$ by applying a threshold to the RSSI. Any signals above -45dBm or below -70dBm are capped to 1 and 0, respectively. Using this normalised distance, the CoAP server linearly interpolates a colour between red and blue (Figure 3) and relays a command to change the colour of the smart bulb to the HA server via HTTP REST API.

[1] https://github.com/insighio/microCoAPy

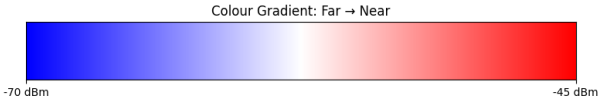


Fig. 3. Gradient of light colours between lower RSSI threshold (-70dBm, blue) and upper RSSI threshold (-45dBm, red).

C. HomeAssistant Server

HomeAssistant OS is a widely used open-source home automation platform that provides 'add-ons' that allow it to interact with devices using various protocols and communication media from various manufacturers. To communicate over the Thread media layer, the server must be configured as an OpenThread Border Router (OTBR) which is easily done via the official OTBR add-on. Similarly, the Matter server add-on allows compatible Matter devices (over Wi-Fi or Thread) to be controlled securely and centrally by the server.

While the Pi that hosts the HA server has built-in IP networking functionality (Wi-Fi and Ethernet), it does not have the necessary hardware to communicate over the IEEE 802.15.4 standard that Thread is built upon. As a result, a Radio Co-Processor (RCP, essentially a radio antenna) is required to bridge between IP networking and Thread networking. The nRF52840 dongle by Nordic Semiconductors can communicate over USB between the HA server and Thread devices after being flashed with the relevant OpenThread RCP firmware that can be built using the Nordic software and toolchain [3].

After adding Thread capability to the Pi server, a Thread network can be created within the HomeAssistant web UI that has itself (the OTBR) as the 'leader' of the network. Using HomeAssistant's RESTful API, the CoAP server can control any devices commissioned by the HA server using a long-lived authentication token, generated through the web UI.

D. Matter, Thread and Smart Bulb

The Nanoleaf Matter Essentials bulb is commissioned by the HA server using BLE (via the HomeAssistant mobile companion app). Network credentials and other metadata are securely transferred to the bulb during the process. After receiving all required information, the bulb uses the credentials to join the Thread network by connecting to the RCP managed by the OTBR. After the bulb is successfully onboarded by the OTBR, no further communication over BLE is used (unless the bulb is re-commissioned through a factory reset).

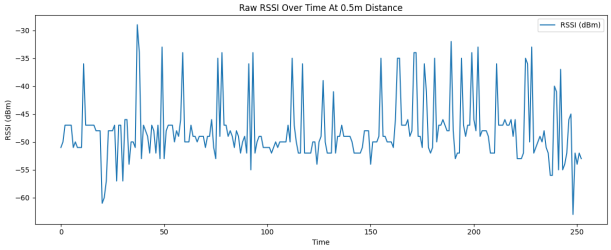
The Matter protocol provides clear, standardised ways of interacting with various IoT devices by listing their capabilities and functions, known as 'clusters'; for example the `ColorControl` cluster that enables the CoAP server to request different colours for the bulb.

IV. Evaluation and Results

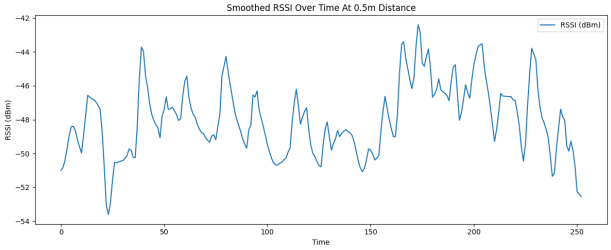
Overall, the system performs well for a networking demonstration of various IoT technologies, and is showcased in the video demonstration (Appendix B). The responsiveness of the system does make it slightly difficult to judge when the user

TABLE I
AVERAGE LATENCY OF VARIOUS SYSTEM NETWORK PROTOCOLS

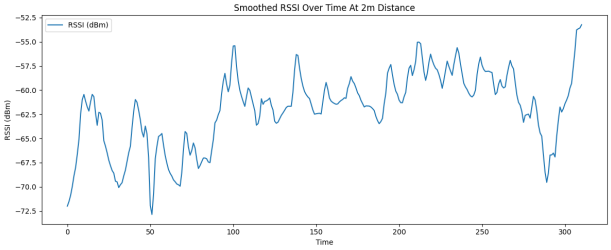
BLE (ms)	CoAP (ms)	HTTP (ms)
21.7	40.9	192.9



(a) Raw RSSI from the hiding device. User stood roughly 0.5m away.



(b) Smoothed RSSI using Kalman filter and moving average. User stood roughly 0.5m away.



(c) Smoothed RSSI using Kalman filter and moving average. User stood roughly 2m away.

Fig. 4. Visualisation of RSSI unreliability over a period of roughly 60 seconds. RSSI becomes more unstable with distance.

is in the correct area to locate the hidden device, and is likely due to the heavy TCP overhead when sending HTTP requests to the HA server, which is on average 5 times slower than the UDP connection managed by the CoAP server (see Table I).

The volatile nature of using RSSI as the chosen measure of proximity also means that the bulb can drastically change colour (e.g., immediately from red to blue) even if the user is stationary. Although smoothing was applied to the raw RSSI values obtained by the beacon, the overall proximity pattern can be unreliable, as shown in Figure 4, where the user is stationary from the beacon for a short period of time. RSSI becomes even more unstable as the distance from the hiding device increases, resulting in a wider range of colours (20dBm difference in Figure 4c) compared to when the user is nearby (12dBm difference in Figure 4b).

A. Limitations

Although the system performs well for a networking demonstration, there are several limitations and challenges of the various aspects of the final system design.

The most visually obvious limitation is the use of a large external power supply for the microcontroller. This limits the creativity of places to hide the device, possibly leading to less interesting games. Thanks to the energy-saving techniques in the BLE protocol, and the lightweight overhead of CoAP compared to other protocols such as HTTP (handled by a dedicated server), the hiding device has very low power consumption in normal use, and could be powered by a single battery for a compact design that is easy to hide and would last many days.

1) Philips Hue: As previously mentioned, this project initially planned to use the Philips Hue system for lighting feedback. Due to the very tight and restrictive proprietary BLE encryption standards enforced by Philips for their products, many attempts to control the bulb using the standard Linux BLE stack failed. Switching to HomeAssistant for lighting control also proved to be unfeasible, as a Philips Hue Bridge (similar functionality to an OTBR, but a proprietary and expensive piece of hardware) is required for an HA server to control and communicate with Hue products. Philips does allow products to be controlled through their mobile app or the Amazon Alexa ecosystem, but both methods were not suitable to integrate with the wider CoAP/BLE network used for this project. As a result, the system uses Thread networking which not only provided a richer and more interesting development and learning experience, but also is the primary emerging technology (along with Matter) for IoT devices, backed by many large companies such as Apple, Google and Amazon.

2) IPv6: The use of IPv6 for all CoAP traffic would help to future-proof the concept, in addition to matching the IPv6 traffic used by the Matter protocol. However, IPv6 support is limited on the microcontrollers used in the project. Many libraries, particularly MicroPython libraries and including `microcoapy`, seem to have been built from the ground up for IPv4, with IPv6 as an afterthought, leading to poor support and functionality. Furthermore, the LAN on which the project was tested had inconsistent IPv6 support, with the router not assigning an IPv6 address to the CoAP server or the microcontroller.

3) Bluetooth Classic: Being designed specifically for microcontrollers, MicroPython has almost no support for Bluetooth Classic; a very heavy protocol. The mobile device used in the project uses the iOS operating system, which supports both Bluetooth Classic and BLE, but does not provide any user interfaces to natively work with BLE. Rather, a dedicated BLE application is required for BLE functionality.

There are many existing dedicated BLE applications available for iOS, almost all of which are free and easy to use. However, being able to use the system Bluetooth settings would have made the set up (connecting and pairing with the beacon) of this system much simpler, as an additional app would not be required; the user could simply connect to the microcontroller in the same way they would pair to a Bluetooth speaker or headphones.

4) Dual Raspberry Pi Servers: For an instance of Home-Assistant to be compatible with Thread devices, it must have a 'Supervisor'. This is a low-level piece of software that enables many HA features such as add-ons, snapshots and more. The Docker installation of HA does not contain a Supervisor[2], and therefore cannot be used to control Thread devices such as the Nanoleaf bulb. HomeAssistant OS (HAOS) is a first-party operating system that includes a Supervisor, is easy to install, and is user-friendly, and was the choice for this project. However, in line with HA's vision of privacy and security, HAOS is locked down and restricted, and does not allow the execution of arbitrary Python code such as a CoAP server. This meant that two Raspberry Pi devices (or similar) are needed compared to a single device acting as a CoAP and HA server and Thread border router.

V. Conclusion

This project successfully demonstrates a proximity-based interaction system in the form of a Hide & Seek game built using modern IoT technologies. By integrating Bluetooth Low Energy, CoAP, HTTP, Thread, and Matter protocols across multiple microcontrollers and servers, the system delivers responsive feedback to guide the user toward a hidden beacon. Despite implementation challenges—such as unstable RSSI readings, proprietary hardware restrictions, the complexity of Thread radio firmware, and protocol-induced latency—the system performs reliably and provided valuable learning opportunities in both IoT and general networking.

The project also highlights the current state of smart home device control, with competing technologies like BLE, Wi-Fi, Thread, and manufacturer-specific ecosystems. Matter aims to unify this landscape with a common, open standard and is increasingly being adopted by the wider IoT community.

A. Possible Improvements and Extensions

1) iBeacon: The use of the iBeacon protocol developed by Apple (built on top of BLE) could have provided more accurate and regular RSSI proximity estimates, leading to a smoother overall user experience. However, it is difficult to implement on non-Apple devices (Android) due to Apple's tight ecosystem, and would require a restructure of the system, as the proximity must be transferred by the mobile device to the CoAP server, rather than from the iBeacon hidden device itself.

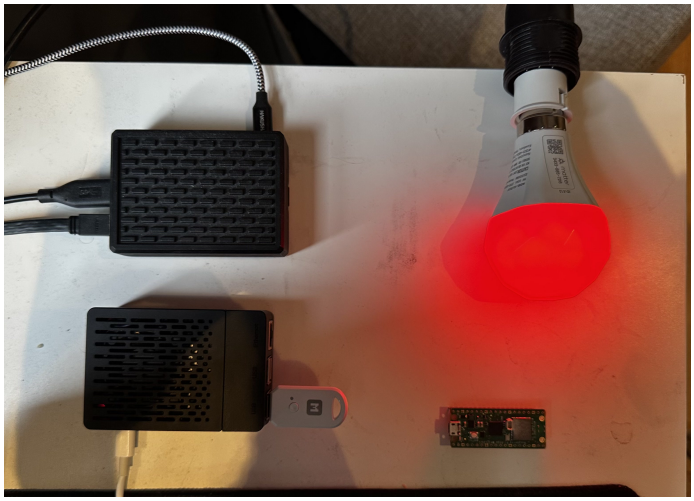
2) Other extensions: As previously mentioned, the lightweight low-energy nature of the system lends itself to the use of battery power for the hiding device, perhaps with a small solar panel to recharge on a window sill or other sunny location when not in use.

In addition to the visual feedback of changing colour, audio or haptic feedback from a smart speaker and a simple motor could further help the user find the device, and make the game more enjoyable.

[2]There are methods of adding a Supervisor to a Docker container installation of HA, but it is extremely complicated. Two servers were sufficient for the purposes of this demonstration project.

Finally, the use of IPv6 in all networking traffic would provide better standardisation and follows a future-first design. However, microcontroller support (or workarounds) would be required and this improvement would not affect the end-user experience.

Appendix A
System Hardware Photo



This appendix shows the devices used in the system. Clockwise from top left: Raspberry Pi hosting CoAP server, Nanoleaf Matter-over-Thread bulb, Raspberry Pi Pico hiding device, Raspberry Pi hosting HA server with white nRF52840 RCP dongle attached over USB.

Appendix B
Video Demonstration

The video demonstration for this project can be found at the following link.

References

[1] Connectivity Standards Alliance. (2024) Matter Specification. https://csa-iot.org/all-solutions/matter/. Accessed: 2025-05-05.

[2] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[3] Nordic Semiconductor. (2024) Product Portfolio and Development Tools. https://www.nordicsemi.com/. Accessed: 2025-05-05.