# Thread Network In Clyde Building Report

Devon Ward, Daniel Harman

April 15, 2024

## Introduction

Thread is a mesh networking protocol which has been developed mainly for smart home devices [Ste]. The main idea is to remove a single point of failure for the networking smart home devices, allowing them to communicate with each other and avoid having a potential problem where one device fails, taking down the entire network. Another beneficial aspect of thread is that it is designed to be low power [RR19], which is also a really good thing for IoT devices, which are often power constrained.

One problem that Thread networks have currently is that they are not very well tested. There are many sources that have successfully tested Thread networks and proven that they can operate in a noisy environment but most publications are focused on throughput achievements, rather than distance and reliability.

Our project seeks to attempt to classify the performance of Thread in the Clyde engineering building at Brigham Young University, and determine if a Thread network is capable of allowing two users to communicate from the graduate research lab in CB 480 to anywhere in the Clyde building. In doing so, we also seek to understand how many nodes are required to allow for uninterrupted communication in the Clyde building.

Our experiments seek to quantify the range of the Thread devices using ESPRESSIF ESP32-H2 Dev boards [Sys] to get an estimate of how many routers we will require for the Thread network. We then plan to strategically place the routers throughout the Clyde building, and attempt to walk through the building without losing connection to the parent node in the graduate lab. We will quantify exactly how many router Thread nodes we will require, and attempt to use the minimum number possible.

We found that we only required one Thread router node, one parent node, and one child node to allow the child node to go all around the basement of the Clyde, and through the rest of the building without losing connection. We were also able to briefly go outside the Clyde building and still communicate with the graduate lab.

## Related Work

There are many sources that have successfully tested Thread networks and proven that they can operate in a noisy environment [RIR18] but very little to classify their performance in indoor urban environments. Work has previously been done to quantify the throughput of Thread networks in an indoor area [Jai+23] [Sis+19] but most publications are focused on throughput achievements, rather than distance and reliability.

Many tutorials exist for building a Thread network with an ESP32 [Ope] as well as details on how to implement the design and program the microcontroller [ESP] to run the thread network.

There are papers that discuss distance as a metric for the Thread network operating [Kha+23] that talk about the jitter in the network. However, they do not discuss the maximum distance for transmission when the Thread mesh breaks down. For our project we wanted to investigate the maximum transmission distance for an indoor environment containing lots of metal doors over multiple floors, something that is not present in the current literature.

## Methodology

We approached our problem in the following steps:

1. Program the ESP32-h to run a thread network
2. Set up a TCP thread connection built on top of UDP to allow secure reliable communication over the thread network to measure performance
3. Measure distance leader and child thread devices can communicate
4. Add router thread nodes until we can map the entire Clyde building

In order to program the ESP32-h devices, we used ESPRESSIF's tutorials which allow us to flash the ESP32 devices with an already functional thread network. This significantly reduced the time for development of the thread network.

Second, we wanted to set up a TCP connection. This is unintuitive because thread by design uses UDP, and there is no real sense of acknowledgements, since the mesh network is utilizing router devices to send UDP packets from the leader to the child nodes. However, we want to see when we lose connection with the child nodes as we walk around in the Clyde building. Therefore, we use a TCP client built on top of the UDP client to see when we stop receiving acknowledgement packets from the child to the leader node.

Once the TCP mesh network is set up and running, we then walk around the Clyde building with only the leader and child, and categorize the number of walls we can transmit through, and the standard distance we can communicate. By doing this we can find the blind spots of the Clyde building that we cannot reach without router nodes in the mesh network. Then we add single router nodes one at a time and see how many router nodes we need to cover all of our previously identified blind spots and have reliable communication throughout the entire Clyde building.

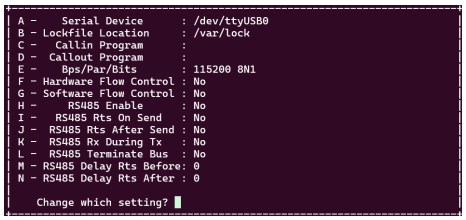


Figure 1: Screenshot of the Minicom Settings

## Implementation/Experimentation

### 0.1 Programming ESP32h2 Module

The ESP32h2 module is a relatively new chip from Espressif, meaning that support for it has not made it to the larger space of IDE's, such as Arduino or VScode's extension Platform.io. Thus, we installed the VScode Espressif IDF extension, which is developed and supported by Espressif as direct access to their software development kit (SDK). The extension conveniently has access to each of their tutorials [ESP]. We used the tutorial "ot_cli", or OpenThread Command Line Interface. The program allows an instance of OpenThread, an open source implementation of the Thread standard to be run via a command line interface with the ESP32 using a serial port connection.

Opening the project in VScode via the Espressif IDF extension allows you to

1. Select a connected USB device/port
2. Build the example project
3. Flash the ESP32

### 0.2 Setting up the Thread Network

Once flashed, a serial connection needs to be established with the ESP32 to setup the thread network and send messages. We used the Minicom serial port terminal emulator for Linux, with settings as displayed in 1. Once connected, we ran the following to setup a Thread network for the network leader. A network leader is a role within a Thread mesh, that provides a key used to join the network and as well

as assuming some necessary centralized functions for the network. To set up the network, we ran:

```
> factoryreset
> dataset init new
> dataset commit active
> ifconfig up
> thread start
> state
> ipaddr
> dataset active -x
```

The last two commands are how the IPv6 address for the leader and the Thread network key are found respectively. These need to be copied over for the other devices to use to join the network and to setup a TCP socket with the leader for testing purposes.

For the child devices (those on the network, but not the leader), we connected to the network using:

```
> factory reset
> dataset set active $PUT THREAD KEY HERE$
> ipconfig up
> thread start
> state
```

Note the state is just a test to establish connection to the network and status within the network.

## 0.3 Setting up the TCP connection

For the server side, we used the following:

```
> tcpsockserver open
> tcpsockserver bind :: 12345
> tcpsockserver send MESSAGE_WITH_NO_SPACES
```

and for the client side we connect to the open socket by using:

```
> tcpsockclient open
> tcpsockclient connect $PUT IPV6 ADDR HERE$
    12345
> tcpsockclient send $MESSAGE_WITH_NO_SPACES$
```

## 0.4 Experiment

The experiment that we ran involved running 3 Thread devices to determine the maximum length of connectivity that the mesh could support. The leader

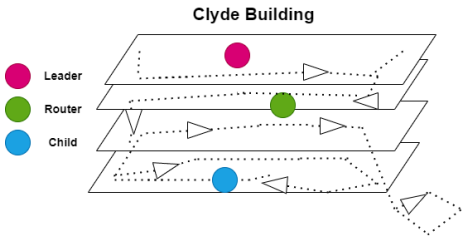


Figure 2: Map of Experiment Path Taken

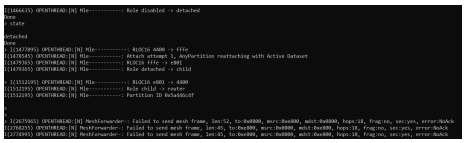

Figure 3: Command Line Mesh Error Response

was in the ECEn graduate student lab as a stationary device. The other two were connected to laptops and moved throughout the Clyde Engineering building. The path taken is shown in Fig. 2. Connection to the leader was strong through out the 4th, 3rd and most of the 2nd floor. The connection began to break down when entering the basement, but strategic placement of the intermediate node allowed the client to get TCP packet through. Even intermittent service outside was possible.

Fig. 4 shows the output of the mesh following a successful setup. Once a device is on the network, there is no way to determine if it acting as a mesh router. This posses the issue of knowing whether or not the placement of an intermediate device is actually beneficial or not. This is helped though by messages that appear on the intermediate device when it fails to pass-along a message, shown in Fig. 3.

To determine if the 3rd device actually played a

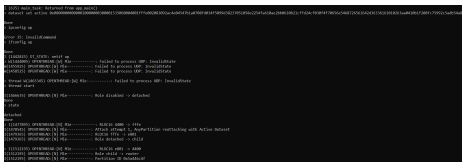

Figure 4: Command Line Mesh Setup Output

significant role, we shut it down for a time and re-walked many of the same areas in the basement and the 1st floor of the Clyde building. The connection between leader and child was essentially the same throughout floor 1, meaning that there really wasn't a significant advantage introduced by the intermediate device, which is unfortunate for Thread, but demonstrates the robustness of Zigbee. However in the basement, Zigbee was unable to reach the entire basement without the help of the intermediate router node.

## Conclusion

Thread is an interesting protocol with potential for many applications. It does not however offer much in terms of the meshing, over direct connections between Zigbee devices. The software support and marketing of these devices will need to increase though before they can reach their full realization and improvements.

# References


[RIR18] Wojciech Rzepecki, Łukasz Iwanecki, and Piotr Ryba. "IEEE 802.15.4 Thread Mesh Network – Data Transmission in Harsh Environment". In: *2018 6th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. 2018, pp. 42–47. DOI: 10.1109/W-FiCloud.2018.00013.

[RR19] Wojciech Rzepecki and Piotr Ryba. "IoTSP: Thread Mesh vs Other Widely used Wireless Protocols – Comparison and use Cases Study". In: *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2019, pp. 291–295. DOI: 10.1109/FiCloud.2019.00048.

[Sis+19] Srikanth Sistu et al. "Performance Evaluation of Thread Protocol based Wireless Mesh Networks for Lighting Systems". In: *2019 International Symposium on Networks, Computers and Communications (ISNCC)*. 2019, pp. 1–8. DOI: 10.1109/ISNCC.2019.8909109.

[Jai+23] Dheryta Jaisinghani et al. "IoT in the Air: Thread-Enabled Flying IoT Network for Indoor Environments". In: *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2023, pp. 142–147. DOI: 10.1109/PerComWorkshops56833.2023.10150308.

[Kha+23] Sohaib Bin Altaf Khattak et al. "Enhancing Wheelchair Communications Utilizing Thread Protocol for Improved Patient Safety". In: *2023 IEEE International Smart Cities Conference (ISC2)*. 2023, pp. 1–6. DOI: 10.1109/ISC257844.2023.10293573.

[ESP] ESP32. *Networking API - Thread*. URL: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_openthread.html. (accessed: 04.12.2024).

[Ope] OpenThread. *Build a Thread Network with the ESP32H2 and ESP Thread Border Router Board*. URL: https://openthread.io/codelabs/esp-openthread-hardware#1. (accessed: 04.12.2024).

[Ste] Steve. *Everything you need to know about the Thread network*. URL: https://nuventureconnect.com/blog/2021/07/12/everything-you-need-to-know-about-thread/. (accessed: 04.12.2024).

[Sys] Espressif Systems. *ESP32-H2-DevKitM-1*. URL: https://docs.espressif.com/projects/espressif-esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/index.html. (accessed: 04.12.2024).