Proceedings of the Federated Conference on Computer Science and Information Systems pp. 501–510
DOI: 10.15439/2020F99
ISSN 2300-5963 ACSIS, Vol. 21

A LoRa Mesh Network Asset Tracking Prototype

Emil Andersen, Thomas Blaalid, Hans Engstad, Sigve Røkenes
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

Frank T. Johnsen
Norwegian Defence Research Establishment (FFI)
Kjeller, Norway

***Abstract*—Long Range (LoRa) is a low powered wide area communications technology, which uses radio frequencies in the industrial, scientific and medical (ISM) band to transmit data over long distances. Due to these properties, i.e., the long range and little restrictions on deployment and use, LoRa is a good candidate for building an asset tracking application on, for example targeting search and rescue operations. This paper describes the development and testing of such a prototype, using commercial off-the-shelf Internet of Things (IoT) consumer devices and a proprietary mesh protocol.**

The prototype enables distributed position tracking utilizing the Global Positioning System (GPS), a gateway to the Internet, a server for data storage and analysis, as well as a Web application for visualizing position tracking data. The devices are small, and our tests have included both personnel on foot carrying the equipment, as well as having the devices on vehicles.

***Index Terms*—Internet of Things, Wireless mesh networks, Web services**

I. Introduction

EXISTING technologies such as cellular networks offer rapid communication across fair distances, but are limited in their scope of operation. Large cellular towers have limited range and rely on extensive infrastructure to provide service, and consequently it is expensive to maintain and expand this type of network. Due to their isolated nature they are also susceptible to a single point of failure [1], [2].

A mesh network circumvents many of these limitations through its distributed design. Perhaps most crucially, a mesh communications network is not reliant on a central station or site. A distributed solution can provide connectivity in near any location, even those without any existing infrastructure. It will also be more resilient to network failure than a centralized solution, because it has no single point of failure. If a network node is destroyed or otherwise left inoperative, remaining units in the network are adaptable and will continue to provide service. Finally, the hardware required to deploy a Long Range (LoRa) mesh network is inexpensive and accessible [3]. This enables swift propagation of a large number of devices, which is essential to establish a robust distributed network. The tradeoff for the advantages of long range and low power use is primarily the low rate of data transfer that LoRa offers [4].

The properties of mesh networks make them particularly suitable for certain applications where traditional communications infrastructure is impractical or insufficient. Examples of application areas include asset tracking, sensor data dissemination, and low bandwidth communication in case of exhaustive infrastructure failure. Due to the low cost and accessibility of the necessary hardware, these, as well as many other applications, are both feasible and pragmatic using affordable commercial off-the-shelf (COTS) devices [5].

A notable example of this is search and rescue operations in areas with challenging geography, such as Norwegian mountain ranges. These areas may not have reliable access to cellular networks, which leaves crucial primary communication channels unavailable. In order to perform a successful search, for instance when looking for a missing hiker in the mountains, it is important to know which area the rescue personnel is currently in (current position) as well as which areas have been searched already (history of positions). An application for location tracking using a LoRa mesh network could provide this, while simultaneously producing a detailed map of all regions of the area searched so far. In this paper, we pursue a prototype implementation of such a search and rescue system, built on COTS Internet of Things (IoT) consumer devices and a proprietary mesh protocol.

The remainder of the paper is organized as follows: Section II outlines the scope of our work, whereas Section III gives an overview of the technologies involved in the prototyping effort. Section IV discusses the design and implementation of our software. The tests we performed using our prototype are summarized in Section V. Section VI presents the analysis of our findings, leading up to a summary of results in Section VII. Section VIII presents related work. Finally, Section IX concludes the paper.

II. Prototype scope

The purpose of our work was to develop and test an initial prototype for search and rescue operations, with the main focus being a distributed network for geographical tracking. To achieve this, we limited our scope to COTS IoT products to build an affordable, highly portable and easily deployable system. Hence, we chose to focus on LoRa, since the protocol offers long range communications while at the same time being battery efficient. LoRa is also one of the few choices out there that you can deploy with few limitations, as it is not reliant on commercial infrastructure, as opposed to e.g., NB-IoT and Sigfox [6]. Due to this, our prototype system uses the LoRa protocol to send messages between devices in a mesh network. The functionality scope is limited to each device reporting its position using its onboard Global Positioning System (GPS).

The mesh network itself is self-healing and not reliant on any infrastructure. Internet is not needed, granted that the

Note that the NTNU authors were equally involved in the work, and hence are listed in alphabetical order by surname. The work was sponsored by FFI project 1431.

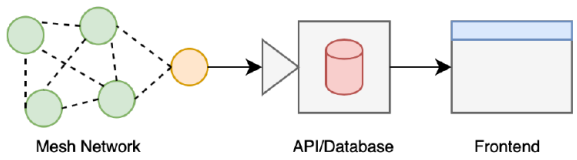

Fig. 1. Prototype system high-level architecture

back-end is available locally. However, for the prototype we deployed the back-end on the Internet, so here one of our devices needs Internet connectivity. These devices that connect to the Internet are referred to as *border routers*. Border routers should receive coordinate data from the whole mesh network, and forward it to a database web server application programming interface (API). The web server is responsible for storing the data persistently in a database. Finally, the location data should be displayed in a frontend web application. In addition to the location data, the application should show the nodes' roles within the network (for debugging), historic positions of nodes (can be toggled on or off, an important feature of the search and rescue application), as well as other relevant metadata collected by the system (used for prototype and protocol evaluation purposes).

Figure 1 illustrates the system architecture. Here, green circles are routers in the mesh network. The orange circle has the special role of the border router, and is responsible for forwarding data from the mesh network to the web server hosting the API and database. Note that even though the figure only shows one border router, there may be multiple for redundancy. In our implementation, a router that is in range of a pre-defined WiFi network SSID and manages to connect to it, automatically becomes a border router. The system needs at least one border router to function as expected, since otherwise the data flow to the database will be disrupted. The mesh network operates on the COTS IoT products over LoRa, whereas the remainder of the system is deployed on the Internet using a Web service API.

III. Technology Background

The COTS products we chose were the Lopy4 [7] and Pytrack [8] from Pycom LTD., since these devices provide an inexpensive IoT prototyping platform for developing with Python. Also, the devices have GPS support as well as several protocols, including LoRa, WiFi, Bluetooth, and Sigfox. Below we present some key properties of the LoRa protocol, followed by a brief introduction to PyMesh [9], which is the mesh network implementation we used in our prototype.

A. LoRa

LoRa is a proprietary physical layer specification that enables a chip with an inexpensive crystal to have high sensitivity, and provides long range wireless transmission with low data rate and low energy usage. The industrial, scientific, and medical (ISM) frequency band that LoRa devices operate on is 915 MHz in the US and 868 MHz in Europe [10].

TABLE I
Relevant EU 868 frequency band data rates

Data rate	SF/Chirp rate	Indicative physical bit/s	Payload size
DR5	SF7/125kHz	5470	242 bytes
DR6	SF7/250kHz	11000	242 bytes

LoRa devices achieve long range due to the transceiver's ability to filter on the constant chirp signals, which enables the device to detect and lock to the LoRa signal. A chirp signal is a rapid increase or decrease in radio frequency over time. The LoRa protocol uses variations of these chirps to establish a connection and encode transmitted data [4].

1) Data rate: The data rate is a direct result of the chirp rate used for transmission. A higher chirp rate enables LoRa to encode more data in the same amount of time. A key advantage of LoRa is its ability to demodulate multiple simultaneous signals at the same frequency if the LoRa devices use different data rates [11]. This increases the capacity of a single LoRa device and enables them to communicate with a large number of devices simultaneously if necessary, as long as the adaptive data rate functionality is enabled. This could however be problematic for continuously moving devices, because higher data rates reduce their range and could prevent their signal from reaching a neighboring device in the network.

2) Spreading factor: Spreading factor (SF) is a parameter in the LoRa protocol that directly affects battery usage, range and how often a device can transmit a message. SF adjusts the number of chirps (the data carrier in the signal) that are sent per second. A lower SF indicates that more chirps are sent per second, whereas a higher SF implies a lower chirp rate. Sending data of the same length with a high SF will create a longer transmission time (known as airtime). More airtime forces the modem to run for a longer duration, and therefore consume more energy. SF is graded on a roughly exponential scale between 7 and 12, where each step is equivalent to doubling the airtime for each unit data and an approximate increase of 2.5 dB in signal strength. A SF of 12 would have the greatest range because the receiving device has more opportunities to sample the signal. Another consideration is that longer airtime result in fewer opportunities to send data (since each message takes longer to transmit). LoRa supports several different data rates (DR). A payload of 11 bytes using a DR0 configuration would only be able to send data roughly once every 2 minutes following appropriate government regulations.

For the sake of our prototype, we experimented with two different data rates: DR6 (which is the default for PyMesh) as well as DR5. Details of these DR are shown in Table I. Lower DR than 5 have decreasing payload sizes, that possibly could affect the performance of the mesh network. Our own prototype data format only has a payload of 53 bytes, and should theoretically (not tested) be usable all the way down to DR3, which offers a payload of 53 bytes.

B. PyMesh

PyMesh is a LoRa based mesh network technology consisting of a firmware and library that we obtained from Pycom

TABLE II
DEVICE ROLES IN THE PYMESH NETWORK

Role	Explanation	Color
Router	Most devices in the network will usually be routers. Routers are devices with neighbors, that are capable of forwarding data towards a border router.	Green
Leader	The leader device is responsible for distributing addresses within the network and making other devices aware of where the nearest border router is located. There is always a single Leader in each network partition, which is dynamically self-elected.	Purple
Child	The child role is given to devices located on the edge of the network graph. These devices are not located in a path to neighboring nodes and will therefore never forward data from other devices.	White
Border Router	The border router role is assigned to devices with an Internet connection. There can be multiple border routers in the mesh network. Ultimately, these devices are responsible for forwarding data from the mesh to a web server through the Internet. A border router may simultaneously act as router or leader as well, depending on the network.	Orange

under a time-limited developer's license. Their technology is implemented using OpenThread [12] which is an open source implementation of the IPv6-based networking protocol Thread [13]. PyMesh was developed by Pycom to enable LoRa MAC addresses to be used over IPv6, and therefore enable an OpenThread mesh network to operate over LoRa. PyMesh removes the need for static gateways, which decentralizes the network's infrastructure and makes it more flexible.

PyMesh will automatically assign a Pycom device to one of four different network roles — leader, border router, router or child (see Table II). The role of a device changes continuously based on several factors, the most important being the radio-link strength between devices (Received Signal Strength Indication (RSSI)). The assignment of roles creates a link local address for every PyMesh device directly connected to another device, as well as a mesh local address for every device in the same network. This enables all LoRa messages to be routed efficiently through the mesh network to a device capable of forwarding it out of the network (border routers), as well as updating mesh information on other devices.

IV. DESIGN AND IMPLEMENTATION

Our prototype uses a client-server architecture. The system has two clients, the PyMesh border router and the frontend web application, and a web server providing the API endpoints and persistent storage. The server interfaces with the clients in different ways: The PyMesh border router sends data from its devices to the server, while the frontend application requests data from the server. Both methods use a Representational State Transfer (REST) API over HTTP. Below we outline the central parts of the prototype. It should be noted that even if the Pycom devices support multiple network protocols, we are only using the LoRa and WiFi capabilities in our prototype. For the complete description of software development methodology and a more detailed system architecture, see [14].

A. PyMesh

There are eight components in the software we built on PyMesh:

1) **Unit**: The Unit component is the most important component, since it is responsible for managing and calling the other components. The methods in Unit are called from the special main.py file, which is automatically executed when a device is powered on. The component is responsible for ensuring the device is checking for an Internet connection at regular time intervals, setting up the correct role for the device, collecting and making sure data is packed correctly, and finally forwarding it to either another device if it is not a border router, or to the server API if it is.
2) **WiFi**: The WiFi component is responsible for trying to connect the device to a specified WPA2 secured WiFi network if it can find a connection. It has a method for returning if the device has a connection, that Unit uses to determine if the device should act as a border router or not. WiFi is the only protocol we support in the prototype at the moment for bridging a border router to the Internet.
3) **Setup**: The Setup component handles the configuration of the device depending on whether it is a regular router or a border router. It is also responsible for initializing the PyMesh configuration on the device making it a part of the network.
4) **Callback**: The Callback component handles message forwarding depending on whether it is a regular router or a border router when receiving a message. If it is a regular router the package will be forwarded throughout the network until it reaches a border router. When data reaches a border router the Callback component will trigger a method in Unit in order to send the package out of the PyMesh network to the server.
5) **DataPacker**: The DataPacker component is responsible for retrieving the data from the GPS component and packaging the data into a format which is suitable for sending through both LoRa and HTTP to the API. Most of the data is provided by the GPS component, but the DataPacker is also responsible for fetching the MAC address and the node type (role) of the device. The component translates the data between the static byte packet format we have defined (see Figure 2) and standard Python dictionaries in order to enable this.

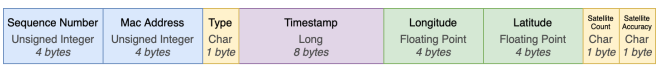

Fig. 2. Our packet format for LoRa GPS transmissions

6) **GPS**: The GPS component handles everything related to the Pytrack GPS. When Pytrack has a GPS fix, this component returns the necessary data to the DataPacker component. The data consists of positional information like longitude and latitude, as well as satellite accuracy and the number of satellites available. It also synchronizes the GPS time with the Real Time Clock (RTC) on the device for timestamping the data packets. If the GPS component cannot find a fix, it will continuously attempt to establish a connection as long as the device is powered on.
7) **Light-emitting diode (LED)**: The LED component is a helper component which is most useful in debugging and showing the user what process is happening. It is useful for observing what role the device is assigned to or when the device is trying to establish an Internet connection. Figure 3 shows the meaning of the LED colors. The device cycle starts with the LED blinking yellow (WiFi color code) if it is not connected, or the LED being on constantly if it has a connection and is acting as a border router. The cycle ends with a constant color representing the role a device has in the PyMesh network at that time.

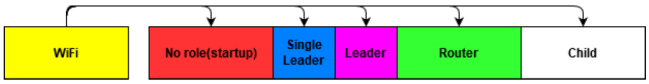

Fig. 3. LED colors assigned to roles and when WiFi is connecting/connected

8) **Node Type**: The Node Type component contains constants with the different roles in the PyMesh network and corresponding LED colors in Figure 3. It is responsible for encoding PyMesh role definitions in the data format used by the DataPacker and API (i.e., the *Type* field shown in Figure 2).

B. Backend

We designed the backend architecture to be light weight and intuitive. There are three entities in the data model, Coordinate, Node and GaiaCoordinate. The GaiaCoordinate entity is implemented on the server for testing purposes, to hold data from the Gaia app [15] we used for GPS comparison, as further discussed in Section V. Each Pycom device will have a unique MAC address, which will be linked to the Node data-model. The Coordinate entity will store each coordinate that it receives from the mesh-network, and link this coordinate to the corresponding Node. The GaiaCoordinate entity will also link its coordinates to a node to be able to visualize a reference in the frontend after testing. The backend consists of two Django-apps [16], which are called “core” and “api”. The core-app contains what can be considered the business-logic and is closer to the database. The api-app defines a REST API which both the frontend and Pycom devices can use.

TABLE III
PYCOM DEVICE FIRMWARE, SMARTPHONES, OPERATING SYSTEMS AND APP VERSIONS

Pytrack firmware	pytrack0.0.8.dfu
LoPy4 firmware	LoPy4-1.20.2.r1
Pycom devices	Pycom A (Mac 4), Pycom B (Mac 5), Pycom C (Mac 3), Pycom D (Mac 7), Pycom E (Mac 6)
Samsung Galaxy J3 SM-J330F	One UI version 1.1, Android version 9, WPA2 WiFi hotspot
Huawei P20 EML-L29	EMUI version 9.1.0, Android version 9, Gaia GPS version 2020.3
Samsung Galaxy S20+ SM-G986B/DS	One UI version 2.1, Android version 10, Gaia GPS version 2020.3
Server	Ubuntu 18.04.3 LTS, GNULinux 4.15.0-76-generic x86 64

C. Frontend

The frontend was written in React [17], and uses a component hook based architecture. Its structure can be divided roughly into two parts, the map for geographic visualization of node positions, and the supplementary user interface for orienting within the application as well as accessing more advanced functions. API hooks are responsible for providing the necessary data for components, which it fetches from the server-side REST API. In order to improve performance and reduce complexity, request parameters such as filtering and search are offloaded to the server. Responses are provided in a Java Script Object Notation (JSON) format, specifically GeoJSON [18] for coordinate data, which, with minimal processing can be visualized in our frontend.

V. TESTS

We wanted to test PyMesh functionality in practice, to see how it would perform as the data carrier for our search and rescue application. The hardware and software we used is summarized in Table III. The goal of our system testing was to gather data to investigate certain metrics:

- Average packet loss and packet loss by range for DR5 and DR6
- Average range for DR5 and DR6
- Average GPS accuracy in meters
- GPS accuracy over time in meters

To have an additional source of GPS tracks, we chose to use the Gaia GPS app [15], which includes functionality to export recorded tracks as easily interpretable Comma Separated Values (CSV) files. This made it easy to perform analysis on gathered data from our system and import Gaia tracks directly into our database and web page for comparison.

Before a test, each Pycom device participating is connected to a power source. When booting they have to be minimum two meters apart and not be started simultaneously, in order to prevent a PyMesh connectivity issue (this occurs due to LoRa transceiver saturation and subsequent mesh initialization

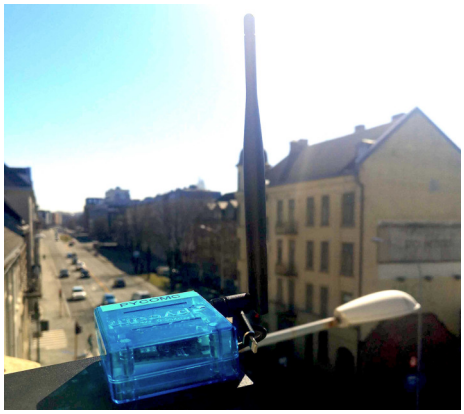

Fig. 4. Photo of our Pycom C node.

failure). A phone with 3G/4G is used as a WPA 2 WiFi hotspot, and the device is placed near the Pycom device that should become a border router. Once all devices get a GPS fix (this can take between 5 and 15 minutes depending on the location), the test can begin. The total setup time is approximately 15 minutes to ensure all devices get a GPS fix, and the test will begin when the recording on the Gaia app tracking is initialized. It should be noted that we do not use an external GPS antenna, which would likely give us a location fix more rapidly. The only external antenna used is for LoRa, both GPS and WiFi in our nodes use the built-in antennas (see Figure 4 for a picture of one of our fully assembled nodes).

At this point, testing can commence. When a *test session*[1] is finished, all Gaia GPS logs have to be exported from the phone manually, transferred to a computer and parsed into the database. This makes the trail visible in the web application. The raw log file is also uploaded to the source code repository to be accessible to our analysis scripts.

A. Limitations

The COVID-19 pandemic severely impacted our collaboration. The quarantine rules posed a major obstacle in gathering data for analysis. Initially, we had planned to test together and gather data with all five devices we had bought, but this was no longer possible. Because of the pandemic, we were not able to do testing with all the Pycom devices simultaneously. Further, we encountered a hardware issue of one of the devices. Rather than replacing the faulty node as we would have aimed to do under normal circumstances, we instead opted to attempt to attempt to repair it so that we could continue our tests. In spite of this we managed to perform some tests and gather useful data, using up to three Pycom devices at a time, which were located in Trondheim. The remaining two devices were in another city, and the team member with access to those were in quarantine, so they were used for software development, but not any of the range tests.

Within these limitations, we performed the test series outlined below.

[1]We define a *test session* as a number of tests in an urban or rural environment with a given data rate. A test session may also include transportation to and from the given environment to gather data for GPS accuracy analysis. The main goal of a test session is to gather data for packet loss as a function of the distance to a border router, or to test the limit in range to a border router. The test sessions are summarized for brevity in this paper. For complete information on all test, see [14].

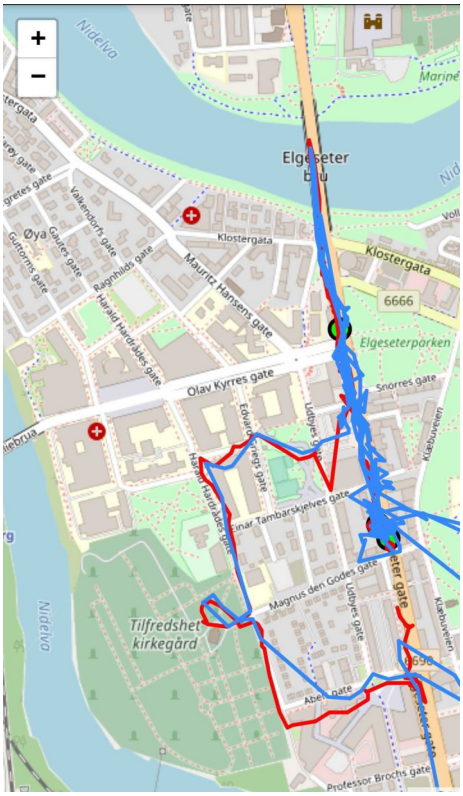

Fig. 5. PyMesh trail (blue) on top of Gaia (red) for tests 1 and 2.

B. Test session 1

This session took place in Trondheim city and the priority was to measure packet loss at various distances, so as to investigate the range of the communication system. The Gaia GPS trail and Pycom trail for both tests 1 and 2 can be seen in Figure 5. All test were performed with DR6.

1) Test 1: Range: Since this was the first range test we slowly increased the distance to the border router while maintaining line of sight. After Pycom D was out of range on Elgeseter bridge and returned back in range, the devices renegotiated and switched roles (Pycom D was a router but was assigned to the leader role).

2) Test 2: Range: After the first range test was complete, a new Gaia recording was started to now measure packet loss with buildings between the Pycom devices. At the end of the test, Pycom D had an error where the LED was blinking yellow rapidly, indicating an error. The test was concluded soon after.

C. Session 2

This testing session took place in the rural outskirts of Trondheim. The testing environment was challenging for the

Pycom devices, since there is a lot of vegetation that blocks the line of sight. Terrain height differences made it more difficult to correctly measure the packet loss with respect to distance. All tests were performed with DR6.

1) Test 6: Rural Forest: Once the test started, both devices already had high GPS accuracy because they had been running for over 30 minutes. We walked up a forest path and lost connection due to a mountain crest in the way, but regained connection once Pycom D was in higher terrain and vegetation decreased. We experienced no issues, and Pycom D reconnected quickly once it returned in range.

2) Test 7: Rural Forest: This test started in a different direction on a steeper path with more trees between the devices. At the top of the hill the path flattened out and the network was subjected to more vegetation than just trees. The packet loss at this location was high, and we noticed that Pycom D had an issue where the LED was rapidly blinking yellow, similar to the one in test 2. It was later discovered that the Pycom D USB connector was loose. This hardware issue can be attributed to the faults seen on this node, since it would affect the power supply. Later, we soldered the connector back on, which improved the stability of this node.

D. Session 3

During this session we tested routing between three devices. The session was conducted in Trondheim city. The test consisted of two team members with one device each, as well as a border router. The border router was located outside a window on the fourth floor, and did not move during the test. The test was performed using DR6.

1) Test 8: Routing: Some issues occurred approximately 10 minutes into the test where both devices lost connection to the network, causing the test to be paused briefly. After some time both devices managed to reestablish their connection to the mesh network, and testing resumed. The two participants in the test walked together until maximum range was reached to the border router. Afterwards, one member continued to walk in the same direction with their device, while the other stayed stationary. Routing worked well between these devices. After a short while an issue occurred where Pycom D did not send its own data. In spite of this issue we were able to test routing successfully.

E. Session 4

This session took place approximately 10-15 minutes from Trondheim center, in the area previously used for session 2. Two members of the team took part in the session, each with their own device, as well as the border router. Two tests were conducted during this session. The border router was placed in a static position. All tests were performed using DR5.

1) Test 9: Rural Forest: This test used the same route as test 6 and both devices were assigned the router role. At the point furthest away from the border router, both devices lost connection to the mesh network and could not reconnect. There seemed to be a hardware or firmware issue causing the devices to be unable to reconnect. Saturation could also be a factor causing the issue, but this is uncertain – it may well be that we encountered the known issue described here [19].

2) Test 10 and 11: Rural Forest: These tests used the same route as test 7. Both devices were assigned the router role. This time the members walked in different directions. Due to an error one of the devices was powered off during some of the test, causing a gap in GPS data until the device was powered on again. Unlike test 9, the device quickly reestablished its connection to the mesh network and the rest of the test was performed without issues.

F. Session 5

The session was conducted in Trondheim city, and the goal was to test range and routing. Two members of the team participated, each with their own device, and the border router was placed at a static location. All tests used DR5.

1) Test 14: Range: This test started with both members walking together in the same direction, in an attempt to find the distance where they would lose connection to the mesh network. During the test both devices had the role of router. After walking approximately 850 meters, connection between the devices and the border router was lost. Both routers managed to reconnect to the network afterward. At this time, the border router experienced an unknown issue leading to corrupt data being sent to the server, so the test was concluded. It should be noted that this issue occurred only this once, so we were unable to further investigate this error.

2) Test 15: Range: This test took place with two devices in a car, where one had the role of router while the other was a leader. The goal was to test how far away the devices could be from the border router when buildings blocked the line of sight. The test started with a few smaller buildings where both devices managed to send data without problems. The range from the border router and size of buildings increased as the test went on, until connection was lost. This test was completed without any issues, and the range from the border router was quite good considering the amount of tall buildings blocking the signal path.

3) Test 16: Routing: This test took place in the same area where we tested routing during session 3. The test followed the same procedures as session 3, with both team members walking together for a while, and then one member remaining at a stationary position while the other walked further away. In contrast to test 8, this time both devices seemed to be able to transmit their data without any issues. The max range achieved with routing during this test was also much greater than in test 8. The device furthest away from the border router managed to reestablish its connection to the network after it was lost, and data was therefore also gathered on its way back to the border router. This is the longest and most stable test throughout all testing sessions.

VI. Analysis

The following sections present graphs, plots and images of data from our analysis of the PyMesh network. The graphs describe various relationships related to packet loss and accuracy.

A. Considerations

We used a collection of scripts that we developed to perform analysis on the data we gathered during testing. During the testing process we did not do logging on-device due to the increased workload and complexity this would cause, which meant that we lacked coordinate data for lost packets. In order to generate better distance metrics we used linear interpolation between the most recent data point before the loss, and the first one after. This allowed us to estimate the position of lost data packets. Because the devices are continuously moving, we divided the data up to 14 distance ranges to facilitate estimation of packet loss by distance. For the sizes of our data sets this number worked well for visualizing trends accurately without excessive variance. We used the haversine formula [20] to calculate distances between coordinates.

Another consideration was the use of Gaia coordinates to measure GPS accuracy of the system. The Gaia tracker did not log coordinate data frequently enough to precisely match every mesh network coordinate with a "ground truth" position. In order to work around this, we allowed a maximum of 10 seconds disparity between timestamps for most tests, and we found this acceptable due to the low distance we would usually move during this time. This allowed us to not lose too many data points while still ensuring that our analysis was valid within the restrictions of these considerations.

B. Environments

Figure 6 illustrates a clear difference between walking in a city with high buildings and limited signal strength compared to line of sight. Tall buildings influence the range and packet loss of a device. DR5 and DR6 differ about 100 to 150 meters in maximum range. DR5 has a significantly lower packet loss at distances under 700 meters.

Figure 7 shows how the environment and distance affects the packet loss of LoRa transmissions. The tests were conducted in the city (left) and the forest (right). Tests 1 and 9, as well as test 2 and 10, used the same routes, making it a better graph to compare between DR5 and DR6 in challenging environments. The spikes in this graph for forest test 1 (blue) and forest test 9 (green) have the same reduction at about a 125 meter distance, as well as a gradual increase towards 225 meters. Graphs 2 and 10 are less similar, but beyond 100 meters their trends are comparable. The graph indicates that DR5 could handle the challenging vegetation much better before a high packet loss occurred.

C. Routing

Our analysis showed interesting differences in potential max range and packet loss using multiple routing devices. The left graph in Figure 8 shows the packet loss differences between DR5 and DR6. The graph (left) shows the same spikes and dips where both tests were closing in on max range. The increase in packet loss is a result of the distance between the devices. The graph to the right shows all the moving Pycom devices participating in each test. We can see that instability of Pycom D at around 350 meters (Test 8, red line) amplified the resulting packet loss for Pycom B at around 800 meters (Test 8, green line). These spikes occurred at the same time, but at different distances.

Our DR5 tests do not display the same level of amplification as DR6 due to device instability. The increase in packet loss beyond 1000 meters for Pycom B (Test 16, blue line) is a result of the device being at a greater distance to Pycom D, than Pycom D is to the border router, Pycom C. See Table IV for accurate distances and Figure 11 for an illustration of the Pycom device positions for Test 16.

D. GPS accuracy

Figure 9 presents the difference in GPS accuracy between the Gaia GPS app and Pycom devices (note that three outliers are omitted from the graph in order to ensure readability). We see a clear trend in the increase of GPS accuracy over time for these tests, although the data set for this graph is small. For this test the Pycom devices are being transported in a vehicle that can travel long distances between each reported PyMesh location. This will influence the GPS accuracy.

Figure 10 shows the distribution and average GPS accuracy for all tests performed in urban and rural environments. The forest tests lasted shorter (up to about 20 minutes), but still had better accuracy overall. This is a result of the open environment making it possible to have more satellite fixes simultaneously. The city tests have an average accuracy of approximately 12 meters. Note that because this graph spans a duration of 50 minutes it does not show the initial increase in accuracy after booting up a device. The tests also had a 15 minute setup time to ensure higher accuracy.

VII. Results

The average packet loss with line of sight for DR5 was 35.1% (336/957) with Pycom D and B, and 32.25% (158/490) for Pycom B alone. This metric was calculated using all tests performed on DR5 (not including accuracy tests). The average packet loss for DR6 was 51.47% (263/511 packets). Due to the hardware instability of Pycom D and the lower amount of data points we consider this metric less reliable than that of DR5.

For short distances below 50 meters we observed a minimum packet loss of approximately 25% for DR5 and 30% for DR6 in forest environments. The packet loss of DR6 increased at a greater rate in shorter distances compared to DR5 in challenging environments. Our tests achieved a maximum range of 608.46 meters for DR6 and 880.88 meters for DR5. This effectively shows a 272 meter increase (44.7%) for DR5 with a direct line of sight, as shown in Table IV.

The PyMesh routing functionality creates an effective doubling in range (up to 1633.37 meters), as shown in Figure 11 and Table IV. Hierarchical routing in a PyMesh network makes it possible to have up to 2.437 square kilometer coverage per Pycom device in a flat environment, as shown in Figure 11.

The average accuracy was approximately 12 meters in urban environments and 10 meters in open forest environments.

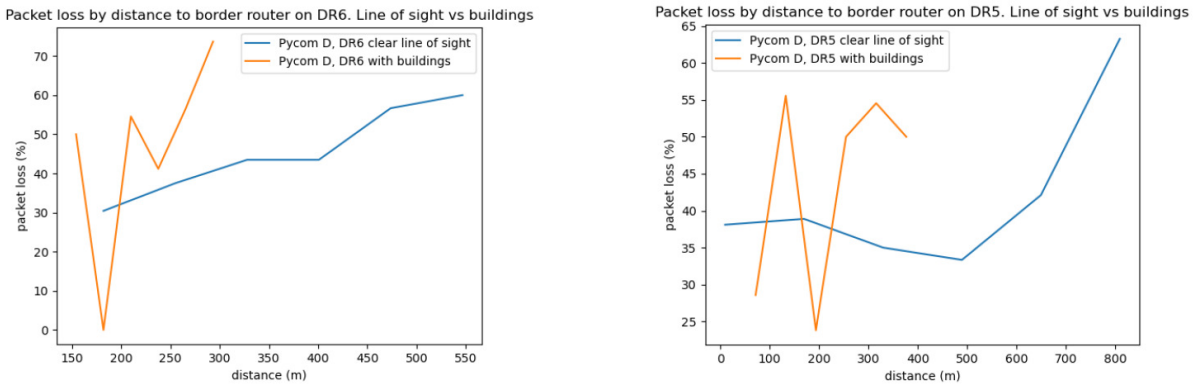

Fig. 6. Packet loss in different city environments on DR6 (left) and DR5 (right)

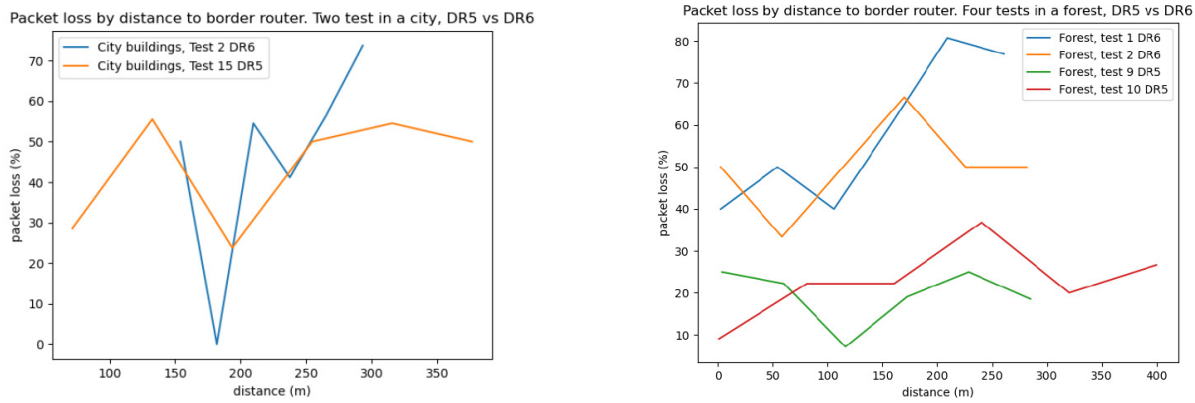

Fig. 7. Packet loss by distance for challenging urban (left) and rural (right) environments

There is a large amount of variance (Figure 10), and achieving a high accuracy usually takes around 15 minutes (Figure 9).

DR5 has provided better performance in terms of both packet loss and range compared to DR6 in our tests. The effect of different data rates on the stability of the PyMesh network itself (not the GPS tracking we perform), is difficult to conclude based on our testing due to hardware issues and amount of data gathered. We think that using DR5 shows promise, and would recommend further research with respect to its stability with a larger number of devices in a PyMesh network.

VIII. Related work

The NATO Research Task Group (RTG) IST-147 titled ”Military Application of Internet of Things” examined applying COTS civilian IoT approaches for military purposes. Typically, the use case was centered around a humanitarian assistance and disaster relief (HADR) coalition operation in a Smart City, where IoT information from the city could be used as additional sensor input to the military situational awareness and hence Command and Control (C2) systems [21]. Following that group’s conclusion in 2019, this work now continues in NATO RTG IST-176 titled ”Federated Interoperability of Military C2 and IoT Systems”. That group continues work on Smart Cities, but also broadens the scope to include such use cases as our recent work on crowdsourcing and crowdsensing [22].

For instance, Mekki et al. [6] have performed a comparison of Low Power Wide Area Networking technologies, including LoRa. They point to LoRa’s main strengths being battery lifetime, capacity, and cost. In their view, LoRa will serve as the lower-cost device, with very long range (high coverage), infrequent communication rate, and very long battery lifetime. Further, LoRa will also serve the local network deployment and the reliable communication when devices move at high speeds. They identify the following application areas as suitable for using LoRa: Smart farming, manufacturing automation, smart buildings, and tracking for logistics.

LoRa is a building block of LoRaWAN [10], which adds security and the means to organize a LoRa network with one or more gateways bridging LoRa to other networks, e.g., the Internet. The military application aspect of LoRaWAN has been investigated by Michaelis et al. [24], who used the USA version of LoRaWAN in the 915 MHz band to track vehicles in an urban environment (downtown Montreal, Canada). Their findings show a usable range of LoRaWAN of up to 5 Km under the conditions tested. In the case where buildings obstructed the line of sight, packet loss increased and the ef-

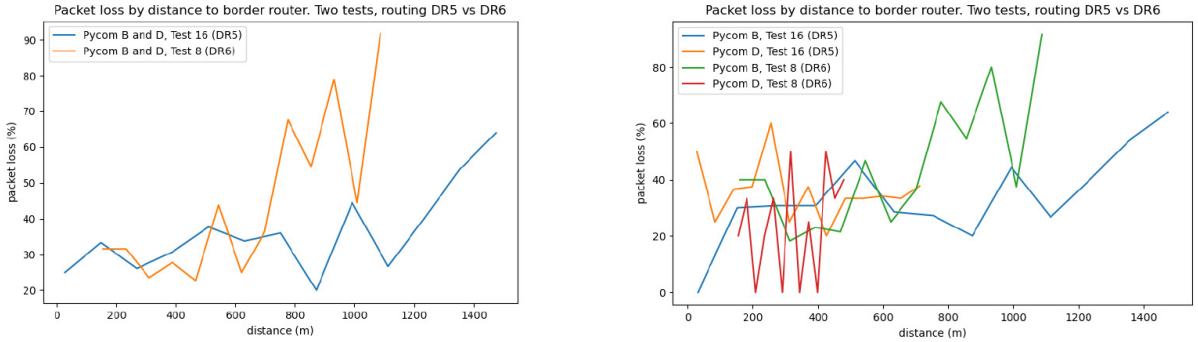

Fig. 8. Packet loss by distance routing DR5 vs DR 6 summary (left) and all devices (right)

TABLE IV
DISTANCES BETWEEN GPS POINTS (CALCULATED USING [23]). FROM LEFT TO RIGHT: PYCOM B TO D (DR5), PYCOM D TO C (BORDER ROUTER) (DR5), PYCOM B TO C (DR5), PYCOM D TO C (DR6, TEST 1).

	Pycom B to D	Pycom D to C	Pycom B to C	Pycom D to C
GPS 1 Latitude	63.43307	63.41855	63.41855	63.418652
GPS 1 Longitude	10.39128	10.39625	10.39625	10.396535
GPS 2 Latitude	63.42521	63.42521	63.43307	63.424006
GPS 2 Longitude	10.39349	10.39349	10.39128	10.394009
Distance apart	880.88 meters	753.18 meters	1633.37 meters	608.46 meters

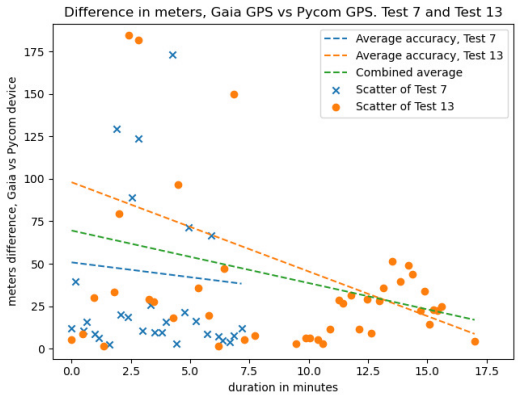

Fig. 9. GPS accuracy over time

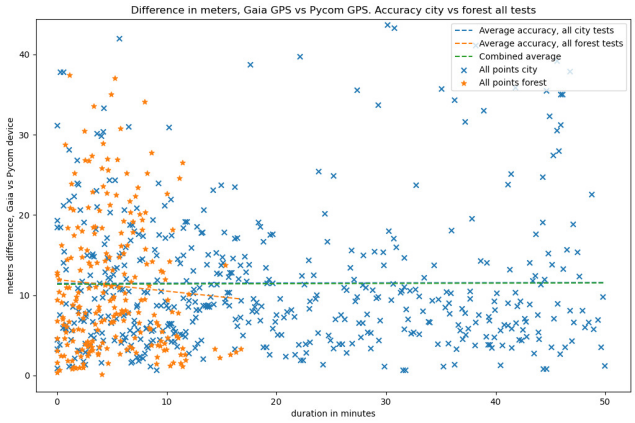

Fig. 10. Aggregated GPS accuracy over time

fective range was shorter, around 2.5 Km. However, to the best of our knowledge, this and other such studies (e.g., [25] which presents a similar experiment with comparable results) all rely on LoRaWAN. We are not aware of any prototype similar to ours as described in this paper, that performs tracking using commercial IoT devices over a LoRa mesh network.

IX. CONCLUSION AND FUTURE WORK

Our prototype uses PyMesh and LoRa to enable you to easily track the location of assets anywhere in the world, though under the limitation that one node, a border router, needs to be within range of a preexisting cellular network. The remaining nodes can operate without other coverage than the LoRa mesh network. Once the necessary software is installed the solution is easy to employ, even without technical knowledge. The prototype includes a modern web application that is intuitive and easy to use, and is supported on all major platforms including mobile. From our tests, the software performed as expected and could be a tool for asset tracking in search and rescue operations. Our findings show that though the Pycom units we used with PyMesh exhibited a shorter range than previous experiments conducted with LoRaWAN (see, e.g., [24], [25]), it should be noted that in our experiment all nodes were equally capable, whereas the LoRaWAN experiments had a dedicated, more capable gateway deployed. Still, coupling LoRa with PyMesh on the Pycom units effectively doubled the operating range, as we found, due to the multi-hop and routing capabilities of the mesh network.

For future work, it would be interesting to experiment with more units in a larger network. The scalability of the network was not specifically investigated by us so far, since we had a limited amount of nodes available. Since we found that of

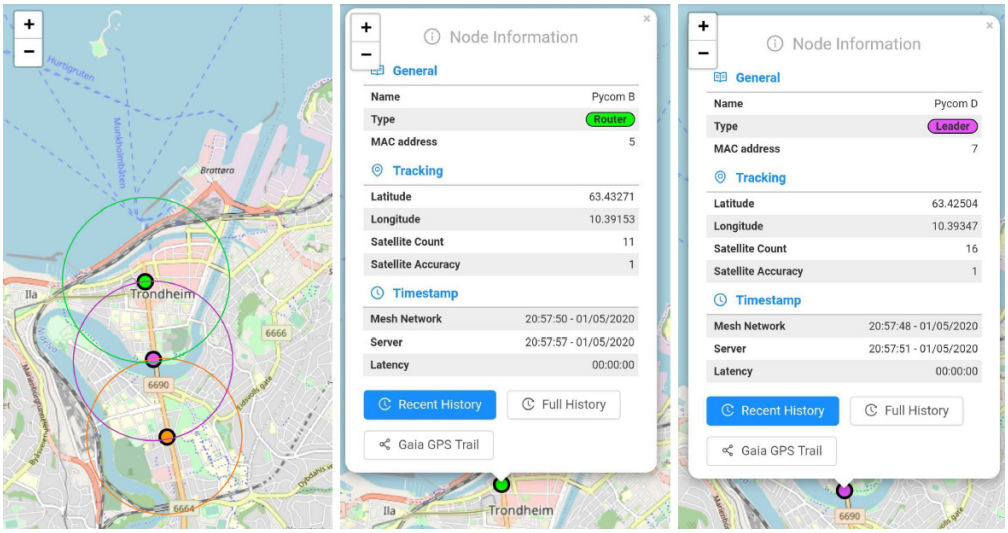

Fig. 11. Theoretical coverage with routing, in difference to coverage with no routing (only orange circle). Taken at approximately max distance (test 16).

the data rates we tested, DR5 has provided better performance in terms of both packet loss and range compared to DR6, we would like to perform more extensive tests with DR5. Also, it would be interesting to try adding additional features to the prototype, like short, pre-defined messages, in addition to the tracking we have implemented now. Such a messaging capability would nicely complement the tracking features, making the prototype much more versatile for asset tracking, search and rescue, and possibly HADR operations.

ACKNOWLEDGMENT

We would like to thank Torkjel Søndrol at FFI, Norway for sharing valuable insights on the Pycom IoT development platform. Finally, a thanks to Katerina Mangaroska at NTNU, Norway, for her efforts as supervisor for Engstad, Andersen, Røkenes and Blaalid when they worked on their Bachelor degree thesis [14], of which the prototype described in this paper is a major component.

REFERENCES

[1] A. Davis. Cellular baseband security. Georgia Institute of Technology, 2012. Available at https://smartech.gatech.edu/bitstream/handle/1853/43766/davis_andrew_t_201205_ro.pdf (2020/02/05).
[2] A. Turner. Tackling cellular vulnerabilities. Available at https://misti.com/infosec-insider/tackling-cellular-vulnerabilities (2020/02/04). MISTI.
[3] Pycom. Pycom company website. Available at https://pycom.io/ (2020/02/03).
[4] Semtech. Lora modulation basics. Available at https://web.archive.org/web/20190718200516/https://www.semtech.com/uploads/documents/an1200.22.pdf (2020/04/27).
[5] N. Suri, M. Tortonesi, J. Michaelis, P. Budulas, G. Benincasa, S. Russell, C. Stefanelli, and R. Winkler. Analyzing the applicability of internet of things to the battlefield environment, 2016 International Conference on Military Communications and Information Systems (ICMCIS), 23-24 May 2016, DOI: 10.1109/ICMCIS.2016.7496574, Brussels, Belgium
[6] K. Mekki, E. Bajica, F. Chaxel, and F. Meyer, A comparative study of LPWAN technologies for large-scale IoT deployment, ICT Express Volume 5, Issue 1, March 2019, Pages 1-7.
[7] Pycom. Lopy4 specsheets. Available at https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf (2020/03/19).
[8] Pycom. Pytrack specsheets. Available at https://docs.pycom.io/gitbook/assets/pytrack-specsheet-1.pdf (2020/03/19).
[9] Pycom. Pymesh documentation. Available at https://docs.pycom.io/pymesh/ (2020/03/19).
[10] L. Alliance. LoRaWAN specification. Available at https://lora-alliance.org/sites/default/files/2018-04/lorawantm_specification_-v1.1.pdf (2020/02/03). Section 2, LoRaWAN Regional Parameters.
[11] Semtech. Understanding lora adaptive data rate. Available at https://lora-developers.semtech.com/library/tech-papers-and-guides/understanding-adr/ (2020/04/27).
[12] OpenThread. Openthread opensource thread implementation. Available at https://openthread.io/ (2020/02/03).
[13] T. Group. What is thread. Available at https://www.threadgroup.org/What-is-Thread (2020/03/19).
[14] H. Engstad, E. Andersen, S. Røkenes and T. Blaalid. Long Range Mesh Networks. NTNU, Department of Computer Science, May 2020.
[15] Gaia. Gaia gps company site. Available at https://www.gaiagps.com/ (2020/04/29).
[16] Django. Django framework. Available at https://www.djangoproject.com (2020/02/04).
[17] React. React framework. Available at https://reactjs.org/ (2020/04/27).
[18] Internet Engineering Task Force (IETF), The GeoJSON Format, RFC 7946 Standards Track, August 2016.
[19] P. forum. wlan causes core panic. Available at https://forum.pycom.io/topic/5765/loadprohibited-core-panic-when-initialize-wlan (2020/04/29).
[20] pypi.org. Haversine formula library for python. Available at https://pypi.org/project/haversine/ (2020/04/29).
[21] F. T. Johnsen, Z. Zielinski, K. Wrona, N. Suri, C. Fuchs, M. Pradhan, J. Furtak, B. Vasilache, V. Pellegrini, M. Dyk, M. Marks, and M. Krzyszton, Application of IoT in Military Operations in a Smart City, 2018 International Conference on Military Communications and Information Systems (ICMCIS), Warsaw, Poland, 22 - 23 May 2018.
[22] M. Pradhan, F. T. Johnsen, M. Tortonesi, and S. Delaitre, Leveraging Crowdsourcing and Crowdsensing Data for HADR Operations in a Smart City Environment, IEEE Internet of Things Magazine, Volume: 2, Issue: 2 , June 2019.
[23] gps coordinates. Gps, distance calculator. Available at https://gps-coordinates.org/distance-between-coordinates.php (2020/04/29).
[24] J. Michaelis, A. Morelli, A. Raglin, D. James, and N. Suri. Leveraging LoRaWAN to Support IoBT in Urban Environments. 207-212. 10.1109/WF-IoT.2019.8767294.
[25] B. Jalaian, T. Gregory, N. Suri, S. Russell, L. Sadler, and M. Lee. Evaluating LoRaWAN-based IoT devices for the tactical military environment, 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), 5-8 Feb. 2018, DOI: 10.1109/WF-IoT.2018.8355225, Singapore, Singapore