Multi-Node Environmental Monitoring

W. Clayton Pannell
Dept. of Electrical and Computer Engineering
The *University of Alabama in Huntsville*
Huntsville, AL, USA
wcp0001@uah.edu

***Abstract*—This paper documents the creation of a performant, reliable Bluetooth Mesh network for the purpose of multi-node environmental monitoring of a large area. The mesh network is then coupled, via a Gateway Node, to a Gateway Server that aggregates and stores the sensor data in addition to making it available to other devices on a Wide Area Network. While environmental monitoring is useful this data can be seen as a stand-in for a wide variety of sensors and devices.**

***Keywords*—Bluetooth Mesh, IoT, Mesh Networking**

I. Introduction

Wireless Mesh technologies allow for the creation of networks that cover a large area where wired networking may be impractical, expensive, or unappealing. Often, these devices solely communicate amongst themselves, with little-to-no provision for logging of the generated data. The systems that can communicate with other devices often travel through the cloud to do so, bringing along data privacy and security concerns.

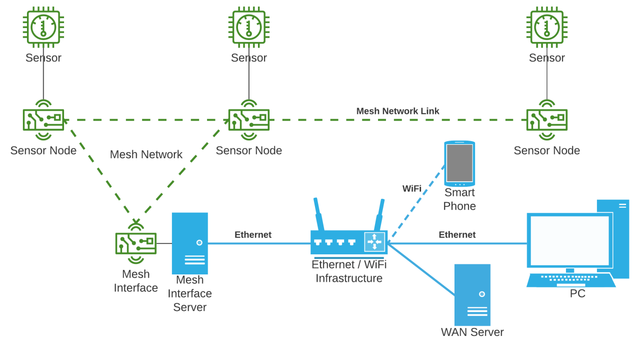

Figure 1: The Proposed Environmental Monitoring System

The system proposed uses the relatively new Bluetooth Mesh Profile to connect a series of Environmental Sensors, strewn across a large area with a mesh-to-http gateway. This gateway will be responsible for aggregating and logging the Temperature and Humidity data produced by these Sensor Nodes, providing a user-friendly Monitoring Interface, and enabling orchestration with other devices that may be present on other mesh networks or the local TCP/IP network (intranet).

The Sensor Nodes are based on the Particle Xenon development board, which was selected because it uses a ubiquitous-in-IoT Nordic NRF52 series System On Chip (SoC), has good support with Zephyr Real Time Operating System due it's feather form-factor, is equipped with a common JTAG style Single Wire Debug (SWD) port, has an onboard lithium ion battery charger with connector, and it was on-hand in the needed quantity. The Nordic nRF Connect SDK with Zephyr Operating System is used to provide the Bluetooth mesh support and hardware abstraction. The DHT20 I2C Temperature and Humidity sensor is used to gather the Environmental data. For the purpose of simplicity, all sensor nodes will Bluetooth Mesh Relay nodes and be mains powered via a 5v USB adapter.

The Gateway-to-Mesh interface node uses the same Particle Xenon development board and NCS/Zephyr operating system as the sensor nodes. The gateway node provides mesh event data to the gateway server over a USB emulated serial port, which will also power the gateway node.

The Gateway server is a Raspberry Pi 4 running the Home-Assistant Operating System. A Home-Assistant integration translates the serial data from the gateway node to update Home-Assistant Sensor Entities. Home-assistant stores the state values for each sensor entity at each change, displays the current state and recent history for user monitoring and provides a means to retrieve stored data for later processing. Home-assistant is privacy-aware and requires no access to the internet for its functionality and, if it were given access, only checks for automated updates, unless intentionally configured otherwise.

II. Market Survey

A. *MESH Project Temperature and Humidity Sensor [1]*

A $60 Bluetooth Mesh connected environmental temperature monitoring solution. This device is essentially an existing commercial solution to the sensor nodes proposed in this paper. The sensors can be bought individually or as a bundle with other nodes (e.g. Button, LED, GPIO, etc). While MESH provides an app that implements a visual orchestration programming language (i.e. pressing the Button node will turn on the LED node), they do not sell a commercially packaged persistent mesh to IP gateway solution. In order to connect to an orchestration / logging server this solution relies on either a phone or tablet dedicated to running their app or a Raspberry Pi3/4 running their packaged Linux distribution. In either case, remote orchestration and logging is provided by IFTTT (If this then that), a commercial cloud service.

B. *Blue PUCK T [2]*

A $47 small temperature sensor node that transmits data via Bluetooth 4.0 Low Energy (in beacon mode) and can be configured via RFID. There is a version that is compliant with EN12830 [3] temperature logging standards for temperature sensitive goods. This version constantly logs the temperature within the node and can transmit the logged data to authorized users. The company that makes the Blue PUCK T makes some Bluetooth Mesh devices and has some documentation referencing a Blue PUCK T MESH device, but it is not yet on the market.

C. *Sonoff Zigbee Sensor and Bridge [4]*

The $18 Sonoff Zigbee Temperature and Humidity sensor is similar the the MESH Temperature and Humidity sensor, but with a little less polish. The device communicates over Zigbee mesh networks. The optional $36 Sonoff Zigbee Bridge [5] acts as a persistent gateway connecting the Zigbee mesh to the user's choice of the eWeLink cloud orchestration service, or to a local orchestration service like Home Assistant.

III. Literature Survey

A literature survey was conducted in order to determine the state of the art, investigate alternative methods, and identify areas that may benefit from further research. The results of the literature survey are listed below. Each paper is presented with a brief summary, a description of the problem the paper attempts to solve, a summary of the paper's contributions, and an explanation of the relevance of the paper to the topic at hand.

A. *An Approach of BLE Mesh Network For Smart Home Application [6]*

Tran et al. presents a case study for a door lock that communicates with a home security system over Bluetooth Mesh in order to allow remote control and monitoring of the door lock through a cloud-hosted website.

1) *Problem*

Devices that communicate via Bluetooth and/or Bluetooth Low Energy are practically ubiquitous and are implemented by not only a wide plurality of very low cost microcontroller system on chips (SoC), but also by many devices the average person already owns. With the introduction of the Bluetooth Mesh protocol, a large-scale device network can operate on top of the low-power BLE protocol.

2) *Contributions*

A Bluetooth Mesh system is designed, built, and tested in a real-world environment. The system consists of the lock node, a remote unlock controller node, two relay nodes, and a provisioner node that also interfaces the mesh network to a cloud server, allowing for remote control and monitoring over the internet.

Four experimental scenarios were developed and tested: open area, one room, two rooms, and two floors. In the open area scenario, two nodes transmitted between each other at a distance of just under 30m without packet loss. The one room scenario takes place in a 30sq meter office environment with metal obstructions and radio interference (e.g. WiFi, Cellular, other Bluetooth devices). 5 Nodes were used for this test with the nodes placed at varying distances and heights. It was found that approximately 5-7m distance at 1.8m height difference between the nodes results in little-to-no drops. The two room experiment uses one node as the relay to communicate between the devices in each room. The relay node is located such that it is line-of-sight with both nodes and is 6.5 meters to the originating node and 7 meters to the end node. The relay to end node link dropped 18-20% of transmitted packets, whereas the slightly shorter start to relay link dropped no packets. The multi-floor tests took two forms, one where the gateway and relay were on the same floor, with the end node 1 floor higher (in a stairwell) and one where all three nodes were on different floors of the same stairwell. The two floor test saw a 2.1 and 5.9% packet drop rate with 6.5m and 3.5m between the start node and relay and the relay and end node The multi-floor test saw similar drop rates, except that the 2nd to 3rd floor nodes experienced a 17.3 % drop rate at only 3.5 meters between the nodes.

Tran et al. also studied the power consumption of their devices and attempted to perform optimizations based on that data. They found that switching the node into low energy mode netted a 35% reduction in power. They were unable to gain any further significant power savings by other means, including a wake/sleep power-saving algorithm. The power-saving algorithm maximized time in the lowest consuming sleep mode and woke the device to send and receive via Bluetooth. The authors blame component selection on the development boards used for consuming too much power regardless of radio state.

3) *Relevance*

The paper presents an application much like the intended application. While the some of the details differ, such as the use of cloud-based orchestration, the system architecture is very similar. The physical domain is also similar. The paper covers the issues with WiFi and physical object interference that would be present in the intended application.

B. *Bluetooth Mesh Analysis, Issues, and Challenges [7]*

This paper provides a detailed summary of Bluetooth Mesh features and problems concerning node types, responsibilities, overhead, and configuration parameters.

1) *Problem*

BLE is nearly ubiquitous. BLE is great for tracking the locations of nodes. Bluetooth mesh is built on top of BLE using it's advertising and scanning states and uses flooding as opposed to a routing protocol. Relay nodes and re-transmissions should be limited and finely tuned to control congestion. The paper focuses on describing and adjusting Bluetooth Mesh parameters and how they interact between the layers of the protocol, based on real-world testing, to achieve reliability, efficiency, low latency, and low packet losses.

2) *Contributions*

A significant contribution is a detailed summary of the Bluetooth Mesh protocol, network topology, and key features. Unlike [8], Á. Hernández-Solana et al. differentiates "low power" nodes (LPN), friend nodes, and relay nodes. The low power node example given is a temperature sensor which rarely needs to receive data from the network, much like the intended application. This is enabled by the friend node which is essentially a relay that stores messages for the LPN until it comes back online to request the messages. A proxy node relays messages to devices outside of the Mesh (e.g. a server). A provisioner node configures a device to join the mesh after the device sends beacon advertisements announcing itself. All nodes must pass encrypted and authenticated messages with separate keys for network, application, and device security. The Advertising Event can transmit up to 47 bytes, but after the protocol overhead through the various layers, only 10 of these bytes are available to the user.

Another contribution is the enumeration of discrepancies between the Bluetooth Mesh specification and the realities of real-world devices and use-cases. The Bluetooth Mesh specification expects the devices to be scanning or advertising with as close to 100% up-time as possible, but due to channel hopping, packet processing, stack processing, and sending connectable advertising packets there are "blind times" that reduce this up-time. Buffering can cause latency and undersized buffers can cause dropped packets or cache flushing (causing unnecessary re-transmission of packets). Repetition of packets provide more reliability at the cost of higher latency , more network congestion, and lower throughput. Inappropriate random delays can result in timing issues that actually increase the frequency of collisions and latency when performing re-transmissions. The random delays for acknowledgment messages may incur a similar fate if not carefully bounded. Likewise, choosing an appropriate time to live value requires some adjustment to prevent data storms and undelivered messages. The high up-time requirements for relaying devices also incurs a power consumption cost. Over-provisioning relays can significantly increase the energy consumption of the network as a whole (due to high radio up-time) while decreasing network reliability due to higher congestion and more frequent re-transmissions.

The paper also identifies areas where further research is needed to improve Bluetooth Mesh Performance and/or power usage. Self-tuning of network parameters (e.g. TTL, re-transmissions, delays) would make mesh networks easier to set-up, reduce fragility to changes (as simple as moving a node), and improve performance of poorly optimized nodes. Combining of copies of the same message received can reduce needless re-transmission. More and/or different bearers can allow for greater byte efficiency and larger payloads (via extended advertising PDUs) or packet chains. The current requirement for high up-time on relay nodes makes it unreasonable to use battery powered relays. The 5ma power consumption figure cited in the paper sounds quite small, but it would drain a 10,000mAh battery pack in under 3 months.

3) *Relevance*

Á. Hernández-Solana et al. Provides a clear summary of the Bluetooth Mesh protocol, warts included. Attention is drawn to some of the configuration pitfalls that lie in wait due-in-part to the wide adjustment range available on some of these parameters. Importantly, some "reality-checks" are dished out that could prevent developers intending to use the protocol from being caught unaware. Particularly shocking is that only 11 bytes of application data can be sent per packet, but that doesn't particularly detract from the intended application as sending environmental data. Another pain point is the high power consumption caused by the apparent need for high on-time for relay nodes, which would make battery-powered applications difficult.

C. *Features of Building MESH Networks Based on Bluetooth Low Energy 5.1 Technology [9]*

This paper considers the Bluetooth standard specification version 5.1, evaluating its features, and (dis)advantages compared to 802.15.4 protocols (Zigbee and Thread).

1) *Problem*

Evaluate performance in a multi-story building made of reinforced concrete. Explore possibilities of building BLE mesh networks using modern microcontrollers. Discuss the new locating features released in Bluetooth 5.1.

2) *Contributions*

Zyulin et al. describes the method by which the Bluetooth 5.1 specification determines the location of the nodes, down to an accuracy of several centimeters. Multiple antennas are required at one node and either Angle of Arrival (multiple antennas at transmitter) or Angle of Departure (multiple antennas at receiver) can be used. The data received from the antenna array is used to determine direction which is combined with the range in order to solve for position.

Bluetooth mesh is compared to the Zigbee, which offers better data transfer security at low speed and range, and Thread protocols, which offers high speeds at short ranges. The paper cites data collected by Silicon Labs [10] which directly compared the three protocols. In this test it was found that, although the data transfer rate of Bluetooth mesh is low, it is almost completely unaffected by the number of hops (relay node receives and re-transmits), unlike Thread and Zigbee, which both decrease exponentially with each additional hop. Data transmission distance is similar between the mesh protocols with roughly a 10% increase from Zigbee to thread and thread to BLE. There is a BLE long range protocol which transmits 2-3 times further than the mesh protocols, but it is not currently implemented in the Bluetooth Mesh protocol.

After this comparison, the authors focus on the performance of the Bluetooth Mesh network. A sample application was created that continuously transmitted the Morse code for S-O-S from the start node, over the mesh, which is received by the end node and displayed on a LED. This application is used to benchmark the signal quality in various connection scenarios throughout the multi-story building. A summary of the error thresholds follows: no errors were found when transmitting down a 40m hallway, errors were found when transmitting across 5 floors (one relay hop per floor), Connection dropped out entirely across 3 floors (no errors on 2 floors), errors were found when transmitting across 5 flights of stairs (in a stairwell), and errors were found when transmitting through 3 classroom walls. The floors were 150-180mm reinforced concrete and the walls were 100mm thick.

3) *Relevance*

Of the papers considered, this covers the nearest to current release version of the Bluetooth Specification (v5.2) and provides a good comparison between the three most common mesh protocols (Bluetooth Mesh and 802.15.4-based Zigbee and Thread). The performance of the Bluetooth Mesh network appears to have much longer range than shown in [6], but this could be attributed to the different Bluetooth interfaces used. The difference may also be due to the method of determining connection quality, where Zyulin et al. used a qualitative approach, that, depending on application design (not discussed in the paper), could be much more tolerant to packet loss than the packet counting method used in Tran et al.

D. *IEEE 802.15.4 Thread Mesh Network – Data Transmission in Harsh Environment [11]*

This paper presents a similar premise to Tran et al. [6], but implements the OpenThread 802.15.4 protocol stack on the NRF52840 SoC instead of Bluetooth Mesh. A mesh connected temperature sensor and raspberry pi based gateway are given as an example application.

1) *Problem*

IPv6 is implemented on top of OpenThread with the 6LoWPAN protocol, enabling internet protocol (IP) v6 to be transmitted over the mesh network via a 6LoWPAN border router. Going further, the Constrained Application (COAP) protocol is used to provide more reliable communication.

The paper also provides a concise description of the Thread network protocol and topology. An example hardware and software solution is developed, tested, and analyzed.

2) *Contributions*

The Thread network topology is very similar to the Bluetooth Mesh network. Packets being transmitted to/from End Devices (which can operate in a low power mode if connected to only one router) are received and re-transmitted by router nodes (equivalent to Bluetooth Mesh relay nodes). Packets are encrypted in transit. With the implementation of 6LoWPAN Thread nodes can communicate directly via IPv6 with external servers on an IP network.

The example solution is an indoor temperature environmental condition monitor that transmits a 128kB packet every minute to a server. The server logs the data and provides a dashboard display of the data, served over HTTP via LAN. The sensor packets are time-stamped so that they can be sent inconsistently (non-real-time). The Border Router server acts as the gateway connecting the Thread mesh network to the IP network and runs the OpenWRT Linux distribution for the operating system. The Thread network interface is a NRF52840 development board running the OpenThread protocol implementation which is connected to the border router via USB. The border router runs a COAP server to provide reliable UDP protocol messaging on top of the mesh IPv6 protocol. COAP adds reliability by requiring message acknowledgment and providing timeouts and re-transmission when a packet is determined to be lost. The sensor devices use the same NRF52840 SoC as the Border Router on a custom PCB with integrated environmental sensors, PCB antenna, and coin cell battery holder. The device can determine whether it is powered from battery or the USB connector. When powered via USB the device configures itself as a Router Device. When powered via battery the device configures itself as a low power Sleepy End Device. The end devices implement a state machine with RTC timer and radio transceiver event interrupts in order to maximize battery life by maximizing sleep time. Once a minute the device wakes up, takes a sensor measurement, packages the measurement data as json data in a COAP packet, and sends it to the logging server over the mesh network. The COAP implementation handles ACK receipt and re-transmission as required.

Testing of the solution was performed in a 5 story research laboratory with brick walls, concrete ceilings/floors, and a large amount equipment producing both RF signals and noise. In order to provide reliable, redundant radio links (at least 3 links per device), four to five Thread router nodes are deployed on each floor, preferring open areas and hallways for best results. The sensor nodes are deployed in random places throughout the building. Testing was done with and without the COAP protocol enabled. When COAP was not used, messages were transported using simple UDP packets (no acknowledgment and re-transmission). In this configuration, packet loss rate was approximately 12% without COAP, and effectively 0% with COAP. Naturally, the 12% loss remains, but re-transmissions ensure that data eventually reaches the desired location. Brief network outages were noticed while the mesh was rearranging, but once the mesh was re-established the affected packets were successfully re-transmitted.

3) *Relevance*

Much like the intended application of this paper, the example application used in Rzepecki et al. is a mesh connected temperature sensor and a Raspberry Pi acting as a gateway connecting the mesh to an IP network. The network topology used is by Thread is very similar to Bluetooth mesh. Also, the IETF is working on a draft standard similar to 6LoWPAN to apply the IPv6 protocol on top of Bluetooth Mesh.

E. *Comparison of the Device Lifetime in Wireless Networks for the Internet of Things [12]*

This paper compares the lifetime (energy consumption) of several prominent wireless networking technologies used in IoT. The lifetime was based on the the platform being powered by a pair of AAA batteries (13.5kJ) All technologies use the 6LowPAN protocol (or the closest approximation available) to keep protocol overhead as similar as possible amongst the technologies. The energy consumption figures account for re-transmission and inactive consumption, thus potentially allowing for an energy intensive technology to still be competitive if it has the bandwidth to transmit more data less frequently.

1) *Problem*

Many IoT devices, like the intended application rely on battery power, and therefor rely on very low power wireless networking technologies to maximize battery life. 802.15.4 networking has long been considered a standard for very low powered devices, but new technologies challenge that status quo. This paper compares estimated battery lifetimes among 802.15.4, 802.11b Power Saving Mode, BLE, 802.11ah, LoRa, and SIGFOX.

2) *Contributions*

The analysis found that BLE platforms had a much higher lifetime than other technologies when transmission was more frequent (1 second Application Period). As transmissions become less and less frequent, sleep-mode power consumption becomes more important than TX/RX power consumption. When a 100 second Application Period was used, the performance was more similar among BLE, 802.15.44, and some 802.11 platforms. In those cases, most BLE implementations outperformed 802.15.4, and most 802.11 implementations outperformed the worst 802.15.4 (TelosB). SIGFOX and LoRa (EU variant) had the worst lifetime, by far and are only competitive when transmitting less than 500 bytes once per day. In all scenarios tested, BLE offered the highest lifetime. The higher bitrate of BLE is what allows it to outperform 802.15.4. This was validated by testing the various bit-rates available to BLE and 802.15.4. It was found that 250kbps 802.15.4 has an equivalent lifetime to BLE at 500kbps at low packet sizes and BLE at 125kbps at higher packet sizes. When 802.15.4 was tweaked to 2Mbps the same was seen when compared to BLE 2Mbps and 1Mbps.

The take-away of this paper is that in order to maximize the lifetime of an IoT device, the device should transmit as little data, as infrequently, and as fast as possible in order to maximize sleep time. Additionally, the next most important finding was that BLE was the best technology tested, in terms of energy consumption, when sending small data packets at medium and high data rates. Neither of these points change when 20% packet losses (and subsequently, re-transmissions) are introduced. In other words, the lifetime *does* go down, as expected, but the ranking of the technologies does not change.

3) *Relevance*

Given that the intended application is to transmit temperature data across a multi-hop network, this paper confirms that BLE and 802.15.4 would be the most appropriate technologies to investigate. Ambient air temperature and humidity can only change so fast. Transmitting data more often than once a minute is unlikely. Likewise, the payload transmitted from the sensor nodes is expected to be only a few bytes. This should result in a multi-year lifetime.

Morin et al also provides a concise summary of the wireless technologies investigated, focusing on their use in a low energy, battery operated node that transmits infrequently. Beacon-enabled 802.15.4 enables low power multi-hop meshed networking. The coordinator / relay nodes send a beacon requesting connector nodes to send data. This method handles the synchronization issue brought up in [11] at a much lower energy cost than constant radio operation. Also, Bluetooth Low Energy specification, as of version 5.0, does not support a true mesh, but instead a scatternet, which is similar, but more like a collection of trees than an interwoven mesh. This is a distinction that I had not seen made, yet and may make mesh configuration more challenging. This may push 802.15.4 as a preferred solution over the somewhat-longer-lived BLE option.

F. *Understanding the Performance of Bluetooth Mesh: Reliability, Delay, and Scalability Analysis [8]*

This paper evaluates the performance and scalability of the Bluetooth mesh protocol. The system is evaluated through extensive simulations, including a model of a real-world office environment with WiFi interference mapped from collected data. The bulk of this paper is concerned with the tuning of protocol parameters for minimal packet loss and corruption, latency, and network congestion.

1) *Problem*

Previous Bluetooth specifications did not support a mesh topology for BLE, opting instead for a star topology, or, at best, a scatternet as discussed in [12]. A variety of proprietary and research mesh schemes have been attempted to rectify this, but all lacked standardization and interoperability required to bring wide adoption. This has been corrected with the 2017 release of the Bluetooth Mesh specifications. Although it is based on the (much older) BLE protocol stack, the mesh network is different enough that it needs it's own performance studies. Because the protocol is so new, there are very few studies that analyze BLE mesh network performance with respect to latency, quality of service, tuning, and scalability.

2) *Contributions*

A significantly useful contribute of the paper was a concise summary of the transmission/receive model of Bluetooth Mesh. Bluetooth Mesh is designed without the need to establish connections among devices in the network by transmitting via the advertising (ADV) / scanning scheme of BLE. The transmitting (ADV) devices send the data packet on each of the 3 channels with a time interval between each transmission denoted as TinterPDU. During the ADV The transmission model is known as managed flooding, where all nodes receive all messages from other nodes in direct radio range. Relay nodes will rebroadcast the message to reach more distant nodes. Time to live (TTL) is implemented to prevent packet storms. This allows messages to reach their destination through multiple network paths, at the cost of increasing collision probability.

Randomization of the advertising event timing (TinterPDU) and idle timing (back-off) can help prevent, or at least recover from, network collisions and prevent continuous masking. Masking is where two nodes' advertising and scanning times never overlap, or do overlap but never on the same channel. In simulation, this randomization was not found to negatively affect the network. However, re-transmitting PDUs can increase packet latency and network congestion. A single repetition was found to be a good trade-off with a 25% improvement in packet loss rate at the cost of a 30ms delay, end to end.

Decreasing advertising event duration (via shorter TinterPDU values) was found to be beneficial with respect to latency and network congestion, but is more susceptible to radio noise. The shorter event makes it more likely that all three PDUs overlap the noise event. Likewise, a longer scan interval is more likely to receive a matching PDU or a noise event. Short TinterPDUs (1-2ms) matched with minimal ScanIntervals (10ms) achieves a 100% end-to-end packet success rate when noise events are not modeled.

In order to understand the behavior of the effects of WiFi on Bluetooth Mesh, a mesh topology was designed to fit within a real office space and tested against WiFi interference map data collected from that office space. Naturally, during times of greatest WiFi traffic Bluetooth Mesh end-to-end packet loss was highest and lowest during times of lowest WiFi Traffic. There was also a clear effect on the node-to-node packed success rate due to the presence of WiFi Traffic, with significant reductions for the nodes with the the most interference. There are newer, secondary ADV channels that became part of the Bluetooth 5.0 core specification, but it was not available in time for publication. The additional channels would decrease channel interference by allowing the network designer to move the channels "further away" from the WiFi channels.

3) *Relevance*

Based on the findings of [6], [12] 802.15.4 and traditional BLE have comparable range in an obstructed, noisy environment while BLE has a power advantage due to the ability. The parameter tuning discussed in this paper, when combined with the "minimal on-time" approach should improve the performance and power consumption of the much more active relay nodes used in a mesh topology. This gives some credence to the feasibility of battery powered relay nodes.

IV. Bluetooth MESH

Bluetooth MESH is a subset of the the Bluetooth standard, first introduced in version X.X. The mesh network messages are built on top of Bluetooth Low Energy (BLE) Advertising bearer (ADV). However, due to the overhead of the mesh network only a maximum of 11 8-bit bytes (out of the maximum of 265 bytes in a link layer packet) can be transmitted in a single, unsegmented payload.

A. *Bluetooth Mesh Node Roles*

1) *Relay Node*

Relay nodes can be thought of as the main trunk and branches of the mesh network. These devices consume a relatively high amount of power because the radio is continuously receiving and transmitting (the NRF52840 specifies roughly 6mA of power during transmit and receive [13]). Each message that the Relay node receives, that is not stored in its recent message cache and has a Time-to-Live (TTL) value greater than 1, is retransmitted to be received by all other nodes within range. See Mesh Sensor Node 1 in Figure 2 for an example of this retransmission.

2) *Friend Node*

A Friend Node stores messages Addressed to one of its registered Low Power Nodes. Once the Low Power Node comes back online, the Friend Node will forward those messages to the low power node. A Friend node is usually also a Relay node since the friend node needs to leave its radio powered on to receive mesh messages in addition to the periodic friendship messages from its low power nodes.

3) *Low Power Node*

A lower power node (LPN) keeps its radio off for the majority of the time to save power. The low power node periodically transmits to and receives from its friend node. In order to facilitate this, whenever the low power node does transmit, it makes an appointment to receive from the Friend node. This appointment consists of a delay and a window. In simpler terms, the LPN tells it's friend that in (for example) 100 milliseconds it will listen for 20 milliseconds and then go back to sleep. After at least 100 milliseconds the Friend node will transmit every message it has queued up for the LPN to the LPN. This friendship allows battery operated devices to join the mesh network without expending their power budget on constant scanning and retransmissions required by Relay nodes. Because of this, Low Power Nodes are leaf nodes in the Bluetooth Mesh tree. They are only able to talk to the node it is friends with. An LPN is only allowed to have one friend. It must cancel its old friendship before it can start a new friendship.

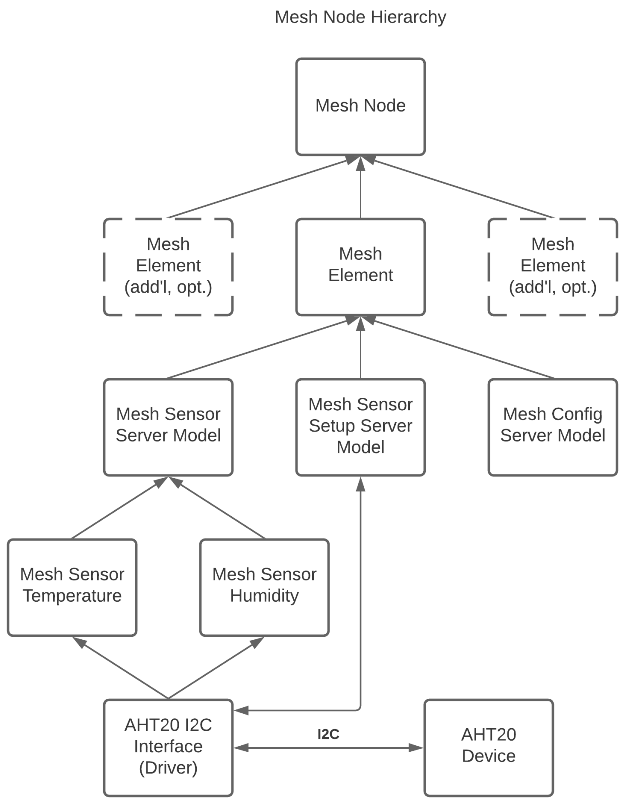

Figure 2: Mesh Node Data Model

4) *Proxy Node*

Proxy Nodes transmit and receive mesh messages to/from devices that do not support the Advertising bearer, but do support the BLE point-to-point GATT bearer. The Proxy node takes the role of the GATT Client. The Gatt server provides the Mesh Proxy Data In (UUID 0x2ADD) and Mesh Proxy Data Out (UUID 0x2ADE) characteristics in addition to the Client Characteristic Configuration descriptor (UUID 0x2902) [14]. The Proxy node (GATT Client) then enables notifications for the Mesh Proxy Data Out characteristic. Messages from the Mesh network are then written (without response) to the Mesh Proxy Data In characteristic whereas messages from the BLE device are read from the Mesh Proxy Data Out Characteristic by the Proxy node as demanded by the notification. It is worth re-emphasizing that this proxy feature transports mesh messages. The BLE-GATT-only device will still need to be able to consume and produce mesh messages in order to communicate.

B. *Bluetooth Mesh Data Model*

The Bluetooth mesh data model follows an object-oriented design pattern [15]. A Node has a set of Elements, each Element has a set of Server and/or Client models, and each Client or Server model has a set of states and properties.

Bluetooth Mesh Properties are instances of characteristics which are identified by Unversally Unique Identifiers (UUID) [15]. These characteristic definitions are shared with the BLE-GATT profile for which the Bluetooth Special Interest Group (SIG) specifies their data structure representations [16]. The UUID of each Mesh Property [17] is specified by The Mesh Working Group, which is also part of the SIG, but the property definitions are not shared with any other part of the larger Bluetooth Specification. There are currently 182 usable properties defined (0x00 is defined as prohibited) to cover a wide variety of data types and use cases.

Bluetooth Mesh Models can be defined by either the SIG or a by a registered vendor. SIG models are identified by a 16bit UUID whereas vendor models are defined by a 32bit UUID consisting of a 16bit company identifier and a 16 bit vendor specific model identifier [18]. Only the SIG models are considered for the purposes of this paper because the Vendor model specifications are vendor specific and are not intended to be used by other parties (unless published by the vendor or reverse engineered). There are two types of SIG models Foundation models and SIG adopted models. There are only 4 SIG Foundation models: Configuration Server and Client, and Health Server and Client. The Configuration server/client pair are used to store and change (respectively) mesh communication settings for each node, such as whether the node is configured as Relay node, how many retransmits the node sends for each message, the keys for the network and applications, heartbeat, and so on. The Health server/client pair deal with faults, the health period, and the attention state (which causes the device to do something to identify itself in the physical domain, like blink LEDs or beep). Each node on the Bluetooth Mesh network must have a Configuration Server and a Health Server. The SIG adopted models are defined in [19], which specifies 52 server and client models. These models are classified into 4 model groups: Generic (on/off, power status, battery status, etc), Sensors, Time and Scenes (time, scheduling, preset states, etc) , and Lighting (lightness, hue, saturation, etc).

A mesh node has at least one element and the number and composition of the elements in a node is static. If the elements need to change (e.g. for a firmware update), the node must be reprovisioned [18]. The elements may only contain one instance of a model. A rectangular box fan with 2 independent fan motors would have 2 elements, each containing a Generic On/Off Server Model that controls a fan. One of the elements would be the primary element, which would contain the Foundation models. Each element is given an identifying unicast address when provisioned, starting from the primary element if more than one element is present. This element address is how mesh messages address their intended target. There are also virtual address groups which can target multiple elements. Keeping the dual fan example, a virtual address could be used to turn both motors on with only one message.

C. *Bluetooth Mesh Dataflow*

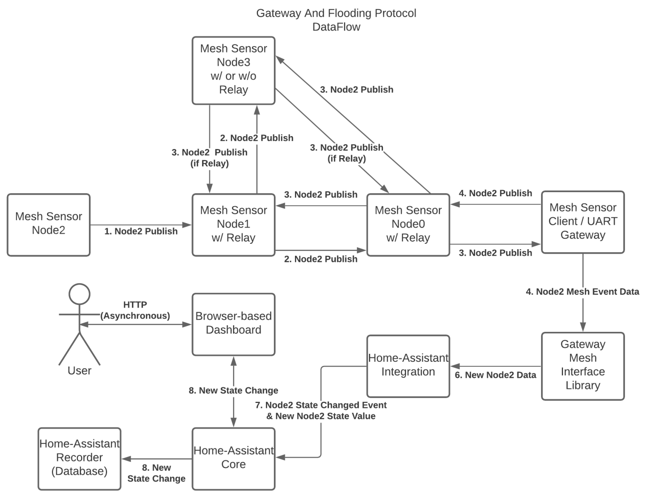
Gateway And Flooding Protocol DataFlow
Mesh Sensor Node3 w/ or w/o Relay
3. Node2 Publish
2. Node2 Publish
3. Node2 Publish (if Relay)
3. Node2 Publish (if Relay)
Mesh Sensor Node2
1. Node2 Publish
Mesh Sensor Node1 w/ Relay
3. Node2 Publish
2. Node2 Publish
Mesh Sensor Node0 w/ Relay
4. Node2 Publish
3. Node2 Publish
Mesh Sensor Client / UART Gateway
4. Node2 Mesh Event Data
HTTP (Asynchronous)
User
Browser-based Dashboard
Gateway Mesh Interface Library
6. New Node2 Data
Home-Assistant Integration
8. New State Change
7. Node2 State Changed Event & New Node2 State Value
Home-Assistant Recorder (Database)
8. New State Change
Home-Assistant Core

Figure 3: Data Flow of Sensor Node Publication Event in Meshed Environmental Monitoring System

Unlike other meshed networks Bluetooth Mesh network eschews routing protocols in favor of a lighter weight, but less deterministic "flooding" dataflow. Each relay node transmits not only its own mesh messages, but also any mesh message it has received that was not already in its recent network messages cache. See Figure 2 for an example of how a packet flows across the mesh network.

D. *Provisioning*

For the node-side of provisioning, the NCS provides a reference implementation of a Zephyr provisioning process handler, intended for use on Nordic development kits. The Provisioning handler implements four of Out of Bounds authentication methods. The Display Number and Display String are not likely to be suitable for a user-facing device because they print a secret number or string, respectively, to the Application Console serial port, which is typically used for diagnostics during development of the Zephyr application. The blink output and push-button input OOB provisioning methods are much more reasonable to for a user to interact with. The blink output method blinks an on-board LED 1-10 times where the number of blinks is the OOB code that is entered into the provisioning application. The push-button input method requires the provisioner to push a button on the node being provisioned a number of times (1-10) equal to the OOB code generated by the provisioner. Both methods were used successfully on the Particle Xenon development kit, so there is hope that other nRF52-based boards that implement the feather footprint may also use the NCS reference provisioning handler without significant modification.

A series of security vulnerabilities were disclosed in the Out-of-band (OOB) secret provisioning process on May 24, 2021 [20]. The only provisioning method that is unaffected by these vulnerabilities is the use of an OOB value with OOB transfer of public keys [21]. This involves storing a randomly generated OOB value and public key pair in the unprovisioned node's non-volatile memory before release to a customer (like a UUID or randomized serial number assigned while programming each node's microcontroller). The unprovisioned node will need to provide a means for the provisioner to learn the provisionee's public key and static OOB value, such as a QR code printed on the device's packaging. During provisioning the the provisioner will use the provisionee's public key to start the key exchange at the beginning of the provisioning process and provide the provisionee with the encrypted OOB value. The provisionee will only accept the provisioning attempt if the decrypted OOB value matches the value stored in the provisionee's non-volatile memory. Unfortunately this vulnerability was not noticed in time to be included in the system described in this paper. Future efforts should follow the security recommendations of the Zephyr Project.

V. Nordic nRF Connect SDK & Zephyr OS

The Nordic nRF Connect SDK (NCS) is based on the Zephyr Operating System. The Zephyr operating system is a light weight OS targeting resource-constrained microcontrollers. The Zephyr Kernel offers threading in various flavors (including POSIX pthreads), interrupt service routines, synchronization services, message/data passing services, and power management services. On top of this, Zephyr uses a Linux-kernel-like devicetree to describe hardware, hardware abstracted drivers, and of course, what pushes many to use an RTOS, a robust networking stack. The NCS builds on top of this with additional driver support, utilities, libraries, and examples. The NCS documentation includes not only documentation for the SDK, but also the documentation for the linked versions of Zephyr and other Nordic Libraries in one convenient place [22].

A. *Zephyr Bluetooth Stack*

Zephyr's Bluetooth stack consists of 3 layers: host, controller, hardware interface. Within a mesh The NCS provides an alternative controller layer, the SoftDevice Controller, that interfaces with its own hardware interface libraries. Although the NCS SoftDevice Controller features a more complete implementation of the Bluetooth 5.2 specification, the NCS documentation recommends using the open-source Zephyr Bluetooth LE Controller for mesh applications. The Host layer is not used within the mesh network. This recommendation was followed for the system implemented in this paper.

B. *Bluetooth Mesh Profile*

The Zephyr OS Bluetooth Mesh Profile implementation closely follows Bluetooth Mesh data model. For the Sensor Nodes implemented in this paper the current value of the temperature sensor is periodically sampled from the sensor by the sensor driver. Once sampling is completed the new value is then converted to the Temperature characteristic representation and used to update the Precise Present Ambient Temperature Property (UUID 0x0075). The Property is a member of the

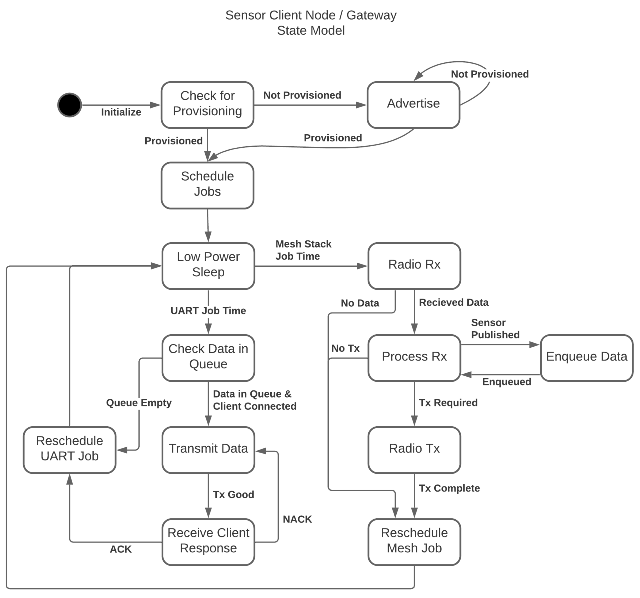

Figure 4: State Model of Gateway Node

Sensor Server Model (UUID 0x1100) and the Sensor Server Model is a member of the primary (and only) element on the Node. Likewise, the Humidity value is translated to the Humidity Characteristic representation, which updates the Present Ambient Relative Humidity Property (UUID 0x0076), which is also a member of the Sensor Server Model. Because these two properties represent the same physical sensor, or from another point of view, are sensors that measure different properties of the same physical object, they can be part of the same Sensor Server Model under the single primary element. If the Sensor Node's sensors measured two locations, then it would need two elements each containing a Sensor Server Model (a server model for each sensor, one instance of a sensor per element).

C. *USB Device Stack*

Zephyr OS has a hardware independent USB that supports a variety of Communications Device Classes and Human Interface Device classes (HID). The interface for the USB CDC ACM implements the UART driver API. This allows using the on-chip usb peripheral to act as a serial interface with almost no configuration. The major constraints to using the USB stack as a UART are that, prior to USB stack initialization, any data written to the UART is lost, and the USB device must be connected to a host that has opened the USB CDC ACM "serial port". Attempt to write to or read from the USB uart in the zephyr application, in my testing, resulted in instability and resets. The USB UART implements a virtual DTR signal that is set once the connection is made. Poll this line before attempting to write the first byte to UART.

D. *Threading*

Workqueues use a dedicated thread to pull jobs from a FIFO queue, perform their work, and yield to any other threads before grabbing a new job from the queue. The Work queue sleeps when the queue is empty. The Sensor and Gateway nodes developed for this paper rely on scheduling delayable work items being processed by the Workqueue thread for their non-mesh-stack functionality. A Delayable work item is similar to the standard Workqueue work item, but is held by the kernel for a time specified when scheduling the work item. After that time has expired the kernel then adds the delayable work item to the workqueue. The Sensor nodes use the workqueue to schedule the Temperature and Humidity measurements every 30 seconds. The Gateway node schedule a job every 100 milliseconds to check if there's a message waiting to be put on the USB UART from the mesh event message queue.

E. *Hardware Abstraction*

Zephyr uses a devicetree and the Linux kconfig configuration system to configure the drivers, pin definitions, and hardware peripheral at compile time allowing for greater abstraction. For example, an application using an I2C peripheral is defined to use pin X and Y on one microcontroller board. To port that application to a different microcontroller on a different board, the device tree is modified to set the I2C device to the new locations and be driven by a different I2C hardware abstraction driver. If the new board has multiple I2C ports, then the unused ports can be disabled in the kconfig. It is worth pointing out that device tree can only configure supported boards. If a custom board is developed, that board needs to be ported in to zephyr. These board support files can be part of the application repository and do not require forking zephyr to add support for proprietary boards.

F. *The West Meta-Build System*

West is a python application developed to manage and automate Zephyr's Cmake build system. West can be used to automate devicetree overlay application for a particular board. This allows the example applications that were developed for Nordic development kits to work on third-party development kits like the feather-compatible Particle Xenon boards used for this paper by only changing the board parameter provided to west. West can also automate running the cmake build, flashing the new build onto the microcontroller, and attach a gdb session to the newly programmed microcontroller. West will also mange the libraries used in the project, including pulling in Zephyr and the NCS. Libraries are specified, with a version number, and a path to a git repository in the west manifest. On first build west will clone all repositories, launch Cmake to build each of them as libraries and link them in to your application. This is a very powerful tool that can ease manually controlled build system headaches.

VI. Home-Assistant

Home-assistant is a software platform intended for the display and orchestration of the variety of IoT devices that can be found in the home. The Home assistant front-end, named Lovelace, provides a convenient dashboard for displaying the current state of the system and configuring automations, add-ons, integrations, and the Home-Assistant platform itself.

One of the most common methods of deploying Home-Assistant, and also the one used for this paper, is to use the Home-Assistant Operating System (HAOS) on a Raspberry Pi single-board computer. HAOS is developed and maintained by the Home-Assistant project as an easy means of deploying and maintaining the Home-Assistant software.

A. *Ingesting Mesh Events Packets with Home-Assistant*

A custom integration was developed that opens the Gateway Node's USB/Serial connection and processes mesh events into a Home-Assistant state_changed event. The integration defines the Sensor Entities on initialization. For a production-ready solution, the integration should be able to create new entities whenever a new sensor node's mesh event is processed. For the sake of time, the Sensor Entities used in this paper are hardcoded to be created during initialization with friendly names and unit-of-measurement correlated to their respective node addresses and sensor property IDs, whether or not data has been received from them.

B. *Home-Assistant Data Organization.*

Each event within Home-Assistant is logged, with context, in its internal database. By default, a local (w.r.t the Home-Assistant server) SQLite instance is used to provide the database. However, the "Recorder" integration which is responsible for saving these events to the database can be configured to use an alternative database, local or remote. Database server software compatibility hinges on whether it is supported by SQLAlchemy, which is what Recorder uses under the hood. The usual suspects for linux-hosted databases, PostgreSQL and Maria DB, are well supported [23].

Changes to states, such as an update event from our meshed environmental sensors, are saved to the database's "states" table. Each new state is assigned a state_id (the index of this table) and is inserted into this table with the entity domain of the state, the entity_id the state belongs to (in the format <entity domain>.<friendly name>), the value of the new state, a JSON string containing the attributes of the state (e.g. the sensor entity attributes are the user-facing name of entity, units of measurement, and its device_class), the event_id of the event that triggered the change (e.g. a state_changed event), timestamps (last_changed, last_updated, and created), and the state_id of the previous state. For the purposes of environmental monitoring, the keys of interest are the created time, the state value, the entity_id, and perhaps the attributes if we do not know in advance which entity stores which measurement type. That said, it is important to note that state change data is considered by Home-Assistant to be short-term data [24]. Home-Assistant retains short-term data for 10 days (by default), after which it is purged on the next daily auto-purge event. The retention time of short-term data can be extended and/or the auto-purge even can be disabled, if storage allows.

Home-Assistant provides a means of long-term storage for Sensor Entities that implement a "state_class" of either "measurement" or "total". Every hour the minimum, maximum, and mean of the state value for those entities over the past hour is calculated and saved into the statistics table of the database. There is also a "statistics_short_term" table which uses the same index and keys as the "statistics" table, but is the data is processed over 5 minute chunks instead of hourly. The documentation for the short term statistics is sparse, but it appears that this is, like the name implies, short term data that will be periodically purged. Both the statistics and statistics_short_term tables use a metadata_id to identify the Sensor Entity that the statistics were generated from. The metadata_id is the index of the “statistics_meta” table which contains the statistic_id (in the same format as the state’s entity_id), and unit of measure. Between one of the statistics tables and the statistics_meta table, all the keys identified as being relevant in the state table are also available here, just in a format that has a smaller data footprint.

C. *Exfiltrating Home-Assistant Data*

There are two main ways to exfiltrate data from a Home-Assistant instance: automated methods using Integrations and Addon, or by interfacing with a database.

If your use-case requires only infrequent data retrieval, such as reviewing long-term statistic data or saving monitoring data from a short-lived (a few days at most) experiment, then manually exfiltrating data from the database may be the optimal solution. The SQLite Web Add-on is available with one-click install in the in-dashboard Home-Assistant Add-on store and provides an easy to use Web UI. Through this Web UI you can browse through Home-Assistant’s database table structure and contents. To exfiltrate the data, SQL Web offers an SQL query editor that can be used to build a query, view the output, and then save the output to JSON or CSV format. The data used in this report was retrieved via this method.

If the use-case demands saving all datapoints (or the 5-minute statistic_short_term values) and daily manual exfiltration is not suitable, then state changes must be exfiltrated to another storage solution, preferably as they occur. For a well supported built-in solution with a Web UI, Home-Assisant Operating System offers a Community Add-on that installs and configures InfluxDB with Chronograf and Kapacitor for administration and data exploration interfaces. [25]. Once configured, the InfluxDB integration transfers each state change to the InfluxDB instance. If a local instance of InfluxDB is not desired, the InfluxDB integration can be configured to transfer state changes to a remote instance of InfluxDB [26]. Alternatively, the Home-Assistant’s core back-end provides REST [27] and WebSocket [28] APIs that can be polled by other services to receive state changes. This method does have a higher overhead Bear in mind that even if Home-Assistant is not used for data storage, it continues to provide value as an aggregator of multi-domain data.

D. *Alternatives to Home-Assistant*

An alternative that was considered for this system was to use the same library that interfaces with the Mesh Gateway Node’s USB/Serial connection to either log directly to a CSV file or offer the mesh events over a REST HTTP server (polled) or client (push). The CSV approach is fast and easy to implement, but effectively limits the gateway server to the role of a datalogger. The REST endpoint or client method does have merit and was considered as a back-up plan for this system if Home-Assistant was found to not be suitable. A REST endpoint can be polled by or push to Home-Assistant or other popular monitoring and logging solutions such as Grafana with Loki or InfluxDB. However, the main drawback to this approach is initial setup. The raspberry pi needs to be setup and hardened for use on a network of presumably modest security. Then the Mesh Gateway Node interface application needs to be configured to point to the correct USB device, with the correct permissions. Finally, the monitoring/logging solution needs to be configured to ingest data from the Mesh Gateway Server, or vice-versa if the Mesh Gateway Server will be pushing data to the monitoring/logging solution. This would be a reasonable solution for a hobbyist or organization with staff that have the ability and time for this setup.

VII. The Complete System

The completed system looks much like the proposed system. Sensor nodes measure temperature and humidity using a low-cost I2C sensor. Bluetooth Mesh was used to communicate amongst the sensor nodes which were physically located in such a manner that the furthest node was unable to reach the gateway node directly, requiring a Relay Node to retransmit the Mesh Messages. A gateway node was used to transmit mesh event data to the gateway server in lieu of using the gateway’s on-board Bluetooth module. The gateway node communicates with the gateway server via a USB CDC ACM Uart, which is recognized by the gateway server as an emulated serial port. Using this connection, a Home-assistant integration library receives mesh events from the gateway node via a simple serial protocol and translates them into state change events on a Sensor Entity (if the new values differ from the old values). Home assistant logs every state change event and state value for short-term retention. For long term retention Home-Assistant stores the average, minimum, and maximum values of the Sensor Entity over one hour periods.

A. *Exfiltrated Data*

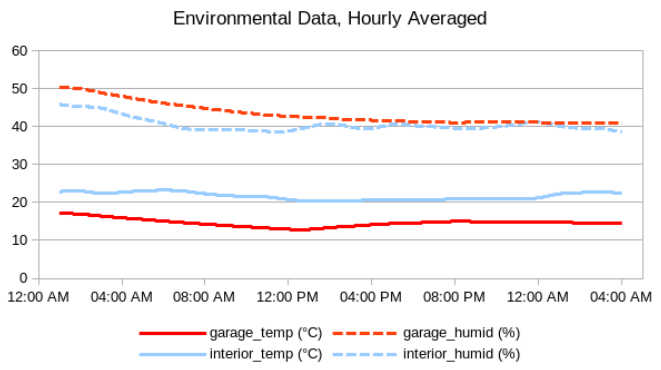

Figure 5: Home-Assistant Statistics Data from Sensor Nodes

The chart in Figure 3 is built from the Home-Assistant one hour statistics values gathered from the Mesh Sensor Nodes over a roughly 24 hour period.

B. *Provisioning the Mesh Network*

By far the easiest method to provision the mesh network is to use Nordic’s nRF Mesh application for Android and iPhone. This application handles unprovisioned node discovery, OOB authentication, network provisioning, binding of application keys, assignment of virtual addresses, setting up publishers and subscribers, and interfacing with the mesh server models through an intuitive GUI. The application does require the device to be within range of a Proxy Node in order for it to function.

C. *Home-Assistant Setup*

Assuming the custom integration developed to translate mesh events received from the Gateway Node's USB serial data was accepted as an official Integration, getting mesh data into Home-Assistant would require little more than inserting the Mesh Gateway Node's USB port into a network accessible computer running Home-Assistant, adding the integration, and provisioning the mesh network. If the Integration was not accepted, then installing it as a custom integration is as simple as creating the "custom_components" directory in the Home-Assistant configuration directory, cloning the custom integration repository into it, and restarting Home-Assistant. Further easing deployment, a Raspberry Pi with Home-Assistant Operating system can be prepared in advance (with any custom integrations pre-installed). All that would be needed for on-site install would be to plug in power and network connections and have the user follow the prompts on the website to set up a supervisor account. The only other configuration would be to provision the mesh network (assuming the gateway and sensor nodes were not pre-provisioned before issuing) and click a few buttons to start the Bluetooth Mesh Sensor Integration and to confirm that the pre-selected USB device is correct.

VIII. Lessons Learned

A. *Difficulties*

A personal difficulty was that, before launching this project, I had no experience with any part of the completed system, except for designing the Gateway Node's serial protocol and writing the serial library to parse it into data objects on the Gateway Server. The common theme was that despite each layer of the system having some of the best developer-facing documentation I have worked with, there was always a large gap between the nearly-trivial examples and what was needed to build my somewhat more complex components.

B. *Failures*

The original intent was to use the Raspberry Pi 4 based Gateway Server's on-board Bluetooth module to communicate with the bluetooth mesh via a Proxy Node. Home-Assistant Operating System does not provide or document a means to install the necessary kernel modules or the supporting BlueZ (the Linux project's out-of-kernel Bluetooth stack) daemons. At this point Home-Assistant OS was swapped for the standard Raspbian distribution, which is preconfigured with the correct kernel modules and now has the mesh-enabled BlueZ tools available in the package manager's repositories. Initial success was had by provisioning the device nodes through the BlueZ mesh-cfgclient. However, minimalist documentation, the need to communicate through DBUS, which I had never as much as looked at, and a promising, but still a work-in-progress python library combined to grind forward progress to a halt after well over 40 hours invested. A third Particle Xenon was acquired and the Gateway Node was completed in much less time.

IX. Opportunities for Future Efforts

Investigate power usage and capabilities of Low Power Nodes, especially for battery operation.

Add more Client Models to Home-Assistant Integration and Gateway Node to widen capability of Bluetooth Mesh control and monitoring. Implement Server Models to Integration and Node to give mesh node clients access to external-to-mesh devices.

Implement and Document a Bluetooth Mesh Integration that uses the Raspberry Pi onboard Bluetooth Module via BlueZ to communicate with mesh Proxy Node. Enable provisioning from Home-Assistant Web UI.

Investigate Existing Integrations to ingest data for other, non-mesh, Microcontroller platforms. There is an integration for automated discovery of and collecting data via Wi-Fi from ESP32 and ESP8266 SoCs running the ESPHome firmware. There is an integration for the Arduino Platform as well.

References

[1] "MESH Temperature & Humidity," *Takeoff Point (A Sony Group Company)*. https://shop.meshprj.com/products/temperature-humidity (accessed Sep. 23, 2021).

[2] "Blue PUCK T - Temperature Bluetooth Sensor | Teltonika Telematics." https://teltonika-gps.com/product/blue-puck-t/ (accessed Sep. 23, 2021).

[3] E. Standards, "EN 12830," *https://www.en-standard.eu*. https://www.en-standard.eu/ilnas-en-12830-temperature-recorders-for-the-transport-storage-and-distribution-of-temperature-sensitive-goods-tests-performance-suitability/ (accessed Sep. 23, 2021).

[4] karinfly, "SONOFF SNZB-02 - ZigBee Temperature And Humidity Sensor," *SONOFF Official*, Jan. 13, 2021. https://sonoff.tech/product/smart-home-security/snzb-02/ (accessed Sep. 23, 2021).

[5] karinfly, "SONOFF ZBBridge - Smart Home Zigbee Bridge," *SONOFF Official*, Feb. 05, 2021. https://sonoff.tech/product/smart-home-security/zbbridge/ (accessed Sep. 23, 2021).

[6] Q. T. Tran, D. D. Tran, D. Doan, and M. S. Nguyen, "An Approach of BLE Mesh Network For Smart Home Application," in *2020 International Conference on Advanced Computing and Applications (ACOMP)*, Nov. 2020, pp. 170–174. doi: 10.1109/ACOMP50827.2020.00034.

[7] ángela Hernández-Solana, D. Pérez-Díaz-De-Cerio, M. García-Lozano, A. V. Bardají, and J.-L. Valenzuela, "Bluetooth Mesh Analysis, Issues, and Challenges," *IEEE Access*, vol. 8, pp. 53784–53800, 2020, doi: 10.1109/ACCESS.2020.2980795.

[8] R. Rondón, A. Mahmood, S. Grimaldi, and M. Gidlund, "Understanding the Performance of Bluetooth Mesh: Reliability, Delay, and Scalability Analysis," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2089–2101, Mar. 2020, doi: 10.1109/JIOT.2019.2960248.

[9] V. A. Zyulin, A. N. Semenova, A. K. Brazhnikova, and D. A. Burilov, “Features of Building MESH Networks Based on Bluetooth Low Energy 5.1 Technology,” in *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, Jan. 2021, pp. 81–84. doi: 10.1109/ElConRus51938.2021.9396530.

[10] “Bluetooth Mesh, Thread, and Zigbee Network Performance Benchmarking - Silicon Labs.” https://www.silabs.com/wireless/multiprotocol/mesh-performance (accessed Sep. 23, 2021).

[11] W. Rzepecki, Ł. Iwanecki, and P. 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)*, Aug. 2018, pp. 42–47. doi: 10.1109/W-FiCloud.2018.00013.

[12] É. Morin, M. Maman, R. Guizzetti, and A. Duda, “Comparison of the Device Lifetime in Wireless Networks for the Internet of Things,” *IEEE Access*, vol. 5, pp. 7097–7114, 2017, doi: 10.1109/ACCESS.2017.2688279.

[13] “Nordic Semiconductor Infocenter - nRF52840 Product Specification.” https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52840.html (accessed Dec. 08, 2021).

[14] “Assigned Numbers,” *Bluetooth® Technology Website*. https://www.bluetooth.com/specifications/assigned-numbers/ (accessed Dec. 08, 2021).

[15] M. Woolley, “Bluetooth Mesh Models - A Technical Overview.” Mar. 27, 2019. Accessed: Dec. 08, 2021. [Online]. Available: https://www.bluetooth.com/bluetooth-resources/bluetooth-mesh-models/

[16] “GATT Specification Supplement 5,” *Bluetooth® Technology Website*. https://www.bluetooth.com/specifications/specs/gatt-specification-supplement-5/ (accessed Oct. 14, 2021).

[17] Mesh Working Group, “Mesh Device Properties 2.” Bluetooth® Technology Website. Accessed: Oct. 13, 2021. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-model-1-0-1/

[18] “Mesh Profile 1.0.1,” *Bluetooth® Technology Website*. https://www.bluetooth.com/specifications/specs/mesh-profile-1-0-1/ (accessed Oct. 12, 2021).

[19] “Mesh Model 1.0.1.” Bluetooth® Technology Website. Accessed: Oct. 13, 2021. [Online]. Available: https://www.bluetooth.com/specifications/specs/mesh-model-1-0-1/

[20] “CERT/CC Vulnerability Note VU#799380.” https://www.kb.cert.org (accessed Dec. 08, 2021).

[21] “Provisioning — Zephyr Project Documentation.” https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.7.1/zephyr/reference/bluetooth/mesh/provisioning.html#bluetooth-mesh-provisioning (accessed Dec. 08, 2021).

[22] “Welcome to the nRF Connect SDK! — nRF Connect SDK 1.7.1 documentation.” https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.7.1/nrf/index.html (accessed Dec. 08, 2021).

[23] H. Assistant, “Recorder,” *Home Assistant*. https://www.home-assistant.io/integrations/recorder/ (accessed Dec. 06, 2021).

[24] “Data | Home Assistant.” https://data.home-assistant.io/docs/data (accessed Dec. 06, 2021).

[25] *Home Assistant Community Add-on: InfluxDB*. Home Assistant Community Add-ons, 2021. Accessed: Dec. 07, 2021. [Online]. Available: https://github.com/hassio-addons/addon-influxdb/blob/764b622119f94135739783d5d9cab8b27e349cde/influxdb/DOCS.md

[26] H. Assistant, “InfluxDB,” *Home Assistant*. https://www.home-assistant.io/integrations/influxdb/ (accessed Dec. 07, 2021).

[27] “REST API | Home Assistant Developer Docs.” https://developers.home-assistant.io/docs/api/rest (accessed Dec. 07, 2021).

[28] “WebSocket API | Home Assistant Developer Docs.” https://developers.home-assistant.io/docs/api/websocket (accessed Dec. 07, 2021).