FireAway

Authors: Karen Abruzzo, Ankita Bhanjois, Arden Diakhate-Palme
Affiliation: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A system capable of signalling the location of a wildfire to emergency services. Currently, it takes time and effort for wildland firefighters to constantly monitor possible wildfire locations. This project intends to create a low power and accurate system that can detect the location of wildfires, by using a wireless sensor network. If a fire occurs near a specific node in the network, that node's data will be transmitted across the network in order for wildland firefighters to then be deployed.**

***Index Terms*—Fire Detection, MAC Protocol, Node, Routing, Scheduling, Spanning Tree Protocol, Web Application, Wireless Sensor Network**

1 INTRODUCTION

In 2022, California has had approximately 6000 recorded wildfires, spanning almost 300 thousand acres. A fire itself takes around 1 hour from the ignition of the fire to become a fully developed fire. This was the inspiration behind our project. We wanted to create a system for wildfire detection to be used by wildland firefighters. The current solution for detecting forest fires is if someone calls 911 or via a system of lookout towers. However, with our solution, firefighters would be able to pinpoint the location of conflagrations in a timely manner. The system is designed for their convenience and safety. Therefore, the goal of the system that we are creating for our project is to have (1) accuracy and (2) low power.

In order to do this, we created a wireless sensor network of eight nodes, in which each node collected data regarding the temperature in its vicinity to indicate if a fire was ignited. The sensor data collected was transmitted across the network and collected by a gateway router, which displayed the node where the fire was detected on a web application. The web application provides the fire department with the location to deploy their personnel. We wanted to create a singular system that does this so there didn't need to be any extra work or components to make wildfire control and eradication unmanageable.

Due to budgeting and time restrictions that come with completing this project in one semester, the project is scaled down. However, the network protocol that is being used is designed to be scaled up. The node architecture is scalable because there is a variable window for sensor data collection. However, a mere temperature sensor mounted on a tree would not be able to detect forest fire, since it is only detecting the a large change in temperature caused by a heat gun. At scale, the web application would be hosted in the cloud to circumvent connectivity issues (i.e. in case the network was to go down). The main part of our project is how to use a wireless sensor network for fire detection and the protocols that make the system useful.

2 USE-CASE REQUIREMENTS

Our use case is rapid wildfire detection for the fire department. Since it could be catastrophic if a fire is not detected, we want to detect a fire with **90-95%** accuracy. This is to allow for some slack in the case of any unexpected interference to the system. Our spanning tree protocol would need to adjust itself to find an appropriate path to the border gateway from the location of the fire, circumventing any nodes that are offline. It takes one hour from the ignition of a fire to become an open fire. Hence, it is critical that the fire is detected during this period, so that wildland firefighters can put out the fire before it becomes fully-developed. Therefore, our web application needs to show active fires within **30 minutes** of conflagration. The nodes also need to be low power to make our system low-maintenance. The purpose of this is to ensure that firefighters aren't constantly needing to check in on the system. The advantage of using a sensor network is that people would not have to be stationed in the forest in lookout tours to keep an eye out for fires. If park rangers need to frequently visit the nodes to replace the battery, then it defeats the purpose of the system. In order to meet the needs of the use case, the nodes need to be able to operate for **one month**. To reiterate, we want our system to be accurate and easy maintenance, so we have decided on the following use-case requirements:

- **90-95%** accuracy with fire detection
- **30 minute** window of when web application is notified
- **1 month** maintenance for overall system

3 ARCHITECTURE AND PRINCIPLE OF OPERATION

A block diagram visualizing our system is provided in Fig. 1, but for clarity we have provided a full-page block diagram of our system in Fig. 13 on page 16. This block diagram has been broken down into individual components in the upcoming sections. Our system is broken down into three major subsystems: Networking, Node Architecture, and Web Application. The first subsystem is the Network

that we are constructing. We are using a spanning tree protocol, a link-state advertisement, and RX/TX timeslot scheduling. This helps transmit data between the nodes about the sensor data being collected and adjust itself in the case of collisions or node failure. We were originally going to include NTP (Network Time Protocol) in our protocols run by the gateway; however, due to time constraints we ended up rescoping to using a timestamp determined by the gateway router. The second subsystem is the Node Architecture which consist of our micro-controller, transceiver, and temperature sensor for fire detection. This system needs to maintain low power. The third subsystem is the web application. This application displays the nodes where the fire is located and alerts the firefighters of that destination. The application and router for the first subsystem are hosted on a Raspberry Pi.

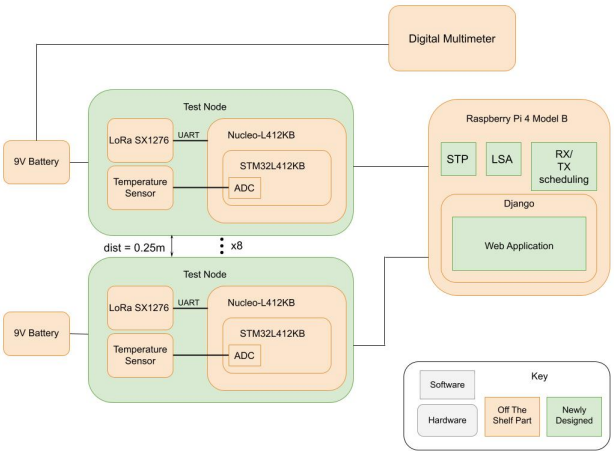
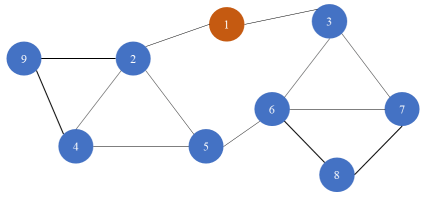
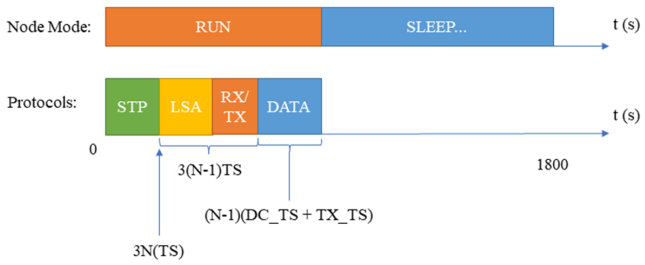


Figure 1: Overall Block Diagram

3.1 Networking



(a)



(b)

Figure 2: (a) Example Mesh Topology (b) Protocol Stack and Operation Overview

Our main objective for this section is to maximize the nodes' sleep time and minimize the amount of time they spend transmitting and receiving data, while remaining resilient to node failures. Figure 2(a) provides an example of the resulting topology after the network administrator has setup the nodes, covering a region in the forest. In this figure, nodes with links to one another are in range (within 15km) of one another and can communicate. In order to establish this network, a network operator would reset all of the nodes at once, and then hike across the forest and fasten nodes on trees above RF-obstructing foliage, ensuring that nodes adhere to the mesh topology rules (described in system implementation).

3.2 Node

This subsystem is how the nodes are constructed to create a system that is low power. Our Nucleo-L412KB with a STM32l4 micro-controller optimizes for low-power, as this STM32 has built in low-power modes. The LoRA transceiver also has a low power mode and has a range that can be used to scale up to a real world scenario. A TMP36 temperature sensor is used for detecting fire. The node is powered by a 9V battery. Both the LoRa and TMP36 are powered by the 3V pin on the Nucleo board.

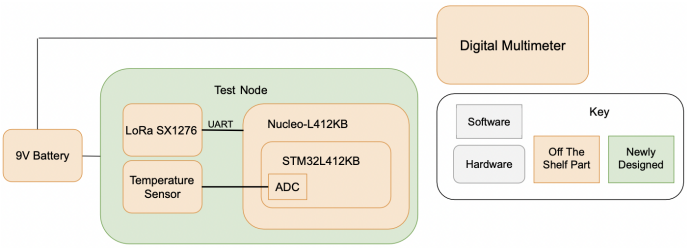


Figure 3: Node Architecture

The image shown above is the test node. This node will be used to show the power consumption.

3.3 Web Application

This subsystem is how the web application is going to be configured. The web application will be written on Django and will consist of a visual interface. The interface takes in the inputs from the sensor that are conveyed using a JSON file and display where the fires are located. The JSON file walks through the graph that is constructed by the network and from there determine which nodes are connected and where the fire nodes and offline nodes are located. The visual interface includes markers that indicate when the node is at its control state, on fire, or offline. It also displays links for the current spanning tree being used by the protocols and shows links that are not being used or are being used by offline nodes.

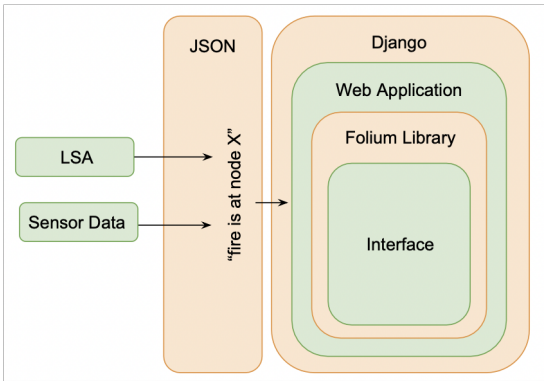


Figure 4: Web Application Block Diagram

4 DESIGN REQUIREMENTS

We would like our network to have a minimum battery life of 1 month, in order to make maintenance cost reasonable. Furthermore, if a node drops offline due to an external event (such as a branch hitting its antenna), we would want the network operator to be informed of this. However, a node dropping offline could also indicate a fire, if the fire has already engulfed the node within the 30 minute window (maybe the node was at the epicenter). In any case, a second verification would have to be performed to ascertain the root cause of this offline node.

In order to minimize the cost of a single node, we should minimize battery cost. Transceiver power consumption, and the MCU will be the primary drain on the battery life. Since propagation delay will differ based on the specific configuration of the LoRa transceivers, we are considering the default settings for power and total runtime of the networking protocol stack (see Fig.6). However, in testing, we used a higher BW than the default setting, leading to a lower ToA or propagation delay.

Ideally, we would run the protocol every 30 minutes to detect node or root-link failure, and to detect a conflagration. We have decided to reduce the mesh network to a tree topology, and will detect node failure in the first phase (TDM STP) of the protocol stack, whilst the tree is being formed. We decided to change these design requirements from our initial design report in order to save even more power (as running the DATA phase is only required every 30 minutes instead of every 15 minutes). Moreover, a 30 minutes detection latency still provides wildland firefighters an advantage as compared to traditional forest-fire detection methods, allowing them to combat the fire before it becomes significantly larger at the 1-3 hour mark.

Furthermore, we want the MCU to be in the lowest power mode for the longest period of time. When the node is not transmitting DATA packets or performing control-plane functions (node failure detection or spanning tree-reconfiguration), we will want to remain in a low-power mode to prolongue battery life. We are able to achieve such low power modes by placing the node's LoRa transceiver into low-power mode (SLEEP mode), and then transitioning the MCU to a low-power mode (STANDBY mode).

The power consumption of a given node is contingent on the efficiency of the routing protocol. If the protocol does not allow nodes to go into low-power since the node has to be able to route packets, we will be losing power unnecessarily.

Our nodes will be placed 0.25 meters apart from each other. This will ensure easier testing as we can keep the node on a single table. This distance was determined knowing that the LoRa transceiver can have a range of 250m and we wanted to scale that down[2].

5 DESIGN TRADE STUDIES

5.1 Network Architecture

In order to enable nodes to sleep for a maximum period of time, we considered using the LoRaWAN MAC protocol. After having read through the Link Layer specification, class A operation of network nodes seemed to be meet our use-case requirements best. According to the specification document, "Class A operation is the lowest-power end-device system for applications that require only downlink communication from the server shortly after the end-device has sent an uplink transmission." [8] The two principal downsides to employing the LoRa-WAN protocol are as follows[8]:

1. **Star-of-stars topology**: The need for intermediary gateways which communicate with the Network server (border gateway) over IP connections

2. **Single-hop** RF communication from nodes to gateways, not applicable for multi-hop sensor networks, gateways would need to have a longer battery life

3. **ALOHA MAC protocol**: Less precise control over node sleep schedule, and higher collision rate: throughput $S = \frac{1}{e}$ for pure ALOHA and $S = \frac{1}{2e}$ for slotted ALOHA.

A multi-hop wireless sensor network would reduce material cost since higher-capacity expensive batteries would

be required for intermediate LoRa-WAN gateways. Furthermore, power grows exponentially in a multi-hop WSN, so we will require technicians to service nodes closer to the border gateway more frequently than nodes deeper in the forest. However, nodes closer to the gateway are assumed to be more accessible. If the same battery was used for sensor nodes and intermediate LoRa-WAN gateways, these gateways would need to be serviced much more often, and depending on the topology these intermediary gateways may not be very accessible (i.e. they might be far from the GW router).
When designing our own protocol for low-power medium access control, we considered two main strategies.

1. **Asynchronous B-MAC**: Nodes periodically wake up to check their children nodes. In this protocol, a node that wants to send a packet sends a preamble for the same time duration as the receiving node's sleep period. Hence, nodes must periodically return to a "receive-only" state to check if their children have a packet to send. For a multi-hop network this approach imposes a prohibitive current draw for all nodes, since the sampling period is once every 30 minutes.

2. **Synchronous MAC**: Nodes use TDMA to avoid packet collisions, which allows them to go to a lower power-consumption sleep state when they are not scheduled to transmit or receive. The tradeoff is that this type of MAC protocol comes at a high traffic control cost, and requires periodic beacons to correct nodes' clock drift. A global notion of time is required so that nodes wake up and transmit in their actual timeslots, and not an adjacent one. Since each node's MCU will have to use its own clock to determine when it is time for them to transmit, periodic clock synchronization is necessary to ensure a global notion of time across nodes.

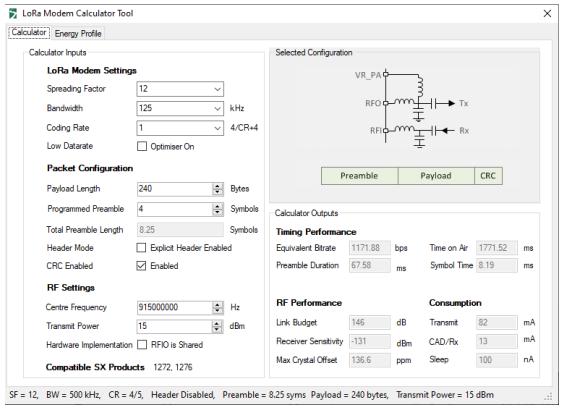


Figure 5: Modem Attributes

We opted for a multi-hop wireless sensor network using synchronous MAC protocol and a basic time-synchronization mechanism in order to increase battery life and reduce network maintenance cost. We have calculated the worst-case power consumption and times as follows based on the chosen synchronous MAC routing algorithm. Here, t_{TS} is the duration of a TS in milliseconds, and t_{DC} is the duration of the data-collection time window (DC_TS) during which the temperature sensor is sampled:

1. **STP**: The protocol will convergence in 3 iterations, and in the worst-case for a single iteration, each sensor node will have to transmit and receive at most $N-1$ times. If we also consider timeslots for the gateway to send STP messages we obtain the following. Furthermore, a node will transmit at most $N-1$ in a star-topology.
   total-duration: $3N(t_{TS})$
   power consumption: $3(Q_{TX}+Q_{RX})(N-1)$

2. **LSA**: A spanning tree of N nodes will have at most $N-1$ links (if the "tree" is a line-topology in the worst case). Since we are transmitting both LSA_REQ and LSA_RESP, and acknowledging each packet, we obtain the following maximum time for this phase. Furthermore, a node will transmit a packet and receive a packet in the worst-case $N-1$ times, if it is the center of a star topology. It is important to note that the two worst-case scenarios for power consumption and total-duration are disjoint, or unlikely to occur at the same time.
   total-duration: $2(N-1)t_{TS}$
   power consumption: $(N-1)(Q_{TX}+Q_{RX})$

3. **RX/TX**: This is the same worst-case time duration as the **LSA** phase, since SCH_ACK's are used to iteratively query nodes and avoid packet collisions. In this case, as a comparison for worst-case power metric calculations, SCH packets take the role of LSA_REQ packets, and SCH_ACK packets take the role of LSA_RESP packets. SCH_ACK packets also provide a form of MAC, since the next child node is not queried before an SCH_ACK is received from the first queried node, so no two nodes transmit simultaneously.
   total-duration: $2(N-1)t_{TS}$
   power consumption: $(N-1)(Q_{TX}+Q_{RX})$

4. **DATA**: For a WSN of N nodes, we have that each node will necessarily need to take sensor readings and transmit them upstream to the GW router. A minimal total duration was obtained via the RX/TX scheduling algorithm. Furthermore, in a star topology, a node would have to receive $N-1$ times, but each node is only transmitting once in this phase.
   total-duration: $(t_{DC}+t_{TS})(N-1)$
   power consumption: $2(N-1)(Q_{RX})+Q_{TX}$

Considering the ToA based on the LoRa modem parameters and a timeslot buffer of 0.5 seconds:
$t_{TX\_TS} = t_{RX\_TS} = t_{TS}$ =ToA+0.5s = 2.271s
$t_{DC} = t_{TS}$
t_{ToA} = 1771.52ms represents the ToA for modem RX and TX functions (see Fig. 5)[9]

$I_{RX} = 13\text{mA}$, $Q_{TX} = 82\text{mA}$, $I_{lora\_low} = 100\text{nA}$ [9]
$I_{RUN} = 3.79\text{mA}$, $I_{SLEEP} = 0.95\text{uA}$ [15]
$I_{TEMP} = 50\text{uA}$ [11].

The average power consumption for one iteration through the protocol stack and node sleep-time (between iteration low-power mode) is as follows:

$$I_{RUN}\frac{(9t_{TS}(N-1))}{1800} + I_{SLEEP}\frac{(1800-9t_{TS}(N-1))}{1800}$$
$$+I_{RX}\frac{6(N-1)(t_{ToA})}{1800} + I_{TX}\frac{(5N-4)t_{ToA}}{1800} +$$
$$I_{lora\_low}\frac{(1800-t_{TS}(9N-6))}{1800} + I_{TEMP}$$

At the time of our design report, we had not yet considered that every packet would require its individual acknowledgment (as part of stop-and-wait). Therefore the following estimate is slightly inaccurate. A 9-node network run at a 30min data sampling period and link/node failure detection period consumes on average 4.34mA. Running this WSN for a period of 1 month would consume $(4.34)(720) = 3124.7mAh$. Hence, a battery with $4000mAh$ of capacity would make a node last one month.

5.2 Low Power Networking VS BMS

One of our ideas for making the system low power was to have a battery management system and a solar panel. The idea was we could power the node and charge the battery when it was sunny, and when it was not sunny we could use the battery to power the node. If the node got low on power, we could change how often we transmit data or put the node into a really low power mode to preserve power until the battery could be recharged. However, we decided not to go with this idea for a few reasons. None of us are familiar with a BMS, and due to the time constraints of the class we were worried about figuring out how to get it to work in time. There was also concern that since the sensors are meant to be placed on trees in a forest, the trees would block too much of the sun and that they would not be able to recharge the battery quickly enough to avoid the battery completely running out of power.

5.3 Node Architecture

For the sensor nodes, we decided to use the Nucleo-L412KB with an STM32L4 as our micro-controller as it is one of ST's ultra-low-power microcontrollers. It has the lowest current in its lowest power mode with RAM retention and the RTC enabled compared to the other ultra-low-power STM32s.[13] It also has a larger variety of low power modes than most of the other ultra-low-power STMs, which allows for more flexibility when entering a low power state.

In addition to the STM32L4, we decided to use LoRa tranceivers because they are relatively low power for the range they offer. 4G routers have a range of 100-300m, and use more power than LoRa [5]. Wifi has a typical range of 150-300ft, which is not enough to meet the requirements of our use case [12]. LoRa, however, has a range of 250m in a forest environment. This would indeed fulfill our use case requirements and help when this node architecture is scaled up in a real world scenario.

5.4 Web Application

Regarding the web application, we wanted to create a way to visualize the location of the fires. We are using a Raspberry Pi to host the web application. We chose to use an RPi for convenience as the RPi will be used as both the border gateway for the network and as the host for the web application. For the webapp itself, we will be using Django as our framework software. Django uses Python which is easier to use in terms of implementation. Django is also well-documented and there are plenty of tutorials that can be of assistance. Django also has a library called Folium [3] that is beneficial for the design of the web application, making it a favorable option. Folium allows using a library that is used for creating interfaces that display maps, making it ideal for our project. It also allows links in between the markers on the map, to help us show our topology being used in the protocols. In addition, using geojson.io [4] allows us to get accurate longitude and latitude coordinates of locations in the world that we can use to show where our project can be applied. Originally, we were going to use GeoDjango as our main library for the web application; however, we found that Folium was more user-friendly and relevant to our project's design. GeoDjango had a lot more complicated documentation which made it a little diffiuclt to use. It is important to note why we are using the RPi to host the web server instead on the cloud. The web application that is being created is simply being used as a visualization of the nodes and the location of the fire. The web application itself is not to scale if this project would be scaled up. A web application would need to be reconfigured and redesigned for it to be used on a larger scale, as it currently only accounts for eight nodes not a whole system. It also has a pop-up window displayed when a fire is located. For the purposes of our project, it is simply a visualization tool.

6 SYSTEM IMPLEMENTATION

6.1 Networking

Our MAC protocol seeks to reduce power consumption, and to be resilient in the face of link or node failures. If a node drops offline or the single link connecting a node to the rest of the WSN fails, a field technician should be informed of that node's failure. Furthermore, the network can recover from a node failure if the node goes offline / unresponsive during the node's SLEEP period (after having run the protocol stack), or before that node's address-specific transmission time slot in the TDM STP phase. In this case, the network creates a new tree topology from the initial links (links which indicate nodes are in range of one another), and re-adjusts the schedule accordingly.

1. **TDM STP**: On node startup, it will run the spanning tree protocol until it has heard a message from the true root (which is the gateway router with network address 1). This will take 3 iterations given the following constraints on the initial mesh topology:

   - In a single loop-free path (GW to end-node), there cannot be more than two monotonously-decreasing nodes in between two monotonously-increasing nodes.

   This constraint disallows some initial line-topology configurations, imposing some limitations on where nodes can be installed in the forest region. However, line topologies are rare, and not robust against node failures: this sensor network would be deployed to monitor a forest region in a mesh topology to circumvent this issue.

   In order to ensure that no packet collisions occur, a node will wait for $a(TS)$ before transmitting, where for a topology of N nodes, $a \in [2, N]$ is the sensor node's address.

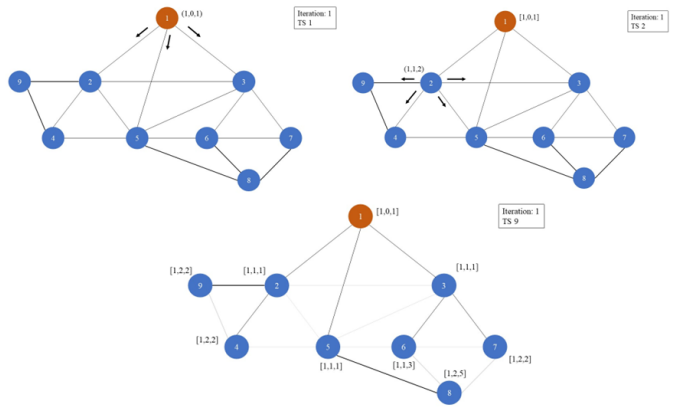


Figure 6: Spanning Tree Protocol with time-division multiplexing

2. **LSA**: Once the spanning tree protocol has converged (in three maximum iterations, given the constraints for the initial topology). Propagate link-state requests down the spanning tree. Each node iteratively forwards a link-state request to each of its children and gathers the link-state response from the querried child before querrying the next one. Once all LSA responses from children sensor nodes have been received, the sensor node sends the response to its spanning-tree parent node.

   In this phase, each node begins in RUN mode with active listening.

   - <u>If the node has children</u>,
     - on receiving an LSA_REQ, forward that request to each child node starting with the child with the lowest address, and wait to receive the complete LSA_RESP from that child node before querying the next one.
     - once the node receives an LSA responses from each of its children, store these responses locally (it now knows the topology of all its children - the entire child sub-tree), and aggregate the responses and send them to the parent node (one packet is sent to the parent node per child node in the child sub-tree). For instance in Fig. 9, node 3 would send 3 response packets: one for its neighbors (6,7), one for node 6's neighbors (3), and one for node 7's neighbors (3).
   - <u>If the node has **no** children</u>,
     - on receiving an LS request, respond with the topology of your sub-tree (the downstream tree topology from the node's perspective) to your parent.
   - If the node (node B) receives a LS request from an inactive port, there must be a uni-directional link from the sending node (node A) to this node (i.e A has an open port to its neighbor B, but the B has a *blocked* port to A). In this case, the node B sends a link state invalidate (LSA_INV) response to node A, signifying to node A to remove its uni-directional link to the B. On receving an LSA_INV response after sending an LSA_REQ to node B, node A would block the port to its neighbor node B.

   We chose to send one LSA response per neighbor in a node's sub-tree because the LoRa transceivers only allow for a maximum packet payload of 240 bytes. However, these transceivers support a 16-bit address space, hence, packing an entire graph structure in a single packet imposes a severe limitation on the scalability of the network. Therefore, the LSA_RESP for each node in a sub-tree (i.e. each node's neighbors the sub-tree) is encoded in a single packet.

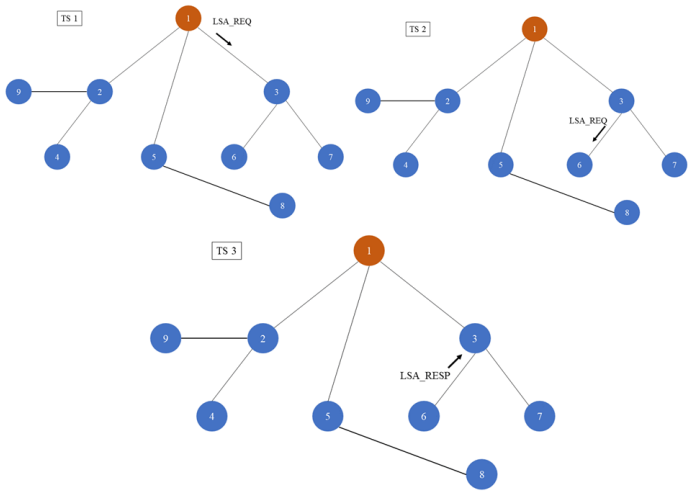


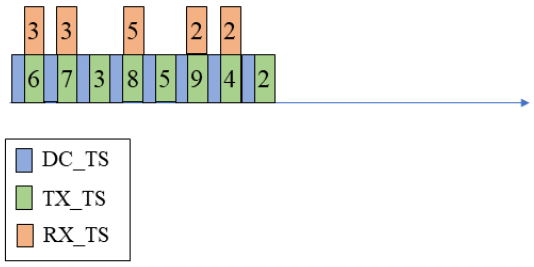
Figure 7: Link State Protocol Overview

3. **RX/TX Scheduling**: Once LSA protocol has run, the gateway router knows the overall topology of the sensor network, hence, it can schedule transmission and receive slots for each node to ensure a power-efficient TDM schedule, which allows all nodes to propagate data up the spanning tree back to the root. Having a global view of the tree from the LSA phase, the gateway router schedules TX and RX data slots as follows:

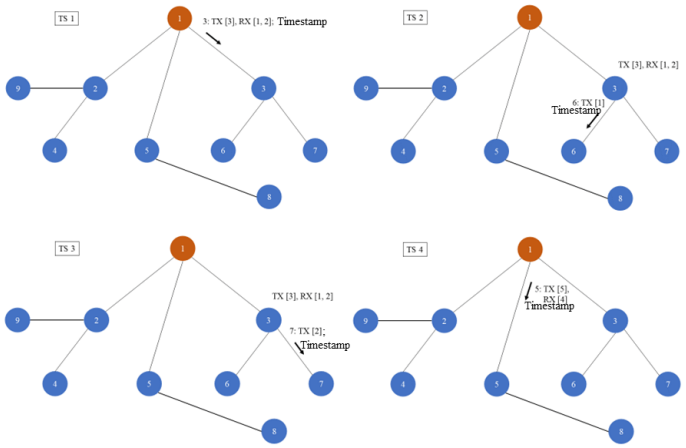
    - nodes which have immediate children nodes in the spanning tree have RX slots when their immediate children have TX slots. In other words, during the DATA phase at a certain DATA_TS, these nodes will be listening for their immediate children's packets. A node also has its TX slot scheduled for the DATA_TS immediately after all of its children's TX timeslots. As a node receives packets from its immediate children during its RX slots, it will record locally in RAM whether one of them have detected a fire.
    - nodes which have no children only have TX slots.

    The transmission schedule accounts for a sampling time t_{DC_TS} during which the node can poll its temperature sensor. Hence, each "slot" in the schedule (each DATA_TS) is set such that $DATA\_TS \geq t_{TS} + t_{DC}$. In its designated TX slot, a node will take a temperature reading at $t_i = 0$ and another at $t_f = t_{DC}$, measuring the temperature increase. If this increase is greater than a predetermined threshold, the node will have detected a fire, and will alert whichever node is listening (whichever node is in RX mode for that DATA_TS) that it has detected a fire, and whether any of its children have detected a fire.

    For instance, in Fig.10(a), node 3 is listening for DATA packets from its immediate neighbor node 6. During that timespan (DATA_TS seconds), node 6 takes polls the temperature sensor, determines whether the temperature increase is sufficient to indicate a fire, and transmits a DATA packet to node 3.



(a)



(b)

Figure 8: (a) Example schedule created by GW router (b) Overview of RX/TX Schedule Distribution Algorithm

After the gateway router computes the schedule for all $N-1$ sensor nodes, it must distribute the schedule to ensure that each node knows at which data timeslots to receive packets and which timeslots to take temperature readings and transmit packets. Each node (including the GW router) sub-divides the schedule before distributing it to its immediate children. From a given node's perspective, each immediate child only needs to know the RX/TX schedule up to the point when it is scheduled to transmit (its TX slot). This subdivision allows for smaller packets since no superfluous information is included. Notice that each schedule has $(N-1)$ DATA_TS's (where N is the number of responsive nodes after STP has run) because all nodes (except for the gateway node) must attempt to detect forest fires.

The gateway router includes a timestamp (phase4_start_time) in each SCH packet. Subsequent sensor nodes receiving a sub-schedule from the gateway router copy this timestamp to their SCH packets before sending a sub-schedule to an immediate child node.

Each node follows this protocol:

- on receiving a sub-schedule (SCH packet),
  - Store sub-schedule and received timestamp (phase4_start_time) in memory.
  - If the node has children,
    - cut the sub-schedule so as to include everything from the beginning of the sub-schedule to the DATA_TS when the immediate child node has its TX slot. Save this as a new schedule
    - Place this new schedule and a copy of phase4_start_time into an SCH packet, and send this packet to the immediate child node.
    - wait for an SCH_ACK packet from the immediate child node before sub-dividing and distributing the schedule to the next child node.
    - Once the schedule has been distributed to all child nodes (and acknowledged via SCH_ACK packets), send an SCH_ACK packet to parent node.
  - If the node has **no** children,
    - send an SCH_ACK packet to parent node.

Once a sensor node has sent its SCH_ACK packet to its parent node, it waits until its millisecond-accurate internal clock (implemented using timer interrupts) reaches phase4_start_time [6].

Since the GW router set the exact timestamp since the start of the iteration, this implies that it is assuming a certain minimum amount of packet timeouts and re-transmission during the schedule distribution phase. If more packet timeouts were to occur, nodes would not enter Phase 4 (DATA sampling phase) simultaneously.

4. **Data Sampling** Since all sensor nodes were reset at the same time, installed throughout the forest region, and all began phase 1 (TDM STP) simultaneously, their clocks will be synchronized to millisecond-level accuracy for the entire first iteration. Therefore, all nodes will reach the phase4_start_time (originally set by the GW) simultaneously, and being executing their schedule.

During the DATA phase, the RX/TX schedule is executed globally. In a given DATA_TS, if a node has a:

- **TX slot**: The node will poll the temperature sensor for DC_TS seconds to detect a fire, then will check its local database for whether any of its children have reported fires, and will transmit to its parent node whether it or any of its children have detected fires
- **RX slot**: The node will receive a DATA packet from one of its children nodes. If one its children nodes sends a packet mentioning that it or any of *its* children have detected a fire, this information is added to a local database.
- **IDLE slot**: The node either has yet to transmit its data or receive from its children because other nodes are busy transmitting or receiving (this provides a form of medium-access control), or the node has already transmitted its data and is waiting for the end of the DATA phase (i.e. the node is waiting for the remaining DATA_TS's).

Once all DATA_TS's have elapsed and DATA phase is complete, the nodes will sleep for the remaining time in the iteration in order and will reset at the 30 minute mark to re-run through the protocol stack. Once the node exits STANDBY after 30 minutes has elapsed, it will reset both the STM and the LoRa transceiver. Note that in our demonstration, the nodes are waiting in standby only 1 minute before running through the protocol stack in the next iteration, in order to facilitate testing.

5. **ACK and Re-transmission Scheme** Nodes required an acknowledgment and re-transmission scheme (Stop-and-Wait protocol) in order to offer reliable data transfer in phases 2-4 (LSA, SCH, DATA). Essentially, we need to ensure that if packets were dropped or had CRC errors at the receiving end, the sender would know to re-transmit the packet. Similarly, the receiving node would need to listen for a re-transmission after responding with an acknowledgment because it is possible that the ACK packet was lost due to channel errors as well. Fortunately, the LoRa modules make it easy to calculate ToA, since on issuing a sending command, the module responds with "+OK" only after ToA milliseconds have already elapsed. In Fig. 9 below, the "+RCV" response from the module indicates packet reception, and the sender (node A) would re-transmit its packet (SEQ 0) if it has not yet received an ACK by the time the re-transmission timer expires. The sending function rdt_send() returns slightly before the receiving function rdt_recv(), so delays are inserted at the application level in order to ensure that packets sent back-to-back can be properly received.

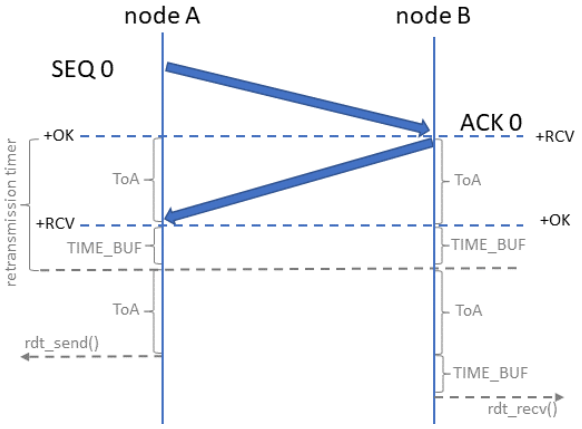


Figure 9: Stop-and-Wait protocol on LoRa module

6.2 Node Architecture

We implemented a sensor network by creating a system of 8 wireless sensor nodes. The sensors used an STM32L4 microcontroller, as it is designed to be low power and has multiple low power modes. A LoRa transceiver, which the STM32 will communicate with using UART, was used to send data between the nodes and the gateway. The nodes had a TMP36 analog temperature sensor in order to detect fire, which the STM32 got data from using ADC. Fig. 10 shows a schematic for the nodes.

The STM32 has multiple different low power modes. The ones we used are run mode and standby. When the node needs to be "awake" the STM32 operated in run mode. When the node was finished executing the schedule during data phase, the STM32 went into standby mode and used the RTC to wake up for the next iteration of protocols [10].

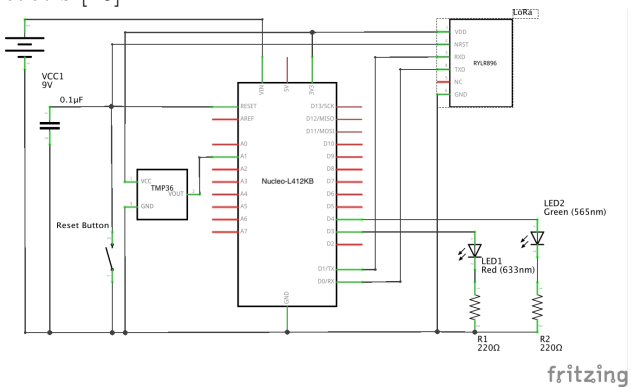


Figure 10: Node Schematic

6.3 Web Application

To create our web application, we used Django as web framework. Upon receiving data from the sensors (i.e. the fire was sensed at node X) and having a JSON file parse the network and obtain the nodes on fire and those that are offline, the web application takes this data as input. Using the Folium library, the web application consists of a topographical map of a forest, which is overlayed with markers. The library lets us choose from different maps and for the purposes of our project, Yosemite National Park was chosen, due to its suffering from wildfires every year. The markers (the pins on the map) represent the individual nodes in our network system. All the nodes have associated longitude and latitude coordinates that are displayed when a user hovers over the node. Each node is represented in a light blue color with the navy blue marker representing the gateway router, where the user would be monitoring the system. When the sensor data is received, the marker that is associated with the node where the fire is located will turn red. and have a pop-up window. If a node was offline, it would turn grey. The topology of the map is also displayed in the web application using black and grey lines. The black lines indicate the paths of the spanning tree being used by the protocols. The grey lines could indicate one of two things: the links associated with an offline node or the links that are not being used by the spanning tree. The web application updates and should be instant after receiving the data from the sensors.

Fig.11 shows an overview of what the web application interface looks like. Note how there are blue nodes, a grey node, and a red fire node, as well as the gateway at the bottom of the tree. It is also important to take note that all the links to the grey node's neighbors are also grey meaning that those are invalid links for the spanning tree to consider. Essentially, the web application is the visualization of what is happening in the "forest" and helps a user understand what is happening in the wireless sensor network.

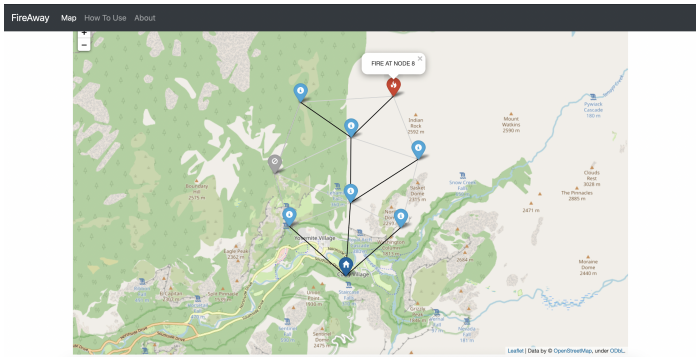


Figure 11: Web Application Interface

7 TEST,VERIFICATION, & VALIDATION

The following section has many references to our team's code and the iterations that we went through to get to our final results [7].

7.1 Unit Tests: Network

1. **4 Node Topologies**
   Before the interim demo, we had worked on getting LSA and STP working on a fixed-tree topology of

four nodes. The root node (hosted on an STM) was informed of the entire topology through a series of LSA packets (one per child node - as described in the system implementation section).
We then tested a fully linked topology, which was reduced into a star topology centered at the GW node (also hosted on an STM). The gateway node received the entire star topology after LSA finished running. Furthermore, if a node died before its transmission timeslot in the phase 1 (TDM STP), it did not appear in the adjacency-list implementation of the spanning tree at the GW node (once LSA finished running).

2. **Integration with RPi**
The RPi used a very different UART library [14] than the interface provided by the STM32's HAL library, so some testing was required in order to test sending AT commands to the LoRa module from the RPi, and to test non-blocking packet receiving calls.
The code had to be ported over from the STM platform to the RPi, which also posed challenges as some STM32 libraries were non-existant on the RPi.

Tests with four nodes and the RPi were performed in order to determine whether it was still able to send out the first LSA_REQ to a child node. Since the function of the gateway router is a subset of the required functions of a given node, this test of a 4-node line topology was quite streamlined.

We then tested a 6-node line topology with the GW router implemented on the RPi, and started to experience more packet errors, which our stop-and-wait re-transmission scheme was still able to recover from.

3. **Line Topology of 8 Nodes**
In order to work up to a test of our final 8 sensor-node mesh topology, we tested a line topology with 8 nodes (one with the GW router implemented on an STM32, and another with the GW router implemented on the RPi). A run through all phases of the protocol stack took significantly longer, and had more failed runs than the 4-node line topology for two principal reasons:

   - A line topology represents the worst case timing for phase 2 (LSA). Given a line of consecutive nodes $[1, 8]$, since we are sending a separate LSA_RESP for each node in a given node's sub-tree, the number of packets sent grows exponentially as you approach the GW router node (address 1). More concretely, node $a \in [1, 8]$ must transmit $(9 - a)$ packets to its upstream parent.
   - Since more packets are being sent in general (without an ACK and re-transmission scheme), there were more opportunities for packets being dropped or CRC errors occurring; errors which were not being immediately addressed.

   Since this test took the longest out of all of the topologies we tested, there was more opportunity for channel interference from the CMU LoRa network. Essentially, since the implementation details of the SX1276 LoRa radio in the REYAX transciever modules were abstracted away to an AT-command set, we could not employ methods like carrier-sense or exponential backoff because we could not detect ongoing transmissions on the frequency bands that we wanted to use. The next step was to implement stop-and-wait for each packet sent in phases after TDM STP.

4. **Reliable Data Transfer** In order to test our stop-and-wait implementation and integrate reliable data transfer functions into the codebase, we ran tests between a pair of STMs, where one STM would send a packet, wait for an ACK from the other, and then the roles would change (i.e the original sender would be recieving a packet and ACK'ing it).
Unfortunately, we quickly noticed that although the tests were passing (calling rdt_send() with sequence number 40 on each end before external reset - 160 packets sent over-the-air in total), they were not consistent due to channel contention. Since the LoRa frequency bands are public use, there was no way to stop external transmissions at the frequencies we were running our tests on. Furthermore, without access to the internal SX1276 radio inside the LoRa transceiver module itself, it was complete guesswork to find a center frequency within the LoRa spectrum allocation (902-928MHz) without channel contention. Therefore, our tests for reliable data transfer, and consequently the integration testing phase, worked inconsistently due to ongoing channel contention at the time of testing.

5. **Final Testing** After having implemented and tested reliable data transfer, we transitioned to testing our final 8-node mesh topology (as shown on the web application in Fig.11) that we used in our demo. We configured the initial links for the topology in all 8 sensor nodes' memory. The logs collected at node 3 (printed via the STM's UART to a serial monitor) show the progression of node 3 through all phases of the protocol stack (see Fig. 14, 15 at the end of the report). Note the highlighted and red sections:

   - in Phase 1: the ports to neighboring nodes are opened (1) or blocked (0), which is the representation of the spanning tree from node 3's perspective.
   - in Phase 2: LSA requests are iteratively sent out to each of node 3's child nodes (note that a uni-directional link to node 5 has been disabled - underlined in the ports:[...] printout)

- in Phase 3: Node 3 receives its sub-schedule and further divides it before sending it off to its actual child nodes (according to the spanning tree i.e which ports are enabled)
- in Phase 4: Node 3 is executing the schedule in sync with all other nodes in the network, it receives in two slots, aggregates data (in this particular run no fires are lit at node 3 or its sub-tree, which is shown marked in red), then transmits in its DATA_TS.

7.2 Unit Tests: Node Architecture

The main unit testing done with the nodes themselves involved the temperature sensors and making sure they were calibrated correctly. The temperature sensors were first connected to the nucleo board and data was retrieved using the ADC. Before using the heat gun used later for testing, a hairdryer was used to safely simulate an increase in temperature. A timer was then added to ensure that an interrupt was triggered after a given wait time to take temperature sensor readings. This ensured that there was some temperature sensor data being calculated and in a given time frame to easily integrate this in with the data collection phase in the networking protocols. Temperature calculations were then done to convert the ADC readings to Celsius. Then, more unit testing was done to decide a threshold that would be used in the final integration to determine whether or not a fire had occurred at a node.

7.3 Unit Tests: Web Application

The web application went through many phases to get to its current iteration. Originally, the plan was to use GeoDjango but as stated in Section 5, the library was a little more complicated than necessary. The next step taken was to create an image map of the map and the nodes which would involve superimposing images over other images. That was being tested originally, until there were some bugs that we were running into with adding images into the Django application. This led to the pivot to use the Folium library to create the web application. The first iteration of the web application was simply adding the gateway router node and the eight other nodes and hardcoding a variable to signify the "fire node." That node would turn red based on what was set in the code. After the interim demo, we decided it would be beneficial to add the topology of the map and show the links between the nodes, so that was added to the web application. The last part of testing involved making sure that dead nodes were treated appropriately. In the case of a dead node, its links to its neighbors would also turn grey, similar to the inactive links in the spanning tree. Before moving the finalized code to the Raspberry Pi, test cases of different paths between the eight nodes were used to ensure that all cases were covered and appropriately displayed on the web application. This included making sure that all fire nodes, offline nodes, and all active spanning tree links were correctly shown to the user.

7.4 Integration Testing

Once the sensor database on the RPi was populated from running phase 4 of the protocol stack (DATA), we needed to get the web application (also hosted on the RPi) to display the results of this DATA phase. This required parsing the results of the protocol iteration (out of the original 8 sensor nodes, which ones detected a fire, and which ones were offline) into a JSON file. Once the JSON file was properly created from the simulation results, we ensured that the web application was able to read this JSON file and adjust its UI accordingly to show the updated results of the last-run iteration.

7.5 Results for Accuracy Testing

Before testing the accuracy of the network and how the system was able to relay the information of the network, iterative tests were completed. This included running tests between two nodes and then adding one or two nodes at a time until we were at a system of eight nodes. Once we were at our final topology, we ran a series of twenty tests in which we started all the protocols and either heated no temperature sensors or heated one or two temperature sensors. We found that 16/20 of the tests went through the set of protocols leading to an **80% accuracy** when it came to the network. The tests that failed often encountered packet errors and led to some of the nodes to get stuck and fail to transmit or receive packets. We also found that the tests ran smoother earlier in the day due to the interference of the on-campus LoRA network being used by other parties. Therefore, all the twenty tests were completed before noon.

7.6 Results for Power Testing

To test if the nodes is low power, we measured the voltage of a 43Ω shunt resistor connected in series to the Nucleo board. The voltage during run mode was **2.75V**, while the voltage during standby mode without the LoRa was **1.8V**. We then connected the shunt resistor in series with the LoRa and put the LoRa in sleep mode, and measured a voltage of **5.46mV**. This gave a current of **0.127mA** for the LoRa in sleep mode. This gave us a run current of **64mA** and an overall sleep current of **42.127mA**. It took **10 minutes** to run the protocols, so in an hour time-span the node would be in run mode for 20 minutes and standby for 40 minutes. This means the average current during the hour is

$$I_{avg} = 64mA * 0.33 + 42.127mA * 0.67 = 49mA \quad (1)$$

Since the battery would need to last a month, it would need to last 720 hours. $49mA * 720H = 35.28AH$. A 9V, **40AH battery** would be sufficient for powering one node for a month. This is much greater than our initial metric for the following reasons:

- We only calculated the power consumption of the STM32L4 chip itself (as we could not find power metrics for the entire Nucleo board)
- The STM32 Nucleo board's on chip ST-LINK debugger is always connected even when in a low-power mode, which led to a constant increase in power consumption regardless of the STM32's mode (Running or Standby), which we did not account for in our initial power calculations

7.7 Summary of Testing Results

Refer to Table 1 on Page 12 for the final testing results.

8 PROJECT MANAGEMENT

8.1 Schedule

Refer to Fig. 12. on page 13 for our semester schedule and the division of our work. Our schedule did shift from the initial creating of the Gantt chart. We found that integration of the individual parts ended up taking longer than we had expected. Also, unexpected bugs caused some backlog in the testing part of our schedule.

8.2 Team Member Responsibilities

- Arden: I was responsible for the design and implementation of the network protocol stack: STP, LSA, SCH distribution, and DATA on both the RPi and STM32 platforms. I covered communication between the STM and the LoRa modules (via UART interrupts) and communication between the RPi and the LoRa modules (via the termios UART library). Using timer interrupts on the STM, and standard C time functions on the RPi, I implemented transceiver functions to send and receive packets (blocking and non-blocking mode), as well as the ACK and retransmission scheme used by the LSA, SCH, and DATA phases.
- Ankita: I was responsible for the web application interface. I hadn't taken the class here at the university prior so it was a new experience having to work with Django and designing the web application. I also helped create the RX/TX schedule that was distributed to the nodes. This also included creating a order for which the nodes would be receive the schedule. In addition, I helped with testing and debugging wherever necessary.
- Karen: I was responsible for implementing the Data Phase on the STM32s, and having the nodes execute the schedules received during the Schedule Phase. I ensured that nodes closely followed the schedule, so that there was no overlap in transmissions and that no transmissions would be missed. I also made sure a large change in temperature would be detected by the STM by polling the TMP36. I implemented the code for the gateway to write the data received from the network into a JSON file for WebApp. I also helped with testing and debugging.

8.3 Bill of Materials and Budget

Refer to Table 2 on Page 13 for the bill of materials.

8.4 Risk Management

One of the risks our project faces is that the timestamps received by the nodes from the border gateway will not be enough to synchronize the clocks to account for clock drift, which will cause nodes to miss their TX/RX slot. If this becomes a problem, we have implemented using ACKs to ensure the packets are received as intended. We also found that our batteries would drain when testing our project, so we had to sometimes manage that by using wires to connect in between the nodes and use a singular power source. We eventually had no need for that once we purchased new batteries.

9 ETHICAL ISSUES

Our product relies on its availability to provide early wildfire detection to fire departments. If a backpacker or a camper was to build a small campfire close to a node, the wind might blow smoke towards a local sensor, causing that node's sensors to react as if a real forest fire were starting. Furthermore, a malicious adversary could light a fire directly under a node to trigger a false positive, effectively calling wildland firefighters to their location.

Such misuse of our system evokes the question of how the fire department should react to a node detecting a fire or dropping offline. If the fire department is informed of a conflagration at node X via the web interface, then they should check via a lookout tower if smoke is even present. If the lookout notices a column of smoke, then a crew of wildland fire-fighter could be dispatched to that node's location. Although this would provide some robustness against a malicious adversary falsifying a fire, it does not fully address the issue, as such an adversary could simply create a bigger fire, to warrant the fire department's attention. Nevertheless, if a node drops offline, the severity of the issue is much less than a node detecting a fire since there could be several root causes of a node dropping offline, namely: a falling branch knocks the node off of its tree, and the antenna cannot receive signals from in-range nodes, or if the node simply runs out of battery power (after a month of operation). Nevertheless a node dropping offline could indicate a fire if the epicenter of the fire is at the node, and if the node is engulfed in the flames before reaching its DATA phase of the protocol stack (before it

takes a temperature reading).

Essentially, responsible use of this early wildfire detection system recommends a secondary traditional evaluation of fire risk (i.e. lookout towers, or local scouts in the area) at the time of fire detection in order to mitigate the amount of false positives. If it has been approximately a month since the last maintenance check, and a node drops offline, and no fire is visually detectable, then it is likely that a node's offline state is due to a lack of battery charge, or that it fell off of its tree. This scenario would necessitate the dispatch of maintenance technician personnel rather than an entire wildland forest fire crew.

We require such a high level of accuracy due to the disastrous consequence of false negatives, and the inconvenience of false positives. If a malicious adversary were to build a jammer by the gateway (blocking the 915MHz LoRa frequency band), they could effectively cause the system to be inoperable. Although the LoRa frequency band is free for public use according to the FCC, this would still count as deliberate misuse, and regulation would have to prohibit transmission on the LoRa frequency spectrum allocation within X kilometers of the wireless sensor network gateway router.

10 RELATED WORK

There are other projects that are documented about creating a wireless sensor networks. For example, there is study completed by researchers in the University of the Coast in Barranquilla where they also created a wireless sensor network for fire detection [16]. This project focuses more on the sensing aspect of the fire detection system and how to determine if a fire is at a location. There are also projects that use machine learning to take the aggregate data to predict the future locations of fires [1]. There are lots of uses of satellite imaging to view the locations of natural disasters like floods and fires [17]. In comparison to those projects, due to our use case as well as the constraints of budgeting and time, our project focuses on the network as we wanted to create a accurate and low power system for fire detection. We wanted to ensure lower maintenance of the system by firefighters. Our project also focuses mainly on the network and how to adjust that architecture to make our project most efficient. More than determining the fire, we are focusing on whether the location of the fire can be relayed correctly.

11 SUMMARY

In our project, we hoped to create a low power and accurate system for wildfire detection. Our system consisted of a wireless sensor network of 8 nodes that transmit data regarding the locations of the fire. This data would be displayed on a web application for wildland firefighters. Our system was intended to be low power, leading to low maintenance as well as high accuracy. The system would help pinpoint the fire's location. Our system didn't end up meeting all our requirements that we had set out for. Using a LoRA network while on CMU campus led to some issues achieving our intended accuracy for fire detection; however, we were able to get the notification of the fire's location within our requirements. Regarding the low power nature of our system, we were able to measure the power consumption of our design though it was not as low as we had hoped. As expected, integration of the individual parts proved to be a challenge. If our team had more time, we would look into testing where there is no interfering LoRA network as that wouldn't apply in the context of a forest. We would also make sure to find hardware that is well-documented and reliable.

Groups working on similar wireless network projects should ensure that the channel they intend to use is available and contention-free for a pre-determined time period, during which they will be able to run tests. This particular issue caused sporadic experiment failures and let to such a hassle in development and testing. There is a pre-exisitng LoRa network on CMU campus, and since the 915Mhz frequency band is public-use (as per FCC regulations), anyone with a radio can broadcast whatever message they want at whatever time suits them on the bandwidth you plan to test your project on!

Glossary of Acronyms

- RPi – Raspberry Pi
- TS - the timeslot duration in seconds.
- TDMA - Time Division Multiple Access
- MAC - Medium Access Control
- LSA - Link State Advertisement
- LS - Link State
- LoRa-WAN - Cloud-based MAC protocol designed for the LORA-PHY (physical) layer

Table 1: Testing Results

Description	Requirement	Results
Fire Accuracy	90 - 95%	80%
Notification Timing	30 mins	10 mins
Low Power	1 mo. maintenance	40AH battery required

Table 2: Bill of materials

Description	Model #	Manufacturer	Quantity	Cost @	Total
Temperature Sensor	TMP36	Adafruit	8	$2.75	$22.00
Breadboards	3.25"x2.2"	Adafruit	9	$4.95	$44.55
Nucleo Board	NUCLEO-L412KB	DigiKey	10	$11.00	$110.00
LoRA Transceiver	RYLR896	REYAX	12	$19.50	$234.00
9V Batteries	Pack of 8	Amazon	1	$12.99	$12.99
9V Battery Clip	5.5mm/2.1mm plug	Adafruit	8	$2.95	$80.00
Barrel Jack	2.1mm	Adafruit	8	$0.95	$7.60
USB cables	USB Micro B	Amazon	1	$13.99	$13.99
USB expansion port	7-port USB 3.0	Amazon	1	$21.99	$21.99
				with shipping costs :	$547.12

- ToA - Time-on-air: time from when a signal is transmitted by a sender until the receiver receives it.
- DC_TS - Data Collection Timeslot (seconds)
- DATA_TS - Data Phase Timeslot (seconds)
- TX_TS - Transmission Timeslot (seconds)
- RX_TS - Reception Timeslot (seconds)

References

[1] U. Dampage, L. Bandaranayake, and R. et al. Wanasinghe. "Forest fire detection system using wireless sensor networks and machine learning". In: *Sci Rep* 12 (2022), p. 46.

[2] Ana Elisa Ferreira et al. "A study of the LoRa signal propagation in forest, urban, and suburban environments". In: *Annals of Telecommunications - annales des télécommunications* (July 2020). DOI: ff10.1007/s12243-020-00789-wff.ffhal-02907283f.

[3] *Folium Library*. URL: https://python-visualization.github.io/folium/.

[4] *GeoJson*. URL: https://geojson.io/.

[5] *How Long is the Transmission Distance of the 4G Wireless Router?* URL: https://www.szceres.com/how-long-is-the-transmission-distance-of-the-4g-wireless-router.html.

[6] Shawn Hymel. *Getting Started with STM32 - Timers and Timer Interrupts*. URL: https://www.digikey.com/en/maker/projects/getting-started-with-stm32-timers-and-timer-interrupts/d08e6493cefa486fb1e79c43c0b08cc6.

[7] Arden Diakhate-Palme Karen Abruzzo Ankita Bhanjois. *Team A1: FireAway*. https://github.com/AnkitaBh/A1-FireAway. 2022.

[8] *LoRa*. URL: https://lora.readthedocs.io/en/latest/#:~:text=When%20a%20signal%20is%20send,device%2C%20or%20system%20is%20operated.

[9] *LoRa Calculator*. URL: https://www.semtech.com/search/results_documents/lora%20calculator*.

[10] *Low power modes in STM32*. URL: https://controllerstech.com/low-power-modes-in-stm32/.

[11] *Low Voltage Temperature Sensors*. D00337-0-5/15. Rev. H. Analog Devices.

[12] Bradley Mitchell. *What Is the Range of a Typical Wi-Fi Network?* URL: https://www.lifewire.com/range-of-typical-wifi-network-816564.

[13] *STM32 ultra low power mcus*. URL: https://man7.org/linux/man-pages/man3/termios.3.html.

[14] *termios(3) Linux Manual Page*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32-ultra-low-power-mcus.html.

[15] *Ultra-low-power Arm Cortex-M4 32-bit MCU+FPU, 100DMIPS, up to 128KB Flash, 40KB SRAM, analog, ext. SMPS*. DS12469. Rev. 8. STMicroelectronics. Nov. 2020.

[16] Noel Varela et al. "Wireless sensor network for forest fire detection". In: *Procedia Computer Science* 175 (2020). The 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC),The 15th International Conference on Future Networks and Communications (FNC),The 10th International Conference on Sustainable Energy Information Technology, pp. 435–440. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.07.061. URL: https://www.sciencedirect.com/science/article/pii/S1877050920317427.

[17] Zhe Zhu and Su Ye. "These AI and satellite mapping techniques are speeding up the process of disaster management". In: *World Economic Forum* (Oct. 2022).

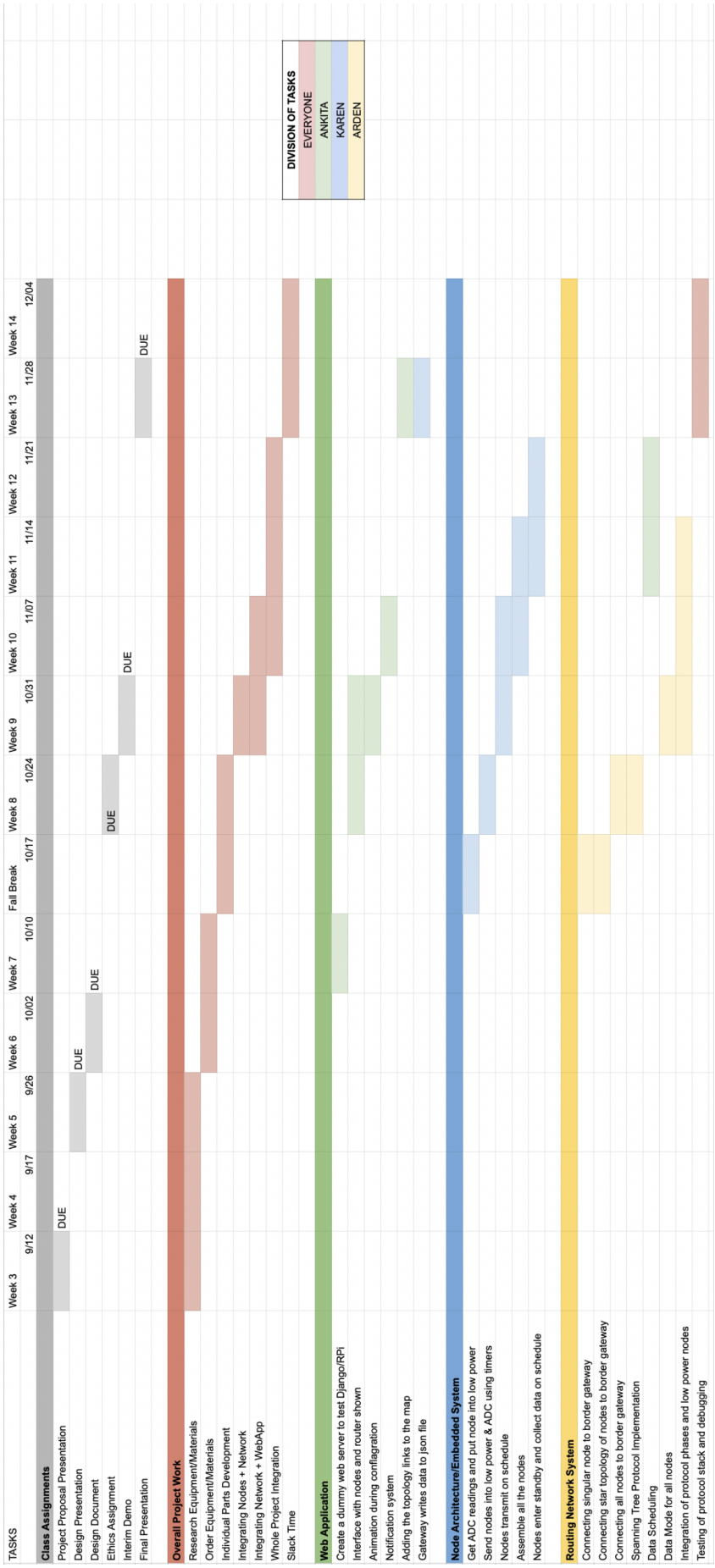


Figure 12: Gantt Chart

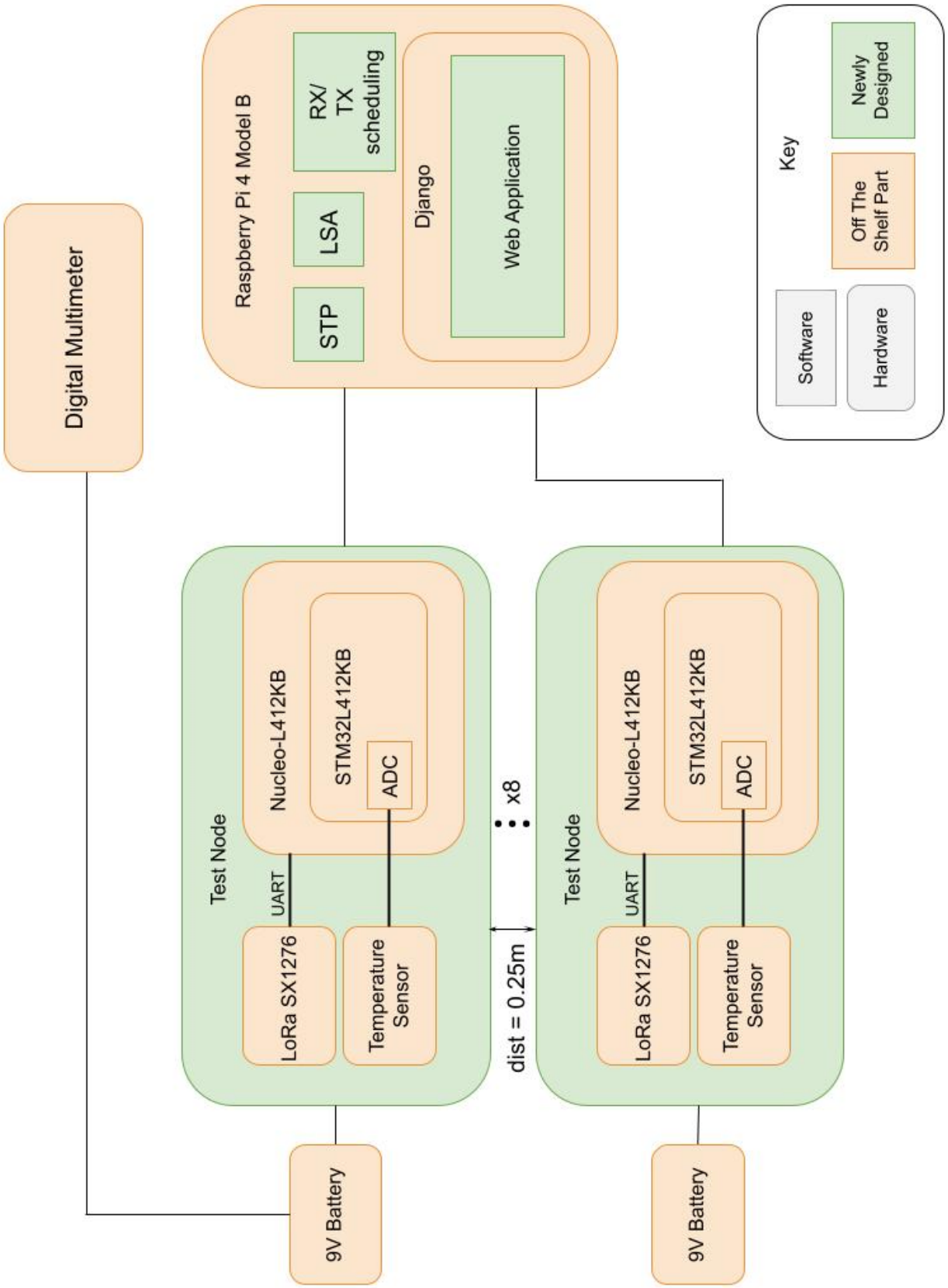


Figure 13: Full-Size Block Diagram

```
[Node 3]
AT+RESET
AT+ADDRESS=3
+OK
[configuration done]
AT+PARAMETER=10,9,1,7
+OK
AT+BAND=863000000
+OK
AT+CRFOP=15
+OK
AT+MODE=0
+OK
[initialization done]
time: 2536
```

```
============ Phase 1: STP ==========
[iteration 1, timeout: 12000]
+RCV=1,180,
[3] received STP packet: (1, 0, 1)
 [iteration 1, timeout: 7491]
+RCV=2,180,
[3] received STP packet: (1, 1, 2)
 [iteration 1, timeout: 3470]
+OK
[3] broadcasting STP update: (1, 1, 3)
[iteration 2, timeout: 28000]
+RCV=4,180,
[3] received STP packet: (1, 1, 4)
 [iteration 2, timeout: 23967]
+RCV=5,180,
[3] received STP packet: (1, 2, 5)
 [iteration 2, timeout: 19981]
+RCV=6,180,
[3] received STP packet: (1, 2, 6)
 [iteration 2, timeout: 15969]
+RCV=7,180,
[3] received STP packet: (1, 2, 7)
 [iteration 2, timeout: 11968]
+RCV=8,180,
[iteration 2, timeout: 7976]
+RCV=9,180,
[iteration 2, timeout: 3962]
ports: [ 1 2 4 5 6 7 ]
       [ 1 0 0 1 1 1 ]
3: [ 1 2 4 5 6 7 ]; (6)
```

```
============ Phase 2: LSA ============
+RCV=1,180,
+OK              (ACK)
Checking for retransmissions
No retransmissions detected
[3] forwarding LSA REQ to 5
+OK
+RCV=5,180,     (ACK)
Checking for retransmissions
No retransmissions detected
+RCV=5,180,
+OK              (ACK)
[3] complete LSA response received from 5
[3] forwarding LSA REQ to 6
+OK
+RCV=6,180,     (ACK)
+RCV=6,180,
+OK              (ACK)
Checking for retransmissions
No retransmissions detected
+RCV=6,180,
+OK              (ACK)
Checking for retransmissions
No retransmissions detected
[3] complete LSA response received from 6
```

```
============ Phase 2: LSA (cont'd)======
[3] forwarding LSA REQ to 7
+OK
+RCV=7,180,     (ACK)
+RCV=7,180,
+OK              (ACK)
Checking for retransmissions
No retransmissions detected
[3] complete LSA response received from 7
[3] Sending LSA Response
3: [ 1 2 4 5 6 7 ]; (6)
6: [ 3 9 ]; (2)
9: [ 6 ]; (1)
7: [ 3 ]; (1)
+OK
+RCV=1,180,  (ACK)
... 2 times ...
+OK
+RCV=1,180, (ACK)
[3] sent complete LSA_RESP
ports: [ 1 2 4 5 6 7 ]
       [ 1 0 0 0 1 1 ]
3: [ 1 2 4 5 6 7 ]; (6)
6: [ 3 9 ]; (2)
9: [ 6 ]; (1)
7: [ 3 ]; (1)
```

Figure 14: Node 3 Logs in final 8-node MESH topology

```
============= Phase 3: SCH =============
+RCV=1,180,
+OK           (ACK)
Checking for retransmissions
No retransmissions detected
[3] received schedule from node 1
RX: [ 6 5 3 3 2 0 0 0 ]
TX: [ 9 8 7 6 5 4 3 0 ]
schedule receive order: [ 2 3 4 5 6 7 8 9 ]
[3] sending schedule to 6
+OK
+RCV=6,180, (ACK)
[3] waiting for SCH ACK from 6
+RCV=6,180,
+OK         (ACK)
Checking for retransmissions
No retransmissions detected
[3] received SCH ACK from node 6
[3] Node 6 has its sub-schedule
```

```
========== Phase 3: SCH (cont'd) =====
[3] sending schedule to 7
+OK
+RCV=7,180, (ACK)
[3] waiting for SCH ACK from 7
+RCV=7,180,
+OK         (ACK)
Checking for retransmissions
No retransmissions detected
[3] received SCH ACK from node 7
[3] Node 7 has its sub-schedule
+OK
+RCV=1,180, (ACK)
[3] sent SCH ACK to node 1
TX TS: 6
RX TS: [ 2 3 ]
total TS: 8
sleep for: 0 TS
current time: 244939
```

```
============= Phase 4: DATA ============
Executing the schedule
timeslot: 0
IDLE
timeslot: 1
IDLE
timeslot: 2
receive slot: 2
receiving...
+RCV=7,180,
+OK             (ACK)
Checking for retransmissions
No retransmissions detected
[3] received DATA packet from node 7
timeslot: 3
receive slot: 3
receiving...
+RCV=6,180,
+OK             (ACK)
Checking for retransmissions
No retransmissions detected
[3] received DATA packet from node 6
timeslot: 4
IDLE
```

```
============= Phase 4: DATA (cont'd) ====
timeslot: 5
IDLE
timeslot: 6
sample 0: 831
sample 209545: 813
Temperature increased by 18
Nodes on Fire: []
Responsive Nodes: [3 7 6 9 ]
transmitting data to 1
+OK
+RCV=1,180,   (ACK)
timeslot: 7
IDLE
Done with protocol stack.
going to sleep for 1374848 seconds
AT+MODE=1
+OK
+OK
```

Figure 15: Node 3 Logs in final 8-node MESH topology (cont'd)