An Advanced Networking Outreach Activity for Kids

DESIGN DOCUMENT

sddec20-05

Dr. Tom Daniels

Grayson Cox | UI Developer | Agile Project Manager

Austin Dvorak | Network Systems Admin

Ryan Newell | Hardware Systems Admin

Spencer Parry | UI Developer

Ross Thedens | Communication System Manager

Malcolm Johnson | Hardware Engineer

Team Email: sddec20-05@iastate.edu

Team Website: http://sddec20-05.sd.ece.iastate.edu/

Revised: Nov 15, 2020 (V4)

Executive Summary

Development Standards & Practices Used

- Standards
  - Quality management (ISO 9001)
  - Developing information for users in an Agile environment (ISO/IEC/IEEE 26515)
- Practices
  - Agile development
  - Code reviews
  - Weekly meetings
  - Rigorous testing for security and reliability

Summary of Requirements

- Support wireless mesh and dynamic ad-hoc networking capabilities
- Setup and use of mesh network requires minimal user input
- Ability to communicate (i.e. request data) between an instructor's computer and mesh nodes
- Ability to stream video and sensor data from mesh nodes to an instructor's computer
- Ability to send network statistics and system information from mesh nodes to an instructor's computer
- User interface on instructor's computer must display a list of nodes and display the nodes' data
- Launching user interface requires minimal user configuration
- Node hardware must be mobile and durable
- Nodes should be rechargeable, with a minimum two-hour battery life
- Nodes must be able to operate without an Internet connection
- Range between nodes should support a 30 foot radius (when in eye-sight)

Applicable Courses from Iowa State University Curriculum

- CprE 489 - Computer Networking and Data Communications
- CprE 430 - Network Protocols and Security
- CprE 537 - Wireless Network Security
- CprE 230 - Cyber Security Fundamentals
- ComS 252 - Linux Operating System Essentials
- SE 319 - Construction of User Interfaces
- SE 329 - Software Project Management
- ComS 309 - Software Development Practices

New Skills/Knowledge acquired that was not taught in courses

- Web Development with Angular/Typescript
- Ad-Hoc/Mesh Networking and Router Configurations
- Raspberry Pi Configuration
- WebRTC and Video Streaming Protocols

Table of Contents

1. Introduction **5**

1.1 Acknowledgement 5

1.2 Problem and Project Statement 5

1.3 Operational Environment 5

1.4 Requirements 6

1.4.1 Engineering Constraints and Non-Functional Requirements 6

1.5 Intended Users and Uses 7

1.6 Assumptions and Limitations 7

1.7 Expected End Product and Deliverables 8

2. Specifications and Analysis **8**

2.1 Proposed Approach 8

2.2 Design Analysis 9

2.3 Development Process 9

2.4 Conceptual Sketch 11

3. Statement of Work **14**

3.1 Previous Work And Literature 14

3.2 Technology Considerations 15

3.3 Task Decomposition 15

3.4 Possible Risks And Risk Management 17

3.5 Project Proposed Milestones and Evaluation Criteria 18

3.6 Project Tracking Procedures 19

3.7 Expected Results and Validation 19

4. Project Timeline, Estimated Resources, and Challenges **19**

4.1 Project Timeline 19
4.2 Feasibility Assessment 24
4.3 Personnel Effort Requirements 25
4.4 Other Resource Requirements 25
4.5 Financial Requirements 26
5. Testing and Implementation **26**
5.1 Interface Specifications 26
5.2 Hardware and software 27
5.3 Functional Testing 27
5.3.1 Unit Tests 27
5.3.2 Integration Tests 27
5.3.3 System Tests 27
5.3.4 Acceptance Tests 28
5.4 Non-Functional Testing 28
5.4.1 Performance Tests 28
5.4.2 Security Tests 28
5.4.3 Usability Tests 28
5.4.4 Compatibility Tests 29
5.5 Process 29
5.6 Results 30
6. Closing Material **33**
6.1 Conclusion 33
6.2 References 33
6.3 Appendices 34
6.3.1 Operational Manual 34
6.3.1.1 Overview of Deployment 34
6.3.1.2 Common Node Deployment 34
6.3.1.2.1 Building 34
6.3.1.2.2 Starting 34

6.3.1.3 Network Master Node Deployment 35
6.3.1.3.1 Building 35
6.3.1.3.2 Starting 35
6.3.1.3.3 Stopping 35
6.3.1.3.4 Viewing Logs 35
6.3.1.4 Video Mesh Node Deployment 35
6.3.1.4.1 Preparing the Raspberry Pi 35
6.3.1.4.2 Building 36
6.3.1.4.3 Running 36

Figures and Tables

Figures

1: Conceptual Sketch 11
2: System Modules 13
3: Gantt Chart CprE 491 (01/19 - 04/04) 20
4: Gantt Chart CprE 491 (04/05 - 05/02) 21
5: Gantt Chart CprE 492 (08/23 - 10/24) 22
6: Gantt Chart CprE 492 (10/25 - 12/12) 23
7: Poor Video Quality 31
8: Acceptable Video Quality 32

Tables

1: Task Estimated Time to Completion Table 25

1. Introduction

1.1 Acknowledgement

We would like to acknowledge our client and advisor Dr. Tom Daniels for his guidance and support on this project. We would also like to acknowledge the help of ETG staff for helping us get our hardware deliveries, especially during the campus closure during the pandemic.

1.2 Problem and Project Statement

Wireless networking has become a fundamental part of many people's everyday lives. This is especially true for students across grades 4-12, who are beginning to spend hours each day on school-issued internet-connected devices. However, few students understand the technology that underpins these networks. The internet is often portrayed as an amorphous portal granting access to huge amounts of information. In reality, this information is stored on servers around the world and delivered to end users via routing protocols. By creating a teaching tool that demonstrates routing protocols, we can illustrate the challenges involved in creating a reliable wireless network, increase technology literacy, and stimulate students' interest in computer networking careers.

Our approach involves creating a teaching toolkit consisting of a set of portable wireless network nodes, a network monitor/configuration application, and a set of lesson plans. A grade school instructor with no computer networking training will be able to operate it. The nodes will automatically form an ad-hoc network with one another when powered on (and within range of one another). Network data will be provided by video cameras and various sensors on the nodes, such as temperature sensors. The data will travel the network to reach a master node which is directly connected to a network monitor application running on an instructor's desktop or laptop (referred to as "station" in this document). The monitor application will control which sensor/video streams are delivered to the master node and display the data received. The lessons will challenge students to transmit data across a long distance (beyond the range of a single node; from the main office to a classroom, for instance) to the network monitor application. As students add nodes into the network and position them to attain connectivity, they will observe the routing of the data (which nodes are reached) in the monitor application. They will use this information to figure out where data is unable to travel and rearrange the nodes accordingly.

Ultimately, our toolkit will act as a miniature mock-up of the real internet. Students will be able to appreciate how data from a web server hops from node to node over the internet before reaching their devices. Following the activities, the instructor will discuss grade level-appropriate topics related to networking from the lesson plan with the class. For younger students, this may involve the fact that wireless signals have a limited range, while for older students it may include a discussion of the routing decisions made by the ad-hoc protocol. Our project will provide a flexible way to teach computer networking to grade school students.

1.3 Operational Environment

A collection of networking nodes will compose the main aspect of the system. It is expected that these nodes will be used indoors, but they should be able to operate outdoors as well. They do not need to be weatherproof; however, they will need to be durable enough to protect the internal

components. The environment is expected to be a public area, such as a school, with many wireless enabled devices. The only node that must connect directly to the local wireless network is the controller node, while the others only interact within the node network. Common wireless network protocols will suffice for effective inter-node communication.

The system does not require special considerations for physical obstacles, and perhaps should be weaker than average Wi-Fi devices, as obstacles will help demonstrate the propagation of the wireless signal.

The primary users of this system will be grade school students, who will not be familiar with the system or safe electronic handling procedures. Additionally, the users are expected to carry the devices around an open space. The devices require measures to prevent tampering by wandering fingers, best achieved by a fully enclosed case secured with screws, as well as sufficient shock resistance to continue functioning if dropped.

There will be a graphical user interface aspect to the system, which will allow users to view sensor and video data transmitted from the nodes. This can be accessed via any web browser on a separate computer, and no internet connection should be necessary as long as there is a connection between the instructor station and the Network Master Node.

1.4 Requirements

The problem definition leads to several core functional requirements. First, each node must support ad-hoc networking capabilities. Nodes need to connect to one another automatically when within range. If a node is moved out of range and back into range, it must reconnect automatically. Multiple networks of nodes should be able to coincide and stay independent of each other.

Another requirement is the ability to communicate between an instructor's computer and the mesh nodes. The instructor's computer will not be connected to the mesh network directly. Instead, we must create a "master" node for the network with an additional network interface that the instructor connects to.

Several types of data must be transmitted through the network to the instructor's computer. Among these are video streams from cameras attached to certain nodes, sensor data streams from a variety of sensors (temperature, etc.) attached to certain nodes, and network statistics (Packet loss, etc.) and system information (RAM/CPU usage, etc.) from each node.

To effectively visualize the network, the instructor must have access to a user interface for the network. The user interface must display the list of nodes in the network. When a node is selected, the user interface program will send the appropriate signals to the network to obtain the node's data.

1.4.1 Engineering Constraints and Non-Functional Requirements

The project has several key constraints based on the assumed experience level of the users and the practical usage of the nodes.

- The students and instructor will likely have little or no knowledge of Linux systems and using a command line interface. Therefore, the mesh network setup must not require the user to obtain access to the command line interfaces of any of the nodes.

- The instructor may have limited rights to install software on their computer, or may be averse to installing unknown software on their computer. Therefore, the user interface application must not require the instructor to install any dependencies on their computer.
- Finally, the nodes must be mobile so that students can carry them throughout their environment. This requires the use of a small, mobile computing platform that can run on battery power.

Several non-functional requirements provide performance requirements for the system.

- Nodes shall be powered by rechargeable batteries that provide at least a two-hour operational time per charge.
- Nodes shall be able to form a mesh network when booted without having to connect to the internet or any existing network.
- Nodes shall have a connection radius of at least 30 feet when in a direct line-of-sight orientation. All node connections shall be wireless.
- Nodes shall be packaged in a durable, drop-proof casing. This includes protection for the power supply, as well as for any peripherals (camera/sensors).
- Recharging the batteries shall be an easy and intuitive process that does not require disassembling the case of the node.

1.5 Intended Users and Uses

The project has three main types of users: students, instructors, and system administrators. The goal of the project is to educate students on how mesh networking works, therefore, the students would be the target audience. The students would not interact with the GUI as much as the instructors or teachers would, but the GUI should be user friendly enough for anyone in the three target audience groups to use. The students largely will be moving or placing the nodes in different places, and the instructors will use the GUI to interpret the data for the class.

The instructors would have knowledge on how to use the nodes and the GUI so they can teach their students general concepts about WiFi such as range and interference. They would know how to position the nodes to display on the GUI how different forms of interference can cause the video to buffer more or less than other forms of interference. The instructor would also use the GUI to change settings on the nodes, like grouping nodes into two separate groups.

The system administrators would have access to the file system and command line of the nodes to manage updates or to diagnose problems with the nodes. The administrators would only have to be involved if there is a problem with one of the nodes, or a mandatory update needs to take place. Students and instructors would not have access to the system administrator settings; however, system administrators would have full system access, as well as access to the student and instructor GUI.

1.6 Assumptions and Limitations

Assumptions

- The user owns a station computer with a web browser.
- The user's environment does not abnormally impede WiFi connectivity.
- The person running this activity is able to charge the nodes when batteries run low.

Limitations

- Internet connection may not be available; nodes must have any software/dependencies pre-installed.
- Each node will be powered by a single-board computer to ensure mobility.
- Nodes will need to be stored in a compact space.
- Battery life may be limited to two hours.

1.7 Expected End Product and Deliverables

The main deliverable consists of a set of at least eight wireless, rechargeable nodes. Some of these are "video" nodes with camera modules attached. Some are "sensor" nodes with sensors (such as a temperature sensor) attached. Others are "relay" nodes that simply carry traffic through the network. These three types of nodes are collectively referred to as "mesh" nodes. A single Network Master Node runs the Backend Application and the User Application and routes data between the UI and network nodes. Sensor data, camera streams, and network statistics (i.e route traces to particular nodes, packets dropped, etc.) are transmitted through the network and displayed on the UI in charts and graphs; camera frames are displayed as a video stream. We were able to deliver the functioning network master node by October 6, with the Relay and Sensor Mesh Nodes delivered on November 8.

The User Application component is in the form of a web-based application that can be accessed through any web browser. The UI offers two types of displays: one that shows an overview of the network and what is connected to it, and another that goes into detail on a specific node when selected. The full UI was delivered on November 10.

2. Specifications and Analysis

2.1 Proposed Approach

Recalling the Summary of Requirements, there are some major objectives that this project needs to meet. The most limiting requirement is the budget. Due to wanting to keep each node at $65, limits the hardware. This means that our approach to what operating system, networking protocols, etc. we use changes due to the limited processing power of the device.

A single board computer was deemed the correct choice due to the low cost ~$35 per board, but also the higher processing power and I/O abilities over embedded boards, along with the high mobility. To increase this mobility, a battery pack would be included to allow the nodes to work without wall power. The next step was deciding how the networking would be handled. Certain distributions of Linux aimed at computer networking come with support for mesh networking protocols, so the decision has to be made about which one to use. These distributions should allow for automatic connecting and disconnecting of links to aid with ease of use. A web server would be set up on one of the links to allow for a cross-platform GUI that would show all of the required information coming from the network, along with easy configuration of the network and nodes.

2.2 Design Analysis

Deciding on the hardware of the project seemed to be relatively easy. Due to the widespread support and price of the board, Raspberry Pi's will be used as the nodes. This also makes figuring out which mesh-networking protocol and linux distribution easy. After research, it was found that IPFire and OpenWRT were the best distributions to use, as there were already examples of these distributions working on the Raspberry Pi. Selecting the ad-hoc networking protocol was the next step.

There are 3 main types of protocols: proactive, reactive, and hybrid. Proactive protocols create a list of connections and their routes at the expense of latency when adding and subtracting nodes, while Reactive floods the network with packets at the expense of latency when sending packets. Hybrid routing creates a table initially and floods only when it cannot find a path to the destination. Due to how much bandwidth flooding a network stream with video packets would require, the decision was made to go with a proactive protocol.

There are 5 main proactive routing protocols: OLSR, Babel, DSDV, DREAM, and BATMAN. BATMAN and OLSR are the two most prominent protocols and are both supported by OpenWRT and IPFIRE as well as Raspberry Pi OS. After initial testing of the system on OpenWRT it was deemed infeasible and we switched to Raspberry Pi OS, as we could not get mesh networking to work with raspberry pis on OpenWRT.

For the user application component, it was decided that Angular/Typescript would be used for the UI due to the ease of use and since the members working on the UI have had previous experience in Angular development. To handle our requirement for video streaming and possibly other forms of live data, we will be using WebRTC, which is an open-source framework that provides real-time communication capabilities in a web browser. With WebRTC, we are able to provide the best video streaming quality with the lowest latency of all the strategies with which we experimented.

Lastly, we have decided that the backend to the User Application would be in the form of a Java Spring Boot application running on the Network Master Node. This "Backend Application" will work with the mesh network directly and will provide an API for the user application to fetch information about the mesh network, such as the list of active nodes, information about each node, network statistics, and sensory data from Sensor Mesh Nodes. By choosing to have a Spring Boot application serving as the interface between the network and the user application, we are also able to take advantage of its web socket capabilities to serve as the signalling server for live media streams via WebRTC.

2.3 Development Process

Because of the lack of certainty in the design of this project, our team requires an iterative development method that allows for design decisions to take place at any point in the project lifecycle. And given the vagueness of our initial project description, we need an approach that will assist us in reaching our final design starting with a high-level understanding of the project. Therefore, we will conform to an Agile development process with a top-down design approach because it mitigates the risks associated with the inherent variability of this project.

The development lifecycle of our project will consist of a number of sprints, each lasting two weeks. Prior to each sprint, the team will hold a sprint planning meeting in which we choose

tasks/features to implement during the coming sprint. Following each sprint, the team will meet for a sprint retrospective meeting, where we will discuss our progress and think critically about our performance.

We will be using the tools provided by GitLab for our artifacts. Individual units of work (i.e., tasks, features, or user stories) will be represented as Issues in GitLab, which can be created by and assigned to any team member. Each Issue is associated with a single Git branch, so Issues are loosely coupled, allowing team members to work independently.

To control the quality of our implementations, all code will go through peer reviews before going into production. These code reviews will be facilitated via merge requests in GitLab. Once a team member finishes work on an Issue, he will open a merge request and assign at least one other team member to review the work.

To track the progress of the team, Issues will appear on sprint boards. We will use different boards to denote the status of every Issue. The board in which an Issue resides will indicate whether it is *Open*, *InProgress*, *InReview*, or *Closed*.

2.4 Conceptual Sketch

Our conceptual sketch of the project is shown in Figure 1.

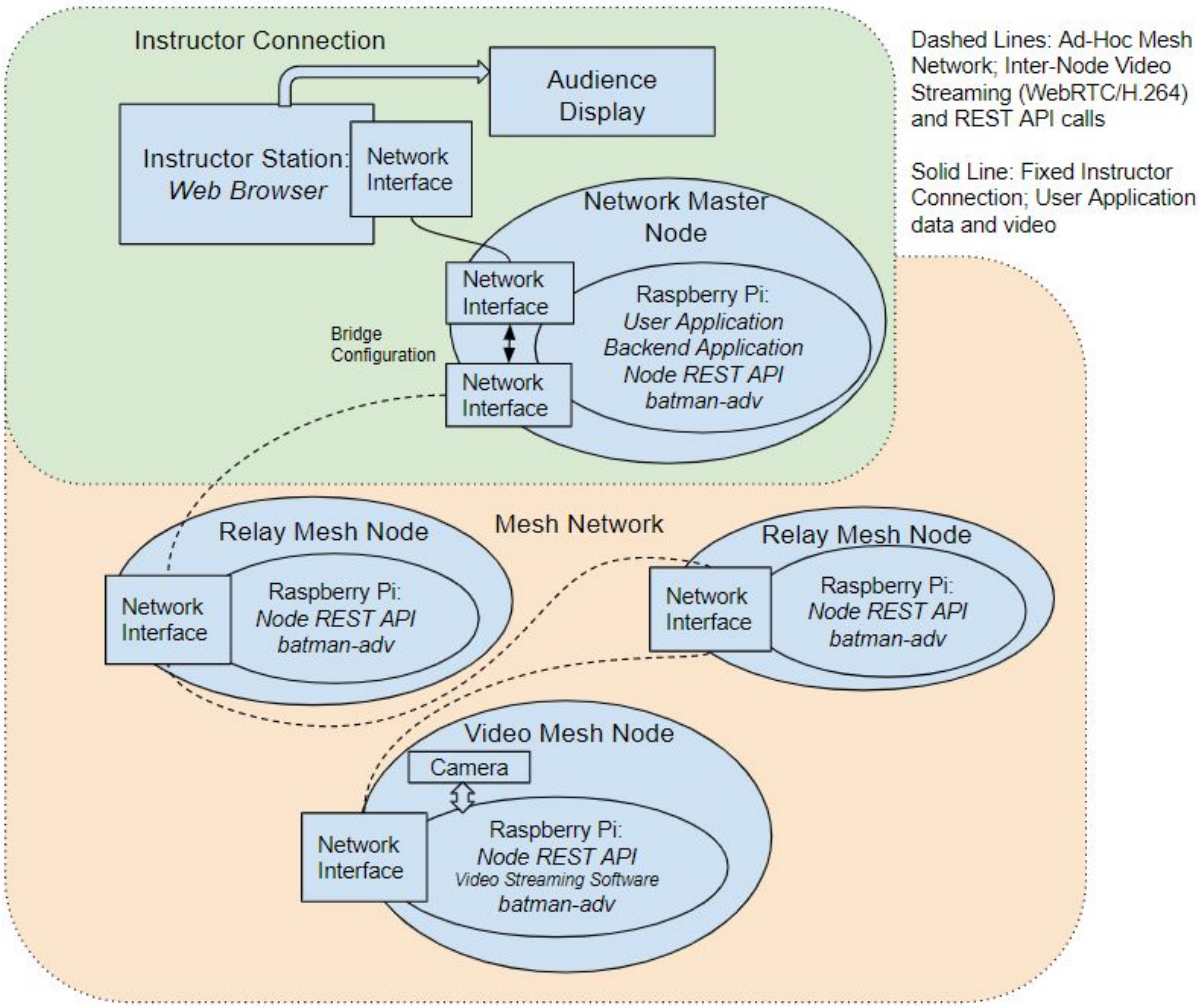


Figure 1: Conceptual Sketch

Our project involves an instructor station and several types of network nodes, including the Network Master Node, Relay Mesh Nodes, and Video Mesh Nodes. The instructor station consists of an instructor's existing laptop, which must be able to run a common web browser (e.g. Chrome, Firefox, etc.) and connect to a WiFi network. The Network Master Node consists of a Raspberry Pi single board computer running the Raspberry Pi OS (formerly Raspbian). The Relay Mesh Nodes consist of the same hardware and operating system as the Network Master Node. The Video Mesh Nodes have the same hardware and operating system as well, but with the addition of a camera module.

The nodes and instructor station communicate using two networks: an instructor connection and a mesh network. The instructor connection is a fixed, wireless network connection between the

instructor station and the Network Master Node. The Network Master Node has a second wireless network interface that connects with the wireless network interface on other nodes to create the mesh network. The ad-hoc connections and routing within the mesh network are handled using the B.A.T.M.A.N. protocol. Each network node (including the Network Master Node) uses this protocol via the batman-adv package available in Raspberry Pi OS. The Network Master Node is configured with a bridge that connects the instructor connection and mesh network, allowing packets to flow from one to the other.

The application software for the project comprises the User Application, Backend Application, Node REST API, and Video Streaming Software. The User Application, which runs on the Network Master Node, is an Angular web app that allows students and the instructor to see a list of nodes in the network with their properties (including IP addresses and types), as well as view video streams from Video Mesh Nodes. It defines the UI that the students and instructor use. The User Application UI is loaded in a web browser on the instructor station and is displayed on the Audience Display, which may be the instructor station's screen or an external display (such as a projector). The Backend Application, which also runs on the Network Master Node, is a Spring Boot application that provides node information, including node types and IP addresses, for the User Application to display. This node information is obtained via calls to the Node REST API, which runs on each network node. The Video Streaming Software, which runs on each Video Mesh Node, consists of two existing software components: FFmpeg and Janus WebRTC Server. The FFMpeg program is configured to accept a video stream from the Video Mesh Node's camera, which it encodes using the H.264 codec and forwards it to the Janus WebRTC server. The Janus WebRTC server runs a WebRTC streaming plugin, which transmits the stream across the mesh network to the User Application, which displays it.

Figure 2 shows the system modules involved in our project for each physical computing device involved, including the network nodes and the instructor station.

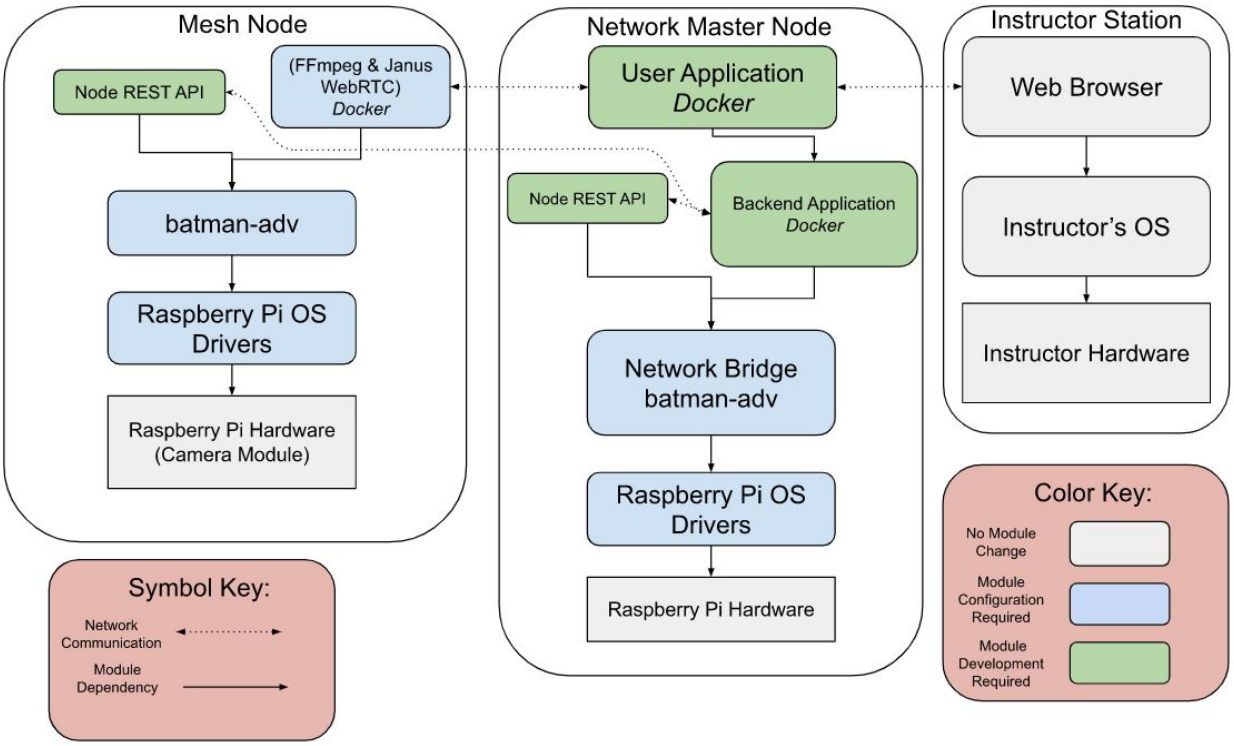


Figure 2: System Modules

The sharp-cornered blocks represent the base hardware layers of each device, while rounded-cornered blocks represent primarily software modules. As shown by the color key, the gray modules existed before the project's inception and need no special configuration. The blue modules existed before the project's inception but require configuration as part of the project. The green modules did not previously exist and are newly developed for this project. Entries in parentheses under Mesh Node (i.e. Camera Module, FFmpeg & Janus) are components that are only required for Video Mesh Nodes. The rest are required for both Video Mesh Nodes and Relay Mesh Nodes.

The arrows with dotted lines represent network communications between modules. For instance, each node runs the Node REST API, which sends information on node properties to the Backend Application. In addition, the video stream from FFmpeg/Janus is sent to a component of the User Application, and the User Application serves web pages to the web browser. The arrows with solid lines indicate dependencies within a node. For instance, the User Application requires input from the Backend Application (i.e. node property information). In the other cases of solid lines, a layer of software depends on the software or hardware below it. The User Application, Backend Application, Node REST API, and Video Streaming Software all require communication over the mesh network, so they all depend on the batman-adv module. This depends on the operating system, which depends on the hardware. The instructor stack is simple: the instructor's web browser of choice runs in the operating system installed on their computer.

Although this is a conceptual sketch, it is important to note that several of the modules are designed to run as Docker containers (and are labeled as such with "*Docker*"). This includes the Backend Application, User Application, and Video Streaming Software. This primarily makes redeploying the project easier, as all the dependencies are automatically installed when the containers are built.

There are several areas in which this final design falls short of the requirements, primarily due to running out of time.

- Our requirements involve powering the nodes via rechargeable battery packs and housing the nodes in portable, drop-proof casings. As of the end of CprE 492, these are still in process; each node is housed in a rudimentary plastic case. We have obtained and tested new batteries, which a future team will need to officially install.
- Our requirements also involve mounting temperature and other sensors on certain nodes (Sensor Mesh Nodes) and displaying the output on the Audience Display. Similarly, this was not finalized. Likewise, system/network statistics from the nodes are not captured. Statistics would be easiest to provide through the Node REST API, while streamed sensor data could be handled with the API or with a data streaming solution such as ZeroMQ or Apache Kafka.
- At present, the mesh network deployment is not fully automated and requires running startup scripts on each node. Similar to the idea to have lesson plans, this has not been completed because the project is still under development. Once the necessary testing and further development is performed by a future team, it will be trivial to configure the Raspberry Pi OS of each node to automatically run the necessary scripts at startup. However, further testing may be required to determine if race conditions occur during startup that affect the configuration of the network. If these issues are uncovered, a messaging framework like ZeroMQ may be helpful in coordinating startup commands.

3. Statement of Work

3.1 Previous Work And Literature

Several alternatives already exist for ad-hoc networking instruction. These primarily consist of computer programs designed to simulate various network types. The ns-3 project [1], supported by the University of Washington NS-3 Consortium, provides a simulated networking environment. Events can be scheduled in ns-3 to cause signals to be transmitted from various nodes at various times. The ns-3 simulator is also capable of interfacing with both virtual and real devices. The mesh package in ns-3 can be used to provide a MAC-layer ad-hoc routing capability for network simulations. Ns-3 is an appropriate tool for graduate and possibly undergraduate study of networking, but it does not effectively meet the needs of a grade-school user base. Ns-3 is primarily configured via the command line, which would be difficult and frustrating for an instructor or grade-school student to use. Students may also have trouble visualizing a network of simulated, virtual nodes; using all physical nodes would provide students with a tangible networking experience that feels more relevant to them.

While our project consists of fully-physical nodes, it shares some points in common with previous academic experiments both physical and virtual. E. Biagioni [2] from the University of Hawai'i at Mānoa created an ad-hoc wireless network using four Raspberry Pi Zero W computers and a station. The network was achieved using Wi-Fi and not Bluetooth, similar to our planned approach. We are wary of Bluetooth, as it may be difficult to obtain the bandwidth needed to stream video through the network. The nodes were placed in rooms around a university building, and a utility similar to traceroute was used to test the network strength. However, the goal in this instance was to evaluate a specific ad-hoc networking protocol, AllNet. In order to keep students' attention and provide a demonstration that feels relevant to students, we need to send live data through the network, such as a video feed.

Another experiment by Sharma and Nekovee [3] is similar to our proposed approach. Unlike our approach, they used virtual nodes. However, their approach does involve sending video over the network. It also centers on the automatic configuration of new nodes as they are added into the network (i.e. assigning IP addresses). This is critical, as we want to avoid having instructors enter arcane commands into a terminal to set up the nodes properly. It also contains a controller with a GUI application, which is very similar to our User Application concept. In completing our project, we hope to combine the aspects of these two prior projects that will benefit students the most and lead to the simplest kit for instructors to set up.

3.2 Technology Considerations

Raspberry Pi's are great for this application due to the amount of support they have. If we were to go with a different board we may not be able to find a suitable Linux distro that supports mesh networking. They are also cheap enough for our budget and have a wide array of accessories designed specifically for the device. The trade off with this is that because the boards are so cheap, they have low power CPUs and low amounts of RAM, along with cheaper Wi-Fi modules. The range of these modules is not very high, and we may have to buy better modules. But, since the boards are so cheap, it will not make the final design infeasible.

For the software, there were a variety of possible approaches. There was a possibility of using Bluetooth mesh networks to design the system; however, we found that Bluetooth LE might make that infeasible for the timeframe and budget given. The best options would be to use either BATMAN or OLSR for our mesh networking protocol over some Linux distribution, and were able to narrow it down to either OpenWRT or Raspberry Pi OS, starting testing first with OpenWRT, due to its inherent Raspberry Pi support and small size. After testing these options, it was decided that BATMAN would be used and run on Raspberry Pi OS. OpenWRT was a viable alternative, but because it doesn't have as many packages available it would have required more work to get software for video streaming and hosting the user application to run. BATMAN was chosen over OLSR because it's able to manage the mesh network and properly assign IP addresses. This was something that OLSR struggled with, among other things.

3.3 Task Decomposition

The following task decomposition reflects our plans at the beginning of CprE 492. Not all of these plans are reflected in the true direction of the final project. Specifically, the ZeroMQ Socket Program was effectively replaced by the Node REST API, and the Camera Stream microservice

concept was shelved in favor of connecting the stream directly to the User Application. In addition, not all tasks listed here were completed; this is described in section 4.

- Nodes
    - Support Ad-Hoc Networking
        - Configure Raspbian Linux
        - Configure BATMAN
    - Battery and System Monitoring
        - Rechargeable Battery Installation
        - CPU, RAM Usage Reporting
    - Mesh Node
        - ZeroMQ Socket Program
        - Case installation
    - Sensor Nodes
        - Stream Camera Feed
        - Obtain/Interpret Sensor Data
        - Case installation (Camera included)
- Network Master Node
    - Host UI Component
    - Host Backend Component
    - Manage Node Connections on Startup
    - Routing Table Configuration (to forward data between ad-hoc network and instructor station)
    - Case installation
- User Interface (User Application)
    - Audience Display
    - Node Selection Page
        - List of Active Nodes in Network
        - Topological Network Graph
    - Node Detail Page
        - Node Information Display
        - Node Information Configuration
        - Data Stream Visualizations
            - Video and Sensor Data (for Video/Sensor Nodes)
            - Network Statistics
        - System Monitoring Indicators
    - Data Model
        - Nodes
        - Stream
    - Network Layer (i.e., Services for communicating with backend)
    - Graphic Design
        - Artwork
            - Application Icon
            - Mesh Node Icon
            - Sensor/Video Node Icons
                - Camera Node Icon
                - Other Sensor Icons

- Backend (Backend Application)
    - Node Status Microservice (node connect/disconnect information)
    - Network Statistics Microservice (packets sent/received/lost, etc.)
    - Camera Stream Microservice (for selection of stream when multiple streams can be displayed)
    - Sensor Data Microservice
    - Network Configuration Microservice (edit node properties such as hostname, etc.)
- User Documentation
    - User Manual
    - Lesson Plans

3.4 Possible Risks And Risk Management

One of the primary risks of the project is the transition to online instruction in Spring 2020 and back to in-person instruction (with restrictions) in Fall 2020 due to the COVID-19 pandemic. This has several implications for the project. Risks are given as main bullet points, while mitigation techniques are given as sub points.

- Group work and bi-weekly meetings with our client are more difficult when we cannot meet in person.
    - Using Zoom or Webex for meetings and Google Docs for group work has made this relatively easy. Even if conceptual drawings are involved, these can be shared via video conferencing.
- Sharing development hardware is difficult given the physical separation of team members.
    - Most of the hardware is kept by team member Austin Dvorak, who has handled most of the mesh network setup work that directly requires the Pi's. Other team members used their personal hardware (older model Raspberry Pi's etc.) to develop the video streaming functionality, User Application, and Backend Application.
    - In Fall 2020, we had several integration meetings among group members in designated group work spaces (Library, TLA).

Among the more conventional sources of risk is the video stream. Below are some of the risks and decisions we made to mitigate those risks.

- The Raspberry Pi has limited hardware acceleration capabilities and may have trouble processing camera streams, depending on the codec used.
    - The H.264 codec has hardware acceleration support on the Raspberry Pi via the GPU. Using this codec ensures the best performance.
- Streaming video to a browser is difficult due to variations in protocols supported by modern web browsers. Many options, such as the VLC video plugin, are obsolete and not supported by modern browsers
    - WebRTC was found to be a well-supported protocol with low latency.
    - The Janus WebRTC Server allows an easy way to stream the video feed across the network using WebRTC, although a complicated setup process must be performed. The setup/dependency installation was simplified through the use of a

Docker image. Firefox is known to work well with Janus and display the video stream with minimal hiccups.

A final source of risk was the development of the mesh network. Below are some of the risks and decisions we made to mitigate those risks.

- The Raspberry Pi network hardware is low-powered and as such may not be sufficient to handle video streaming over a wireless mesh network.
    - By using an external wireless adapter we were able to ensure that the hardware could support the bandwidth and speed requirements of the mesh network.
- Integrating the mesh network with the user application and camera streaming may lead to issues as this project is a very specific use case probably never implemented in this capacity.
    - By starting testing of the project early we were able to ensure that the mesh network was not the source of any networking issues.

3.5 Project Proposed Milestones and Evaluation Criteria

Our first milestone, the completion of one or more Relay Mesh Nodes, was evaluated by our ability to send and receive data in a network of Relay Mesh Nodes. The Relay Mesh Nodes should also be able to report network statistics. This does not include any other components of the system, but ad-hoc mesh networking between the nodes is required. This was completed at the end of the first semester.

Our second milestone was the completion of a functional Video Mesh Node and video streaming functionality. This was finished second semester and entailed the ability to connect a Video Mesh Node to the network via the Network Master Node and then view live camera feed in the User Application.

The third milestone was marked by the completion of the Network Master Node software including the Backend Application with microservices supporting sensor, camera, network statistics, and node status data request and retrieval from the network. It was finished around the same time as the second milestone. It also includes a partial implementation of the User Application. Our Network Master Node will be able to connect to the Mesh Nodes in the network, and the User Application should be able to query that information and show all active nodes on a display. The User Application will also allow selection of camera streams, network statistics, and sensor data for viewing.

Our fourth milestone will be the completion of the battery installation. Each network node should be connected to a rechargeable battery. The battery will need to be securely fastened to the node to prevent its loss. This milestone will also include the implementation of CPU and RAM usage monitoring, which will be reported back to the UI.

Our fifth milestone will include a finished User Application with a dynamic list of all active nodes and a topological network graph showing the connections of the ad-hoc network. At the same time, the User Application and Backend Application will be optimized for the Raspberry Pi to ensure the Network Master Node can handle its processing load effectively and the network is not overrun with unnecessary packets.

Lastly, the sixth milestone will be the completion of our project, accompanied by a set of lesson plans and a user manual. Additionally, we may add one or more new types of Sensor Mesh Nodes (with additional sensor types) that can be connected to the network.

While we were able to finish milestones 1-3, milestone 4 was partially completed, as we obtained and tested the batteries but did not fully install them. We did not complete the system statistics monitoring either. Likewise, milestone 5 was partially completed as we had a fleshed out User Application with no topological graph or any User or Backend Application optimizations. The team had not started the sixth milestone as of the end of senior design; however a wiki page for each component and a README were included for each function to allow for easy deployment and use of the project.

3.6 Project Tracking Procedures

To track our progress throughout the lifetime of the project, we will use the project management tools offered in GitLab. We have set up Issue boards to denote certain Issues as *Open*, *Blocked*, *InProgress*, *InReview*, or *Closed*. These boards will be used to track the progress of every iteration (represented with a Milestone in GitLab) of the project. For every Milestone, GitLab offers a burndown chart showing the amount of work completed over time. These burndown charts will also show progress for the entirety of the project as we complete our deliverables.

3.7 Expected Results and Validation

The determination of our desired outcome comes down to three main goals: functional lesson plans, an easy-to-use user interface, and lack of interference from other networks. The project should include lesson plans that will teach students how mesh networking works, and those should all work seamlessly with the provided equipment. The user interface needs to be designed in a way that an instructor or a student can operate it with minimal knowledge on how it all works. Finally, in order for the first two goals to be fulfilled, the system should not encounter any outside interference from other types of networks.

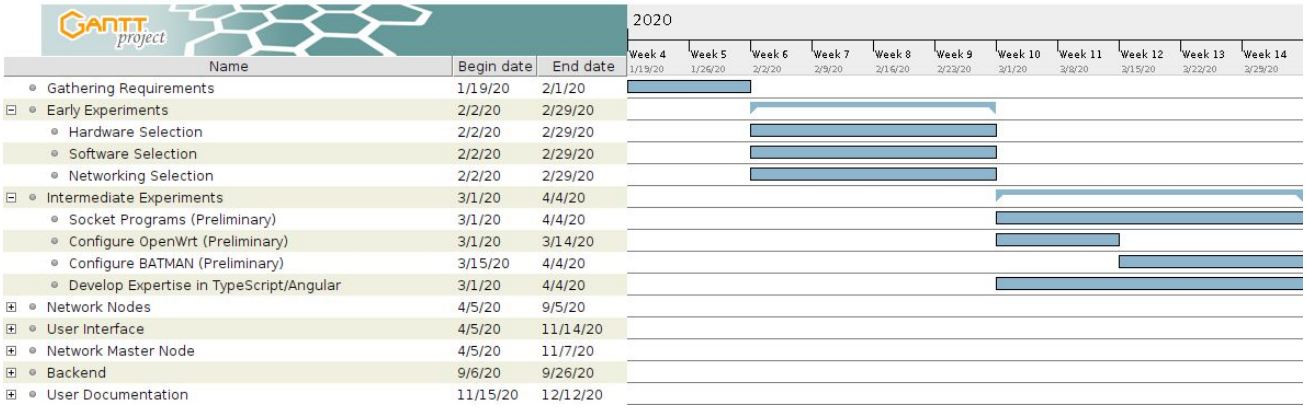
We have a number of test cases that we are going to use to confirm the results work at our expectations. We intend on testing the system in multiple environments, such as inside one of the university buildings where there would be other networks that could potentially cause interference, or outside where we could test maximum range of the product. The goal of this test is to ensure that the system will work as intended during real world use. We also plan on having a test user from outside of the project test the user interface, making sure that an outside individual can operate the lesson plans, user interface, and system with ease. These tests are outlined further in section 5.

As of the end of senior design, the project's main goals were completed. A mesh network of Raspberry Pi's was created that can communicate with each other, as well as a user interface for the demonstrator to use. A Video Mesh Node was created that can stream video to the user interface, and the Backend Application is able to communicate with the User Application, however at this time, the access point interface and mesh network interface are unable to communicate with each other.

4. Project Timeline, Estimated Resources, and Challenges

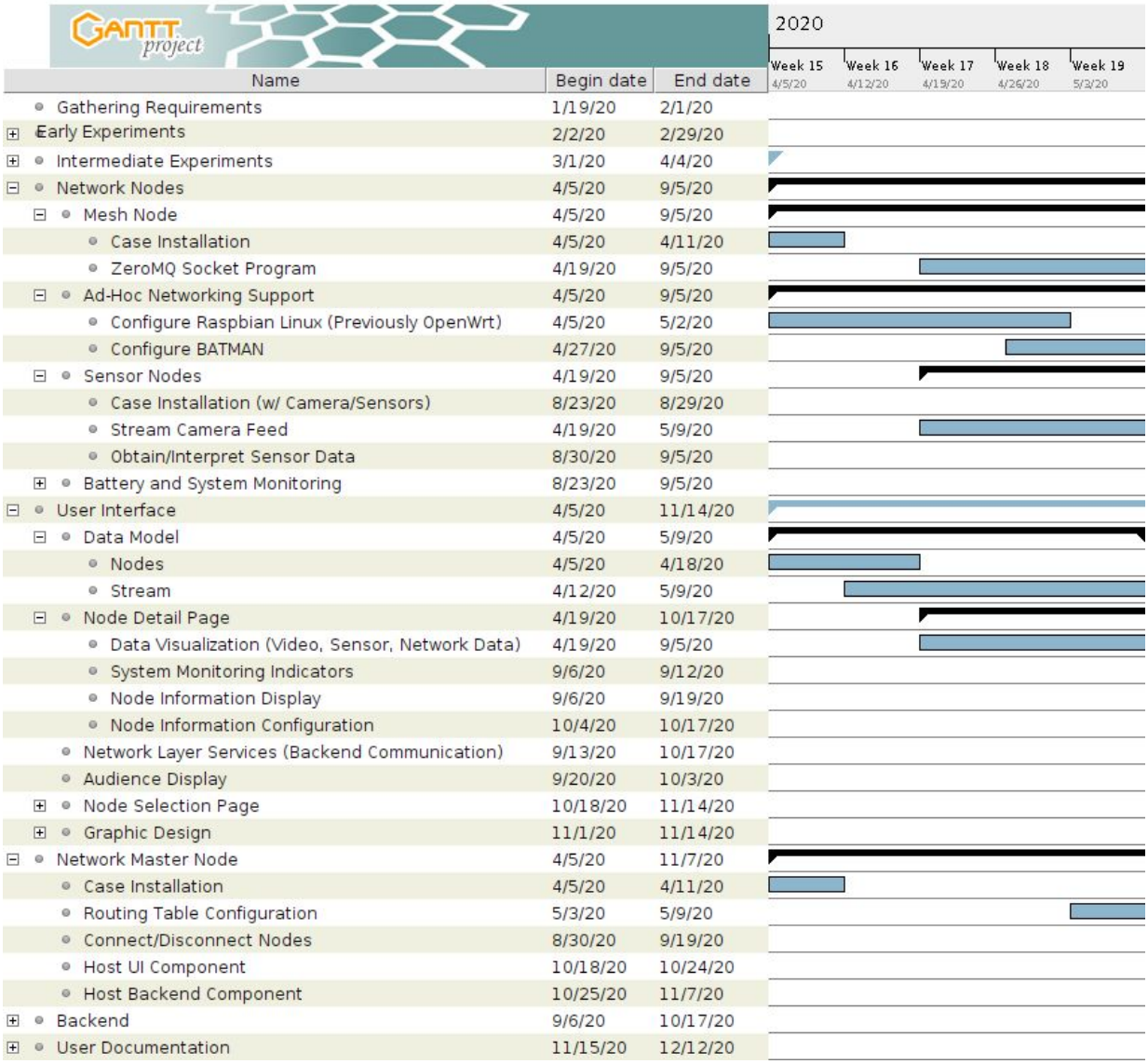
4.1 Project Timeline

Figures 3, 4, 5, and 6 below show the entire Gantt chart, which outlines our initial development plan for our two-semester senior design project. The figure captions state the time period each image represents. Tasks with subtasks that are not present in a given time period have been collapsed. Figure 3 shows the exploratory steps we are performing prior to the receipt of our official hardware for the project.



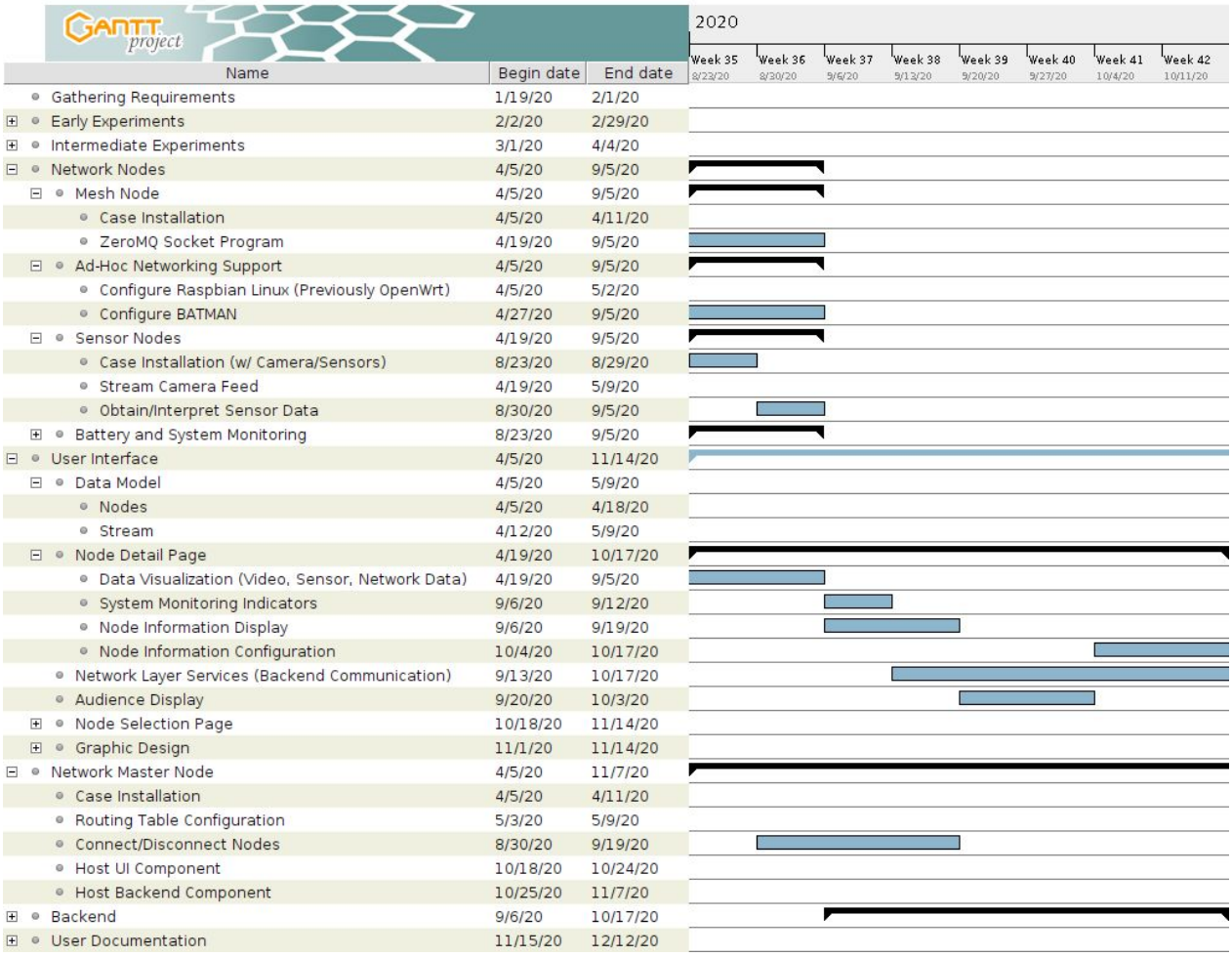
Name	Begin date	End date
Gathering Requirements	1/19/20	2/1/20
Early Experiments	2/2/20	2/29/20
Hardware Selection	2/2/20	2/29/20
Software Selection	2/2/20	2/29/20
Networking Selection	2/2/20	2/29/20
Intermediate Experiments	3/1/20	4/4/20
Socket Programs (Preliminary)	3/1/20	4/4/20
Configure OpenWrt (Preliminary)	3/1/20	3/14/20
Configure BATMAN (Preliminary)	3/15/20	4/4/20
Develop Expertise in TypeScript/Angular	3/1/20	4/4/20
Network Nodes	4/5/20	9/5/20
User Interface	4/5/20	11/14/20
Network Master Node	4/5/20	11/7/20
Backend	9/6/20	9/26/20
User Documentation	11/15/20	12/12/20

Figure 3: Gantt Chart CprE 491 (01/19 - 04/04)



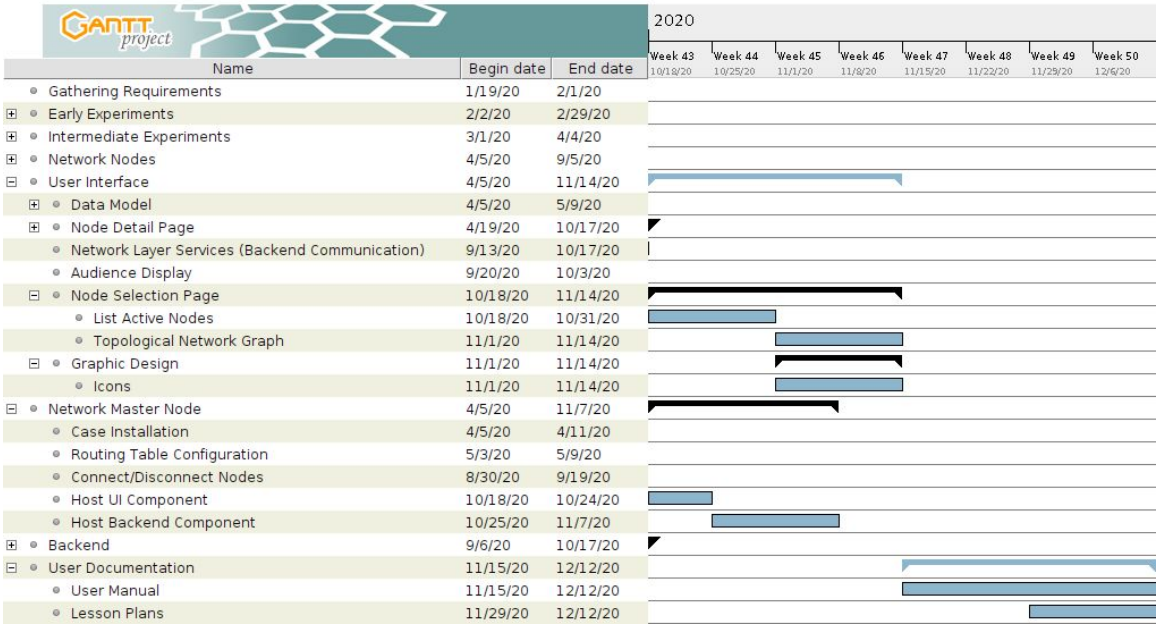
Name	Begin date	End date
Gathering Requirements	1/19/20	2/1/20
Early Experiments	2/2/20	2/29/20
Intermediate Experiments	3/1/20	4/4/20
Network Nodes	4/5/20	9/5/20
Mesh Node	4/5/20	9/5/20
Case Installation	4/5/20	4/11/20
ZeroMQ Socket Program	4/19/20	9/5/20
Ad-Hoc Networking Support	4/5/20	9/5/20
Configure Raspbian Linux (Previously OpenWrt)	4/5/20	5/2/20
Configure BATMAN	4/27/20	9/5/20
Sensor Nodes	4/19/20	9/5/20
Case Installation (w/ Camera/Sensors)	8/23/20	8/29/20
Stream Camera Feed	4/19/20	5/9/20
Obtain/Interpret Sensor Data	8/30/20	9/5/20
Battery and System Monitoring	8/23/20	9/5/20
User Interface	4/5/20	11/14/20
Data Model	4/5/20	5/9/20
Nodes	4/5/20	4/18/20
Stream	4/12/20	5/9/20
Node Detail Page	4/19/20	10/17/20
Data Visualization (Video, Sensor, Network Data)	4/19/20	9/5/20
System Monitoring Indicators	9/6/20	9/12/20
Node Information Display	9/6/20	9/19/20
Node Information Configuration	10/4/20	10/17/20
Network Layer Services (Backend Communication)	9/13/20	10/17/20
Audience Display	9/20/20	10/3/20
Node Selection Page	10/18/20	11/14/20
Graphic Design	11/1/20	11/14/20
Network Master Node	4/5/20	11/7/20
Case Installation	4/5/20	4/11/20
Routing Table Configuration	5/3/20	5/9/20
Connect/Disconnect Nodes	8/30/20	9/19/20
Host UI Component	10/18/20	10/24/20
Host Backend Component	10/25/20	11/7/20
Backend	9/6/20	10/17/20
User Documentation	11/15/20	12/12/20

Figure 4: Gantt Chart CprE 491 (04/05 - 05/09)



Name	Begin date	End date
Gathering Requirements	1/19/20	2/1/20
Early Experiments	2/2/20	2/29/20
Intermediate Experiments	3/1/20	4/4/20
Network Nodes	4/5/20	9/5/20
Mesh Node	4/5/20	9/5/20
Case Installation	4/5/20	4/11/20
ZeroMQ Socket Program	4/19/20	9/5/20
Ad-Hoc Networking Support	4/5/20	9/5/20
Configure Raspbian Linux (Previously OpenWrt)	4/5/20	5/2/20
Configure BATMAN	4/27/20	9/5/20
Sensor Nodes	4/19/20	9/5/20
Case Installation (w/ Camera/Sensors)	8/23/20	8/29/20
Stream Camera Feed	4/19/20	5/9/20
Obtain/Interpret Sensor Data	8/30/20	9/5/20
Battery and System Monitoring	8/23/20	9/5/20
User Interface	4/5/20	11/14/20
Data Model	4/5/20	5/9/20
Nodes	4/5/20	4/18/20
Stream	4/12/20	5/9/20
Node Detail Page	4/19/20	10/17/20
Data Visualization (Video, Sensor, Network Data)	4/19/20	9/5/20
System Monitoring Indicators	9/6/20	9/12/20
Node Information Display	9/6/20	9/19/20
Node Information Configuration	10/4/20	10/17/20
Network Layer Services (Backend Communication)	9/13/20	10/17/20
Audience Display	9/20/20	10/3/20
Node Selection Page	10/18/20	11/14/20
Graphic Design	11/1/20	11/14/20
Network Master Node	4/5/20	11/7/20
Case Installation	4/5/20	4/11/20
Routing Table Configuration	5/3/20	5/9/20
Connect/Disconnect Nodes	8/30/20	9/19/20
Host UI Component	10/18/20	10/24/20
Host Backend Component	10/25/20	11/7/20
Backend	9/6/20	10/17/20
User Documentation	11/15/20	12/12/20

Figure 5: Gantt Chart CprE 492 (08/23 - 10/17)



Name	Begin date	End date
Gathering Requirements	1/19/20	2/1/20
Early Experiments	2/2/20	2/29/20
Intermediate Experiments	3/1/20	4/4/20
Network Nodes	4/5/20	9/5/20
User Interface	4/5/20	11/14/20
Data Model	4/5/20	5/9/20
Node Detail Page	4/19/20	10/17/20
Network Layer Services (Backend Communication)	9/13/20	10/17/20
Audience Display	9/20/20	10/3/20
Node Selection Page	10/18/20	11/14/20
List Active Nodes	10/18/20	10/31/20
Topological Network Graph	11/1/20	11/14/20
Graphic Design	11/1/20	11/14/20
Icons	11/1/20	11/14/20
Network Master Node	4/5/20	11/7/20
Case Installation	4/5/20	4/11/20
Routing Table Configuration	5/3/20	5/9/20
Connect/Disconnect Nodes	8/30/20	9/19/20
Host UI Component	10/18/20	10/24/20
Host Backend Component	10/25/20	11/7/20
Backend	9/6/20	10/17/20
User Documentation	11/15/20	12/12/20
User Manual	11/15/20	12/12/20
Lesson Plans	11/29/20	12/12/20

Figure 6: Gantt Chart CprE 492 (10/18 - 12/12)

Our Gantt chart reflects several primary goals. Most importantly, we were pushing to complete the initial ad-hoc networking prototype by the end of CprE 491 and finalize it in the first two weeks of CprE 492 (see Figure 4, Figure 5). This required the proper configuration of Raspbian and BATMAN, as well as the streaming and forwarding of data in the network. Prioritizing this goal would help us take our biggest risks first, when there is less pressure. Another large risk was the video stream prototyping. There are many possible configurations for video codecs (H.264, H.265, etc.) and browser-based streaming techniques (WebRTC, etc.), but a lot of testing was required to determine that WebRTC is our strongest option. Once we were to get the network prototype and streaming working, we anticipated that later tasks would be more straightforward.

We were also being cautious with the development of our user interface and backend (User Application and Backend Application, respectively) (see Figure 5). These tasks required a lot of coordination, especially when different team members are working on each part. This also required some work on the Mesh Node software (i.e. to obtain the data requested or set the desired node configuration). We distributed the frontend user interface and backend work from early September to mid October. In some cases, the frontend development superseded the backend development (i.e. the main audience display will be completed before the sensor data microservice is created). In these scenarios, both frontend and backend developers were to collaborate to determine an interface beforehand, so that the frontend developers could test their UI with mock data following the interface pattern. The risk of having to change both UI and backend due to implementation issues on the backend is small, as these microservices were to be very similar in their implementation (relying on ZeroMQ to communicate with nodes).

The end of our timeline (Figure 6) consisted of UI improvements and production of two items of documentation: lesson plans and a user manual. Putting these tasks last was intended to help us avoid the possibility of a last-minute technical flaw upending the entire project. These tasks also served as an assessment of our project goals. By writing our user manual, we would have been able to assess whether our project is easy enough for an instructor to use. By writing the lesson plans, we would be able to assess whether our project has real teaching value.

Now that we have reached the end of our second semester of working on this project, we can observe the ways we have diverged from this development plan. Of course, many complications arose with the pandemic and consequential changes to the academic calendar, so our plan could not be followed closely. We actually finished the bulk of the User Application and Mesh Network configuration within the first reporting period of our second semester. At this point, we had successfully set up a mesh network of nodes and had also completed both web pages required for the User Application. This put us slightly ahead of schedule.

Unfortunately, our early lead was lost because of complications with our video streaming implementation. We intended to finish the video streaming capability by September 6; however, this was delayed until early October. This task proved more difficult than anticipated because of the complexity of video streaming and the lack of support for live video streaming within web-based applications.

By the end of October, we had successfully completed all the configuration of the mesh network, the implementation of the Video Mesh Node, and a containerized implementation of the Backend Application and User Application for the Network Master Node. At this point, our Network Master Node was capable of communicating directly with the Camera Node, but we were still in the process of making the Backend Application communicate with the Mesh Network.

Now, at the end of the project, we have a Network Master Node, consisting of the Backend Application which can communicate with the other nodes in the network and display this information in the User Application; a Node REST API implementation, which allows the Network Master Node to access and configure the properties of any node; and a Video Mesh Node implementation, which serves live video to the User Application.

Because of the final state of our project, we decided to forgo writing the lesson plans. Other features that we did not have time to implement included the topological network graph visualization, network statistics visualization, battery level and system statistics reporting, Sensor Mesh Nodes, and node-specific graphics.

4.2 Feasibility Assessment

Given the time frame of the project, creating all software components ourselves is not practical, nor is it necessary per the requirements. Therefore, pre-existing software will be used for networking, video streaming, and other complex tasks that arise in the future.

Mesh and ad-hoc networking software is abundant for the Raspberry Pi 4. The system hardware is powerful enough for networking and streaming video and meets project requirements. Similar to mesh networking, streaming video is a matter of finding the right combination of codecs/software to achieve acceptable latency. The task of exhausting the possible options is relatively simple.

The User and Backend Application implementations are similar to mesh networking in their feasibility, but easier as multiple team members have prior experience in web development. Other requirements, such as the two-hour battery life, are simple to meet, as USB power banks already exist with the necessary capacity. Generally, our design is feasible within our schedule.

Given the specific requirements for any casing, it is unlikely that a commercial option exists. Due to the late consideration of the exterior, a simpler case will be chosen with a recommendation for a more appropriate design created later.

4.3 Personnel Effort Requirements

Table 1 shows our original estimates of hours required to complete the major tasks of the project.

Task	ETC (hrs)	Task	ETC (hrs)	Task	ETC (hrs)
Configure Raspbian	10	Network Master Node Routing	8	User Application Stream Viewing	7
Configure BATMAN	15	Deploy Backend Application on Network Master Node	10	User Application Backend Communication	8
Mesh Node Communicatio n Program	10	Deploy User Application on Network Master Node	10	User Application Graphic Design	5
Report Node System Statistics	5	User Application Active Node List	10	Backend Application Node Property Microservice	8
Video Stream from Video Mesh Node	15	User Application Node Graph	10	Backend Application Stream Microservice	8
Sensor Data from Sensor Nodes	7	User Application Node Detail Display	10	Network Configuration Microservice	8
Camera, Case, Battery Installations	5	User Application Node Control	10	User Manual / Lesson Plans	15

Table 1: Task Estimated Time to Completion Table

4.4 Other Resource Requirements

This project will only need materials and parts to build each node. Each node consists of a Raspberry Pi 4, an SD card, a rechargeable battery pack (including the charger), and a case. In addition to that, the Video Mesh Nodes will also have the Raspberry Pi Camera Module attached.

To accommodate the camera, the Video Mesh Nodes will reside in a case that includes a spot to hold the camera. Additional sensors were considered, but left out due to time constraints.

4.5 Financial Requirements

The base of each node is the Raspberry Pi Starter Kits that includes the Raspberry Pi Model 4. These kits also include a power adapter and SD card which is the bare minimum to get started with the Raspberry Pi. Six of these kits were obtained from CanaKit for $90 each.

To power these wirelessly, we found a shield for the Pis that acts as an Uninterruptible Power Supply and holds two 18650 type batteries. These shields are made by GeekWorm and are $50 on Amazon. The 18650 batteries that will be used in conjunction with the UPS are Panasonic 3400mAh batteries that are $4.75 each.

The last part needed to complete the basic node was a case that fit both the Pi and the UPS shield. We ended up going with the GeekWorm DIY case kit because of its ability to be custom sized depending on our needs. These cost $6 each.

In total, the base of each node included one Raspberry Pi starter kit, a case, a UPS, and two batteries. The cost of each node was about $155 at the time of purchasing. Originally we only got three UPS shields and six batteries to test them out before buying the set up for each node. After testing the UPS and batteries, we found that they work as needed but didn't have time to order the configuration for the remaining three nodes. So the full cost of the completed nodes was $155 while the three without batteries only cost $96.

The last cost of building the nodes was for the node that included a camera for streaming video. For this, a camera module was needed as well as a camera case mount to protect the camera. The Raspberry Pi Camera Module was picked along with a camera case that fits it. Together these cost $34. As only one node needs the camera, one camera and case are needed. The camera case will mount nicely on top of the case that we picked out so no additional case or materials are needed to attach the camera to the node.

Overall, the total cost of the project came out to be $787.75. This includes three completed nodes and three nodes that need the UPS and batteries still. To complete the last three nodes, an additional $180 would be needed bringing the total cost to $966.

5. Testing and Implementation

5.1 Interface Specifications

The major components to test included the User Application, Backend Application, Video Streaming Software, mesh network setup, and batteries. All tests of UI components were conducted using the real implementation of the User Application. To help in testing the Backend Application and User Application, we created a mock Backend Application that contains fake nodes for testing without the real mesh network. The Video Streaming Software was tested using the real User Application and camera, although it was mostly done over a local area network instead of the real

mesh network. Battery testing was performed by connecting the batteries directly to a Raspberry Pi and did not require any specialized interfaces.

5.2 Hardware and software

Three Raspberry Pi 4 Complete Starter Kits were obtained from CanaKit for the testing of the mesh network as well as a Raspberry Pi camera module. Three Raspberry Pi's were purchased to allow for a more complete network rather than simply two connections. The camera module was purchased to test the streaming functionality. Three Geekworm X728 UPS battery boards with rechargeable lithium ion batteries were also acquired.

For the Backend Application, which was implemented using Spring Boot, we wrote unit tests using JUnit5 and Mockito. These tests were purposed with covering all execution paths of every method in each class.

To test the video streaming functionality, we created a basic JavaScript application to act as the WebRTC client so that the video streaming could be tested independently of the User Application before integration.

5.3 Functional Testing

Our functional testing plan includes unit, integration, system, and acceptance testing. The unit testing covers specific features of the Backend Application. The integration testing covers the connections between components, such as between the User Application and Backend Application, or User Application and Video Streaming Software. System and acceptance testing focuses on testing the whole system's functionality. Below are some tests that we have performed. Italicized tests were completed, while unitalicized tests were not completed.

5.3.1 Unit Tests

1. *Backend Application can get a list of nodes from the node service.*
2. *Backend Application can get a specific node from the node service using its IP address.*
3. *Backend Application can update node properties using the node service.*

5.3.2 Integration Tests

1. *Backend Application receives requests for data from User Application (node IP addresses) and serves them appropriately.*
2. *Video stream can be transmitted from the Pi camera to the User Application.*
3. *A single Mesh Node, when booted, pairs correctly with its Network Master Node; mesh network connectivity can be verified between the two (using a simple utility such as ping).*
4. *The Backend Application is able to perform a scan of the Mesh Network and fetch all the properties of each node.*

5.3.3 System Tests

1. *Ensure all Mesh Nodes connect automatically to their associated Network Master Node when booted and within range.*
2. *Ensure all nodes indicated as available in the User Application UI are reachable from the Network Master Node (using ping).*

3. Ensure all Mesh Nodes reconnect to Network Master Node when brought into range after being isolated and disconnected.
4. Ensure video feed from a Video Mesh Node can be viewed from the User Application over the mesh network.

5.3.4 Acceptance Tests

Since we did not fully complete the project, we did not attempt acceptance testing. Below are our plans for acceptance testing.

1. The instructor must not be required to use any command-line tools to begin an experiment.
2. Video must have a high enough frame rate and resolution so that students are able to see one another on the feed and appreciate the real-time changes in the stream when the surroundings on camera are manipulated.
3. The instructor must only need to connect to the Network Master Node's network and open the web app from their computer station to begin.

We also originally planned tests related to acquiring and displaying sensor data and network/system statistics, but these were scrapped when we ran out of time to implement these features.

5.4 Non-Functional Testing

Our plan for non-functional testing originally involved testing the project on its performance, security, usability, and compatibility. Below is a list of tests performed for each category.

5.4.1 Performance Tests

1. *Check that network speeds within the mesh network are high enough to support video streaming.*
    - *Calibrate video streaming bitrate to match the available bandwidth.*

We originally planned to test the network connection ranges for the nodes, but this was scrapped as system testing with the mesh network ran into issues.

5.4.2 Security Tests

No tests were performed. Potential ideas for security testing include ensuring that the mesh network does not allow foreign devices to freely connect and ensuring that video streams cannot be viewed by foreign devices.

5.4.3 Usability Tests

1. *Check that nodes are portable when batteries are connected.*
2. *Check that battery life is at least two hours per charge.*
3. *Check that an instructor station does not have to install any dependencies to load the User Application.*

We originally planned to test the durability of the node cases, but we ran out of time to do so, as our batteries and durable cases arrived late.

5.4.4 Compatibility Tests

1. *Check that the User Application runs correctly in a variety of common browsers.*

5.5 Process

For each component, we followed a somewhat different pattern of implementation and testing. These processes are detailed below.

For the mesh network, BATMAN's configuration wiki was used to configure the mesh network. Each node was configured with the same mesh SSID, channel, frequency, and interface types to ensure that they would connect to each other. A wireless access point was created using hostapd that would allow a user's computer to connect to the node for either accessing the UI or manually configuring the node. A bridge interface was created that would connect the hostapd interface to the BATMAN interface so that the proper forwarding of packets would occur when the user accesses any of the sensor node pages.

For the video stream, we initially created a basic HTML page with a video element and JavaScript code to connect to the WebRTC stream from the Pi. We soon transitioned to testing within the User Application. Most testing was performed with the Pi connected to a wireless access point within a dorm room. We ran the User Application and Backend Application on a personal computer connected to the access point, with the video stream URL hardcoded with the IP address of the Pi. The dorm connection was measured at about 8 mbps by running the *iperf* client and server commands (in UDP mode) on the Pi and a personal computer connected to the network. Three parameters of FFmpeg were determined that impact the data rate required by the video stream: framerate, bitrate, and resolution. We tested several combinations of these, determining what set of parameters led to the best picture without leading to excessive choppiness. We then sampled the speed of the mesh network and modified the parameters again based on these findings. Due to last-minute issues with the mesh network and Docker, we were unable to view the video stream inside the actual mesh network.

We began the User Application by creating a simple two-page Angular application with a home page, which displayed a list of all the nodes in the network, and a node detail page. At this point, the Backend Application hadn't been created, so we tested the User Application with a Postman API that served fake sample data. Once we implemented the Backend Application, we changed the proxy configuration so that the HTTP requests would be forwarded to the Backend Application instead of the Postman API. Not much of the User Application has changed at this point in terms of UI design, except for the video streaming integration, which will be described elsewhere.

The Backend Application started as a simple Spring Boot application that provided a simple REST API with only the two methods that were required by the User Application at the time. At the beginning, this didn't actually connect to the mesh network; it simply served fake sample data so that we could test the integration with the User Application independently of the mesh network. Here, we performed basic smoke testing to ensure that everything was functioning between the User Application and Backend Application and then wrote unit tests for all components of the Backend Application. Once we had full test coverage, we implemented the last of the required functionality, which was to make the Backend Application communicate with the nodes in the network. This last feature was tested by an integration test, in which we set up the Mesh Network,

deployed the updated Backend Application, and manually tested the User Application to verify that the network scan was returning all the nodes in the network.

Battery implementation and testing was performed late in the semester and was not comprehensive. We attached the batteries and board to a Pi for roughly three hours while performing intensive tasks as part of the Video Mesh Node deployment process (large package downloads and installations via a package manager, compiling Janus WebRTC Server and a support library, and running the video stream to the User Application without the mesh network).

5.6 Results

After trying multiple different configurations for the mesh network for both BATMAN and OLSR on OpenWRT, we could not get the network to work. We decided to move over to Raspberry Pi OS as there seemed to be more support for it on Raspberry Pi's that was written for the most modern version of batman-adv. After updating the nodes, we were able to see some limited success with the network, as now the wireless interface could communicate with batman-adv, however an external wireless adapter was required. After testing the network with wireless dongles, we were able to verify that the network was working. After some modifications to the setup, we were able to include full modification of the network through an API. In order to connect to the master node, we created a wireless interface connected to the internal wireless hardware on the Pi. This allows a user's computer to connect to the master node by selecting the interface in the Wi-Fi settings as if it were a regular wireless network. Through this testing, we were able to verify integration test case 3 from 5.3.2 and system test cases 1 and 2 from 5.3.3.

We ran into issues at the very end with packet forwarding between the access point interface and the mesh network. We created a bridge attached to both the access point interface as well as the mesh network and enabled IPv4 packet forwarding. While this temporarily works, for some reason the bridge fails at some point, resulting in both interface's inability to forward packets, either in the mesh network or the access point. A possible reason for this may be because the User Application is running in a Docker container, and the bridge only goes down when the user connects to the UI through the browser. Another possible reason is that the Raspberry Pi's simply can't handle the amount of traffic going through the network, and the bridge that connects the two interfaces goes down due to this. As a result, we were unable to verify system test cases 3 and 4 from 5.3.3.

Our video streaming tests revealed several important limits to the stream parameters. For one, we recognized that the bitrate of the stream could only be increased to about 60% of our measured network speed (on the dorm network, about 5 mbps) before the video became excessively choppy. After measuring the speed of the mesh network and determining that it ranged from 200 kbps to 2.5 mbps, we decided to set the bitrate at 320 kbps. This assumes that the network will typically allow at least ~533 kbps data transfers, based on the previous finding. We then tested resolutions and framerates. FFmpeg restricts resolutions to several specific sizes. We originally chose the 4CIF resolution, which is 704x576 pixels. This resulted in poor video quality at frame rates of 10 fps or higher, as shown in Figure 7, which was captured with a framerate of 15 fps. The color inaccuracies seen in the figure were especially pronounced when the image in the frame changed quickly (i.e. when the camera was moved).

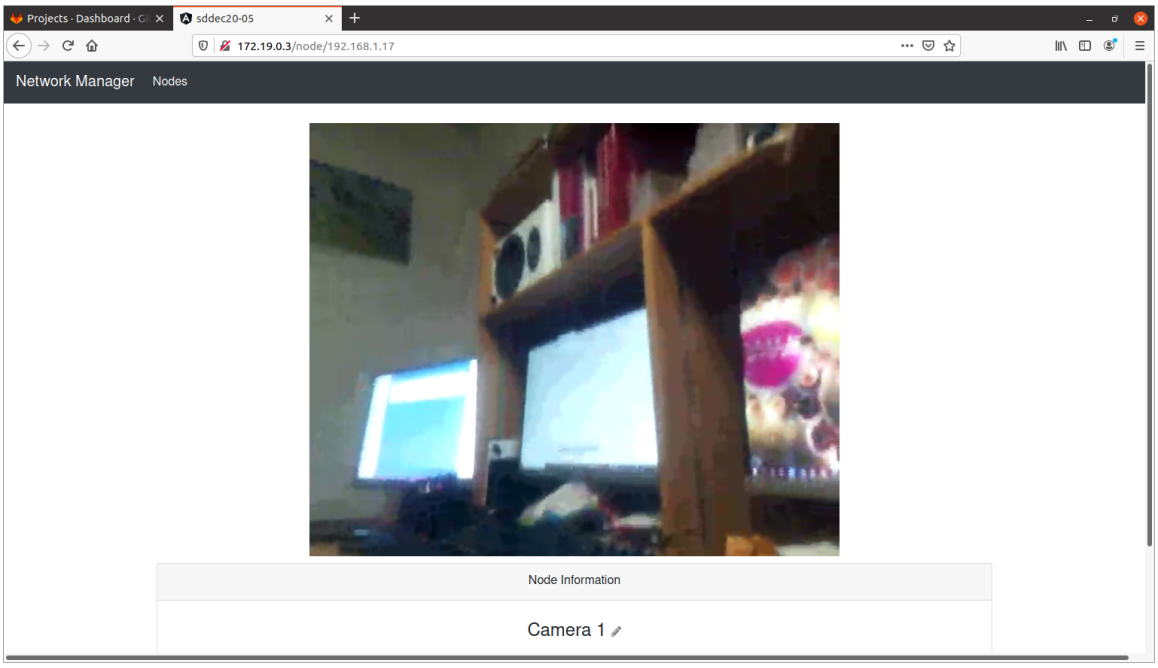


Figure 7: Poor Video Quality

After this, we chose the HVGA resolution for the video, which is 480x320 pixels. This resolution fits well with the other node detail page elements on a typical laptop screen. We found that the color accuracy was acceptable at 15 fps. This is seen in Figure 8.

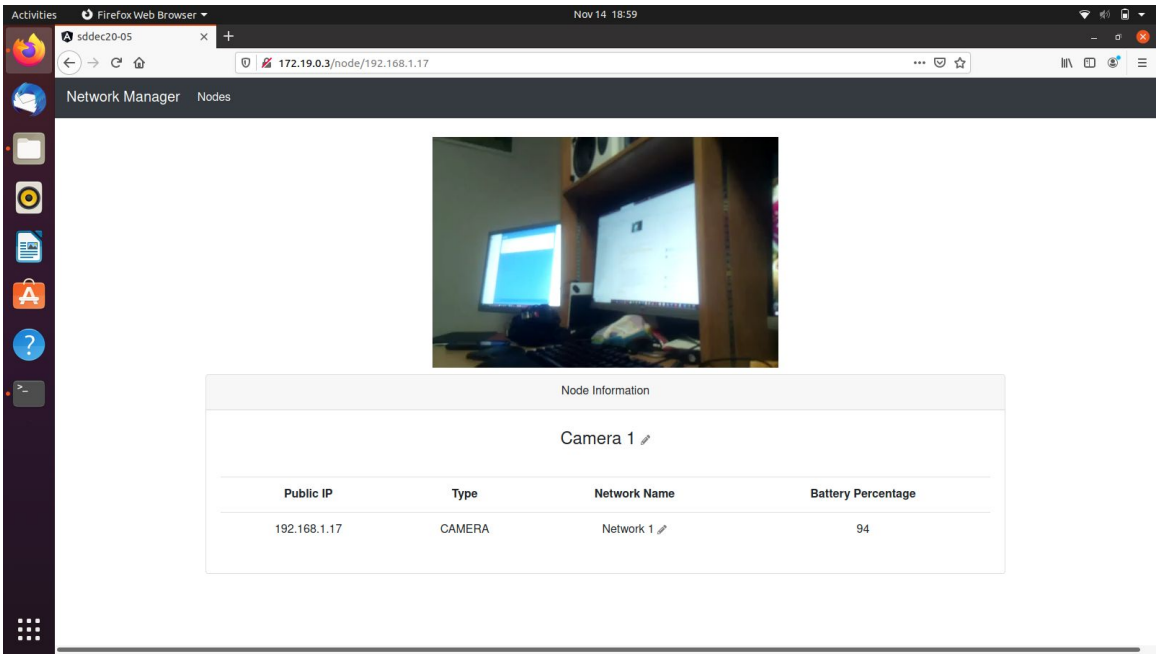


Figure 8: Acceptable Video Quality

From our video tests, we were able to verify integration test case 2 from 5.3.2. We were unable to verify system test case 4 from 5.3.3, as the network bridge issues prevented us from sending video over the mesh network. Our testing with the bitrate verified performance test case 1 from 5.4.1, and our use of the User Application throughout this process verified usability test case 3 from 5.4.3 and compatibility test case 1 from 5.4.4, although we found that Chrome has trouble connecting to the Janus WebRTC server.

Our simple battery tests showed that a node could be powered for up to three hours while performing intensive tasks, and was still highly portable. This verified usability test cases 1 and 2 from 5.4.3.

For the Backend Application, the automatic testing was entirely successful. We achieved 100% unit test coverage (excluding model and configuration classes). For the integration test, which verifies that the Backend Application receives requests from the User Application, we received perfect results. More specifically, we were able to run the application and test each of the endpoints of the REST API to verify correctness. This verified the unit test cases 1, 2, and 3 from 5.3.1 as well as integration test case 1 from 5.3.2.

The last test for the Backend Application was integration test 4, which tested the Backend Application's ability to scan the network to find all the nodes in the Mesh Network and fetch their properties. This test yielded mostly positive results, albeit not reliably. It was capable of discovering all the nodes in the network, but because this scan process was held to a strict timeout limit, the scan sometimes didn't find every node in the network. More effort will be required of the next

group to improve the performance of this feature. But for now, we consider integration test case 4 from 5.3.2 to be verified.

To summarize, here are some of the key implementation issues and challenges:

- batman-adv and IPv4 packet forwarding do not work well with Docker, which results in network connectivity issues. We may want to rethink the network configuration of the Docker containers.
- Video bandwidth is extremely limited, and a fine balance between resolution, framerate, and bitrate must be established.
- Janus WebRTC seems to be compatible with Firefox, but compatibility issues with Chrome are a concern.
- It may not be ideal for the Backend Application to use nmap to discover all the nodes in the network because the scan time can be very lengthy.

6. Closing Material

6.1 Conclusion

At the end of this project, we have created a set of nodes that configure into a mesh network. We are able to display the list of nodes in the network and query node properties via the User Application and Backend Application. The software can stream video from a camera on a Video Mesh Node to the User Application. However, difficulties with Docker and its effect on the network setup have prevented us from demonstrating the full functionality. Due to time constraints, several other features, such as sensors and a network topology graph, were left on the drawing board.

For each node, we have a Raspberry Pi 4, the UPS hat used as the power supply, and a simple acrylic case. The UPS hat and acrylic case have been obtained but not installed. Further testing is required to ensure that these options meet the needs of the project.

Ultimately, another team will need to finish this project. This document, along with our project Wiki, will help such a team understand the original requirements and the current state of the project. This will allow the team to better plan the work that remains.

6.2 References

[1] *NS-3 | A Discrete-Event Network Simulator for Internet Systems*, 2020. Accessed on: Mar. 29, 2020. [Online]. Available: https://www.nsnam.org/

[2] E. Biagioni, "A Network Testbed for Ad-Hoc Communications using Raspberry Pi and 802.11," in *Proc. of the 52nd Hawaii International Conference on System Sciences*. Accessed on: Mar. 29, 2020. [Online]. Available: http://hdl.handle.net/10125/60191

[3] S. Sharma and M. Nekovee, "Demo Abstract: A demonstration of automatic configuration of OpenFlow in wireless ad hoc networks," *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Accessed on: Mar. 29, 2020. [Online]. doi: 10.1109/INFCOMW.2019.8845307

6.3 Appendices

6.3.1 Operational Manual

The following sections give a brief instructional overview of the process of setting up the Mesh Nodes and deploying our project's software. These instructions can also be found in the various *README.md* files in our project repository.

6.3.1.1 Overview of Deployment

The steps for setting up a Mesh Node are as follows.

1. Put the project's Git repository on the Raspberry Pi.
2. Follow the common node setup instructions.
3. If the node is not a Relay Mesh Node (i.e., the Network Master Node or a Video Mesh Node), then proceed to follow the instructions for deploying that specific node type.

6.3.1.2 Common Node Deployment

This software runs on every Raspberry Pi you intend to use for the project. Each Pi must have Raspberry Pi OS installed on it as well as a connection to the internet (preferably via ethernet).

6.3.1.2.1 Building

When deploying the Common Node software for the first time, you will need to run the *build.sh* script in the directory *NodeCommon*. This will download the required software for the mesh network and set up the access point to allow the user computer to connect to the node wirelessly for changing the configuration of the node.

6.3.1.2.2 Starting

Before starting the network, navigate to *NodeCommon/network_settings.cfg* and change the information in there to match the information for the node.

- node_ip_address
  - This should be a unique IP for the network you are trying to connect to.
- node_name
  - This should be a unique name identifying the type of node this is (like "Camera 1", "Camera A", etc.).
- node_type
  - This should be the type of node this is (right now, either *CAMERA*, *MASTER*, or *RELAY*).
- mesh_network_name
  - This should be the name of the mesh network you want to connect to, must not include spaces.

Then to start the Mesh Network, access point, bridge and Node REST API, run the *start.sh* script.

6.3.1.3 Network Master Node Deployment

When deploying the Network Master Node software for the first time, you will need to install *docker-compose* on the Raspberry Pi. Instructions for doing this are located in the *README.md* file of the *NetworkMasterNode* directory of the project repository.

The following instructions will outline the deployment operations that should be carried out via the command line. These commands must be executed inside the *NetworkMasterNode* directory.

6.3.1.3.1 Building

Before building, look in *NetworkMasterNode/BackendApplication/src/main/resources/application.properties* and make sure that the value of *network.manager.hostMachineIpAddress* is the same as the value of *node_ip_address* in *NodeCommon/network_settings.cfg*.

Once the IP address is correctly defined, run `make build` to build the *UserApplication* and *BackendApplication* Docker containers.

6.3.1.3.2 Starting

Run `make start` to run the applications in the background. This simply starts the containers that were built in the previous step.

6.3.1.3.3 Stopping

Run `make stop` in this directory to stop the Docker containers. This will only stop the containers; they can be started again with `make start`.

6.3.1.3.4 Viewing Logs

You can view the logs from the User Application and Backend Application by running `docker-compose logs` in this directory. If you want to view the logs continuously, you can do `docker-compose logs -f`.

6.3.1.4 Video Mesh Node Deployment

6.3.1.4.1 Preparing the Raspberry Pi

Boot up the Raspberry Pi with the camera connected via the ribbon cable.

In a terminal, run `sudo raspi-config`. Enable the camera in the Interface Options menu. Exit raspi-config by selecting Finish and reboot the Pi.

In a terminal, run the following commands (primarily for testing; these are not required in future power-ons).

- `sudo apt install python3-picamera`
- `modprobe bcm2835-v4l2`
- Optionally, you can run `raspistill -o test.jpg` to test the camera. Even in the non-desktop version of Raspberry Pi OS, you will see a preview of the camera as it takes the picture. This will take a few seconds.

6.3.1.4.2 Building

You are now ready to build and run the *CameraNode* Docker container on the Pi. Inside the *CameraNode* directory of the repository, execute the *build.sh* script. You will probably need to use sudo to execute it, i.e. `sudo ./build.sh`.

6.3.1.4.3 Running

Now that you have built the Docker container, you are ready to run it. Run start.sh in the *CameraNode* directory. You will likely need to use `sudo` for this script as well. The console is interactive, although output is printed to the console regularly. To stop the stream, you can type `exit` in the console.

To run the stream within the mesh network, follow the procedures for connecting the Pi to the mesh network instead of your internet connection. Use the *start.sh* script to start the stream once the Pi is connected.