Project Proposal - Team 15

WifiLMR

October 22, 2018

Metadata

Team members:

- Armaan Amirani
- Zane Cersovsky
- Amrit Thapa
- Jack Weber
- John Ying

Synopsis

A cost-effective high fidelity two-way radio for campus-scale deployments leveraging pre-existing WiFi infrastructure to avoid the need for licensed spectrum while improving indoor performance.

Description

Rationale

Current solutions used for team communication on small to medium campuses have poor voice quality and limited range or are too expensive to be deployed by organizations with limited resources. By utilizing already-common wireless networks (WiFi) and Voice over Internet Protocol, these problems are addressed without requiring costly high-power transceivers and licensed spectrum allocations.

Use Cases

Our product will be useful in locations such as:

- Hospitals and other healthcare facilities
- Schools and higher education campuses
- Large corporate offices and campuses

Teams targeted by our product include:

- Maintenance personnel
- Nursing staff
- Information Technology technicians

End Product

The product will consist of a handheld end-user device that functions as a two-way radio. There will also be a server component to relay traffic from user to user as well to provide a way to administer a fleet of devices.

Milestones

First Semester

1. **Core functionality proof of concept:** This entails having a working application that can take voice from an input, send it to our backend, then have the voice played back on another application. Estimated completion date: November 23
2. **Digital hardware designed and ordered:** Know exactly how our radio will look and feel. Have all designs and prefabs organized and materials ordered. Estimated completion date: December 1
3. **Management tool prototype:** Our project will have a console that a user can log into on their local network. The console will allow the user to manage radios on the network. Estimated completion date: December 14

Second Semester

1. **Core functionality finalized:** Our core applications (RadioApp & RadioManager) will be tested and working according to spec. Estimated completion date: February 14
2. **Mechanical design completed:** The physical housing design will be complete and ready to be integrated with the digital hardware. Estimated completion date: March 21
3. **Product integrated:** The radio will be built and be running our application. The radio should be roughly customer ready. Estimated completion date: April 14
4. **Stretch goals (optional):** After the radio's core requirements are met, stretch goals will be attempted. Estimated completion date: End of semester

Budget

Interim Bill of Materials

Item	Quantity	Estimated Unit Cost	Vendor(s)	Vendor Part Number
Software				
.NET Core SDK	1 per team member	$0.00	Microsoft	
Visual Studio Code	1 per team member	$0.00	Microsoft	
Buildroot	1 per unit	$0.00	Buildroot Project	
Mumble Server	1 per deployment	$0.00	Mumble Project	
Hardware				
Raspberry Pi Zero W	1 per unit	$19.99	Microcenter	486575
Screen breakout	1 per dev unit	$19.95	Adafruit	358
Microphone breakout	1 per dev unit	$6.95	Adafruit	3421
I2S Amplifier breakout	1 per dev unit	$5.95	Adafruit	3006
Screen (raw)	1 per non-dev unit	$9.95	Adafruit	618
Speaker	1 per unit	$3.95	Adafruit	3351
D-pad/joystick	1 per unit	$1.95	Adafruit	504
Battery	1 per non-dev unit	$5.95	Digikey	1568-1488-ND
Fuel gauge	1 per non-dev unit	< $2	Digikey	unknown
Battery charge	1 per non-dev	< $2	Digikey	unknown

controller	unit			
Microcontroller (power sequencing)	1 per unit	$0.34	Digikey	ATTINY9-TS8RCT-ND
Channel knob	1 per unit	$0.50	Adafruit	2047
Volume knob	1 per unit	$0.50	Adafruit	2047
Channel encoder	1 per unit	$1.25	Digikey	987-1398-ND
Volume potentiometer + power switch	1 per unit	< $2	Digikey	unknown
Passives	Some per unit	< $5	Digikey	unknown
Connectors (internal)	Some per unit	< $5	Digikey	unknown
Connector (charging)	1 per non-dev unit	< $1	Digikey	unknown
Printed Circuit Board	1 per non-dev unit	< $5	Macrofab	
Fasteners	Some per unit	< $2	McMaster Carr	unknown
ABS filament	Some per unit	< $3	unknown	unknown

Budget Estimate

Fixed costs (software, tooling):	$0.00
Cost per development unit:	< $84.33
Cost per final unit:	< $75.38
Desired number of development units:	2
Desired number of final units:	10
Estimated cost before shipping:	< $842.58

Note: The development units will be scavenged for the Raspberry Pi and screen modules in order to meet this budget.

Special Training

All of the tools selected for this project have excellent online documentation, so there will be no need for any special training. Some of the libraries we will likely use as dependencies have lacking documentation, but are obscure enough that there is no training available. Luckily, these libraries are simple enough that this will not be too much of an issue. For hardware, we have access to experienced mentors for both the digital and the mechanical design.

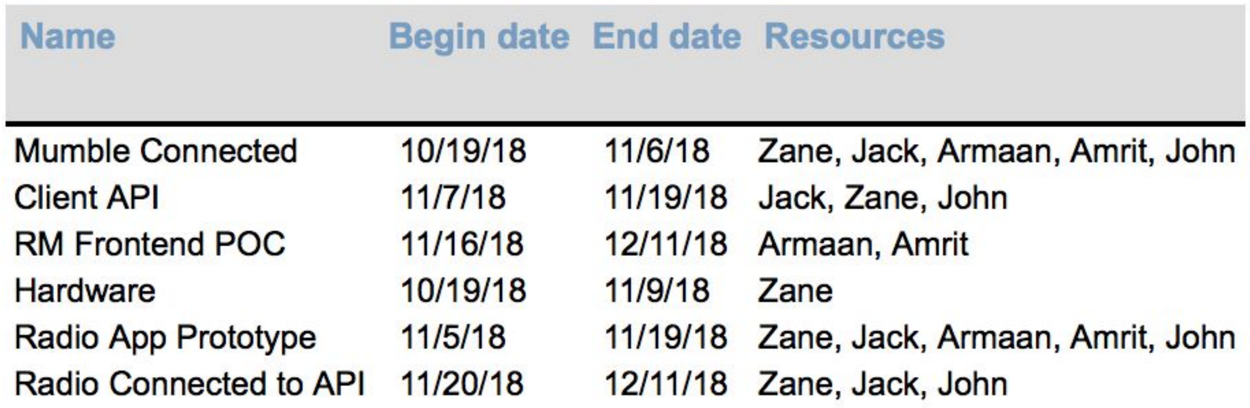
Required Delivery Dates

The software resources are all free and open source, so they can be acquired as soon as needed. The hardware resources will be needed before the end of the semester so that work can commence in integrating hardware and software, but the software architecture permits software development to proceed without having possession of actual hardware. There will likely be two phases of hardware acquisition, for development hardware and for final hardware. This will allow the final design to be changed during the development stage. Some pieces of the hardware that are known to be in both designs may be ordered in the first round to save shipping costs and get the price break of ordering ten items.

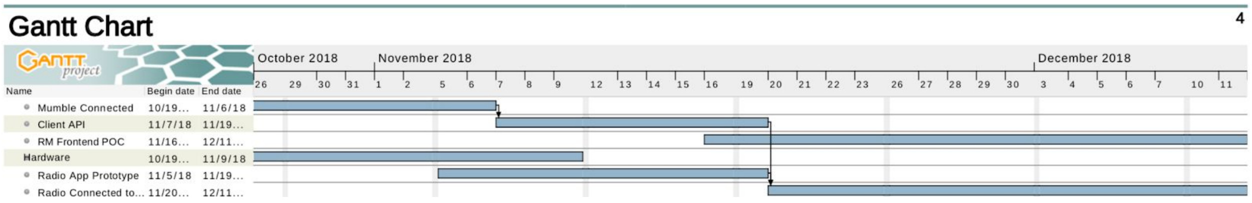
Gantt Chart & Work Plan

First Semester

(RM = Radio Manager)



Name	Begin date	End date	Resources
Mumble Connected	10/19/18	11/6/18	Zane, Jack, Armaan, Amrit, John
Client API	11/7/18	11/19/18	Jack, Zane, John
RM Frontend POC	11/16/18	12/11/18	Armaan, Amrit
Hardware	10/19/18	11/9/18	Zane
Radio App Prototype	11/5/18	11/19/18	Zane, Jack, Armaan, Amrit, John
Radio Connected to API	11/20/18	12/11/18	Zane, Jack, John

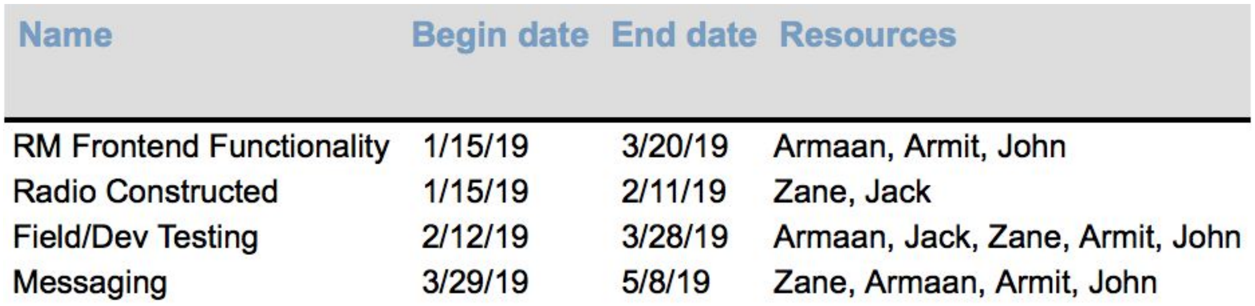


Gantt Chart

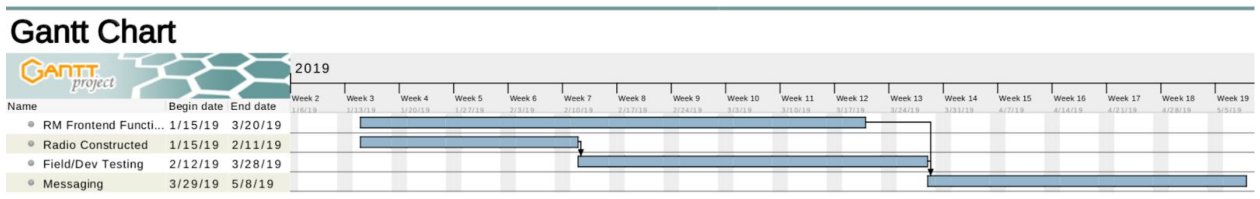
4

Name	Begin date	End date
Mumble Connected	10/19...	11/6/18
Client API	11/7/18	11/19...
RM Frontend POC	11/16...	12/11...
Hardware	10/19...	11/9/18
Radio App Prototype	11/5/18	11/19...
Radio Connected to...	11/20...	12/11...

Second Semester



Name	Begin date	End date	Resources
RM Frontend Functionality	1/15/19	3/20/19	Armaan, Armit, John
Radio Constructed	1/15/19	2/11/19	Zane, Jack
Field/Dev Testing	2/12/19	3/28/19	Armaan, Jack, Zane, Armit, John
Messaging	3/29/19	5/8/19	Zane, Armaan, Armit, John



Name	Begin date	End date
RM Frontend Functi...	1/15/19	3/20/19
Radio Constructed	1/15/19	2/11/19
Field/Dev Testing	2/12/19	3/28/19
Messaging	3/29/19	5/8/19

Preliminary Project Design

Overview

The package consists of the client, RadioApp, and a service backend, RadioManager. Voice traffic originates on the device, is encoded by the RadioApp, then is sent over the local network infrastructure to an instance of RadioManager. The RadioManager then distributes the voice traffic to the units that are currently joined to the logical channel that the traffic originated in. The RadioManager also handles configuration and authentication of the connected clients.

Software On Device

RadioApp

Overview

The RadioApp is the primary software component on the devices. It contains the user interface and audio sending and receive functionality.

User Interface

The user interface module should provide three things: the appearance of the UI, input handling, and the transitions between different UI states (screens). All of these items will be delegated to a pre-built user interface library or framework.

Core

The Core module encapsulates the business logic of RadioApp.

State

The State submodule provides stateful, persistent stores of three types of information: configuration, user preferences, and persistent UI state. Configuration is defined in this context as values that can only be set at provisioning time or by configuring the radio via the Management Console. User preferences are values such as backlight level that are set via a local menu. Persistent UI state includes values that are set by the end user or by some radio action that are not preferences but must survive reboots. The backing store for this data will provide forward compatibility in the case of schema changes.

Service Discovery

The Service Discovery submodule is responsible for locating and identifying backend instances. In normal operation, it must find the address of the backend that was configured in the provisioning mode. In provisioning mode, it must discover and return the names of all available backend instances present on the local network. This will accomplished via a standard service discovery protocol, multicast DNS (mDNS).

Backend Interface

Management and authentication will require the use of some sort of communications channel between RadioApp and RadioManager.

Calling

The Calling submodule receives and transmits audio from the backend. To control the scope of this project, the sending and receiving of audio is delegated to an external piece of software known as “Mumble”. Mumble is a voice chat solution with a client-server architecture. As such, the Calling submodule is primarily concerned with the setup and control of a Mumble client.

Hardware Abstraction Layer

Any hardware-specific device driver interactions will be handled in the HAL submodule. This is what allows the RadioApp to run on normal Linux machines to ease development. This module should be rather minimal and will consist mostly of configuration values (such as device path names and boolean flags) that describe the platform that the RadioApp is running on. There will be only two of these platforms: a dummy platform for development use and a platform describing the actual radio hardware. There may be some classes that provide abstracted access to hardware that is too custom to use standard drivers. The only likely use of this will be the hardware channel knob if it is not mapped to a keyboard-like device.

The HAL also will provide an interaction layer to the WiFi hardware via communicating with wpa_supplicant over DBus. This will be used to connect to the configured network, provision the network connection via WiFi Protected Setup, and retrieve current connection status information.

Board Support

Since we will be using a Raspberry Pi, all of the devices we will use have standard Linux drivers. Thus, most of our kernel-level board support will consist of Device Tree definitions and bootloader configuration. The current plan for the userland is to use a Buildroot-derived userland. Buildroot will take care of assembling the userland and the runtime dependencies we require. The fallback option if Buildroot does not work for us is to modify Raspbian, the official GNU/Linux for the Raspberry Pi. This would be less ideal, but is guaranteed to work.

Software Off Device

RadioManager

The RadioManager is the backbone of our service. There will be one instance of the RadioManager, and it will act as a server to host the clients, or RadioApps. Like RadioApp, it will be built with .NET Core. The main purpose of the Radio Manager will be to segregate clients into specific channels, and route voice data sent to it from the clients. This will happen through an API defined in the RadioManager. This will be an abstraction of the Mumble API, that makes finding channels and sending voice easy from the client's view point.

Management Console

The Radio Manager will also have a front end that will allow a user to configure and manage the clients. This front end is called the Management Console. It will consist of a web interface and full authentication system. This authentication system will be used to verify administrators and give them access to several tools. Some of these tools will include remotely configuring clients, sending full one-to-all broadcasts and possibly sending text messages to devices. (This is a stretch goal.)

Digital Hardware

Overview

To minimize the amount of effort needed to design the hardware, there is an emphasis on using prebuilt modules or highly-integrated solutions wherever possible. As such, the digital design revolves around a Raspberry Pi Zero W. It is common to attach daughter boards (HATs) to the top of the Pi and this will be how our custom hardware will be implemented.

Audio

The only true audio hardware on the Raspberry Pi is an I^{2}S bus exposed on the expansion connector supporting two audio channels. This will be used in a full-duplex mono configuration with one channel feeding an I^{2}S enabled amplifier and the other channel sampling a I^{2}S microphone.

Battery Management

There are three main battery management components: the fuel gauge, the charge controller, and the power hand-off circuit. The first two components are available as integrated circuits that communicate over I^{2}C. The last component may have to be built from scratch, but is relatively simple as its only responsibility is to switch from battery to charger power when the charger is connected.

Power Management

The Raspberry Pi has no way to fully shutdown on its own. To implement this, a very small microcontroller connected to a small, always-on power supply will be used to run a simple state machine. The inputs to this machine will be the state of the power switch and the TX serial line on the Pi (which is pulled high while the processor is running). The outputs will be a signal to the Pi to warn about an impending shutdown and a signal used to switch main power on and off. The power switch will be debounced in hardware so that the microcontroller can spend most of its time asleep.

Screen Connection and Management

This portion of the daughterboard will route the Serial Peripheral Interface lines from the Pi to the screen module. In addition, one of the Pulse Width Modulation outputs on the Pi will be routed to the backlight line to allow dimming of the screen.

User Controls

All user controls (navigation D-pad, knobs, PTT) will be routed through this part of the board. The digital inputs (all except for the volume knob) will be pulled high by the Pi so that the only components required will be resistors. The volume knob will be sampled by an analog-digital convertor connected via I^{2}C.

Mechanical Hardware

The outer casing will consist of two major pieces: the primary shell and a rear cover. The shell will consist of the front face and the sides as one piece. The rear cover will attach via screws to the front. Internally, most of the components will be on the daughterboard, which, along with the Pi, will attach to the front cover. The exceptions to this are the PTT button and knobs, which are affixed directly to the front cover.

Design Constraints

Software

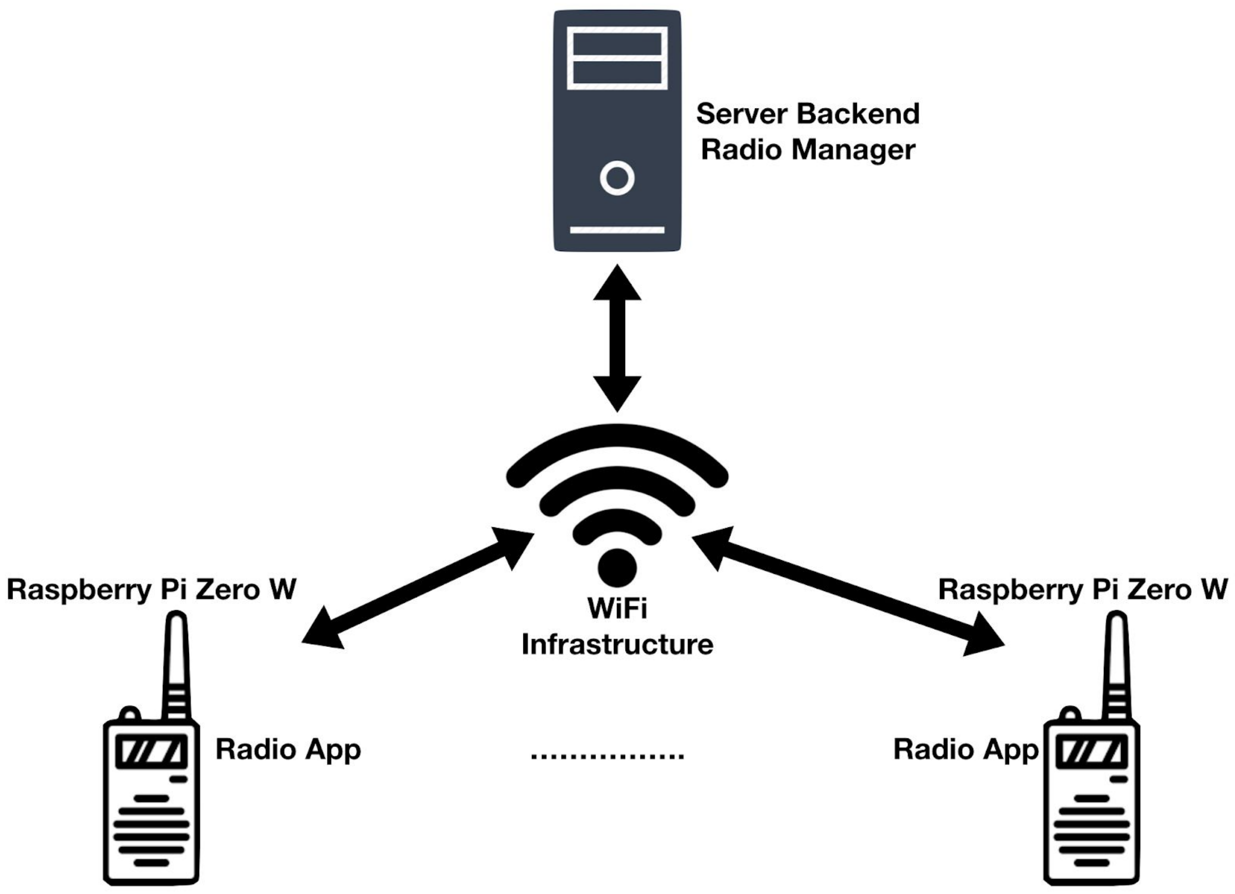
- The device UI should be usable by non-technical users with little or no training
- The device UI should easy to read even in poor lighting conditions or by users with poor eyesight
- The device should boot into a state that allows transmit and receive
- The main UI screen should display vital statuses (WiFi connection strength, RX/TX status, battery life) as icons

- The UI should be navigable using only a 4-way directional pad plus a select button
- Network traffic should be less than 60kbit/s during RX/TX
- Audio quality should exceed normal PSTN telephone voice quality
- The software architecture must be flexible enough to allow RadioApp to run on a normal desktop Linux machine

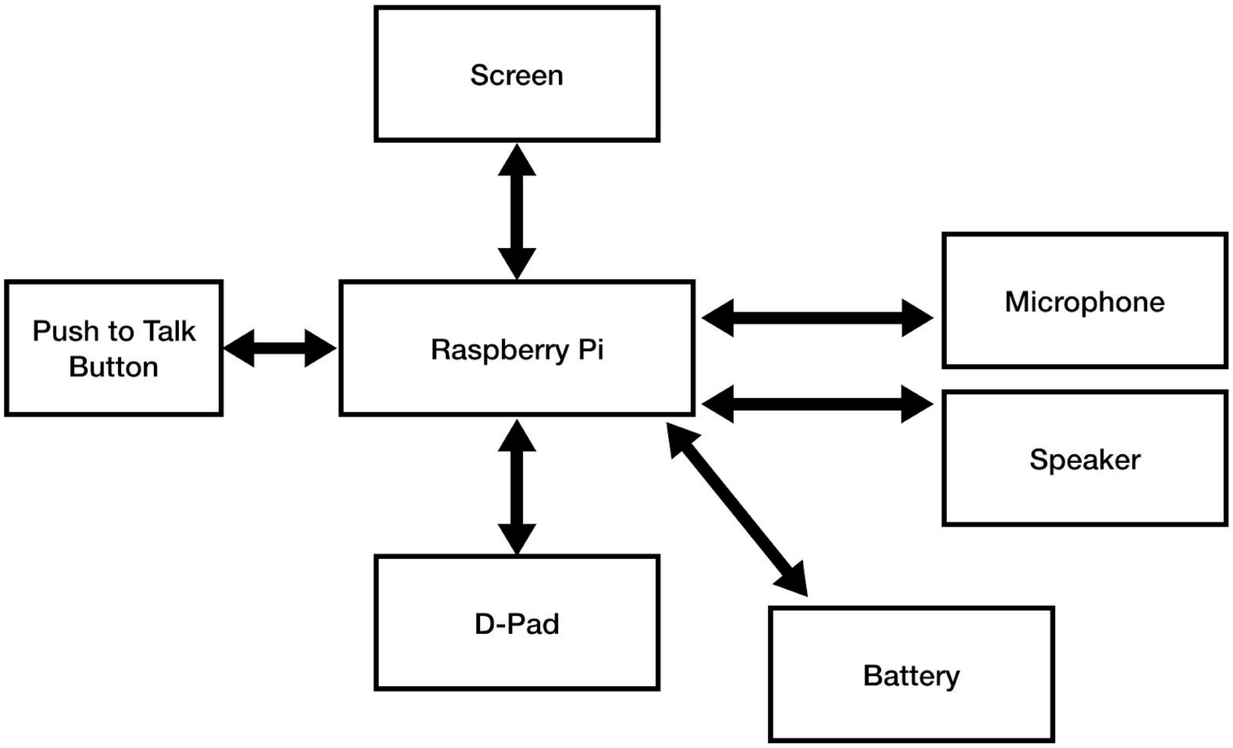
Hardware

- Hardware complexity should be converted into software complexity wherever possible to take advantage of team resources
- All frequent user interactions (volume and channel changes, push-to-talk input) should have hardware inputs
- Battery life should be long enough to last an entire 8-hour shift on a single charge
- The physical casing should be roughly 6 inches tall. The other dimensions should be adjusted to maximize ergonomics
- The power switch should trigger a graceful shutdown when switched off
- Minimize cost and manufacturing complexity

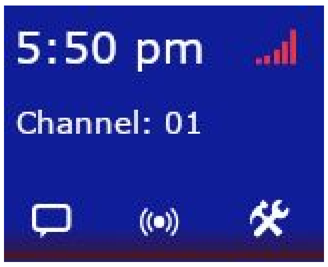
System Components Diagram



Radio Hardware Diagram



Initial Status Screen UI Mock



Ethical and Intellectual Property Issues

Ethical Issues

Current ethical issues remain small with our product. One possible area of issue is security of the communication network as it is a communication device. If an outside person were to listen in on the channel that is supposed to be secure, that would provide a responsibility for us to make sure that does not happen. Our product, however, is not labelled as being capable of secure communications.

This product also makes no guarantee of the reliability required to be used in mission critical situations. It could be possible to extend this product into both of those areas, but it would require more design and a rigorous quality assurance process that we do not have the resources for.

Intellectual Property Issues

Software Licensing

We presently have no plans to commercialize our project. As such, we have decided to keep our design open source so that it may benefit others. The software components, where possible, are MIT licensed to allow commercial reuse. The hardware components will be licensed under an equally permissive license wherever possible.

We are making an effort to use primarily MIT and compatible dependencies where possible. Several parts of the project will be forced to use dependencies with GPL. These will be kept separate from our software so contamination is avoided. Certain portions of our work product will have to tightly integrate with GPL dependencies and will be licensed with appropriate GPL derivative since the redistribution clauses of the GPL would render any MIT-licensed code effectively GPL in any other case.

Employment Agreements

Several of the members of our group are presently employed in software engineering positions. As is common in the software engineering field, they are bound by intellectual property ownership clauses in their employment agreements. None of these employers are interested in enforcing these clauses in the case of this project. This is due to the project being too far from their respective target markets.

Appendix: Change Log

Our team had temporarily switched to a different project idea around the time that the Initial Project Proposal came due. As such, that document described this other idea (“MetaRank”, a HackerRank-like service that generated pseudo-random toy programming languages to test the adaptability of candidates for software engineering positions). Every section shared by this document and the initial version has changed except for the team information.