Northern Arizona University

Team Radio Pi

Capstone Team	*Project Sponsor*	*Mentor*
Tyler Plihcik	**General Dynamics**	David Failing
Brandon Click	**Mission Systems**	
Jefferey Williamson	Randy Derr	
Kaelen Carling		
Saurabh Jena		

Software Design Document

Version 2.1

February 10th, 2021

Contents

Introduction

The world runs on multitudes of IoT (Internet of Things) devices; everything from smart home devices, to wifi extenders, speakers, even cars and kitchen appliances. These devices communicate via radio waves and run on what are called embedded chipsets, which are inherently small, in both footprint and power consumption. As our technology continues to advance, the need for smaller and smaller embedded systems becomes more urgent.

For nearly three decades software defined radios (SDRs) have been revolutionizing the way these IoT devices communicate. Leading the charge into this new era of is our project sponsor, General Dynamics Mission Systems. The need for newer, smaller, more efficient yet more powerful systems is great. Current solutions are often times built on legacy software, and although they still work, are definitively not the best solution given our technological advances since the late twentieth century. Because of this, Software Defined Radios are prone to:

- Failure
- System crashes
- Memory overloads
- General decay

Our envisioned solution is to redesign General Dynamics SDR Command and Control System from the ground up. We will be simulating this solution on a Raspberry Pi, a very small yet powerful, computer. We will establish an interconnected database and web interface in the most efficient language(s) available, including networking protocols and database languages. We will then ensure that the end user has complete and total control, given they have the proper authorization, over the system. From querying the database to collecting, sorting, and storing data to controlling permissions and viewing system diagnostics, our envisioned solution will attempt to right all the wrongs of current command and control interfaces for embedded systems.

The following will serve as the design “blueprint” of our redesigned SDR command and control system (C2 system). Our implementation must ensure that the overall footprint of the C2 system remains very small as is will be running on an embedded chipset. Our implementation must also be scaleable, that is, to be able to control and monitor many devices, as well as our backend being able to handle a very high volume of requests, at times simultaneously, with no degradation of quality of service. Our backend database will support a variety of data types including but not limited to:

- Temperature
- Device ID primary key

- Log file pointers
- Device status
- Time
- Size of database

The remainder of datatypes and database support information can be found in the architectural overview. In addition to these functional requirements, we must also ensure security, speed, and quality. The C2 system must be useable and efficient under load.

This document will be used as a reference by Team Radio Pi members throughout the development process, as well as by the team's client and mentor as a vision and high level overview of our products design.

Implementation Overview

Our envisioned solution, coupled with our directive from the project sponsor, is to build a completely redesigned and reimagined C2 system to interact with GD's software defined radios, simulating real hardware with a Raspberry Pi 3 B+. Our solution will feature:

- Fast and efficient transfer of data
- Modern hardware
- A slick user interface
- Small overall footprint
- High throughput
- Low power consumption
- Elegant continuity of tasks and functions

Our solution will be a simulated prototype, which we will present to the project sponsor as a proof of concept.

To accomplish this, as mentioned earlier, we will be using a Raspberry Pi 3 B+ as our hardware platform, using the most recent release of Raspberry Pi OS, installed via noobs. In the realm of software, our solution relies heavily on our database and networking protocol performance. We will be implementing a SQLite database, which will store the datatypes mentioned above and throughout this document.

Our networking protocols will be UDP and TCP. UDP will be used when the data being transferred over the network is non system critical and time sensitive, this is because UDP does not have functionality to ensure data delivery, but is faster than TCP. TCP will be used when the data being transferred is system critical data or sensitive, this is because TCP favors confirmation of data delivery over speed. This network will have a backbone of socket programming, written in C, to allow for device communication.

Our front end user interface (UI) will rely on the python micro web framework Flask. Flask makes our web development process smooth and painless while adding some python functionality that is essential to our system.

Architectural Overview

The diagram pictured below (Figure 1) represents a high-level architectural view of our solution. It containing three distinct major components, each with its own smaller yet still very important components inside.

We will look at this diagram from the bottom up. The first, and most basic component of the solution will be the SDR embedded device which itself has 3 smaller components in it, in our case this is a Raspberry Pi 3 B+. The device will have an its own database, storing all device specific information locally. This will communicate with our data manager which receives, formats, and stores the data from the device. The data manager connects to the network we have built for communication, this component connects the simulated SDR devices to the next major component, our Shepherd C2 System.

The Shepherd C2 System is the “go-between” for the SDR devices and our front end user interface (UI). We have dubbed this component the “Shepherd System” because it acts as the shepherd, controlling his flock (our devices) and keeping them all in one place This major component also contains three smaller components within it. Firstly, we have the Shepherd System itself, which is networked to each SDR device over the network mentioned above. This will contain the addresses for our front end, and the SDR device database. It will process, send, store, and retrieve data. This SDR device database will be the staging area for all of our embedded devices, where you can see all devices, on and offline, and is the access point to individual device data. Driving this component is our custom built Shepherd API, which receives and parses the requests from our front end to the Shepherd System..

Finally, we have the front end UI. This will be the component which allows the user to actually command and control the devices being monitored on our command and control system. In the interface, you will be able to see the status of all devices registered in our SDR device database, and access the information in each one. The user will also be able to request specific information by performing certain queries, after which the data will be displayed to the user in such a way that is easily digestible.

Our solution as a whole, in short, works as follows: the simulated SDR device will store its data locally, and be networked with our Shepherd System, this system will contain all SDR devices, and the user will be able to access the Shepherd system and the devices it contains via the front end user interface.

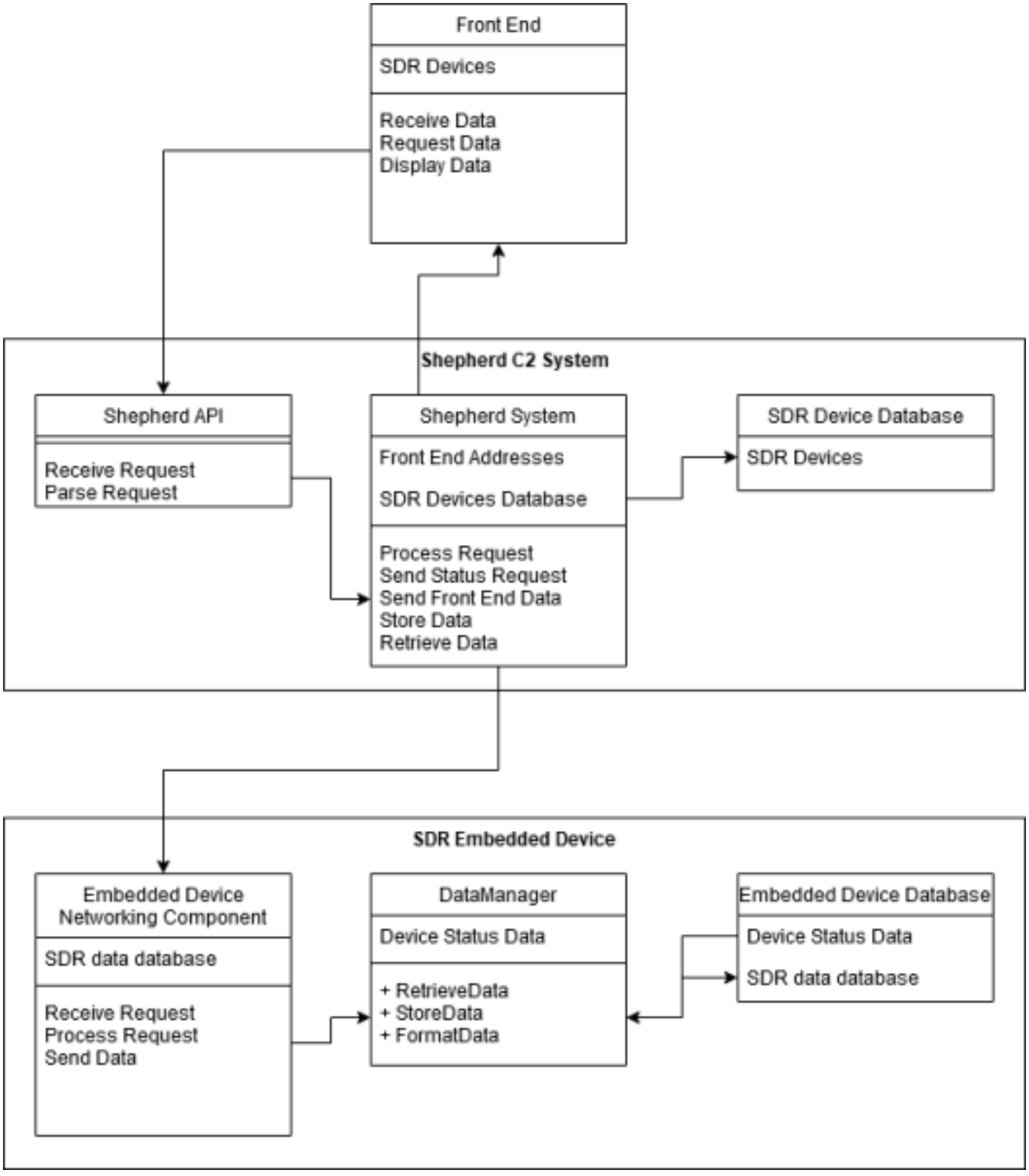

Figure 1. Architectural Diagram of Team Radio Pi C2 System.

Module and Interface Descriptions

The following is a continuation of the component descriptions from above, in greater detail. Broken into three separate sections, each with its own sub sections which will describe each smaller component.

Section 1:

SDR Embedded Device

The SDR embedded device will be simulated on a Raspberry Pi 3 B+, because of its low resource consumption and availability it is a perfect substitute for an actual SDR device.

Section 1.1:

Embedded Device Database

The embedded device database is one of the core elements of our solution. It will house all essential device information that is needed for the command and control system.

As you can see in the Entity Relationship Diagrams (ERD's) below (Figures 2 and 2.1), the database will have four main tables in it:

- Temp Log
- Device Information
- Process / Transition
- Connections

The Temperature Log table, or temp log for short, will contain the temperature of the devices processor, and the time that temperature measurement was taken.

The Device Information table will contain the current status of the device, online or offline, the device ID, which will serve as the primary key for the SDR device database, as well as for each embedded database. It will also contain the size of the database to ensure we continue to be efficient with our memory usage.

The Process / Transaction table will contain the status of the current running process:

- New
- Ready

- Waiting
- Running
- Terminated

As well as the time that process was loaded into the queue, and the instruction. In this case the instruction will be database commands, select, update, delete, insert into, create, or alter.

The Process / Transaction table is related to the Connections table as follows. Each process / transaction can have one and only one connection, and each connection can have zero or many processes / transactions.

The connections table will contain the time in which the connection began, the time it ended (an optional entry into the database as the connection may be viewed before ending). There will also be a connection ID, which will serve ads the primary key for this particular table as well as a foreign key in the process / transaction table. Finally, the connections table will store the type of connection being used.

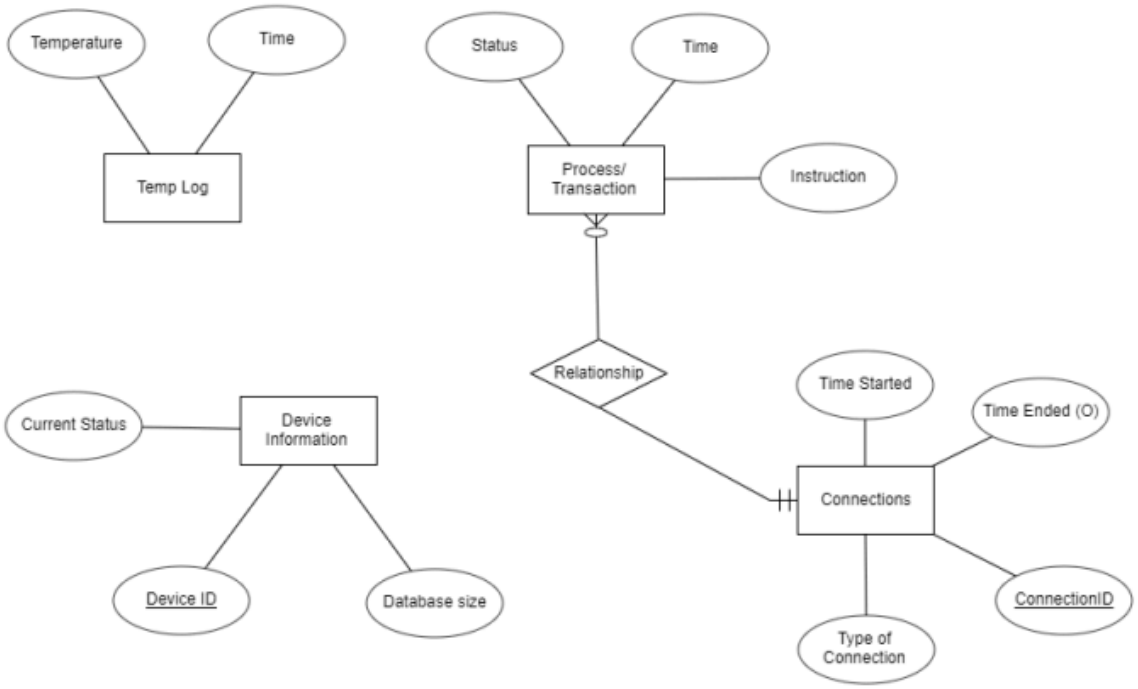

Figure 2. Embedded Device Database ERD

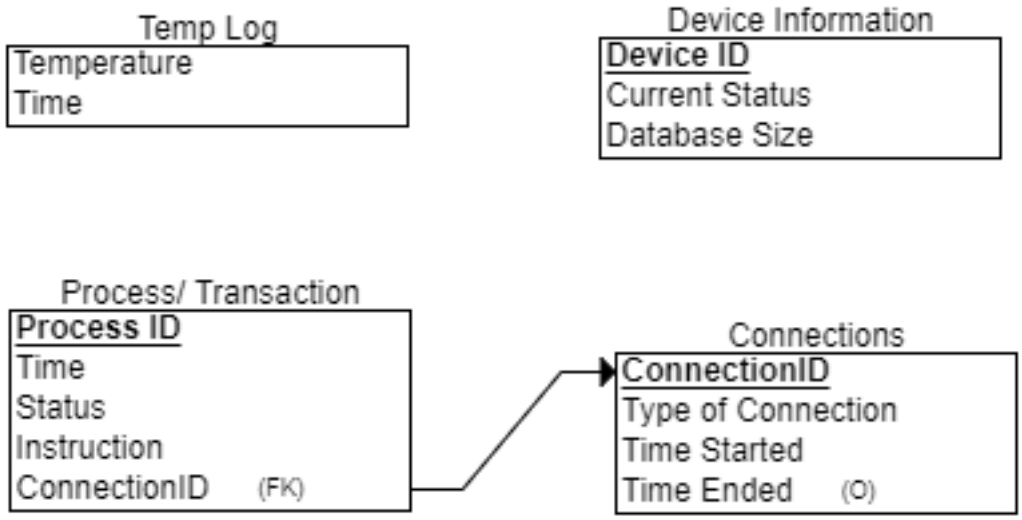

Figure 2.1. Embedded Device Database ERD

Section 1.2:

Data Manager

The device data manager will interact with the embedded device database by storing device stats data, and be able to access and query the device database. This includes retrieving data from the device, formatting that data, and then storing it.

Section 1.3:

Embedded Device Networking Component

The embedded device networking component is the data pipeline driving our solution. This will implement two different protocols, UDP and TCP. As mentioned earlier in this document, TCP will be used for system critical, non time sensitive data transfer. UDP will be used for non system critical, time sensitive data. The networking component will receive data requests from the Shepherd C2 System, process those requests, and send them to the correct destinations within the system.

Section 2:

Shepherd C2 System

The Shepherd C2 System the backbone of our solution. Its purpose is to have all of the SDR devices stored in one place, virtually speaking. This will streamline the management of the devices.

Section 2.1:

SDR Device Database

The SDR device database will be a small database as far as tables are concerned, as it compares to the embedded database contained on each device. It will simply be a table containing the SDR device ID, which is used to then access the individual device's data. A visual is shown to the right (Figure 3).

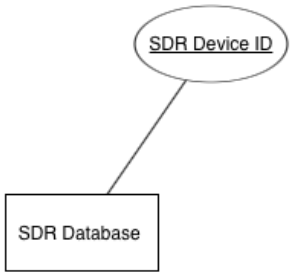

Figure 3. SDR Device Database

Section 2.2:

Shepherd System

The Shepherd system is the most complex part of the overall shepherd component. It will use front end addresses and a connection to the SDR devices database to process requests, send status requests, send data to the front end UI, store data in the device database (in the case a new device is registered, or an old device is removed), and to retrieve data from the SDR device database. This system interacts closely with the custom Shepherd API described below.

Section 2.3:

Shepherd API

The Shepherd API is our mediator between the front end user interface, and the Shepherd C2 System. This component will receive requests from the UI, and parse those requests out to the Shepherd System, which initiates contact with the SDR database, which obtains information from each individual device over the network, using either UDP or TCP depending on the task.

Section 3:

Front End User Interface

The front end user interface is a stand alone component of our system and connects to the Shepherd C2 System. This interface will allow the user to view all registered devices, their status access the data stores in their database, query specific information from specific devices, as well as add or remove devices from the system as a whole, depending on the level of access the user has within the system.

Implementation Plan

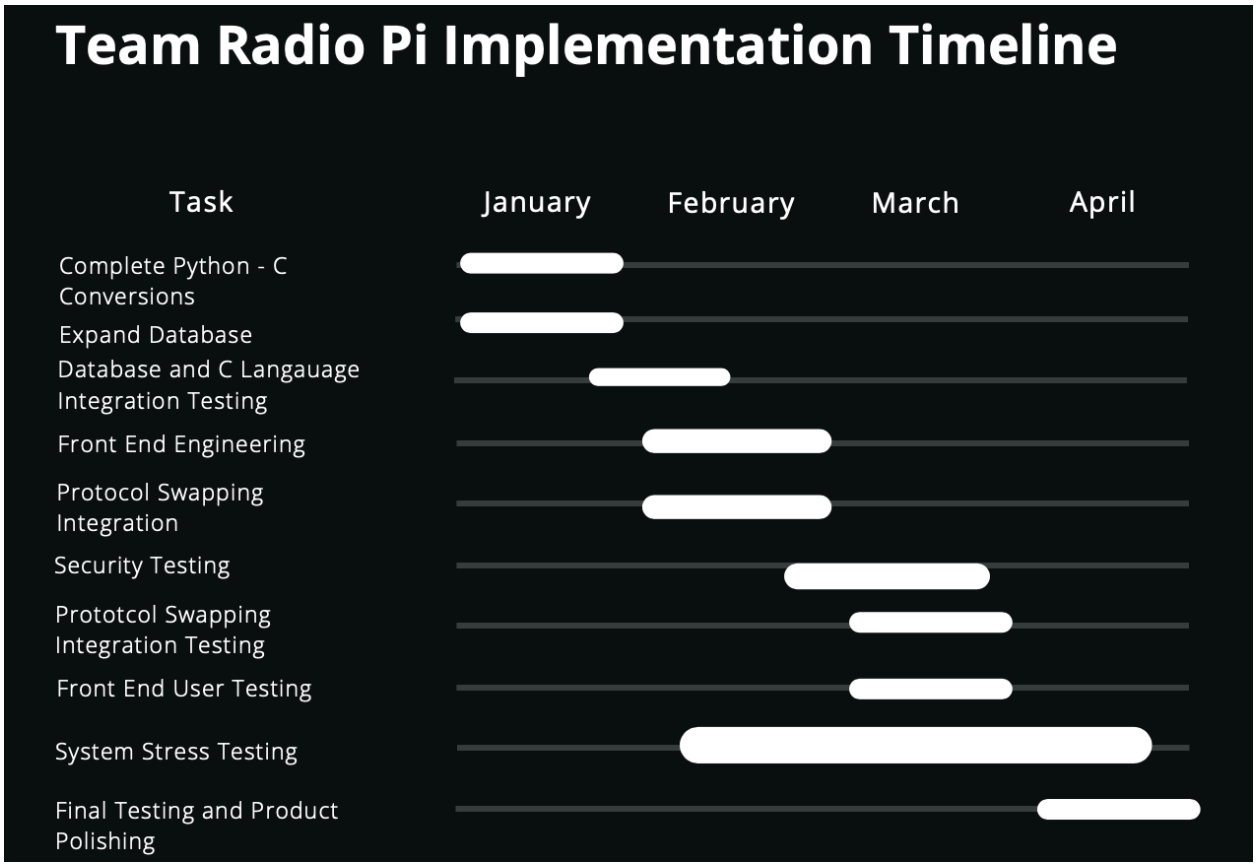

Figure 4. Team Radio Pi Implementation Timeline, visualized as a Gantt Chart.

As seen above in the project time line (Figure 4), we have a few large tasks that must be completed in January, those being Python to C language conversions of certain parts of our source code, and database expansion. For our prototype, we decided to use python to prove that what we had proposed to our client was feasible, for its ease of use and practically unlimited functionality, we chose python. Now that our prototyping phase is over and we are in full development, it is essential that we convert specific parts to C for the C language's low overhead high efficiency properties. In the prototyping phase we also only used one table with two attributes for our database, the temperature table, containing temperature and time of measurement. Now that we have proven feasibility, we expand our database to conform to what is stated in our signed requirements document. Starting in February, following regularly scheduled end of month integration testing on completed modules, we being to do the heavy lifting on the front end user interface. In addition to that, our main large task in February is to implement protocol swapping based off of the task, as mentioned throughout this document. This

module will be essential to providing the quickest and most reliable service to our client. Additionally, we will begin security testing to ensure the integrity of the data being transferred back and forth on our network. These two tasks will continue into early March. At the end of this phase we will perform the integration testing of new modules, as per usual at the end of the month, or thereabouts. Beginning in March, the front end engineering will be nearing completion, it will be important in meeting our non functional requirements of usability and ease of learning of the UI for us to have extensive user testing. This testing will be done internally, only by those who have signed the necessary legal documents provided to us by the client to ensure authorized access only. We will also being stress testing our system at this time to ensure that it can handle extremely large volumes of traffic and have no degradation of service whatsoever. As March comes to a close, we will be mostly finished with all engineering, it is at this point we will be making final touches and finishing polishes on the product, compiling user guides and manuals, and look forward to final product delivery in late April.

Conclusion

This document stands as a “blueprint” for our solution. Containing a high level overview of the solution components, followed by an in-depth description of each individual component, we have laid the ground work for a successful development phase looking forward to final product delivery in late April, 2021. All milestones and tasks have been completed on time and engineered very well to this point at the writing of this document, and no roadblocks are foreseen by team leadership that would cause a delay in delivery. Our solution will be delivered to General Dynamics Mission Systems as a proof of concept to a proposed solution of a completely redesigned command and control system for embedded products. Using modern hardware and well engineered software, we believe that our solution will exceed the expectations set forth to Team Radio Pi by GDMS, and the NAU Capstone staff.

Glossary

Noobs - New out of the box software, noobs is a compressed configuration file making the install of a new system easy and fast

UDP - User datagram protocol, a core member of the internet protocol suite

TCP - transmission control protocol, a core member of the internet protocol suite

UI - user interface, the interaction point between human user and computer system

ERD - entity relationship diagram, a structural diagram used in database design

Overhead - an excess of computing time and resources spent on a certain process