UNIVERSITY of WASHINGTON | BOTHELL

In affiliation with

PACCAR Inc

Vehicle-to-Vehicle Update Delivery System

Final Report

Version 1.2

Updated June 11, 2021

Austin Gilbert **#1737454**
Aashima Mehta **#1970936**
Cameron Ufland **#1061818**

Table of Contents

1. Abstract

Commercial vehicles operate in many different environments and an internet connection may not be available most of the time. Due to this lack of internet connectivity, many commercial vehicles have software that is out of date, leaving their owners vulnerable to a lack of software support and security breaches. Our team has therefore created a close-range receptionless Vehicle-to-Vehicle update delivery platform consisting of two or more Raspberry Pi development boards. With pre-installed updates of varying versions, each board creates a mesh Ad-Hoc network in tandem and decides via version number the direction of the update file transfer. This update platform was designed to be used as a proof of concept in PACCAR's fleet of trucks, where trucks with later versions can find, update, and disseminate updates to trucks with earlier versions.

2. Introduction

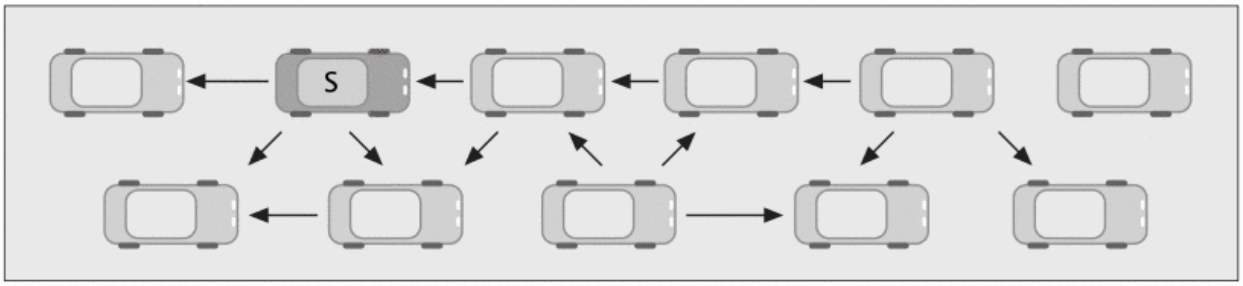
2.1 Background

Although first conceived in the early 2000s, recent advances in networking technology and in-vehicle regulations have allowed Vehicular Ad-Hoc Networks (VANETs) to begin implementation in vehicle infotainment systems. This technology allows for vehicle-to-vehicle (V2V) and vehicle-to-roadside communication and can be used for a variety of applications including improved road safety, traffic awareness, emergency response, driving, and vehicle maintenance. VANETs also have many advantages over internet-based communication as they are more reliable and much faster for close-range V2V communication.

Although VANETs hold great potential, they have not yet made their way to the trucking industry. Currently, trucks used for logging, mining, or transportation purposes are operated mostly in areas with little or no internet coverage. During maintenance, technicians usually must drive out to these remote locations making it difficult to update trucks to the latest software. Additionally, many trucks used for transportation have software that is out of date due to the lack of internet connectivity at truck stops and other remote areas. Trucks with outdated software are more vulnerable to a lack of software support and security breaches, which can be dangerous for truck drivers who rely on the same software to carry out their work.

In areas with little or no internet access such as rural highways and truck stops, VANETs can work to replace functionality that currently can only be performed via an internet connection. One such functionality is downloading and distributing the latest vehicle software. Since VANETs can be used to disseminate information amongst multiple vehicles as shown in Figure 2.1.1, this technology can be used to download and distribute vehicle software updates.

Figure 2.1: Vehicle Update Dissemination



The motivation for the V2V Update Delivery System is therefore twofold: to transfer software updates from one vehicle to another to stave off security breaches and to distribute such updates across an entire fleet of trucks fairly quickly. In combination with downloading updates from the internet, this technology can ensure the trucks from our industry sponsor, PACCAR, stay up-to-date no matter the location or connectivity. Additionally, this technology will allow PACCAR to more easily support their software and keep it secure for their customers.

2.2 Project Deliverables

As per specification, the goal of this project is to present a proof of concept for the automatic delivery of updates between two devices in proximity, with the direction of update transfer determined via version number. Using two Raspberry Pi single-board computers, each with a 3.5-inch touchscreen and EAFU portable power bank, we have tested for functionality, usability, and performance of the project. In addition to our testing results, deliverables include the hardware, firmware, software, and graphical user interface (GUI) to be reviewed by the advisors.

3. Hardware Specification

The following sections serve to address all hardware components, the reasons why we selected them, and how they add to our project.

3.1 Overall Hardware Design

The V2V Update Delivery System consists of two Raspberry Pi 4 Model B (2 GB) single-board computers, each with a 3.5-inch touchscreen, an EAFU portable power bank, a 16 GB Micro-SD Card, and the essential power and data cables. Specifically, the Raspberry Pi (RPi) is used to run all firmware and software for the project when powered by the power bank and can display such interactive software via the touchscreen. When all of this hardware is put together, our devices are representative of two vehicle infotainment systems that can send cached update files to one another or a tertiary device at great distances and speeds. Although the three main devices can take up space, the sizes of each and how they are joined together make the overall hardware compact and portable.

Overall, our hardware is limited to 2 GB of RAM, 16 GB of storage, 150 Mbps of signal strength, and a 10,000 mAh battery capacity, which is more than enough for the needs of the firmware and software.

Figure 3.1.1: Hardware Top View

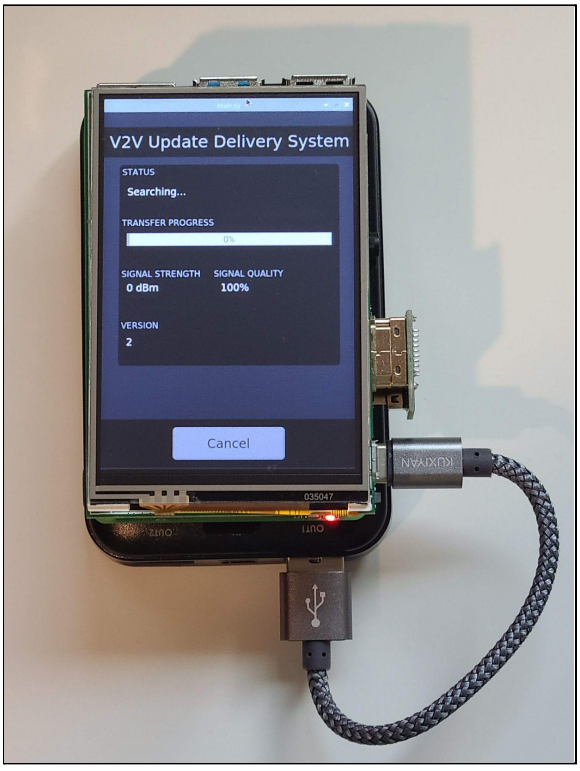


Figure 3.1.2: Hardware Angled View

Figure 3.1.3: Hardware Side View

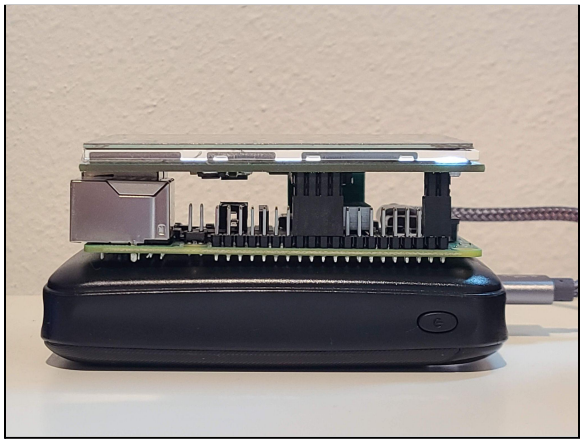
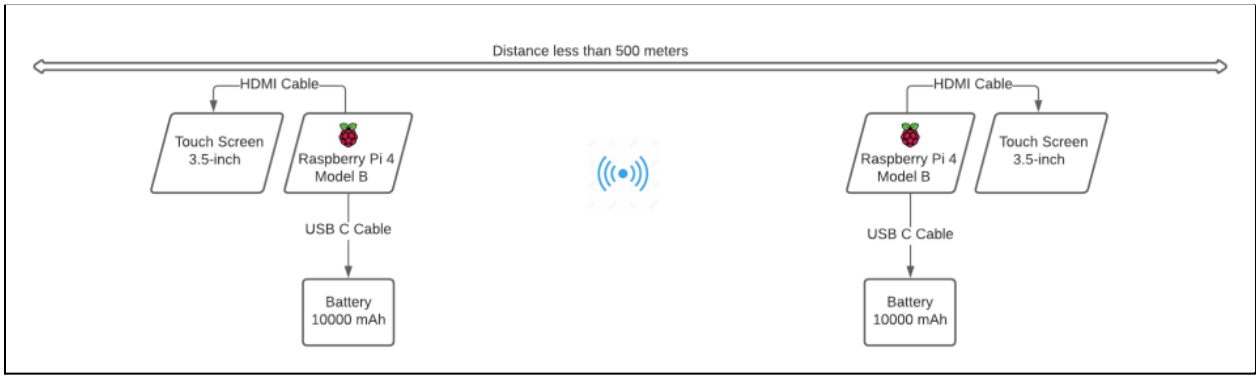


Figure 3.1.4: Hardware Design Diagram



3.2 Raspberry Pi

The Raspberry Pi 4 Model B (2 GB) single-board computer acts as a small computer with a Linux-based operating system called Raspberry Pi OS. The Raspberry Pi features a 1.5 GHz Quad-core 64-bit ARM Cortex A72 processor with 2 GB of LPDDR4-2400 SDRAM and 16 GB of storage via an attached microSD-HC card. With these specifications, the Raspberry Pi can be a powerful tool for a number of IoT and other embedded applications as its simple user interface and operating system make it easy to create and run relatively high-intensive programs.

In the context of our project, we chose the Raspberry Pi over other single-board computers due to its reliability, active community, powerful processor, competitive price point, and wide range of hardware support. We specifically chose the most basic, newest 2 GB model to have enough RAM to run our firmware/software and the 3.5-inch touchscreen that is attached to each Pi.

Figure 3.2.1: Raspberry Pi 4 Model B

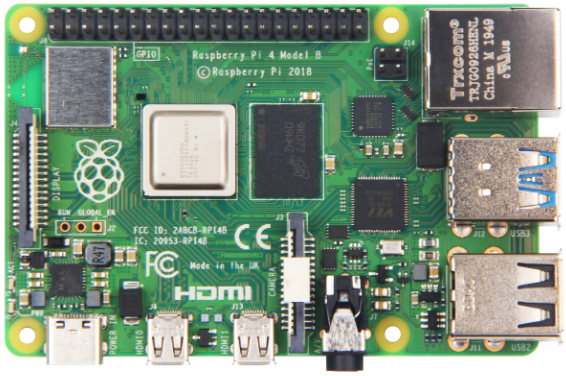


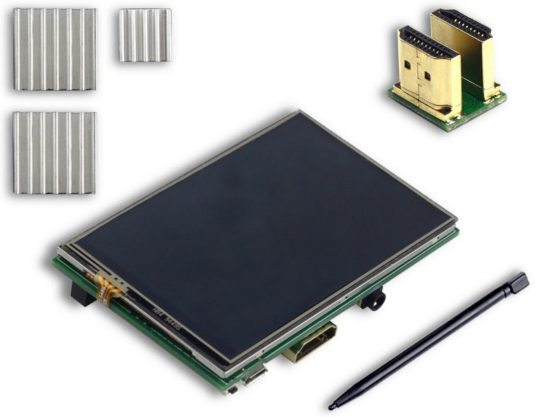
Figure 3.2.2: Attached 16 GB Micro SD-HC Card

3.3 3.5” Touchscreen

To display our program’s GUI and make it interactable, we used two UCTRONICS 3.5” LCD HDMI Display Model B0106s and connected them to the top of each Raspberry Pi via 12 pins and an HDMI interface connector. The touchscreens also came with heatsinks and a stylus to prevent overheating of the Pis while they are connected to screens and to more accurately tap the display respectively.

In the context of our project, we chose this touchscreen because it had great reviews for its price point and was entirely compatible with the Raspberry Pi 4 Model B. Additionally, it was very easy to set up and use with the provided drivers and has consistent software and technology support from UCTRONICS.

Figure 3.3: UCTRONICS 3.5” HDMI Display



3.4 Portable Power Bank

To make our project portable and enable field testing, we used an EAFU Portable Power Bank to supply the Raspberry Pi and touchscreen with power. This power bank uses a 10,000 mAh lithium-polymer battery, giving enough power to the Raspberry Pi for multiple hours of use.

For our project, we chose this specific power bank because of its small size, lightweight design, great reviews, LED numeric display, and most importantly, 3 A of output current required to fully power the Raspberry Pi 4.

Figure 3.4: EAFU Portable Power Bank

3.5 USB-C to USB-A Cable

To connect the Raspberry Pi to the portable power bank, we used USB-C to USB-A cables rated for 3.1 A of current. As the portable power bank delivers 3 A over USB-A and the Raspberry Pi uses 3 A over USB-C, this was the perfect connector to power our device.

We chose this particular cable because its 6-inch long size allowed it to discreetly connect the battery and the Pi without using much space. Additionally, this cable had great reviews and included a 56 kΩ resistor and military-grade sheathing to protect the USB circuit.

Figure 3.5: USB-C to USB-A Cable

4. Software Specification

The following sections serve to address the project's firmware and software development, including the program's theory of operation, organization, and choices of libraries or methods of programming. Additionally, this specification describes how each library and file add to our project.

4.1 Overall Program Design

The program that runs on each of our Raspberry Pis can be divided into two major design components: firmware and software. In terms of firmware, B.A.T.M.A.N. Advanced is a tool used to allow our Raspberry Pis to create and connect to an Ad-Hoc mesh network in the background. Since it handles all connections and disconnections dynamically, our team did not need to worry about handshake between devices or reconnection if a device loses connection. In terms of software, we created a multi-tiered, multi-threaded software package that runs a set of Python files to broadcast, connect, and transfer an update file to another device all while displaying transfer statistics on a simple GUI. *Figure 4.1.1* compartmentalizes the program into visual blocks while *Figure 4.1.2* illustrates the step-by-step operation of this package.

Figure 4.1.1: Program Software & Firmware Layout

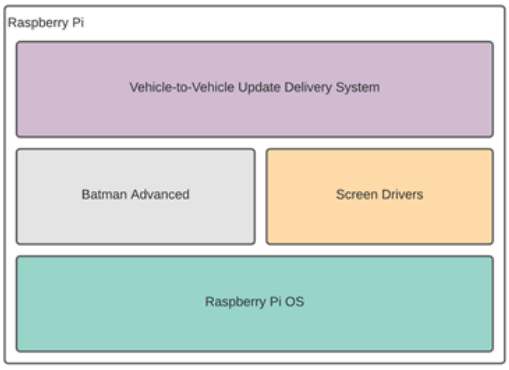
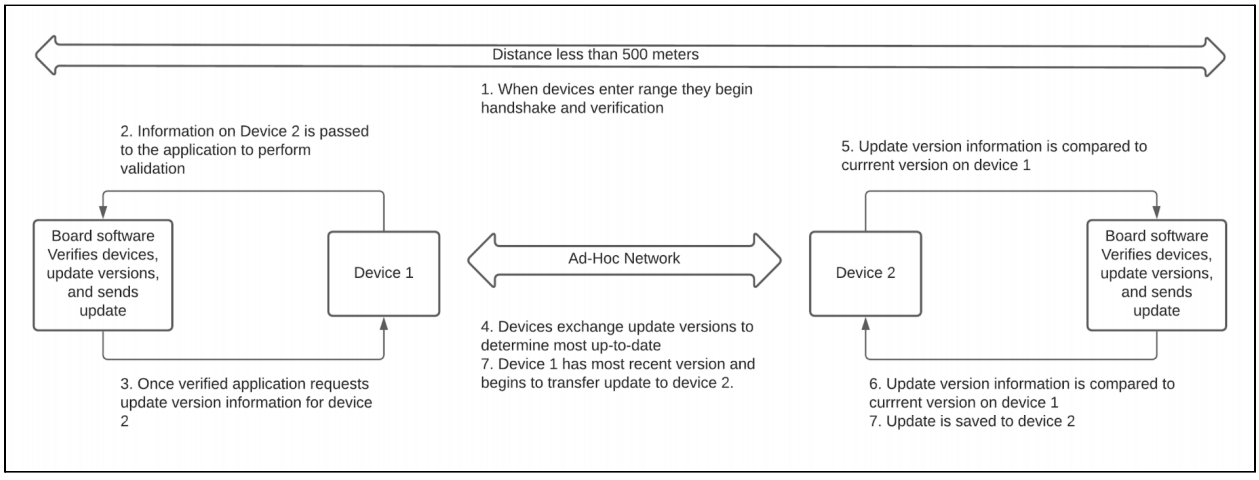


Figure 4.1.2: Step-by-step Operation of Program



On startup, the Raspberry Pi will automatically run multiple bash scripts which have been edited according to the SETUP.md guide that we created as a repair manual for our project. Following everything in this guide will allow the Raspberry Pi to first rotate its screen vertically, then configure B.A.T.M.A.N. Advanced and the mesh network, then set up the touchscreen interfaces, and finally launch the program after waiting 15 seconds for all necessary processes. The GUI will then open as a full-screen tappable window and begin updating statistics while searching for other Pis in an infinite loop. To refresh the program, we have included a touch-enabled “Cancel” button, and to exit out of the program, we have included the “X” button in the upper right portion of the GUI window.

4.2 Firmware Design

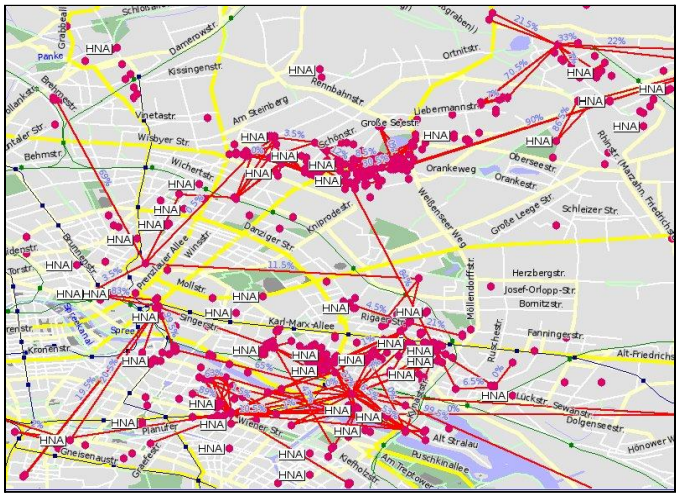
4.2.1 B.A.T.M.A.N. Advanced

Standing for “Better approach to mobile Ad-Hoc networking,” B.A.T.M.A.N. Advanced (batman-adv) is the most popular implementation of the B.A.T.M.A.N. Ad-Hoc network multi-hop routing protocol and has been part of the Linux kernel since 2010. Batman-adv operates on ISO/OSI layer 2 (ethernet layer), which means that all data traffic is handled by the program itself and all devices in the network are “unaware of the network’s topology as well as unaffected by any network changes” (“B.A.T.M.A.N. advanced”). To translate, this means that all data flowing through the mesh network is optimized automatically by batman-adv, and each new device that joins the network is doing so by first deploying the same network individually. In this way, if a connection becomes severed, batman-adv will just split the network into separate smaller networks that can freely combine again when reconnecting just like two combining water droplets.

Figure 4.2.1: B.A.T.M.A.N. Logo

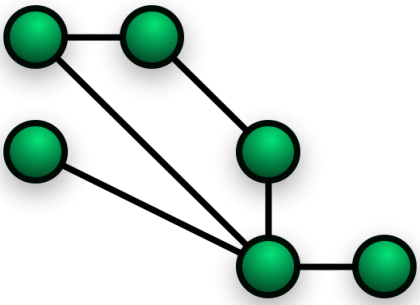
In the background, batman-adv is constantly broadcasting a device’s existence to its neighbors, which in turn relay the message to other neighbors, and so on. Routing to different devices uses logs of these messages to determine the best possible path on a device-by-device basis. Overall, batman-adv is commonly compared to a giant network switch that routes data packets across multiple devices to a destination node, working almost like a game of Chinese checkers. When a large number of devices are connected to such a mesh network, it can look something like the Berlin community-driven network shown in *Figure 4.2.2*, where each dot represents a device and each line represents a connection between devices.

Figure 4.2.2: Freifunk Berlin Community Mesh Network



Although most use cases for batman-adv involve connecting the network to an ethernet gateway and sending internet packets amongst devices, this project will not involve an internet connection. Therefore, in the context of this project, the batman-adv mesh network can be visualized as a simple mesh of connections like in *Figure 4.2.3* where each green dot represents a Raspberry Pi, and each line represents a connection between two Pis.

Figure 4.2.3: Simple Mesh Network



As soon as a Raspberry Pi is within range of another Pi, they will make a connection, creating a WLAN (WiFi) mesh network connection with a coverage area larger than the range of one Raspberry Pi. Once connected, every device within the mesh network will be able to communicate with one another. All handshakes in the network will be managed by batman-adv, allowing the routing of data to begin when there is at least one connection. Since we wanted static IP addresses for each of our Pis during each session, we chose to use an IPv4 interface for our mesh networks where addresses are automatically assigned when the device turns on.

Additionally, another implementation tool for B.A.T.M.A.N. is batctl, which helps configure, debugs, and pulls status from batman-adv. Batctl is used to set up the batman-adv network on startup and has some commands in various startup-related bash scripts.

Setting up this firmware disables normal WiFi access, but does not affect ethernet access. However, connecting ethernet to one of the Pis will interfere with the software and is recommended against while the software package is running.

4.2.2 Touch Screen Drivers

For the UCTRONICS 3.5” touchscreen to function correctly with touch, various drivers must be installed and several files must have extra commands written to them. The reason for extra commands in several files is that we must rotate the touch interfaces in addition to rotating the screen for the vertical GUI. Instructions on exactly how to do this as well as links to the drivers can be found in the final section of SETUP.md.

4.2.3 Raspberry Pi OS

Formerly known as Raspbian, Raspberry Pi OS is a Debian-Linux-based operating system with a familiar Desktop and menu interface to many other operating systems. Citing the official setup documentation for the Raspberry Pi (first section of SETUP.md), we imaged our Micro SD card with Raspberry Pi OS (32-bit) and plugged the card into the Pi to boot into it. Once installed and configured, we used the terminal and editor of choice, Microsoft VSCode, to edit, test, and run our code. We chose Raspberry Pi OS over other Linux-based operating systems since it was the simplest to use and had the most support for Raspberry Pi.

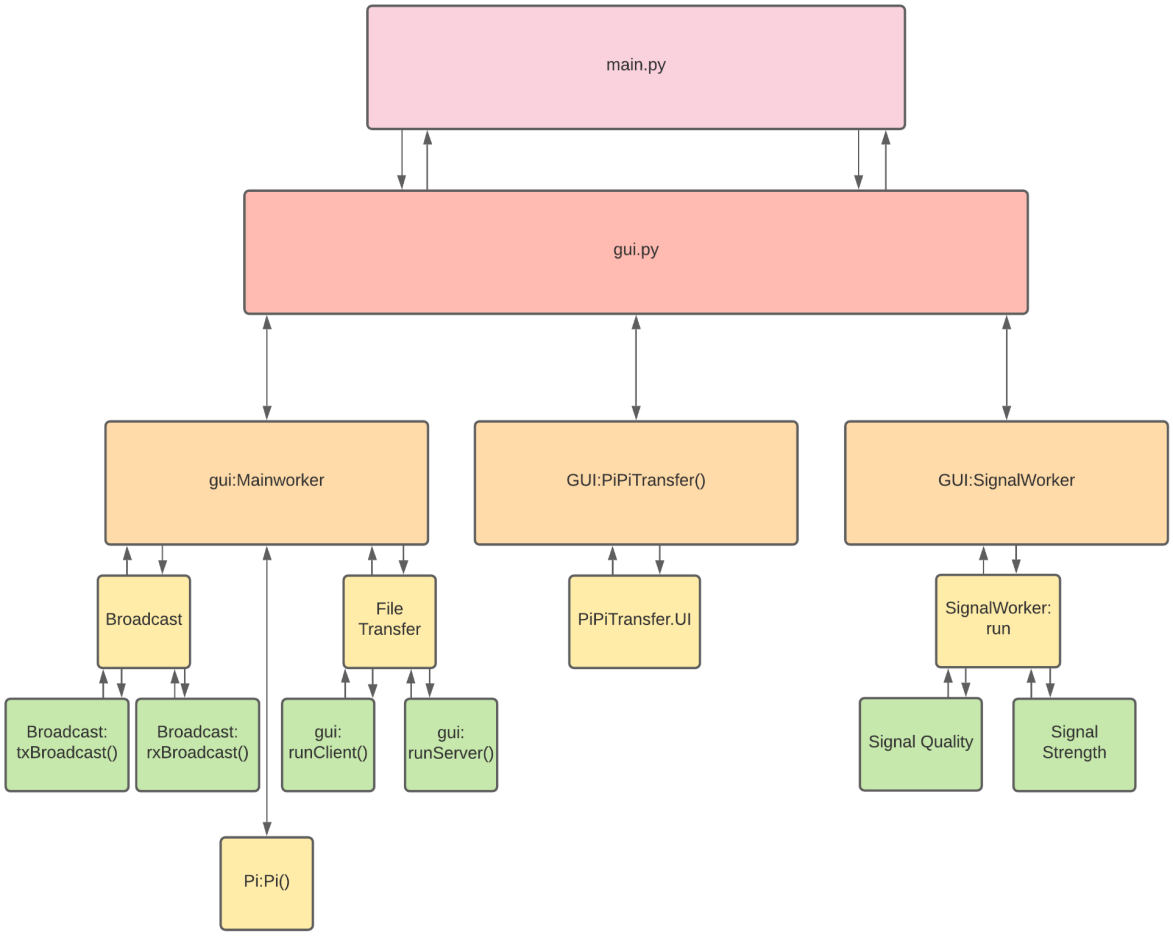
4.3 Software Design

Although batman-adv creates a mesh network, all of the logic for broadcasting version numbers, determining the direction of file transfer, sending/receiving update files, and updating the GUI is contained within the **gui.py** script, **Pi** class, and **Broadcast** class. A second script, **main.py**, lies at the top of the program hierarchy, calling all other files, scripts, and classes. The program is structured such that the user only needs to run **main.py** (in addition to all firmware) to execute the entire project. This hierarchical structure can be seen in *Figure 4.3.1*, where the program has the following order:

> **main.py** calls **gui.py**, which contains the three thread classes **MainWorker**, **SignalWorker**, and **PiPiTransfer**. The former of which calls the **Pi** class, **Broadcast** class, and **update.txt** file while the latter of which calls the **pipitransfer.ui** file.

All of these files can be found in the Appendix of this document or on GitHub using the following link: https://github.com/capstone-paccar/v2v_interface.git.

Figure 4.3.1: Program Hierarchy

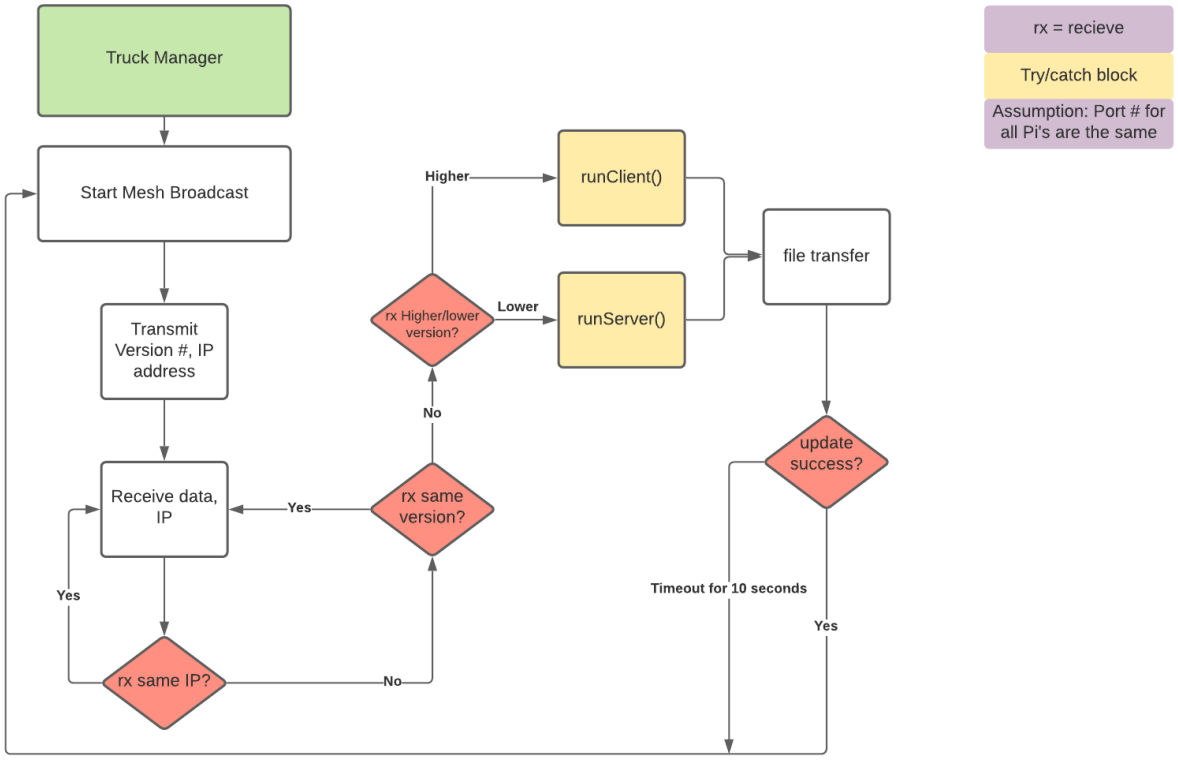


To create such a program, our team first began researching how to send and receive data across devices and came across a software structure called network sockets. Upon finding that sockets used IP addresses and different network protocols to communicate with other devices, we started looking for ways to first broadcast a version number and IP address, and second, send file data over the network using sockets. As our group learned more about networking and sockets, we began to focus our attention on UDP broadcasting sockets and TCP networking sockets to send version number information and file data respectively. Through this research, we discovered that TCP sockets can send data packets more reliably yet slower than UDP sockets. Therefore, although UDP was the only socket option for broadcasting, TCP was our protocol of choice in transferring file data. After creating many iterations of test files for each process, we finally had an overall design which we combined into one temporary truck manager program.

This simple truck manager program worked by first broadcasting a version number and IP address via UDP broadcasting sockets and subsequently searching for other broadcasts on the network. If it found another broadcast, the program would move into a file transfer mode and

attempt to connect to the broadcasted IP address as either a client or server depending on the broadcasted version number. If the version number read from a text file on the Raspberry Pi was lower than received from a broadcast, the Pi would become a server; otherwise, the Pi would become a client. From here the program would enter a loop where it would try to connect to the other Pi and either send or receive the version number from the other Pi. Upon successful file transfer, the Pi receiving a version number would update its version text file to that number. Regardless of if the connection was successful or unsuccessful, the program would repeat itself in an infinite while loop until the program was forcefully terminated. The flow of operation of this program is illustrated in *Figure 4.3.2* and visually shows how the program functions.

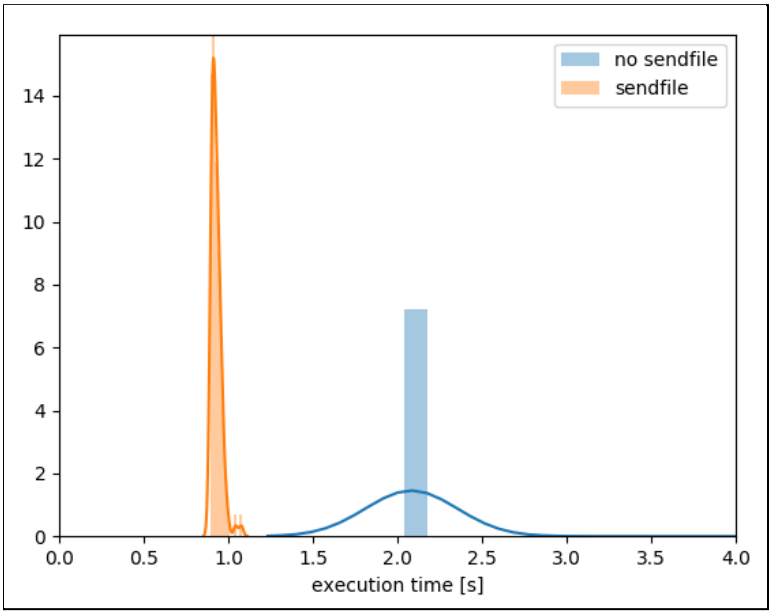
Figure 4.3.2: Flow of Truck Manager Program Operation



In this first iteration of the software design, we had successfully created the overall structure of the file-transfer program. However, the program was still incomplete and posed a few issues that needed to be addressed. Firstly, this simple truck manager could only send 1 kB of data at a time, which was much too small for sending presumably large-sized update files. In the most recent iteration at the time, the program was only reading and sending a single integer value and could not handle sending files of any type. Secondly, this program was unreliable and would fail during nearly half of all tests. Upon researching more, the solution to this problem was to introduce the **socket.sendfile()** method which not only sent files quicker and more efficiently, but had no limit to the file size or file type that it could send. The following graph (*Figure 4.3.3*) from a research article shows that the **socket.sendfile()** method can send large files nearly

twice as quickly and with much greater stability than the previous file transfer method we were using.

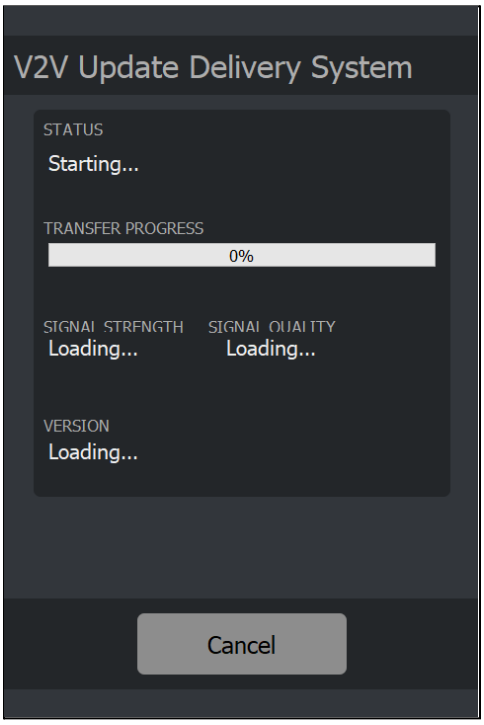
Figure 4.3.2: Sending 4 GB Files with Different File Transfer Methods



After implementing this faster method for sending files into the program and adding a larger update text file to test, our program was much more efficient and reliable across devices. Organizing the program further with Broadcast and Pi classes completed the bulk of our program; the only task left was to implement the GUI for the touchscreen.

Since one of the group members had experience with a graphical user-interface (GUI) python library called PyQt5, our team decided on this library and its attached tools to display our program statistics and status. After embarking on research and finding helpful tutorials online, we were able to design a GUI with the intuitive drag-and-drop application called Qt Designer. The GUI design that we settled on took inspiration from Google's Material Dark theme and used high-contrast text and numbers to conserve battery power and make it easy for the user to read when performing outdoor testing. As seen in *Figure 4.3.3*, this GUI design incorporates the title of the project, dynamic text and a progress bar representing file transfer status, information about signal strength, quality, and version number, and an interactable "Cancel" button.

Figure 4.3.3: UI File Exported from Qt Designer



However, when we attempted to implement the GUI's buttons and dynamic text into our code, we ran into a major problem: every time the program was run, the GUI would freeze and the program would instantly crash. After researching more about this problem, we found that we were attempting to run two threads, or two separate computer processes, at once while the program could only handle sequential execution of code. Since the program running the GUI runs on a separate thread called the event loop, running longer background tasks on this same thread will cause the event loop to slow to an unresponsive state and crash.

Therefore, the solution to this problem is to introduce multithreading into our program, which uses multiple cores within a processor to simultaneously run all programs in different threads without any one program interfering with another. Since this problem is common in PyQt5, the library includes a **QThread** class to create special threads that can communicate with the GUI's event loop thread via PyQt5's system of signals. In this way, every time the truck manager program would like to update the GUI, it can do so by emitting one of several signals to be picked up by the main event loop thread and subsequently displayed on the GUI. Additionally, a third thread was created in addition to the truck manager and main event loop threads to consistently fetch the signal strength and signal quality of the mesh network via bash commands.

Once the GUI was fully implemented into the program, we renamed each thread and placed our entire truck manager program into a thread called the **MainWorker**. Thus, the GUI script

became the top-level script in our program, calling the truck manager program and all other subsequent methods and classes. After testing our program for many hours at a time, we were confident that it was finished, and began editing our program for style and readability. During the editing process, we combed over each file in the program, adding comments to clear any confusion in our code and Python DocString to label each file, method, and class with a description and important parameters.

The following subsections explain how each file functions, is organized and contributes to the overall program.

4.3.1 main.py

This python script is meant to be a small user-friendly file that calls gui.py, launching and running the program. The python code in this file is solely meant to create the GUI window for the program to use.

4.3.2 gui.py

This python script enables a Raspberry Pi 4 to send files via Ad-Hoc to another Pi running the same program and launches a graphical user interface to display file transfer characteristics. The script is organized into one event loop thread, two worker threads, and a classless method. The event loop thread, **PiPiTransfer**, is the first to start when the program is first run, opening and setting up the GUI window as well as starting the **MainWorker** and **SignalWorker** threads by calling their **run()** methods.

SignalWorker's run() method uses the subprocess library to run the "ifconfig" bash command and emit information on the signal strength and quality to be picked up by the **PiPiTransfer** GUI thread.

MainWorker's run() method runs the truck manager program, broadcasting the Pi's version number to other Pis on the network and running either the **runServer()** or **runClient()** method depending on the version number received from other Pis. Additionally, this method will emit status, progress, and version number signals to the PiPiTransfer GUI thread.

MainWorker's runServer() method runs a TCP server on the local Pi, thereby receiving an update file, taking in a Pi object, and returning a boolean representing the success of the file transfer. TCP sockets are used to start the server, connect to a remote client, and download a file in 4 KB chunks. Once the file data has fully downloaded, it will be written to the local Pi's **update.txt** file and the program will enter the **run()** loop once again.

MainWorker's **runClient()** method runs a TCP client on the local Pi, thereby sending an update file, taking in a Pi object, and returning a boolean representing the success of the file transfer. TCP sockets are used to start the client, connect to a remote server, and send the file in 4 KB chunks using socket.sendfile(). Once the file data has been fully sent, the program will enter the run() loop once again.

If the Cancel button is pressed, the **MainWorker's stop()** method will refresh the program. The classless **getSystemIP()** method works in tandem with the **MainWorker's run()** method and uses the subprocess library to run the "hostname -I" bash command to get the IP address of the local Pi in a string format.

4.3.3 broadcast.py

This python class designs a transmission and receive socket to broadcast and receive IP address, Port, and version data. The **Broadcast** class contains these attributes as broadcastAddress, port, and version respectively.

Broadcast's rxBroadcast() method handles receiving broadcasts and returns a tuple object with socket and version information.

Broadcast's txBroadcast() method handles transmitting broadcasts to the network.

4.3.4 pi.py

This python class designs a **Pi** object to store and set a version number and IP address. This class contains these attributes as version and address respectively.

Pi's getVersion() serves as the getter for the Pi's version number.

Pi's getIP() serves as the getter for the Pi's IP address.

Pi's setVersion() serves as the setter for the Pi's version number.

Pi's setIP() serves as the setter for the Pi's IP address.

4.3.5 update.txt

This text file acts as the update file to send across Pis, containing the local version number on the first line and text to simulate an actual update file on the third line.

4.3.6 pipitransfer.ui

This UI file serves to create the GUI design and is loaded using the initialization of the GUI script's **PiPiTransfer()** method and the "loadUi" command.

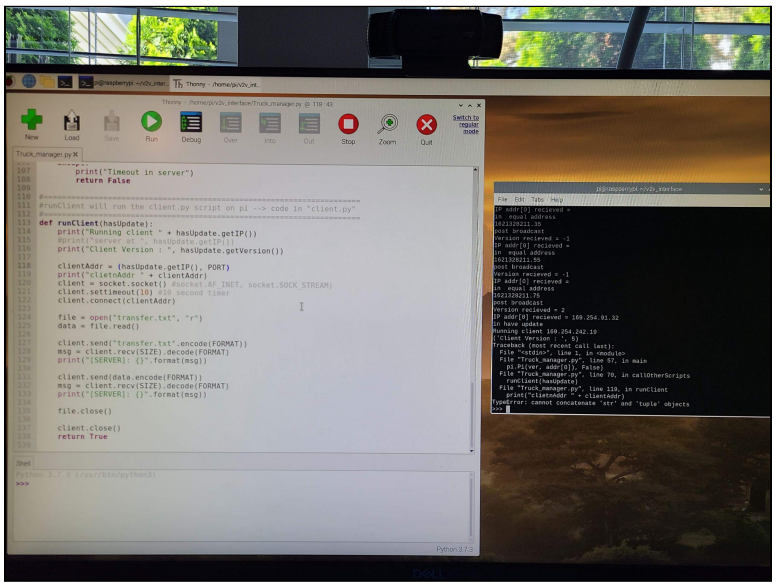
5. Testing

To ensure the performance of the V2V Update Delivery System, we consistently wrote and tested our program after each iteration of changes. This process guaranteed the success of our project from the first lines of code written to the final comments created. Once the truck manager program was finished, we began putting our devices through more rigorous field testing to see how well our devices and program would perform if integrated into a vehicle system. The following sections each cover a specific area of testing meant to evaluate the success of our project.

5.1 Software Integration Testing

Since our IDE of choice, Microsoft VSCode, was available on Raspberry Pi OS, it became fairly easy to test our program with another nearby Pi after writing or making changes to each Pi. This allowed us to debug our program in real-time throughout the quarter, slowly building our program up to the final version as seen in this report. Therefore, instead of saving software integration testing for the final version of our program, we opted to perform such testing throughout the development of our project.

Figure 5.1.1: Performing Software Integration Testing Mid-project

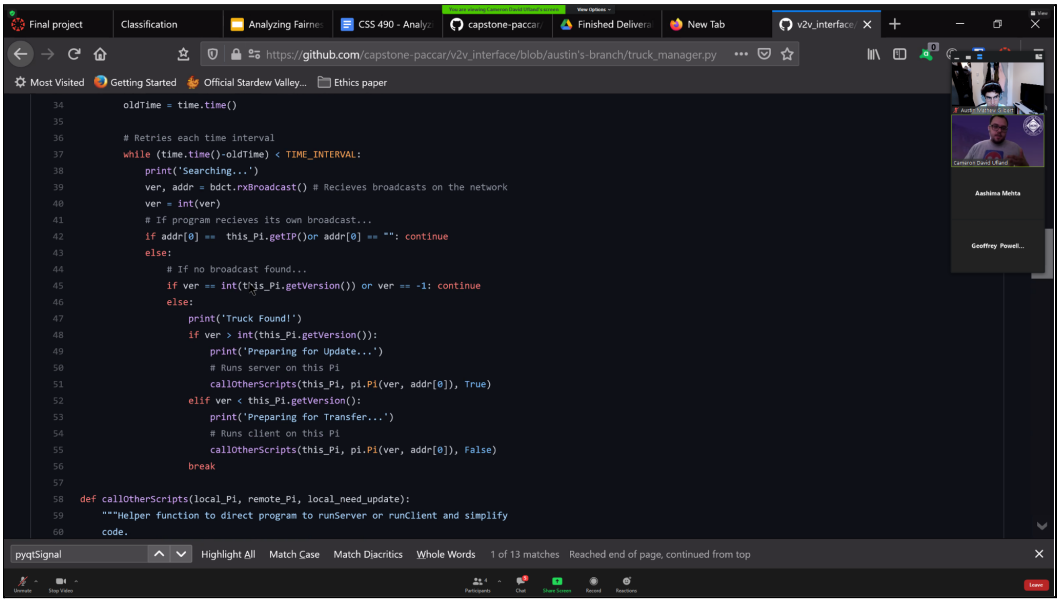


We performed these tests by splitting the program into smaller sections of the software, testing these sections individually, then adding each section to the main program after it was proven to work. Repeating this process, we were able to successfully add all the various required functionalities of the project to our program.

On the 31st of May, our team tested our software design further by hosting a formal design review meeting with a fellow computer engineering student, Geoffrey Powell-Isom. During the meeting, we reviewed the program file-by-file to explain our thought process behind each design

choice and to make sure that the code made sense within the contexts of our project. A screenshot from the meeting is shown in *Figure 5.1.2*.

Figure 5.1.2: Formal Design Review



At the end of the meeting, we received intuitive recommendations and commentary from Geoffrey on how to improve our software design, each of which made it into the final version of our software.

5.2 Distance Testing

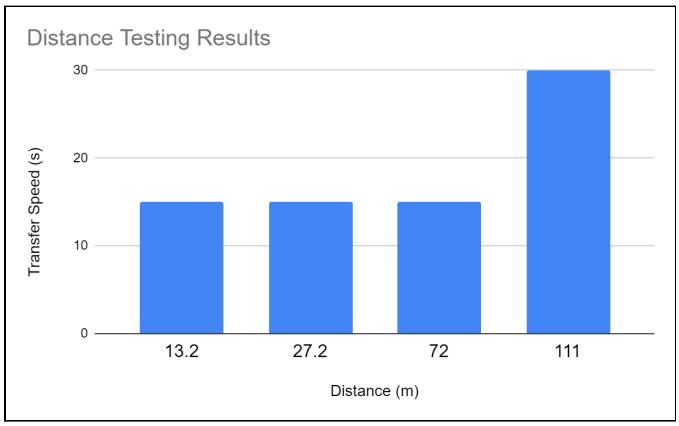
Our team conducted multiple tests across different distances to determine the maximum distance that the project could successfully transfer files across. The distance of transfer was an important criterion of our project as it determined how well this technology could be translated into a vehicle system; since most file transfers from V2V would likely occur in large truck-stop lots, distance testing on a field in Shoreline's Cromwell Park would give a gauge as to how far our project could still function.

Starting preliminary tests with small distances of five to ten meters, we first confirmed our program was working correctly between Pis. We then tested at four different distances by having two team members spread out with Pis in hand across the field at specific park landmarks so we could measure these distances later with Google Earth Pro's "measure" feature. With the distance increasing with each test, we could accurately test the maximum distance at which the Pis could communicate. Each distance can be measured out as seen by *Figure 5.2.1*, where the yellow line is 13.2 meters long, orange is 27.2 meters long, red is 72 meters long, and blue is 111 meters long.

Figure 5.2.1: Measured out Distances

Surprisingly, each distance test resulted in a success, with the final test reaching a distance of 111 meters. However, we performed only four tests in total since it took the Pis much longer to connect during the final test and since we ran out of time and distance to test more. These results can be summarized by the following *Figure 5.2.2*.

Figure 5.2.2: Distance Testing Results



Therefore, based on these results, the V2V Update Delivery System is successful in delivering updates up to around 100 meters apart and possibly more. Since 10 parked trucks take up about 50 meters of space, this system could effectively update 20 trucks in each direction.

5.3 Environment/Field Testing

In addition to testing the speed of transfer on large fields, our team also tested transferring in Cromwell park's parking lot. The purpose of this test was to mimic real-world environments with obstacles such as trees and cars that our technology would need to overcome if translated into a vehicle system. Since the parking lot was small, this was purely a test of the environment rather than distance. *Figure 5.3.1* shows the following locations in which we tested connectivity between two Pis.

Figure 5.3.1: Measured out Environmental Distances

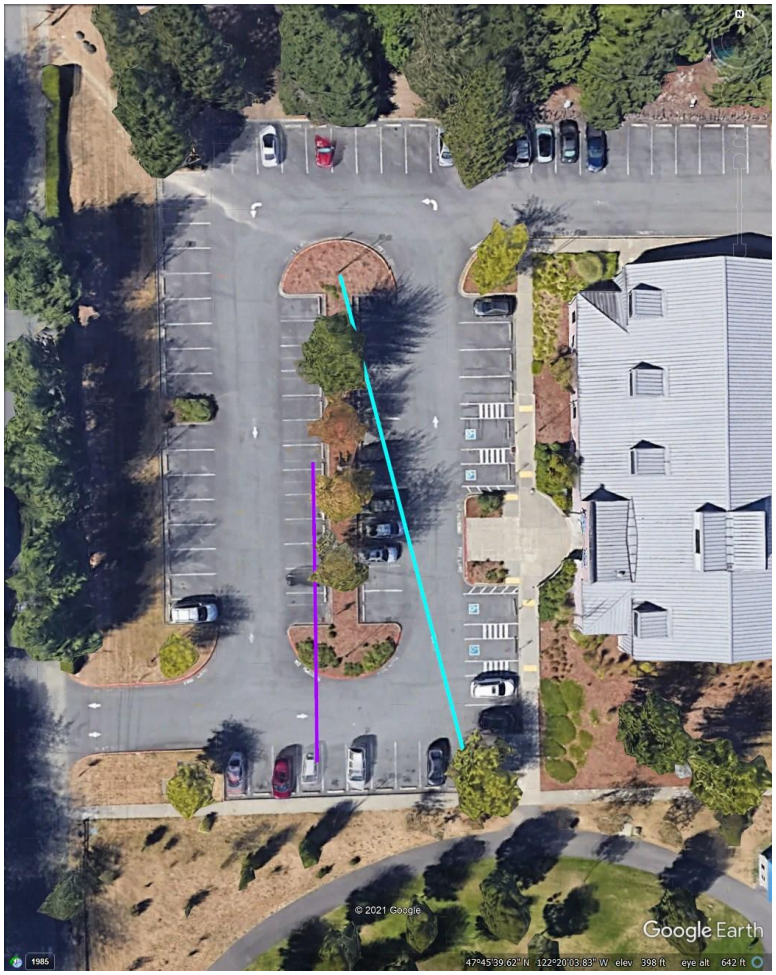


Figure 5.3.1: Testing in the Park

These two distances were measured to be about 57 and 33 meters respectively, and although both tests were successful, the Pi struggled to connect and took longer to successfully transfer an update file. In the cyan test, we proved that our project could transfer files with obstructions such as trees and the tops of cars. In the purple test, we attempted to transfer while inside two separate vehicles, but could not connect until walking outside each car. Therefore, if this technology was to be implemented in a vehicle system, our team would recommend attaching antennae to the tops of vehicles such that a line of sight between devices could be achieved.

5.4 Speed Testing

To test the speed of connecting between Pis and successfully transferring files, we started a stopwatch when main.py was first launched and stopped the stopwatch when a file had been successfully transferred between Pis. The file used to transfer updates in our program, “update.txt” was sent between Pis while logging such data over five separate trials.

These tests yielded consistent results of 10.1 seconds for all five trials. From this data, we can conclude that transfer times are consistent and reliable. Since update files will likely contain a list of variables in a compact format, sending 750 bytes of data in just 10 seconds is a relatively decent metric when compared to the overall speed of the program. Thus, if this technology was to be translated into a vehicle system, customers could expect slow to medium-speed transfers.

5.5 Download File Size Testing

To test the speed and accuracy of transferring files of variable sizes, we again timed the program. The following graph (*Figure 5.5*) represents such data logged over 5 separate trials.

Figure 5.5: Download File Size Testing Results

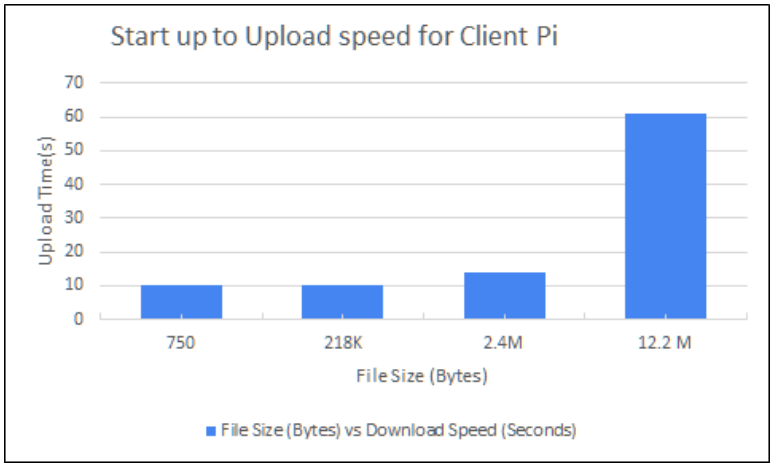


Table 5.5: Download File Size Testing Results

Startup to Upload speed for Varying File Sizes *(trial times in seconds)*						
File Size	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Average**
750 B	10.1	10.1	10.1	10.1	10.1	10.1
218 KB	10.2	10.1	10.2	10.2	10.1	10.16
2.4 MB	12	22	11	12	12	13.8
12.2 MB	57.1	56.9	60.7	64.8	64.4	60.78

From this data, we can conclude that transfer times are consistent and reliable across different file sizes. However, there is a somewhat linear trend as the trial times increase with increasing file size. Although a minute is a decently long time to transfer files over WiFi, our WiFi speeds on the Raspberry Pi's antenna are limited to 150 Mbps while using batman-adv. Although transferring a 2.4 MB file during Trial 2 yielded an outlier result, an error such as this is few and far between and most likely was due to the Raspberry Pi's hardware.

If this update delivery system were to be implemented in a vehicle system, we would recommend using a more powerful and faster WiFi antenna and possibly altering the program's current 4 KB chunk size to send more chunks of data in the same amount of time. We would also advocate implementing file compression or optimization on each file before sending the files across Pis.

5.6 Usability Testing

To test the ease of use of our software and hardware design, we chose the internationally standardized Post-Study System Usability Questionnaire (PSSUQ), explaining our project to three randomly selected people and having them answer sixteen questions about the project's

usefulness, information quality, and interface quality. Originally created and distributed by IBM, the PSSUQ is used amongst technology companies to measure the perceived satisfaction of a system. In the context of our project, we will focus less on the information quality section of this test, but rather on the interface quality and system usefulness of our project as it is a small prototype of what the technology is meant to become.

The PSSUQ is scored on a scale where 1 is the best and 7 is the worst. The following scores (*Table 5.6*) are the results of three people answering the survey about our product.

Table 5.6: PSSUQ Results

Overall Score	1.75
System Usefulness (SYSUSE)	1.93
Information Quality (INFOQUAL)	1.60
Interface Quality (INTERQUAL)	1.56

From this data, we can see that the system usefulness score is lower than the rest of the scores which makes some amount of sense since the program performs most tasks automatically and in the background, away from the user. Additionally, none of the people completing the questionnaire were told about how exactly to interact with the program since there were so few possible interactions. Our other categories deal with the information displayed on the GUI as well as the GUI's overall design and prove that those areas will not need as much future work as the system usefulness category due to their higher scores.

6. Conclusions

Our team was able to successfully create a proof of concept for the automatic delivery of updates between two devices in proximity, with the direction of update transfer determined via version number. By integrating our program with multiple Raspberry Pi-based devices and testing under numerous criteria, we were able to deliver a prototype with recommendations on possible methods for vehicle system integration. Overall, our team has fulfilled the following requirements set in the original internal specification:

- **BR_01 -** The device should be able to automatically detect another device within 500 meters.
- **BR_02 -** The device should handle the delivery of updates from device to device in both directions without human interaction.
- **BR_03 -** The device should interrogate another device to determine the direction of update delivery.
- **BR_04 -** The device should handle the delivery of updates via an Ad-Hoc mesh network with no internet dependence.
- **BR_06 -** The device's touchscreen ~~and mobile application~~ should have an easy-to-use user interface.
- **BR_08 -** The device should handle pausing update delivery due to connection range and waiting for reconnection for a specified period.
- **BR_10 -** The device should support communication with multiple other devices.

Out of the original ten requirements, we have completed six and partially completed one. Out of these completed requirements, all of the high-priority requirements and three of the medium- to low-priority requirements resulted in a successful implementation into our software and hardware.

7. Future Work

Since not all of the requirements were successfully completed and since our project was largely a prototype, future work may need to be completed before translating the V2V Update Delivery System into an in-vehicle system. The following sections cover such future work, including ideas relating to the hardware design and software development of each prototype device.

7.1 Phone-Pi Update Delivery

Most uncompleted lower priority requirements dealt with a secondary mode of our program that would enable technicians to more easily update trucks in remote locations without nearby trucks using a connected mobile application. Such an application would act as a one-directional node, only transferring stored update files to qualified devices. Technicians would use this application

to download updates from an internet database when the mobile device is connected to the internet, store such update files, then send them on command to a device via an Ad-Hoc network with no internet connection.

Although our team was on track to complete such a deliverable, problems with the main Pi-Pi program set our team back, and we were unable to finish this portion of the project. This could be built upon further by integrating a second “widget” layer into the program’s GUI as well as building an Android application to interface with the Raspberry Pi via Bluetooth sockets. Additionally, a program that facilitates Bluetooth connectivity on the Raspberry Pi called “bluez” must be installed and integrated into the software design.

7.2 Long-Range Antenna

Although not an official requirement of our project, our team purchased two KuWiFi 1 W 9 dBi long-range antennae. Each antenna contained an RT3070 chipset that was proven via our own tests to function with Raspberry Pi OS and was unproven to work with the IEEE 802.11 Ad-Hoc mesh network standard. Implementing such antennae could benefit this project significantly as it would increase the range of any Ad-Hoc mesh network from the current 111-meter maximum to over 1500 meters in all directions along the azimuth plane. This would allow this project to theoretically send and receive an update file even in the largest truck stops in the world and would allow the device to continue to update for a good distance upon exiting a truck stop, helping prevent download cancellations.

7.3 Security Implementation

Although implementing security features to protect the transfer of update files was out of scope for this project, any future team looking to implement this technology into a vehicle should consider securing, encoding, and optimizing the file transfer process. Batman-adv contains no inherent security measures, although the creators of the software recommend integrating encryption and authentication on the layers above the WiFi layer using WPA_NONE, IBSS RSN, or IPsec. Integrating encryption and security into this project will also support the project’s motivation of securing the software on trucks by ensuring such software is consistently up to date and free of vulnerabilities.

8. Acknowledgements

David Sasaki

Industrial Advisor

We would like to thank David Sasaki for taking his time to provide us with technical and insightful assistance on our project.

Elaine Reeves

Academic Advisor

We would like to thank Elaine Reeves for giving us numerous valuable insights, feedback, and many hours of her time to help make the project a success.

Geoffrey Powell-Isom

UWB BSCE Student

We would like to thank Geoffrey Powell-Isom for his participation in our software design review.

Dr. Arnold Berger

UWB Faculty

We would like to thank Dr. Arnold Berger for organizing the Capstone project as well as the canvas page with project instructions, advice, and examples.

Appendix

Appendix A. Definitions, Acronyms, Abbreviations

- **Ad Hoc -** Wireless device-to-device network protocol
- **B.A.T.M.A.N. -** Better approach to mobile Ad-Hoc networking
- **batman-adv -** B.A.T.M.A.N. Advanced
- **FCC -** Federal Communications Commission
- **IDE -** Integrated Development Environment
- **IP -** Internet Protocol
- **Mesh Network -** A non-hierarchical network topology
- **OS -** Operating System
- **Project A -** Vehicle-to-Vehicle communication hardware and software
- **Project B -** Phone-to-Vehicle communication hardware and software
- **PSSUQ -** Post-Study System Usability Questionnaire
- **RPi -** Raspberry Pi
- **TCP/IP -** Transmission Control Protocol / Internet Protocol
- **UDP -** User Datagram Protocol
- **UI -** User Interface
- **V2V -** Vehicle-to-Vehicle
- **VANET -** Vehicular Ad-Hoc Network
- **Wi-Fi -** A wireless network protocol managed by the Wi-Fi Alliance
- **WLAN -** Wireless Local Area Network

Appendix B. Datasheets

Raspberry Pi 4 Model B

https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf

UCTRONICS 3.5” LCD HDMI Display

https://www.uctronics.com/download/Amazon/B0106.pdf

Appendix C. Bill of Materials

Supplier	Item	Qty.	Cost Per Item	Total Cost	Link	Notes
CanaKit	Raspberry Pi 4 Model B - 2GB	5	$35.00	$175.00	Link	
Amazon	3.5" Touch Screen for RPi4	2	$27.99	$55.98	Link	- Includes Stylus & Heatsinks
Amazon	EAFU Portable 10,000 mAh Battery	2	$13.99	$27.98	Link	- Outputs 3A necessary for RPi4
Amazon	2 x Short USB C to USB A Cable	1	$7.99	$7.99	Link	- Power Cable
Amazon	5 x 16GB Micro SD-HC	1	$17.49	$17.49	Link	
Amazon	Micro HDMI to HDMI Adapter	3	$5.99	$17.97	Link	
TOTAL				**$302.41**		

Appendix D. References

"Antenna Basics." *SimpleWiFi*, www.simplewifi.com/pages/antenna-basics.

"Antenna Gain." *Wikipedia*, Wikimedia Foundation, 26 Dec. 2020, en.wikipedia.org/wiki/Antenna_gain.

"Antenna Patterns and Their Meaning." *Cisco*, 7 Aug. 2007, www.cisco.com/c/en/us/products/collateral/wireless/aironet-antennas-accessories/prod_white_paper0900aecd806a1a3e.html.

"B.A.T.M.A.N." *Wikipedia*, Wikimedia Foundation, 25 Apr. 2021, en.wikipedia.org/wiki/B.A.T.M.A.N.

"Dark Theme." *Material Design*, Google, 25 June 2014, material.io/design/color/dark-theme.html#usage.

Doll, Tyler. "How to Setup a Raspberry Pi Ad-Hoc Network Using BATMAN-ADV on Raspbian Stretch." *Tyler Doll - Medium*, Medium, 18 Apr. 2019, medium.com/@tdoll/how-to-setup-a-raspberry-pi-ad-hoc-network-using-batman-adv-on-raspbian-stretch-lite-dce6eb896687.

Dul, Michal. "File Transfer over Sockets without User-Space Memory in Python 3.5." *Notes on Data Processing*, Jeykll, Minimal Mistakes, 4 Feb. 2018, michaldul.com/python/sendfile/.

Innes, Brian. "Part 1 - Mesh Networks." *GitHub*, GitHub, Inc., 21 Jan. 2020, github.com/binnes/WiFiMeshRaspberryPi/blob/master/part1/PIMESH.md.

JamesH65. "BCM2711 - Raspberry Pi Documentation." *Raspberry Pi*, Raspberry Pi Foundation, 23 June 2019, www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/README.md.

JamesH65. "FAQs - Raspberry Pi Documentation." *Raspberry Pi*, Raspberry Pi Foundation, 23 June 2019, www.raspberrypi.org/documentation/faqs/#:~:text=It%20uses%20a%201.2GHz,wireless%20LAN%2C%20and%20Bluetooth%204.2.

Jin, Chuan. "Transfer File over TCP/UDP." *DevNotes*, Hexo, NexT.Muse, 3 Aug. 2016, chuanjin.me/2016/08/03/transfer-file/.

"KuWiFi High Gain 13 DBi Dipole Antenna: Product Description." *Amazon*, www.amazon.com/KuWFi-Distance-Waterproof-Improving-Reception/dp/B00L8PZ8DA.

Linder, Marek, and Simon Wunderlich. "B.A.T.M.A.N. Advanced." *Open-Mesh*, Redmine, www.open-mesh.org/projects/batman-adv/wiki/Wiki.

Lindner, Marek, and Simon Wunderlich. “Debian Batman-Adv AutoStartup.” *Open-Mesh*, Redmine, www.open-mesh.org/projects/batman-adv/wiki/Debian_batman-adv_AutoStartup.

“Network Socket.” *Wikipedia*, Wikimedia Foundation, 27 May 2021, en.wikipedia.org/wiki/Network_socket.

newsordice. “Re: Rotate to 90 or 270 Degrees.” *Raspberry Pi Forums*, Raspberry Pi Foundation, Broadcom, 17 Dec. 2016, 7:24 pm, www.raspberrypi.org/forums/viewtopic.php?t=166959#p1083497.

ninedraft. “Python Udp Broadcast Client Server Example.” *GitHub Gist*, GitHub, Inc., 25 Nov. 2020, gist.github.com/ninedraft/7c47282f8b53ac015c1e326fffb664b5.

“PyQt5 FULL Modern Gui Tutorial #1 - Welcome Screen [for Beginners].” *YouTube*, Uploaded by Code First, 3 May 2021, youtu.be/RxGlB9U64fg.

“PyQt5 FULL Modern Gui Tutorial #2 - Login Form: GUI & Database [for Beginners].” *YouTube*, Uploaded by Code First, 5 May 2021, youtu.be/pleHwE1swbY.

“PyQt5 FULL Modern Gui Tutorial #3 - Create Account Form: GUI & Database [for Beginners].” *YouTube*, Uploaded by Code First, 9 May 2021, youtu.be/kpIbcFeNKCw.

Quinten, Jörg. “Re: Official 7' Touchscreen, Rotate Touch.” *Raspberry Pi Forums*, Raspberry Pi Foundation, Broadcom, 30 June 2015, 1:35 pm, www.raspberrypi.org/forums/viewtopic.php?f=108&t=219333&p=1358066#p1348198.

Ramos, Leodanis Pozo. “Use PyQt's QThread to Prevent Freezing GUIs.” *Real Python*, Real Python, 21 Dec. 2020, realpython.com/python-pyqt-qthread/.

Reddy, Vivek. “TCP 3-Way Handshake Process.” *GeeksforGeeks*, 6 Sept. 2019, www.geeksforgeeks.org/tcp-3-way-handshake-process/.

“Requirements Specification Template.” 17 Apr. 2013.

Rockikz, Abdou. “How to Transfer Files in the Network Using Sockets in Python.” *Python Code*, Feb. 2021, www.thepythoncode.com/article/send-receive-files-using-sockets-python.

“RPi USB Wi-Fi Adapters.” *ELinux.org*, Wikimedia Foundation, 10 Mar. 2021, 15:18, elinux.org/RPi_USB_Wi-Fi_Adapters.

Schoch, Elmar, et al. “Communication Patterns in VANETs.” *IEEE Communications Magazine*, vol. 46, no. 11, 25 Nov. 2008, pp. 119–125., doi:10.1109/mcom.2008.4689254.

"Send UDP Broadcast Packets." *Svn Python*, Apache Subversion, svn.python.org/projects/python/trunk/Demo/sockets/broadcast.py.

Tomar, Nikhil. "File-Transfer-Using-TCP-Socket-in-Python3." *GitHub*, GitHub, Inc., 5 Jan. 2020, github.com/nikhilroxtomar/File-Transfer-using-TCP-Socket-in-Python3.

usabiliTEST. "Usability Testing & Information Architecture." *UsabiliTEST*, 2011, www.usabilitest.com/.

"Usability Evaluation Basics." *Usability.gov*, Department of Health and Human Services, 8 Oct. 2013, www.usability.gov/what-and-why/usability-evaluation.html.

Walicki, John, and Brian Innes. "Configuring Mesh Networking for the IoT Edge." *IBM Developer*, IBM, 6 Mar. 2019, developer.ibm.com/technologies/iot/tutorials/create-iot-mesh-network#1-set-up-the-mesh-network .

Whitehorn, Jack. "'Sudo Batctl If' Command on Gateway Node Results in 'wlan0: <error reading="" status="">"." GitHub, GitHub, Inc., 28 Feb. 2021, github.com/binnes/WiFiMeshRaspberryPi/issues/6#issuecomment-787487859. </error>

"WiFi Transmit Power Calculations Made Simples." *DigitalAir Wireless*, www.digitalairwireless.com/articles/blog/wifi-transmit-power-calculations-made-simples.

Zeadally, Sherali, et al. "Vehicular Ad Hoc Networks (VANETS): Status, Results, and Challenges." *Telecommunication Systems*, vol. 50, no. 4, 9 Dec. 2010, pp. 217–241., doi:10.1007/s11235-010-9400-5.

Appendix E. GitHub Documentation

E.1 Program Description (README.md)

Vehicle-to-Vehicle Update Delivery System

This project involves the design and creation of a close-range reception-less Vehicle-to-Vehicle update delivery platform using two Raspberry Pi development boards.

UNIVERSITY of WASHINGTON | BOTHELL

In affiliation with

PACCAR Inc

Installation

Clone the repository from Github to Raspberry Pi to install v2v_interface.

```
git clone https://github.com/capstone-paccar/v2v_interface.git
```

See SETUP.md for setting up Raspberry Pi for Ad-Hoc communication.

Usage

If you choose not to integrate this program with **/.bashrc** for automatic run-on-startup, run the following command in the Pi's terminal.

```
cd v2v_interface
python3 main.py
```

See SETUP.md for setting up Raspberry Pi for automatic run-on-startup.

Authors and Acknowledgement

Created by Austin Gilbert, Aashima Mehta, and Cameron Ufland for the *University of Washington, Bothell* in affiliation with *PACCAR Inc*.

Designed for the Electrical Engineering Department's Senior Design Capstone Project under supervision of staff lead Elaine Reeves and industry lead, David Sasaki.

Background

Our team has developed a proof of concept for this technology to be used in PACCAR's fleet of trucks for the delivery of updates in locations with slow internet speeds. Commercial vehicles operate in many different environments and an internet connection may not be available most of the time. Due to this lack of internet connectivity, many commercial vehicles have software that is out of date. This Vehicle-to-Vehicle Update Delivery project will allow trucks to communicate with one another in receptionless areas with many trucks, such as truck stops.

Furthermore, this project implements a mobile application for maintenance technicians to more easily update trucks in remote locations without nearby trucks. Through these two means of update delivery, our industrial sponsor, PACCAR, can ensure that important safety and general updates can reach more vehicles.

User Scenarios & Use Cases

This project allows devices to communicate with one another in receptionless areas with many other nearby devices. The main use case is trucks with later versions of updates delivering updates to other trucks with earlier versions at truck stops, lots, and on the road.

Both trucks, once updated, should be able to deliver updates to even more trucks, spreading the latest update files much like the spread of a virus. The following figures illustrate Project A in a diagram and step-by-step process.

Figure 1: Diagram Illustrating Project

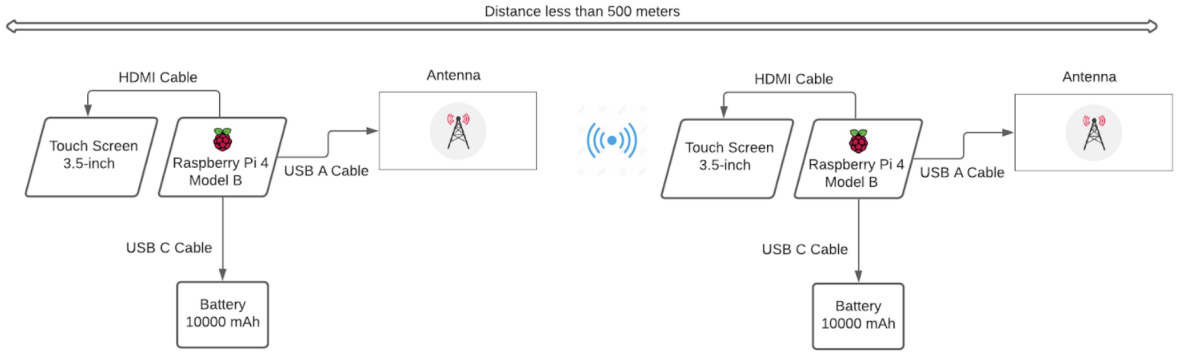
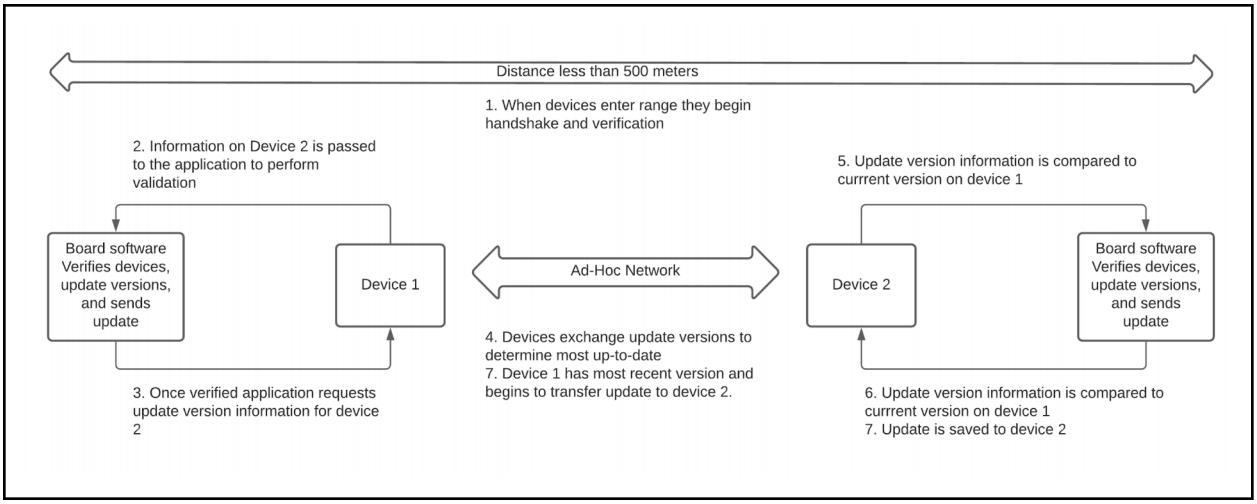


Figure 2: Step-by-Step Process



E.2 Program Installation & Device Setup (SETUP.md)

Raspberry Pi Setup

This document describes the process for setting up batman-adv, which configures an Ad-Hoc mesh network on a Raspberry Pi.

Background

Standing for "better approach to mobile ad-hoc networking," B.A.T.M.A.N Advanced or **batman-adv** is the Linux-based network routing protocol used to manage Ad-Hoc mesh networks in the Vehicle-to-Vehicle Update Delivery System. To learn more about this protocol, here is the official documentation.

Performing Initial Raspberry Pi Setup

See the official setup documentation to set up and start up your Pi.

Instead of using a wall-to-USB-C adapter, you may want to use a high mAh capacity power bank to make the module portable. Additionally, you may want a 3.5-inch touchscreen to connect to the top of the Pi as a display for program statistics and progress.

Setting up batman-adv

We will configure **batman-adv** such that it takes over the WiFi interface **wlan0** and sets up a tandem WiFi interface **bat0**. This enables the Pi to send network sockets over the wlan0 antenna while using IP addresses set by bat0 and deploys a mesh network.

1. Install **batctl**

   batctl is a control tool for batman-adv. To install, use the command:

```
sudo apt-get install batctl
```

   - If the batctl present in the Raspberry Pi repository is out of date, run this command to install the "latest version" (as of June 1, 2021):

```
sudo apt purge batctl -y &&  \
sudo apt install libnl-genl-3-dev libnl-3-dev -y && \
wget
https://downloads.open-mesh.org/batman/releases/batman-adv-2020.4/batctl-2020.4.tar.gz &&  \
tar -xvf batctl-2020.4.tar.gz && \
cd batctl-2020.4 && \
make -j 4 && \
sudo make install
```

2. Create **start-batman-adv**

   start-batman-adv will serve as an executable that runs the necessary commands to start the mesh network. Use the following commands to create and edit the executable:

```
cd ~ && touch start-batman-adv.sh && chmod +x start-batman-adv.sh
sudo nano start-batman-adv.sh
```

   On the nano editor, add the following to start-batman-adv:

```
#!/bin/bash

# Tell batman-adv which interface to use
```

```
sudo batctl if add wlan0
sudo ifconfig bat0 mtu 1400
sudo ifconfig wlan0 mtu 1500

# Activates the interfaces for batman-adv
sudo ifconfig wlan0 up
sudo ifconfig bat0 up
```

3. Configure Interfaces **wlan0** and **bat0**

   First ensure that the interfaces are set up correctly by opening the interfaces file using the following command:

```
sudo nano /etc/network/interfaces
```

   And ensure the following line is written in the file:

```
# Include files from /etc/network/interfaces.d
source-directory /etc/network/interfaces.d
```

   Next, create/find **wlan0**:

```
sudo nano /etc/network/interfaced.d/wlan0
```

   And ensure the following is written in the file:

```
auto wlan0
iface wlan0 inet6 manual
    mtu 1532
    wireless-channel 1
    wireless-essid my-mesh-network
    wireless-mode ad-hoc
    wireless-ap 02:12:34:56:78:9A
```

   Finally, create **bat0**:

```
sudo nano /etc/network/interface.d/bat0
```

   And ensure the following is written in the file:

```
auto bat0
```

```
iface bat0 inet6 auto
    pre-up /usr/sbin/batctl if add eth0
    pre-up /usr/sbin/batctl if add wlan0
```

4. Enabling **Automatic Run-on-Startup**

   Running the following commands will enable the Pi to start deploying a mesh network on boot:

```
# Starts batman-adv at boot
echo 'batman-adv' | sudo tee --append /etc/modules

# Prevents DHCPCD from automatically configuring wlan0
echo 'denyinterfaces wlan0' | sudo tee --append /etc/dhcpcd.conf

# Enables start-batman-adv to execute at boot
echo "$(pwd)/start-batman-adv.sh" >> ~/.bashrc
```

5. **Reboot**

   Reboot with the following command to put all changes into effect:

```
sudo reboot
```

6. **Test** for a Functioning Mesh Network

   Test by running the following two commands:

```
sudo batctl if
sudo batctl n
```

   If the first command gives the following response, then the network is functioning:

```
wlan0: active
```

   If the second command gives at least one MAC address, those devices are connected to the mesh network.

Some Things to Note

- When running on multiple Pis, ensure that all files are the same and that the hostname is different for each Pi.
- To test further, try pinging another device by using the **ping** command in the terminal followed by the IP address of another Pi.
- Ethernet must be unplugged for the v2v_interface program to run correctly.
- Following this guide will disable your Pi's integrated WiFi.

Setting Program to Run on Startup

Assuming the program was downloaded, to set the v2v_interface program to run on boot, run the following command:

```
sudo nano ~/.bashrc
```

And scroll to the bottom of the file. At the very bottom of the file, add two lines which contain the following:

```
sleep 15
cd v2v_interface
python3 main.py
```

Then run the following command:

```
sudo nano /etc/xdg/lxsession/LXDE-pi/autostart
```

And at the very bottom of the file, add a line which contains the following:

```
@lxterminal
```

Then reboot to put the changes into effect:

```
sudo reboot
```

Setting up the Touchscreen

Depending on which screen you use, instructions may vary. The following instructions use an UCTRONICS 3.5 Inch Touchscreen for program input.

1. Install **UCTRONICS Drivers**

   See the official instructions to set up the Raspberry Pi for the touchscreen.

   After the driver installation process reboots your Pi, you may have to change the display resolution. To do this on Raspberry Pi OS, click the Raspberry Pi button in the upper-left corner and go to **Preferences -> Raspberry Pi Configuration -> Display**. Click the button titled **Set Resolution...** and set the resolution to **CEA Mode 2 720x480 60Hz 4:3**. When you exit or press **OK**, the Pi will restart with a better resolution. If you get a black border around your screen, go back to **Raspberry Pi Configuration -> Display** and check **Disable** where it says **Overscan:**.
2. **Rotate Display** 90°

   Since the program has a vertical GUI, rotating the screen 90° will allow it to be displayed. To do this, run the following command:

```
sudo nano /boot/config.txt
```

3. And scroll to the bottom of the file. At the very bottom of the file, add one line which contains the following:

```
display_rotate=1
```

4. Do not reboot yet. We must also rotate the touch interfaces. To do this, run the following command:

```
sudo nano ~/.bashrc
```

5. And again scroll to the bottom of the file. At the very bottom of the file, add one line which contains the following:

```
xinput set-prop 'ADS7846 Touchscreen' 'Coordinate Transformation
Matrix' 0 1 0 -1 0 1 0 0 1
```

6. Then reboot to put the changes into effect:

```
sudo reboot
```

E.3 Program License (LICENSE.md)

Appendix F. Program Code

F.1 main.py

```
"""Main

This script enables a Raspberry Pi 4 to send files via Ad-Hoc to another Pi
running the same program and launches a graphical-user-interface to display
file transfer statistics.

This tool reads and writes from a text file 'update.txt', calls Python
files
'truck_manager.py' and 'gui.py', and classes 'Broadcast' and 'Pi'.

This script requires that 'batman-adv' be installed on the Pi you are
running
this program on.

Example
------------------------------------------------------------------------
----
To run this program, type the following into the correct directory in
terminal:

    $ python3 main.py

Notes
------------------------------------------------------------------------
----
Created by Austin Gilbert, Aashima Mehta, and Cameron Ufland for the
University
of Washington, Bothell in affiliation with PACCAR Inc.

Attributes
------------------------------------------------------------------------
----
Vehicle-to-Vehicle Update Delivery System README:
    https://github.com/capstone-paccar/v2v_interface/blob/main/README.md
"""

from gui import PiPiTransfer
from gui import PiPhoneTransfer
import sys
```

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication

# Creates application screen
app = QApplication(sys.argv)
userScreen = PiPiTransfer()

# Creates a stackable widget with multiple screens
widget = QtWidgets.QStackedWidget()
widget.addWidget(userScreen)
widget.setFixedHeight(720)
widget.setFixedWidth(480)
widget.show()

# Runs main event loop thread and exits when "X" is clicked/tapped
sys.exit(app.exec_())
```

F.2 gui.py

```
"""Graphical User Interface (GUI)

This file designs a graphical-user-interface which runs all Pi-Pi
management
via three threads and four classes.

Notes
--------------------------------------------------------------------------
----
Created by Austin Gilbert, Aashima Mehta, and Cameron Ufland for the
University
of Washington, Bothell in affiliation with PACCAR Inc.
"""

import sys, time
import socket
import pi
import broadcast
import subprocess
import logging
from PyQt5.uic import loadUi
from PyQt5.QtCore import QObject, QThread, pyqtSignal, pyqtSlot
```

```
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QDialog, QApplication, QLabel, QWidget

logging.basicConfig(format="%(message)s", level=logging.INFO)

class MainWorker(QThread):
    """The main worker thread used to run all Pi-Pi management code and
update
    status, progress, and version number on the GUI.

        The Pi-Pi management code broadcasts the Pi's version number to
other
        Pis and sets up a server or client depending on the version number
        received from other Pis.

    Attributes
    ----------
    status : pyqtSignal -> str
        the signal to update logging status and display on GUI
    progress : pyqtSignal -> int
        the signal to update the progress bar and display on GUI
    version : pyqtSignal -> int
        the signal to update the version number and display on GUI
    finished : pyqtSignal
        the signal to indicate finishing the thread

    Methods
    -------
    run()
        runs the Pi-Pi management program and emits all signals
    stop()
        stops the thread by setting boolean "running" to false
    """

    status = pyqtSignal(str)
    progress = pyqtSignal(int)
    version = pyqtSignal(str)
    finished = pyqtSignal()

    def __init__(self):
        super(MainWorker, self).__init__()
        self.running = True
```

```
    def run(self):
        """Runs the Pi-Pi management program and emits all signals. The
Pi-Pi
        management program broadcasts the Pi's version number to other Pi's
and
        sets up a server or client depending on the version number recieved
from
        other Pis.
        """

        self.status.emit('Starting...')
        # Reads version number of this Pi
        with open('update.txt', 'r') as file:
            version = int(file.readline())
        self.version.emit(f'{version}')
        this_Pi = pi.Pi(version, getSystemIP())
        time.sleep(1)

        # Loops until Pi is found
        while(self.running):
            # TODO: if 'Cancel' button pressed, exit while loop

            bdct = broadcast.Broadcast(this_Pi.getVersion())
            bdct.txBroadcast() # broadcasts version number to network
            oldTime = time.time()

            # Retries each time interval
            while (time.time()-oldTime) < 1.0:
                self.status.emit('Searching...')
                ver, addr = bdct.rxBroadcast() # Receives broadcasts on the
                                               # network
                ver = int(ver)

                # If program receives its own broadcast...
                if addr[0] ==  this_Pi.getIP()or addr[0] == "": continue
                else:
                    # If no broadcast found...
                    if ver == int(this_Pi.getVersion()) or ver == -1:
continue
                    else:
                        self.status.emit('Truck Found!')
```

```
                        time.sleep(1)

                        # If this Pi has outdated version...
                        if ver > int(this_Pi.getVersion()):
                            self.status.emit('Preparing for Update...')
                            self.progress.emit(10)
                            time.sleep(1)

                            # If server runs on this Pi successfully...
                            if (self.runServer(this_Pi)):
                                this_Pi.setVersion(ver) # Updates this Pi's
                                                        # version number

                        # If this Pi has an up-to-date version...
                        elif ver < this_Pi.getVersion():
                            self.status.emit('Preparing for Transfer...')
                            self.progress.emit(10)
                            time.sleep(1)

                            # Runs client on this Pi
                            self.runClient(pi.Pi(ver, addr[0]))

                        break

    def runServer(self, localPi):
        """Runs a TCP server on this Pi, thereby receiving an update file.

        Parameters
        ----------
        localPi : Pi
            the Pi object to designate this Pi

        Returns
        -------
        bool
            a boolean representing the success of the function
        """

        # Tries running a TCP server...
        try:
            serverAddr = (localPi.getIP(), 1750) # Creates server address
with
```

```
                                                    # local IP and Port 1750
            self.status.emit('Establishing Connection...')
            time.sleep(1)

            # Starts server
            with socket.socket() as server:
                server.bind(serverAddr)
                server.listen(1) # Listens for remote client
                server.settimeout(10) # Starts 10 second server timeout
timer
                conn, connaddr = server.accept() # Connects to remote
client
                self.status.emit('Connection Successful!')
                self.progress.emit(20)
                time.sleep(1)

            # With successful connection...
            with conn:
                self.status.emit('Downloading...')
                self.progress.emit(40)
                time.sleep(1)
                chunk = conn.recv(4096) # Downloads first chunk (4 kB)
                data = chunk # Stores the first chunk

                # While more chunks exist...
                while chunk:
                    chunk = conn.recv(4096) # Downloads chunks
                    data = data + chunk # Stores chunks
                self.progress.emit(90)
                time.sleep(1)

                # Writes stored data to local update file
                with open('update.txt', 'wb') as update:
                    update.write(data)

                self.status.emit("Update Downloaded Successfully!")
                self.progress.emit(100)
                time.sleep(1)
                return True

        # Error in running TCP server...
        except:
```

```
            self.status.emit("Error: Timeout in server.")
            time.sleep(1)
            return False

    def runClient(self, remotePi):
        """Runs a TCP Client on this Pi, thereby sending an update file.

        Parameters
        ----------
        remotePi : Pi
            the Pi object to designate the remote Pi

        Returns
        -------
        bool
            a boolean representing the success of the function
        """

        # Tries running a TCP client...
        try:
            clientAddr = (remotePi.getIP(), 1750) # Creates client address
with
                                                  # local IP and Port 1750
            self.status.emit('Establishing Connection...')
            time.sleep(1)

            # Starts client
            with socket.socket() as client:
                client.settimeout(10) # Starts 10 second client timeout
timer
                client.connect(clientAddr) # Connects to remote server
                self.status.emit('Connection Successful!')
                self.progress.emit(20)
                time.sleep(1)

                # Reads and sends stored data to server
                self.status.emit('Sending...')
                self.progress.emit(40)
                time.sleep(1)
                with open('update.txt', 'rb') as update:
                    client.sendfile(update, 0) # Sends file
                self.progress.emit(90)
```

```
                time.sleep(1)

                self.status.emit("Update Sent Successfully!")
                self.progress.emit(100)
                time.sleep(1)
                client.close()
                return True

        # Error in running TCP server...
        except:
            self.status.emit("Error: Timeout in client.")
            time.sleep(1)
            return False

    def stop(self, restart):
        """Stops the thread by setting boolean "running" to false"""
        self.status.emit('Ending Broadcast...')
        time.sleep(1)
        if restart:
            self.status.emit('Restarting...')
            time.sleep(1)
        self.running = False
        self.finished.emit()

class SignalWorker(QThread):
    """The secondary worker thread used to run subprocess terminal commands
and
    update the signal strength and signal quality on the GUI.

        The subprocess code uses the "ifconfig" bash command to get
information
        on signal strength and quality in the form of a dBm and percentage
        value respectively.

    Attributes
    ----------
    signalStrength : pyqtSignal -> str
        the internal signal to update the signal strength and display on
GUI
    signalQuality : pyqtSignal -> str
        the internal signal to update the signal quality and display on GUI
```

```
    Methods
    -------
    run()
        runs the subprocess program and emits all signals

    Notes
    -----
    This method will run until the window exits on a click/tap of the "X".
    """

    signalStrength = pyqtSignal(str)
    signalQuality = pyqtSignal(str)

    def __init__(self):
        super(SignalWorker, self).__init__()

    def run(self):
        """Runs the subprocess program and emits all signals."""
        while(True):
            batcmd = 'sudo iwlist wlan0 scan | egrep -C 2
"my-mesh-network"'
            getSignal = str(subprocess.check_output(batcmd, shell = True))
            try:
                self.signalStrength.emit('{}
dBm'.format(int(getSignal[50:53])))
            except ValueError:
                self.signalStrength.emit('{}
dBm'.format(int(getSignal[50:52])))
            except:
                self.signalStrength.emit('Error')

            try:

self.signalQuality.emit('{:.0%}'.format(float(getSignal[30:32])
                                                    /70.0))
            except ValueError:

self.signalQuality.emit('{:.0%}'.format(float(getSignal[30:31])
                                                    /70.0))
            except:
                self.signalQuality.emit('Error')
```

```
class PiPiTransfer(QDialog):
    """The class used to design a window with signals and slots for the
Pi-Pi
    management program.

    Methods
    -------
    runProgram()
        manages main worker thread and its signals
    collectSignal()
        manages secondary worker thread and its signals
    technicianMode()
        stops the main worker thread and switches screens to Pi-Phone
screen
    cancelled()
        restarts all threads; called when cancel button is pressed
    writeStatus(label)
        writes the status to the GUI
    writeProgress(value)
        writes the progress bar value to the GUI
    writeSignalStrength(label)
        writes the signal strength to the GUI
    writeSignalQuality(label)
        writes the signal quality to the GUI
    writeVersionNum(label)
        writes the version number to the GUI
    """

    def __init__(self):
        super(PiPiTransfer, self).__init__()
        loadUi("pipitransfer.ui",self) #editor must only open "Pi GUI
Folder"
        self.cancelButton.clicked.connect(self.cancelled)
        self.runProgram()
        self.collectSignal()

    def runProgram(self):
        """Manages main worker thread and its signals."""
        self.worker = MainWorker()
        self.finished.connect(self.worker.deleteLater)
        self.worker.status.connect(self.writeStatus)
        self.worker.version.connect(self.writeVersionNum)
```

```
        self.worker.start()

    def collectSignal(self):
        """Manages secondary worker thread and its signals."""
        self.worker2 = SignalWorker()
        self.finished.connect(self.worker2.deleteLater)
        self.worker2.signalStrength.connect(self.writeSignalStrength)
        self.worker2.signalQuality.connect(self.writeSignalQuality)
        self.worker2.start()

    def cancelled(self):
        """Restarts all Threads.
            Is called when the cancel button is pressed.
        """
        self.worker.stop(restart = True)
        self.runProgram()

    def writeStatus(self, label):
        """Writes the status to the GUI.

        Parameters
        ----------
        label : str
            text label to update on the GUI
        """
        self.status.setText(label)
        logging.info(label)

    def writeProgress(self, value):
        """Writes the progress bar value to the GUI.

        Parameters
        ----------
        value : int
            int label to update on the GUI
        """
        self.progressBar.setValue(value)

    def writeSignalStrength(self, label):
        """Writes the signal strength to the GUI.

        Parameters
```

```
        ----------
        label : str
            text label to update on the GUI
        """
        self.signalStrength.setText(label)

    def writeSignalQuality(self, label):
        """Writes the signal quality to the GUI.

        Parameters
        ----------
        label : str
            text label to update on the GUI
        """
        self.signalQuality.setText(label)

    def writeVersionNum(self, label):
        """Writes the version number to the GUI.

        Parameters
        ----------
        label : str
            text label to update on the GUI
        """
        self.versionNum.setText(label)

def getSystemIP():
    """Uses a subprocess to get the IP of the local Pi.

    Returns
    -------
    str
        a string representing the IP of the local Pi
    """

    batcmd = 'hostname -I'
    getIP = subprocess.check_output(batcmd, shell = True).strip()
    return getIP.decode("utf-8")

"""The following must be run in both gui.py and main.py:"""
# Creates application screen
app = QApplication(sys.argv)
```

```
userScreen = PiPiTransfer()

# Creates a stackable widget with multiple screens
widget = QtWidgets.QStackedWidget()
widget.addWidget(userScreen)
widget.setFixedHeight(720)
widget.setFixedWidth(480)
widget.show()

# Runs main event loop thread and exits when "X" is clicked/tapped
sys.exit(app.exec_())
```

F.3 broadcast.py

```
"""Broadcast

This class designs a transmission and receive socket to broadcast and
receive IP Address and Port data.

Notes
-----------------------------------------------------------------------
----
Created by Austin Gilbert, Aashima Mehta, and Cameron Ufland for the
University
of Washington, Bothell in affiliation with PACCAR Inc.
"""

import socket

class Broadcast:
    """The class used to represent a Broadcast socket for sending or
receiving.

    Attributes
    ----------
    broadcastAddress : str
        the IP address to broadcast
    port : int
        the port number to broadcast on
    version : int
        the version number to broadcast
```

```
    Methods
    -------
    rxBroadcast()
        handles receiving broadcasts
    txBroadcast()
        handles transmitting broadcasts

    """
    broadcastAddress = ''
    port = 0
    version = 0

    def __init__(self, version, port = 1200, broadcastAddress =
'<broadcast>'):
        self.broadcastAddress = broadcastAddress
        self.port = port
        self.version = version

        # Sets up transmission socket
        self.tx_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.tx_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.tx_sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)

        # Sets up receiving socket
        self.rx_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.rx_sock.settimeout(0.2)
        self.rx_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.rx_sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        self.rx_sock.bind(('', self.port))

    #Handles the receiving of broadcasts, it adds IP addresses to the
dictionary
    def rxBroadcast(self):
        """Handles receiving broadcasts.

        Returns
        -------
        (version, (address, port)) : tuple
            tuple object with socket information
        """
        try:
```

```
            data, addr = self.rx_sock.recvfrom(1024)
            return (str(data.decode()), addr)
        except:
            return(-1, ("", ""))

    #Handles the transmission. and then broadcasts the version number to
the network.
    def txBroadcast(self):
        self.tx_sock.sendto(bytes(str(self.version), "utf-8"),
(self.broadcastAddress, self.port))
```

F.4 pi.py

```
"""Pi

This class designs a Pi object to store and set a version number and IP
addresses.

Notes
-------------------------------------------------------------------------
----
Created by Austin Gilbert, Aashima Mehta, and Cameron Ufland for the
University
of Washington, Bothell in affiliation with PACCAR Inc.
"""

class Pi:
    """The class used to represent a Pi.

    Attributes
    ----------
    version : int
        the version number of the Pi
    address : str
        the IP address of the Pi

    Methods
    -------
    getVersion()
        the getter for the Pi's version number
    getIP()
```

```
        the getter for the Pi's IP address
    setVersion()
        the setter for the Pi's version number
    setIP()
        the setter for the Pi's IP address
    """

    version = 0
    address = ''

    def __init__(self, version, address):
        self.version = version
        self.address = address

    def getVersion(self):
        """The getter for the Pi's version number."""
        return self.version

    def getIP(self):
        """The getter for the Pi's IP address."""
        return self.address

    def setVersion(self, incomingVersion):
        """The setter for the Pi's version number."""
        self.version = incomingVersion

    def setIP(self, incomingAddress):
        """The setter for the Pi's IP address."""
        self.address = incomingAddress
```

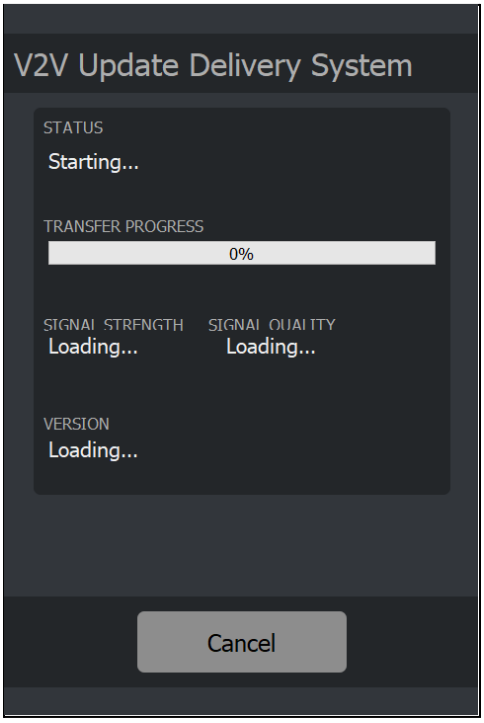
F.5 update.txt

```
2

Did you ever hear the tragedy of Darth Plagueis The Wise? I thought not.
It's not a story the Jedi would tell you. It's a Sith legend. Darth
Plagueis was a Dark Lord of the Sith, so powerful and so wise he could use
the Force to influence the midichlorians to create life... He had such a
knowledge of the dark side that he could even keep the ones he cared about
from dying. The dark side of the Force is a pathway to many abilities some
```

```
consider to be unnatural. He became so powerful... the only thing he was
afraid of was losing his power, which eventually, of course, he did.
Unfortunately, he taught his apprentice everything he knew, then his
apprentice killed him in his sleep. Ironic. He could save others from
death, but not himself.
```

F.7 pipitransfer.ui




```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Dialog</class>
 <widget class="QDialog" name="Dialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>480</width>
    <height>720</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Dialog</string>
  </property>
```

```
  <widget class="QWidget" name="widget" native="true">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>480</width>
     <height>720</height>
    </rect>
   </property>
   <property name="sizePolicy">
    <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <property name="maximumSize">
    <size>
     <width>480</width>
     <height>720</height>
    </size>
   </property>
   <property name="styleSheet">
    <string notr="true">background-color: rgb(50, 54, 59);</string>
   </property>
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>30</y>
      <width>480</width>
      <height>60</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">font: 75 24pt &quot;MS Shell Dlg 2&quot;;
background-color: rgb(35, 38, 41);
color: rgb(173, 173, 173);</string>
    </property>
    <property name="text">
     <string> V2V Update Delivery System</string>
    </property>
   </widget>
```

```
   <widget class="QPushButton" name="cancelButton">
    <property name="geometry">
     <rect>
      <x>135</x>
      <y>615</y>
      <width>212</width>
      <height>62</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">border-radius:7px;
background-color: rgb(141, 141, 141);
font: 18pt &quot;MS Shell Dlg 2&quot;;</string>
    </property>
    <property name="text">
     <string>Cancel</string>
    </property>
   </widget>
   <widget class="QLabel" name="label_2">
    <property name="geometry">
     <rect>
      <x>30</x>
      <y>105</y>
      <width>422</width>
      <height>392</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <family>MS Shell Dlg 2</family>
      <pointsize>12</pointsize>
      <weight>50</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">font: 12pt &quot;MS Shell Dlg 2&quot;;
background-color: rgb(35, 38, 41);
color: rgb(173, 173, 173);
border-radius:7px;</string>
    </property>
```

```
    <property name="text">

<string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;STATUS&lt;/p&gt;&lt;p
&gt;&lt;br/&gt;&lt;/p&gt;&lt;p&gt;&lt;br/&gt;TRANSFER
PROGRESS&lt;/p&gt;&lt;p&gt;&lt;br/&gt;&lt;/p&gt;&lt;p&gt;&lt;br/&gt;SIGNAL
STRENGTH     SIGNAL
QUALITY&lt;/p&gt;&lt;p&gt;&lt;br/&gt;&lt;/p&gt;&lt;p&gt;&lt;br/&gt;VERSION&
lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
    </property>
    <property name="textFormat">
     <enum>Qt::AutoText</enum>
    </property>
    <property name="scaledContents">
     <bool>false</bool>
    </property>
    <property name="alignment">
     <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignTop</set>
    </property>
    <property name="wordWrap">
     <bool>false</bool>
    </property>
    <property name="margin">
     <number>5</number>
    </property>
    <property name="indent">
     <number>5</number>
    </property>
   </widget>
   <widget class="QLabel" name="label_3">
    <property name="geometry">
     <rect>
      <x>0</x>
      <y>600</y>
      <width>480</width>
      <height>92</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">font: 75 16pt &quot;MS Shell Dlg 2&quot;;
background-color: rgb(35, 38, 41);
color: rgb(173, 173, 173);</string>
    </property>
```

```
  <property name="text">
   <string/>
  </property>
 </widget>
 <widget class="QProgressBar" name="progressBar">
  <property name="geometry">
   <rect>
    <x>45</x>
    <y>240</y>
    <width>392</width>
    <height>24</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <pointsize>12</pointsize>
   </font>
  </property>
  <property name="styleSheet">
   <string notr="true">background-color: rgb(30, 30, 255);</string>
  </property>
  <property name="value">
   <number>0</number>
  </property>
  <property name="alignment">
   <set>Qt::AlignLeading|Qt::AlignLeft|Qt::AlignVCenter</set>
  </property>
  <property name="textVisible">
   <bool>true</bool>
  </property>
 </widget>
 <widget class="QLabel" name="status">
  <property name="geometry">
   <rect>
    <x>45</x>
    <y>135</y>
    <width>392</width>
    <height>47</height>
   </rect>
  </property>
  <property name="font">
   <font>
```

```
      <family>MS Shell Dlg 2</family>
      <pointsize>14</pointsize>
      <weight>9</weight>
      <italic>false</italic>
      <bold>false</bold>
     </font>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: rgb(35, 38, 41);
color: rgb(239, 239, 239);
font: 75 14pt &quot;MS Shell Dlg 2&quot;;</string>
    </property>
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
style=&quot;
font-size:16pt;&quot;&gt;Starting...&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt
;/html&gt;</string>
    </property>
   </widget>
   <widget class="QLabel" name="signalStrength">
    <property name="geometry">
     <rect>
      <x>45</x>
      <y>330</y>
      <width>122</width>
      <height>32</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: rgb(35, 38, 41);
color: rgb(239, 239, 239);
font: 75 14pt &quot;MS Shell Dlg 2&quot;;</string>
    </property>
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
style=&quot;
font-size:16pt;&quot;&gt;Loading...&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;
/html&gt;</string>
    </property>
   </widget>
   <widget class="QLabel" name="signalQuality">
    <property name="geometry">
```

```
     <rect>
      <x>225</x>
      <y>330</y>
      <width>122</width>
      <height>32</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: rgb(35, 38, 41);
color: rgb(239, 239, 239);
font: 75 14pt &quot;MS Shell Dlg 2&quot;;</string>
    </property>
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
style=&quot;
font-size:16pt;&quot;&gt;Loading...&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;
/html&gt;</string>
    </property>
   </widget>
   <widget class="QLabel" name="versionNum">
    <property name="geometry">
     <rect>
      <x>45</x>
      <y>435</y>
      <width>122</width>
      <height>32</height>
     </rect>
    </property>
    <property name="styleSheet">
     <string notr="true">background-color: rgb(35, 38, 41);
color: rgb(239, 239, 239);
font: 75 14pt &quot;MS Shell Dlg 2&quot;;</string>
    </property>
    <property name="text">
     <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;&lt;span
style=&quot;
font-size:16pt;&quot;&gt;Loading...&lt;/span&gt;&lt;/p&gt;&lt;/body&gt;&lt;
/html&gt;</string>
    </property>
   </widget>
   <zorder>label_3</zorder>
   <zorder>label_2</zorder>
```

```
   <zorder>label</zorder>
   <zorder>cancelButton</zorder>
   <zorder>progressBar</zorder>
   <zorder>status</zorder>
   <zorder>signalStrength</zorder>
   <zorder>signalQuality</zorder>
   <zorder>versionNum</zorder>
  </widget>
 </widget>
 <resources/>
 <connections/>
</ui>
```