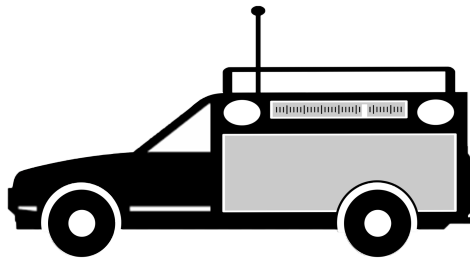




Boston University
Electrical & Computer Engineering
EC 463 Senior Design Project

User's Manual

C-V2X Misbehavior Detection System



Submitted to

David Starobinski
8 St. Mary's St Boston, MA 02215 Room 431
(617) 353-0202
staro@bu.edu

by

Team 13
C-V2X Misbehavior Detection System

Team Members

Michael Aliberti mjali@bu.edu
Max Ellsworth maxell@bu.edu
Jason Inirio jasonini@bu.edu
Sam Krasnoff krasnoff@bu.edu
Julia Zeng zjulia@bu.edu
Yixiu Zhu zhuyixiu@bu.edu

Submitted: March 24, 2022

User Manual

Table of Contents

Executive Summary	3
1 Introduction	4
2 System Overview and Installation	5
2.1 Overview block diagram	5
2.2 User Interface	5
2.3 Physical Description	5
2.4 Installation, setup, and support	5
3 Operation of the Project	6
3.1 Operating Mode 1: Normal Operation	6
3.2 Operating Mode 2: Abnormal Operation	6
3.3 Safety Issues	6
4 Technical Background	7
5 Relevant Engineering Standards	8
6 Cost Breakdown	9
7 Appendices	10

Executive Summary (by Michael Aliberti)

Vehicle to Everything Technology enables cars and IOT devices with an LTE connection to broadcast positional information such that vehicles can perceive their surroundings without line of sight. The goal of this project is to provide insight into the strengths and vulnerabilities of the Vehicle to Everything protocol. Before it is widely adopted, its reliability and security must be verified. To this end, we are developing a monitoring system that listens on Vehicle to Everything frequency blocks, verifies message integrity, and leverages machine learning to detect denial of service attacks. This system will consist of a dedicated radio and a web application to display information about frequency block resource usage. We will also be modeling examples of ordinary communication and denial of service attacks to provide data to the monitor.

1 Introduction (by Michael Aliberti)

As the information age continues to push the boundaries of interconnectivity, vehicular communication is beginning to incorporate modern innovations. Vehicle-to-Everything, or V2X, technology is poised to revolutionize the way that cars interact with their surroundings. By leveraging LTE signals, the V2X protocol allows for information to be exchanged between cars and any wirelessly connectable device, which includes civilian smartphones, bikes, and even other cars. By creating this network of devices, vehicles will be notified of road conditions, accidents, and other unexpected events with unprecedented speed and accuracy. Estimates predict that signals pertaining to left-turn warnings and blind-spot detection alone could prevent upwards of 600 thousand crashes and save over 1,000 lives each year.

However, with the arrival of this new technology comes new challenges and malicious actors. In normal use, V2X-capable devices will be transmitting and receiving vital data, like GPS position and velocity. If the transmissibility of messages is poor or if an attacker is able to jam the network through a Denial-of-Service (DoS) attack, then vehicles will be rendered unable to properly assess their surroundings. This compromises the integrity of the mesh network, and the likelihood of collisions skyrockets.

This project aims to not only examine the speed and efficacy of a cellular V2X network, but also to explore the detection and possible avoidance of malicious signals sent by attackers. Through machine learning and cyber security principles, we have developed a software-defined radio (SDR) application to detect and warn users in real-time about ongoing DoS attacks. By verifying the reliability and security of V2X technology, we can hopefully reap its benefits while avoiding its pitfalls.

The purpose of this project is to build a monitoring infrastructure to observe communication (e.g. channel resource usage) over a Cellular Vehicle-to-Everything, or C-V2X, network and detect potential anomalies, including DoS attacks. There are four main components to the project as outlined by our customer: (1) to demonstrate the ability to transmit and receive C-V2X messages, (2) to visualize resource usage, (3) to implement DoS attacks through a jammer, and (4) to design and validate a misbehavior detection system to thwart such attacks.

To gauge reliability and end-user safety in cellular Vehicle-to-Everything communication, we have implemented a functional model of communication. This model includes two radios functioning as ordinary vehicles. Normal basic safety messages are sent on multiple orthogonal frequency bands in the V2X range between these two radios, serving as a source of control data and as a testbed for attack models. A third radio acts as an attacker or team of attackers in the form of a jammer, attempting to prevent normal operation and cause instability through DoS attacks. This radio also serves to monitor the entire V2X frequency spectrum, using GNURadio to save all in-phase and quadrature (IQ) data observed to files for later use. This data is then used to train a machine learning algorithm in order to provide insights into V2X resource utilization and attack patterns.

2 System Overview and Installation (by Samuel Krasnoff)

2.1 Overview block diagram

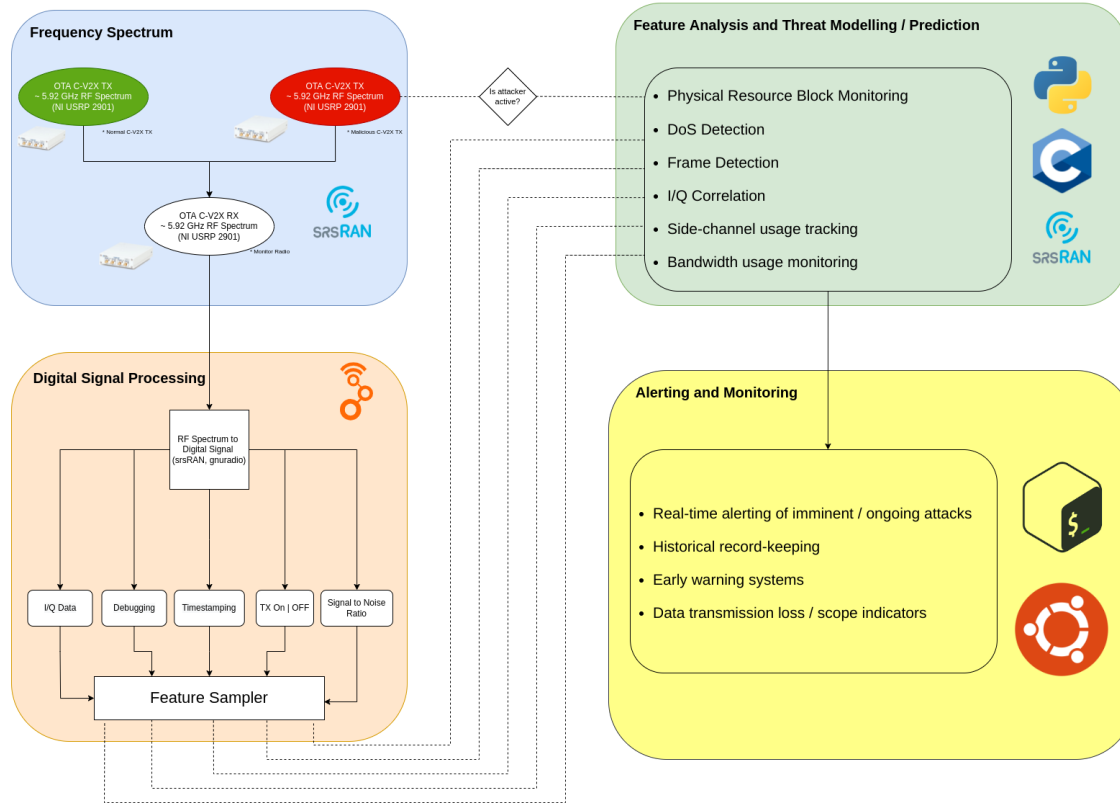


Figure 2.1 A diagram identifying the individual components of the C-V2X Misbehavior Detection System. The hardware setup comprises the Frequency spectrum block, SrsRAN performs signal processing, and data collection and machine learning cover the final two components.

2.2 User Interface

Gaining insight into C-V2X traffic requires a number of tools, including GQRX and GNURadio for IQ data visualizations and the SrsRAN library for traffic generation and reception. Some of these tools have their own interface built-in, such as GQRX and GNURadio, whereas SRSRan is accessible through the command line interface. Due to the variety of tools incorporated and the intended audience of this project being a research-oriented, technically savvy group, no overarching user interface has been incorporated beyond the command line.

However, to streamline the user experience and to allow for rapid data generation, a number of scripts have been included in our code base, such as the `run_hw_loopback` script in the modSrsRAN repository, which creates two SrsRAN communication endpoints and begins collecting analytics for a configurable time frame with one command as opposed to six or more.

2.3 Physical Description

The system, at its heart, runs from 3 NI 2901 USRPs. The transmit and receiver devices each synced through a connection to a Keysight 33500B Series waveform generator. Each USRP may be connected to the same computer through a USB connection for both power and data transfer. The rest of the content is digital, and is reliant on the configuration of the computer.



Figure 2.1 The setup for producing and monitoring C-V2X traffic. The left computer is attached to the central radio and acts as a receiver. The right computer is connected to the two rightmost radios and conducts transmission and monitoring. The two central signal generators synchronize all radios to ensure valid over-the-air communication.

2.4 Installation, setup, and support

Installation

The installation area must have sufficient space and power outlets for two computers, two waveform generators, and up to four software defined radios.

Setup

1. Connect the Base Station and User End USRPs to power sources and separate computers.
2. Ensure these USRPs have been properly connected and are visible to the computer by running `uhd_find_devices` on the command-line. Look for the device name “b200.”
3. Configure signal generators with the following settings:
 - a. A “Pulse, Off, 50 Ohm” wave which is “AM modulated by sine”, frequency of 1 Hertz, amplitude of 10 dBm.
 - b. A “Square, Off, 50 Ohm” wave which is “AM modulated by sine”, frequency of 10 Megahertz, amplitude of 100 millivolts (this should probably be 10 dBm).
4. Connect the base station and user end USRPs to both signal generators. Ensure that each signal generator is connected to the same relative input on each USRP.
5. Clone and build the `modSrsRAN` and `c-v2x-traffic-generator-fork` repositories from the C-V2X-Senior_Design github page [here](#).
6. Open up a terminal on the base station; run `srsepc` and `srsenb`.
7. Open a terminal on the user end and run `srsue`.

8. Activate the trace on the User End and Base Station.
9. Connect the jammer/monitor USRP to the computer hosting the Base Station.
10. Open GQRX on the PC attached to the jammer/monitor. Configure it to read data from the monitor by checking "Ettus B200" in "File => I/O Devices" and typing "ctrl+D."
11. Observe traffic in the 5.92 GHz range when `srsepc` and `srsehb` are and are not communicating with each other. If there is a difference in activity, the system is working.

3 Operation of the Project (by Max Ellsworth, Jason Inirio, and Julia Zeng)

3.1 Operating Mode 1: Normal Operation

Traffic Generator

During normal operation, the traffic generator will be activated through a terminal command with no flags. It will continuously output formatted V2X packets for any radios listening in the near radius. The traffic generator is used for stress testing of communications systems on the physical layer and the SCI transmitted is used to indicate used resources to COTS C-V2X modems to constrain the available time-frequency resources externally. The C-V2X transmission code runs on top of the SrsRan library which has capabilities for building LTE uplink and downlink networks for both virtual and USRP-equipped systems. The C-V2X transmit code adds sidelink communication capabilities and configuration standards associated with the protocol. To generate traffic, turn on the USRPs and make sure they are attached to the signal generator. Next, navigate to the `cv2x_traffic_generator` directory under the build folder and run `./cv2x_traffic_generator --help` to print information about available configuration flags. Run `./cv2x_traffic_generator -d UHD -a clock=external -o logfile.csv -n 5 -l 10 -s 0` to test a static transmission where the same resource blocks are allocated for every packet.

Receiver

During normal operation, the receiver `pssch_ue` will be executed from the console to process all data being transmitted from the traffic generator. This will parse and save the data for use by the machine learning algorithms. The receiver outputs resource block allocation data, along with the associated subframe index and system timestamp in milliseconds. It generates an output whenever there is valid SCI data being transmitted. To generate the output, navigate to the `pssch_ue` directory under the build folder and run the `pssch_ue` binary. Run `./pssch_ue --help` to print information about available configuration flags. For the purposes of our data-collection we have left it at the default configuration for the sub-channel size (10) and number of sub-channels (5). The default configuration allocates a resource pool of 50 resource blocks. The C-V2X receiver-jammer network connection can be automated using the shell script which accepts a command-line argument for time length of connectivity in seconds. Ensure that separate radios are transmitting and receiving traffic by checking that one radio has the red TX LED on and another has the green RX LED on.

Jammer

The jammer we are using is a modified version of one written by YaYa Brown and Cynthia Teng of Worcester Polytechnic Institute (WPI). This jammer does not work with the latest version of gnuradio due to the `ofdm_mod` block being phased out. The workaround to this was to use a Docker image provided by our graduate student advisor and client, Stefan Gvozdenovic, which uses gnuradio version 3.7.11. The Docker image is available [here](#). YaYa Brown and Cynthia Teng's paper (which links to jammer) is available [here](#) and the citation for their work is the following:

Brown, YaYa Mao, and Cynthia Teng. *LTE Frequency Hopping Jammer*. : Worcester Polytechnic Institute, 2019.

Our modified version of the WPI jammer is available [here](#) (top_block.py). Instructions for how to run the Docker image are available on the Github repo linked above. Once the Docker image has been downloaded and installed, the jammer can be run as a Python executable by running `./jammer.py`. In order to get the jammer to quit correctly, it is important to close it by selecting “x” in the GUI box titled “top block.”

Machine Learning Model

Before getting started with implementing and evaluating machine learning models. You must set up a virtual environment with the necessary libraries needed. To accomplish this, use the command:

```
Python -m venv .
```

Where you can activate the python virtual environment by using “`source ./bin/activate`” for MacOS/UNIX or “`.\Scripts\activate`” for Windows. After this step is done, you can download the necessary libraries by simply running:

```
Pip install -r requirements.txt
```

The machine learning models are all located as an inheritance of the **model** class from the models library (`models.py`). To add models, you can follow the template located in the models library to create your own model. The models class also holds values for input shape, name, and other various functions needed for training and evaluation.

`CreateModels.py` is the primary script used for training and testing the models located inside the models library. To properly utilize it, you must invoke your new model inside the script under the heading “ADD MODELS HERE” by using the line:

```
models.append(m.YOUR_MODEL_CLASS())
```

After doing so, the model will begin to run and train over a period of samples (N) created by a simulator or data collector library. The **simulator** (`simulator.py`) and **preprocessing** (`preprocess.py`) library are both used under the assumption of a 10x50 Resource Pool input. To change this value, simply input the arguments of the **simulator** in `CreateModel.py` and change the input in the **model** class. The values in the **model** class are SubCh and SubFr that represent subchannels and subframes respectively.

After your models are created, you can evaluate them (without the need of invoking `CreateModels` again) by running `evaluation.py`.

* By default, the evaluation trains and tests SciKit Learn algorithms against your machine learning model. This broadens the various models we can use on implementing misbehavior detection on signals with resource blocks.

3.2 Operating Mode 2: Abnormal Operation

In the case of a USRP crashing or not being recognized, a reseating of the power cord in the radio generally resolves this issue. Reseating the usb connection between the radio and the computer is also recommended. Device recognition may then be verified by running of *uhd_find_devices* and ensuring that the name of the attached USRP appears.

In the case that two USRPs are unable to connect to each other when running the *main_hw_loopback* script, ensure that the antennae on each radio are secure, verify your installation of modSrsRAN, and repeat the approach above.

One important note about the modSrsRAN and c-v2x-traffic-generator-fork repositories is that both codebases contain immutable loggers which record traffic analytics. These loggers have no limit on how much data they record when run outside of pre-made scripts, so running code from either of these repositories through unintended pass-throughs may lead to data corruption.

3.3 Safety Issues

As these are open electronic components, one must be careful when handling liquids near the radios, computers, and waveform generators. Additionally, generated radio waves can interrupt other broadcast activities, so this setup should be kept separate from critical devices.

4 Technical Background (by Yixiu Zhu)

Our project has three primary components: hardware & signal generation, software and signal processing, and machine learning.

For the signal generation, it was important to also cause denial of service, in the form of a jammer. We adapted a jammer written by Ya Ya Brown and Cynthia Teng of Worcester Polytechnic Institute (WPI) in order to fit our needs for a frequency-hopping jammer. The WPI jammer was written using gnuradio-companion version 3.7.11 and contains the ODFM Mod block not found in the latest iteration of gnuradio-companion, version 3.8.1.0, so we decided to use an older version of GNURadio.

We adjusted the center frequency range and jamming duration in the jammer code in order to get it to meet our needs. Within the WPI python code `top_block.py`, the random center frequency threshold was changed to $5.9165e9 < \text{center_freq} < 5.917e9$. It takes approximately 100 ms for a full V2X packet to transmit, due to the resource block allocation of the LTE structure it is based on. Thus, 25 millisecond shifts of the jammer would not produce enough interference to ruin transmissions. By altering the timing to 1 second shifts, this allows the jammer to thoroughly disrupt communication between C-V2X enabled devices. The three physical radios were configured to be in close-proximity with each other in order to minimize issues associated with variable gain.

Due to the complete randomness of this jammer, this is considered an “oblivious” jammer, as it does not actively search for occupied channels. Using gnuradio-companion, we wrote receiver code with the purpose of monitoring the IQ behavior of the signals. While setting bandwidth and gain was fairly straightforward, we encountered difficulties in setting the squelch such that the final IQ data would be restricted to pick up C-V2X / jammer transmission and not noise. The IQ plots and data created from the transmissions of the traffic generator and jammer separately, and then overlaid, allow for unique signatures that can help our machine learning algorithm train. Once the data was being produced, it was up to the data collection team to process the IQ data into numerical values.

As our primary goal is to distinguish jammed states from unjammed states, we are using two different types of data as points of analysis: IQ and resource block allocation data. IQ data is both easily accessible and provides the most complete insight into the behavior of C-V2X traffic, but is cumbersome to train with in that each second of transmission generates tens of thousands of IQ data points. Collecting resource block information provides more specific insights and generates fewer data points, but is more limited in terms of scope.

The main libraries we used to simulate and receive C-V2X traffic were SrsRAN and Fabian Eckerman's C-V2X traffic generator, both of which are written in C++. These libraries are built with ease of communication in mind, but data interfaces for extracting and analyzing metrics are not built-in. To make these libraries more amenable to our needs, we have created and hosted the dual forked repositories `modSrsRAN` and `c-v2x-traffic-generator-fork`, which include built-in scripts for running native code and codebase modifications which extract and record traffic analytics like resource block allocation and IQ data.

IQ data plotting natively ships with SrsRAN, but there is no built-in API for extracting the data itself. The modSrsRAN library ameliorates this by adding functions dedicated to printing IQ data to the library, and calling these functions at appropriate code segments so as to print IQ data to a file in the *probes/* directory when running SrsRAN communication programs like *srsenb* and *srsue*. These programs are in-turn called by the user-facing script *main_hw_loopback*, which safely automates the data-collection process.

A similar approach was taken to extracting the resource block group (RBG) data from the srsRAN source code. By locating and printing *rbgmask_t* variables that are reached during runtime, the allocation arrays on the transmission side are parsed and printed to a file in the *probes/* directory via the same *main_hw_loopback* script. To provide further insight into resource block allocation, this data is also extracted on the receiver-end in the c-v2x-traffic-generator-fork repository, where sidelink control information (SCI) resource blocks are decoded and printed to a file in the same fashion.

The above data is used in training machine learning models to detect jamming. We incorporated a series of python scripts that would help with automating and training these models.

The binary map implication was influenced by studies from Keysight Technologies in their studies on LTE physical layers.

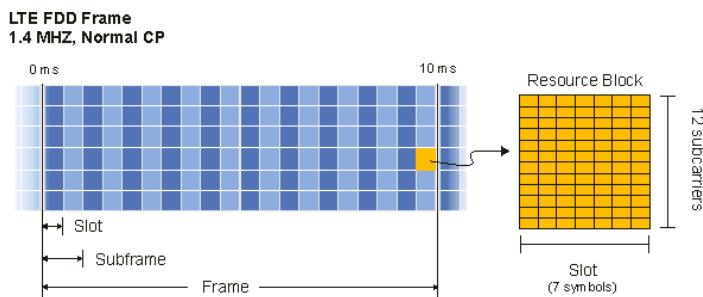


Figure 4.1 A visualization of resource block allocation on the LTE physical layer. Each frame spans 10 ms and is subdivided into subframes each of which is capable of carrying at most one transmission. A transmission can use one or more resource blocks within a subframe.

The following represents the final product of our simulator and how we thought of implementing it to match our criterias for training the simple convolutional neural net (CNN) model. The model was able to take in an input of 10x50 “images” from a serialization of the entire resource pool that spans for 5 frames (10 subframes each) with 10 subchannels. With the simulator’s output, we engaged with the idea of having a jammed signal to be a simple binary representation of 0 versus a binary representation of 1 for unjammed. Therefore, the entire resource pool must be a map of 1’s otherwise it is considered jammed. Despite the simplicity of the first design, this model’s unreliability and the lack of viable supporting data led to this model being abandoned in favor of a newer, more promising model.

Our next approach is to utilize the components of the physical layer and produce either a Q-learning reinforcement learning model or a deep learning model. We first chose to proceed

with the deep learning approach. Deep learning models have a better model architecture compared to other approaches, e.g. our simple CNN with Q-learning integrated, so it can utilize our limited computing resources much more efficiently. Furthermore, deep learning models have a huge number of learnable weights and possibly can outperform our previous approaches.

For the first trial we did for deep learning, we selected the similar input mentioned above. We tried a very simple DL model and to see how it could extract features from our self-generated dataset. We tested the model based on three layers and five layers of 2D ConvNet followed by a Max Pooling layer. The result we obtained was not that satisfying, we also found our self-generated data has already cut down many of the trainable features from the raw radio signal, so the model is not that responsive as we expected.

We started our second trial with a considerable new input: the waterfall diagram directly from the traffic generator. This is a completely new approach so we are still in the midst of making three trials based on a graduate team's research in Noiselab UCSD. In short, the CNN approach had data plotting issues, but due to a similar construction model for layers, this model still has some promise. The ResNet allowed for better tuning to our dataset, but may not be as well suited for outputs. The CLDNN has given us the most trouble, with numerous dataset import issues, but we are in the process of using the Pickle format to load preprocessed data for more precise training of the model.

As a precursor to training the aforementioned deep learning models on self-generated data, we trained them using the DEEPSIG RADIOML 2016.10A synthetic dataset produced by DeepSig Inc. which contains RF data generated in GNU Radio. The results demonstrate the capability of our deep learning method for feature extraction of RF data. The trained model provides a framework to drop in future self-generated data and insight into suitable data extraction methods for GNU Radio.

Lastly, we'd like to mention more about our layer construction experience. For the deep learning trials we executed above, we found adding batch normalization layers after each convolutional layer can effectively make the loss converge faster and lead to a higher accuracy.

5 Relevant Engineering Standards (by Sam Krasnoff, Julia Zeng)

C-V2X uses 3GPP standardized 4G LTE or 5G mobile cellular network to transmit and receive signals from other vehicles. The 3GPP standard covers cellular telecommunications technologies, including radio access, core network and service capabilities, and for interworking with non-3GPP networks.

C-V2X is also an alternative to 802.11p, the IEEE standard for V2V communications protocols.

3GPP Release 14, finalized in 2017, includes specifications for C-V2X communications, adapting upon the V2X standards set in 3GPP Release 13, finalized in 2014.

The United States Federal Communications Commission has very thoroughly divided the airwaves to split between private and public communications. In May of 2021, a document was drafted which finalized the allocation of the 5.895 - 5.925 GHz band from the deprecated and much less robust protocol Dedicated Short Range Communications (DSRC) to C-V2X. The document additionally specifies the steps and timeline for the Intelligent Transportation System, a set of applications and programs to monitor and optimize modes of transportation and traffic management. Finally, it discusses the technical specifications/requirements of qualified devices as well as transmitter power and emission limits, in an effort to standardize the use of this communication vector.

6 Cost Breakdown (by Michael Aliberti)

Below is a breakdown of all the equipment used and their costs. As a research project where all components were provided by the client, this project came at no personal cost, and market viability was generally not considered a priority.

As this project was conducted in accordance with the Air Force Research Laboratory's (AFRL) Beyond 5G challenge, the usage of NI 2901 USRPs was required, making the usage of these USRPs and all related costs unchangeable. The oscilloscope and signal generators used, on the other hand, could be replaced with cheaper alternatives without altering functionality and were simply used due to their availability within the Boston University Photonics building where our client resides. Replacing the signal generators with significantly cheaper GPSDO modules, which serve to synchronize radios in the same fashion and also technically reflect field conditions of the C-V2X protocol more accurately, could significantly reduce the price of this lab setup. However, the heavy interference within the Photonics building makes our hard-wired solution more viable for our client. The desktop computers used in this setup are not uniquely significant and could also be replaced; the operating system, DragonOS, is freely available online and may be installed to any system.

One addition that could improve this setup would be a true C-V2X modem which could be used in conjunction with our USRPs in order to verify the integrity of our generated C-V2X traffic as well as our receiver. The only issue

No.	Project-related Device & Materials	Specs of Device & Material		Total Cost (\$)
		Quantity	Unit Cost (\$)	
1	Desktop with Dragon OS	2	Around 500	1,000
2	NI 2901 USRPs (AFRL provided)	3	2010	6,030
3	Keysight 33500B signal generators	2	1979	3,958
4	LeCroy Wavesurfer 422 Oscilloscope	1	5141	5,141
Total Cost				16,129

7 Appendices (by Michael Aliberti)

7.1 Appendix A - Specifications

Requirements	Value, Range, Tolerance, Units
Frequency Band	5.92 GHz
Frequency Accuracy	2.5 ppm
Misbehavior Detection Accuracy	50% (in-progress)
Operable Range	1+ m (untested further due to setup limitations)
Power Requirement	6 V, 3 A DC per USRP

7.2 Appendix B - Team Information

Michael Aliberti

Computer engineering student involved in SrsRAN modification, data collection, documentation, and visual asset creation. Future software engineer at Tulip Interfaces.

Max Ellsworth

Computer engineering student involved in radio setup and synchronization, over-the-air communication, and jammer design. Future member of United States Air Force.

Jason Inirio

Computer engineering student involved in the creation of a machine learning model based on resource block usage. Future software engineer at IBM.

Sam Krasnoff

Computer engineering student involved in radio hardware configuration, documentation, and jammer design.

Julia Zeng

Computer engineering student involved in SrsRAN modification, data collection, protocol comprehension, and documentation. Future software engineer at Wolverine Trading.

Yixiu Zhu

Electrical engineering student involved in the creation of a deep learning model based on in-phase and quadrature data.