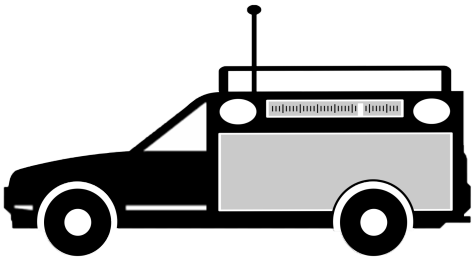
Boston University
Electrical & Computer Engineering
EC 463 Senior Design Project

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 (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.

Glossary (by Michael Aliberti)

Air Force Research Lab (AFRL) - organization which conducts research into radio operations

Basic Safety Message (BSM) - the payload of a C-V2X transmission which communicates position, velocity, and acceleration

Beyond 5G Challenge - a competition run by the AFRL into which this project was entered

Cellular Vehicle to Everything (C-V2X) - wireless communication protocol which uses sidelink channels to allow devices to directly communicate with each other

Denial of Service (DoS) - an attack which limits the availability of a protocol or resource

DragonOS - a linux operating system which contains pre-installed radio software

Global Position System (GPS) - used in C-V2X to synchronize devices and provide position, velocity, and acceleration information for basic safety messages

GNURadio - program with graphical user interface used to receive, transmit, and modulate radio signals

GQRX - program used for radio signal analysis and modulation

In-phase and Quadrature (IQ) - physical layer data which characterizes radio signals

Long-term Evolution (LTE) - protocol for over-the-air telecommunications

Resource Block (RB) - subdivision of time and frequency along which individual cars transmit messages within the C-V2X protocol; a grid of resource blocks make up a subframe

Sidelink Control Information (SCI) - resource blocks in C-V2X which provide metadata for transmissions

Software-defined Radio (SDR) - an over-the-air communication device which operates with a connected computer as its interface

SrsRAN - an open source software suite which enables the execution of cellular protocols on software-defined radios

Subframe - periodically repeating subdivision of time and frequency which constrains C-V2X traffic; a grid of resource blocks

TX/RX - transmit and receive

Universal Software Radio Peripheral (USRP) - radio hardware which interfaces with a computer to form a software-defined radio

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.

With the arrival of this new technology however, 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 to detect potential anomalies, including DoS attacks. There are four main components of 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, (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, which serve 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 monitors the entire V2X frequency spectrum, using GNURadio to save all in-phase and quadrature (IQ) data observed to files for later use. These data are 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

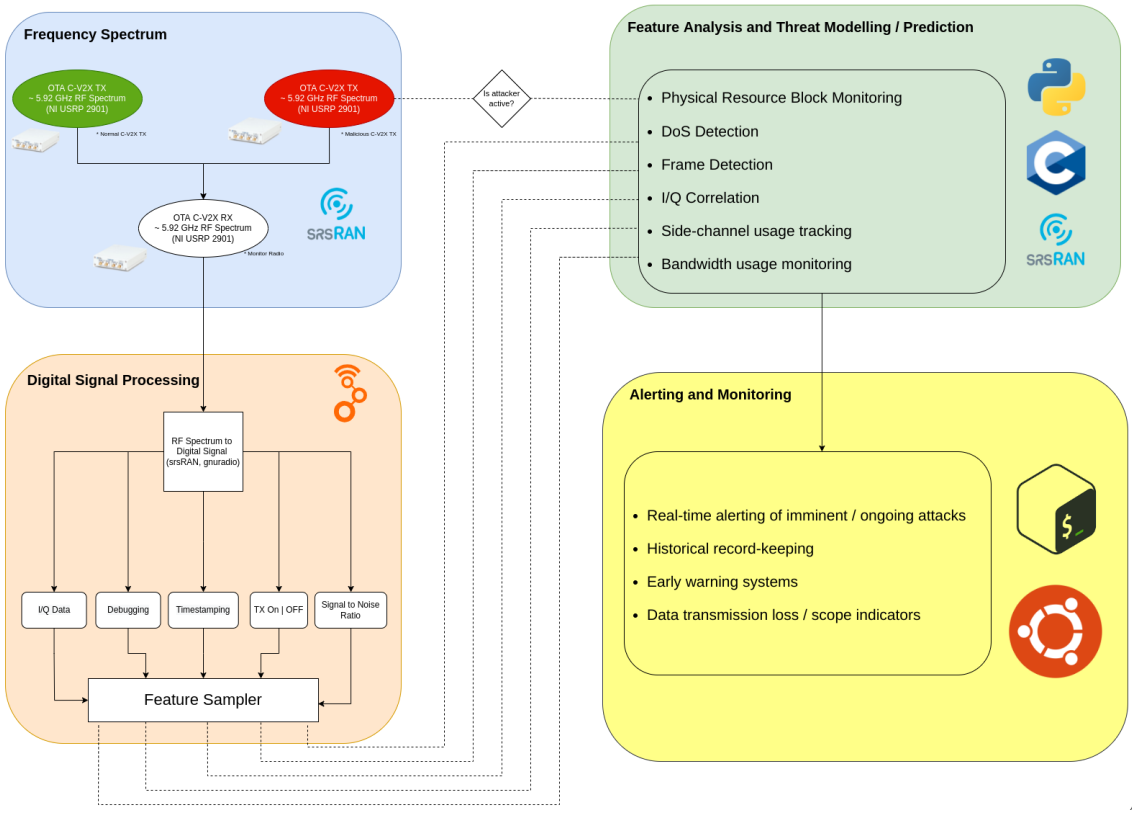


Figure 2.1 A block diagram identifying the individual components of the C-V2X Misbehavior Detection System. The hardware setup is outlined in the Frequency Spectrum block and SrsRAN performs Digital Signal Processing. Data collection and machine learning cover the final two blocks.

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 built-in interface, 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.

To streamline the user experience and to allow for rapid data generation however, a number of scripts have been included in our code base. The *run_hw_loopback* script in the modSrsRAN repository, for example, creates two SrsRAN communication endpoints and collects analytics for a configurable time frame with a single command.

2.3 Physical Description

The core of the system consists of 3 NI 2901 USRPs. The transmitter and receiver devices are synchronized through a shared connection to a Keysight 33500B Series waveform generator. Each USRP is connected to a computer through a USB connection to enable both power and data transfer. The rest of the content is digital, and is reliant on the configurations of the computers.

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 `srsenb` 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 to stress test communications systems on the physical layer, and the SCI transmitted is used to indicate occupied 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

The receiver, *pssch_ue*, is executed from the console to process all data being transmitted from the traffic generator. This program parses and saves 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 its red TX indicator light on and another has the green RX indicator light 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 as follows:

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

A virtual environment with the necessary libraries is needed before implementing and evaluating machine learning models. To begin, use the command:

Python -m venv .

Activate the python virtual environment by using "*source ./bin/activate"* for MacOS/UNIX or ".*\Scripts\activate"* for Windows. Then, download the necessary libraries by 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, follow the template located in the models library to create a new 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, your new model must be invoked inside the script under the heading "ADD MODELS HERE," 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, 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, which represent subchannels and subframes respectively.

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

* By default, the evaluation trains and tests SciKit Learn algorithms against the machine learning model. This broadens the range of usable models for implementing misbehavior detection on signals with resource blocks.

3.2 Operating Mode 2: Abnormal Operation

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

If 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.

The modSrsRAN and c-v2x-traffic-generator-fork repositories both 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) 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 better reflect the C-V2X protocol. Within the WPI python code top_block.py, the random center frequency threshold was changed to 5.9165e9 < 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. Altering the timing to 1 second shifts allows the jammer to thoroughly disrupt communication between C-V2X enabled devices. The three physical radios are configured to be in close-proximity to each other in order to minimize issues associated with variable gain.

This is considered an “oblivious” jammer, as it uses random channel selection instead of actively searching 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 so that the final IQ data would be restricted to pick up C-V2X / jammer transmission and not noise. As a workaround, we found that generating IQ plots from the traffic generator and jammer separately and combining the results in-post produced a higher quality dataset and reduced machine learning algorithm training time. Once the data were 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 are both easily accessible and provide the most complete insight into the behavior of C-V2X traffic, but are cumbersome to train with because 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. The modSrsRAN library solves this by adding functions which print IQ data to a file in the *probes/* directory whenever SrsRAN programs like *srsenb* and *srsue* are executed. 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_loopbakc script*. To provide further insight into resource block allocation, these data are 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 are 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.

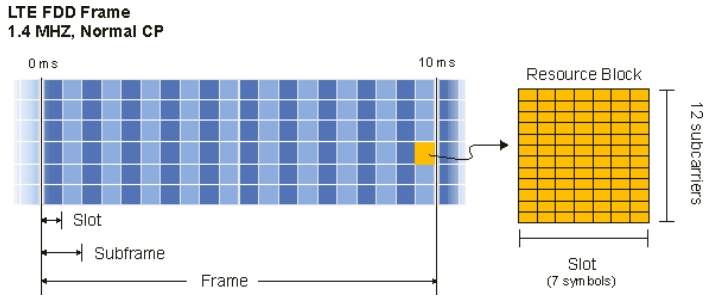


Figure 4.1 A visualization of resource block allocation on the LTE physical layer. Each frame spans 10 ms and is subdivided into subframes each of which is capable of carrying at most one transmission. A transmission can use one or more resource blocks within a subframe.

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. In the simulator’s output, jammed signals are represented as binary zeros whereas unjammed signals are represented as binary ones. Therefore, the entire resource pool must be a map of 1s; otherwise, it is considered jammed. Despite the simplicity of the first design, this model’s unreliability and the lack of viable supporting data led us to abandon this model in favor of a newer, more promising one.

Our next approach was 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 more efficiently. Furthermore, deep learning models have a huge number of learnable weights and may outperform our previous approaches.

For our first deep learning trial, we selected the similar input mentioned above. We tried a very simple DL model 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 unsatisfactory, and we also found that our self-generated data had already cut down many of the trainable features from the raw radio signal, so the model was not as responsive as we expected it to be.

We started our second trial with a considerable new input from the traffic generator: a waterfall diagram. This was 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.

We trained our deep learning models using the DEEPSIG RADIOML 2016.10A synthetic dataset produced by DeepSig Inc. As this dataset contains RF data generated in GNU Radio, its features are identical to those that may be extracted from our radios. The results demonstrate the capability of our deep learning method for feature extraction of RF data. The trained model can assess newly-generated data in the future and provides insight into suitable data extraction methods for GNU Radio.

Lastly, , we found that adding batch normalization layers after each convolutional layer can effectively make the loss converge faster and lead to a higher accuracy in the deep learning trials we executed above.

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 Commision has divided the airwaves 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 not a priority.

NI 2901 USRPs were required as part of the Air Force Research Laboratory's (AFRL) Beyond 5G challenge and were therefore unsubstitutable. The oscilloscope and signal generators, on the other hand, could be replaced with cheaper alternatives without altering functionality– they were 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 could also be replaced; the operating system, DragonOS, is freely available online and may be installed on 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. Modem producers like Qualcomm, however, do not appear to directly sell C-V2X hardware, making such an improvement difficult.

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.