Misbehavior Detection System for Cellular Vehicle to Everything Technologies

Michael Aliberti, Max Ellsworth, Jason Inirio, Sam Krasnoff, Julia Zeng, Yixiu Zhu

Executive Summary— 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.

Index Terms— Distributed Networks, Machine Learning, Network Protocols, Vehicle Operation

- *Michael Aliberti is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: mjali@bu.edu.*
- *Max Ellsworth is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: maxell@bu.edu.*
- *Jason Inirio is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: jasonini@bu.edu.*
- *Sam Krasnoff is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: krasnoff@bu.edu.*
- *Julia Zeng is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: zjulia@bu.edu.*
- *Yixiu Zhu is with the Department of Computer and Electrical Engineering, Boston University, Boston, MA 02215. E-mail: zhuyixiu@bu.edu.*

1 Project Overview

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.

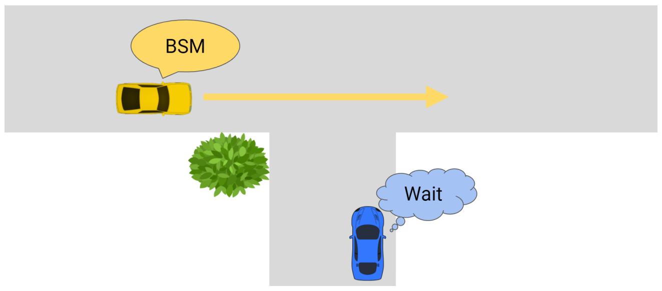

Fig. 1. Within the V2X protocol, cars broadcast Basic Safety Messages (BSMs) which convey location and velocity. This information facilitates inter-vehicle awareness without line of sight.

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.

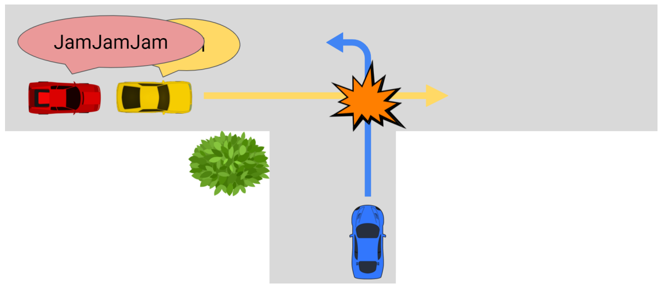

Fig. 2. When an attacker disrupts the transmission of basic safety messages, V2X provides limited information and collisions are far more likely to occur.

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, a monitoring system will be set up 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.

1.1 Problem, Purpose, and General Approach

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) demonstrate the ability to transmit and receive C-V2X messages, (2) visualize resource usage, (3) implement DoS attacks introduced in [2], and (4) 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 were tasked with creating a functional model. 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 data observed to files for later use. This data can then be visualized and used to train a machine learning algorithm in order to provide insights into V2X resource utilization and attack patterns.

1.2 Deliverables

The functional hardware implementation of V2X behavior detailed above serves as a deliverable in-and-of-itself, as it may be used independently to explore other practical dimensions of the protocol. A DoS attack in the form of a jammer has been modeled in hardware, with documentation of all code added upon a preexisting jammer for ease of replication. The final hardware-adjacent deliverable is a monitoring system to record data across all V2X frequency bands and store said data locally. The monitored data and the overall resource usage of the V2X frequency band can be monitored as the data is collected, which is accomplished in GNURadio.

At the very core of the project is the misbehavior detection system itself. This is a piece of software running on the monitor which uses data generated by the model to ascertain whether or not a specific snapshot of V2X traffic represents an attack. This takes the form of a machine learning algorithm which uses classification techniques to identify normal behavior versus jammed data.

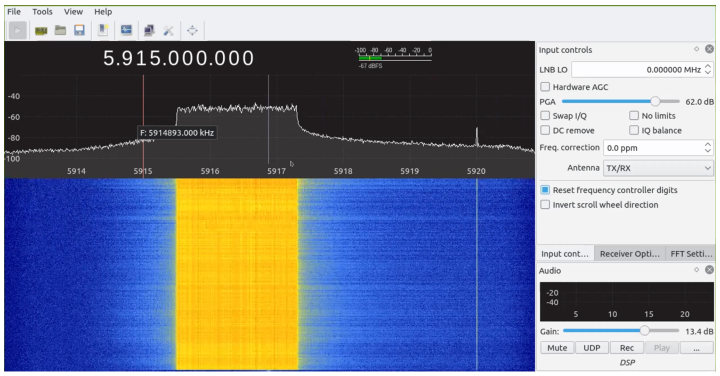

Fig. 3: A screencap of GQRX, displaying C-V2X packet transmission in the 5.915 GHz band

1.3 Current State of the Technology

The purpose of this project is to further the efficacy and security of the V2X standard, which is still in the earlier stages of development. Due to this, a lot of our early research centered on other simulators and verification tools for V2X communication. One such piece of technology is the SA8700A C-V2X Test Solution from Keysight Technologies. This machine has a built-in high-power RF signal generator that allows it to accurately send messages in the V2X frequency band. It also possesses a large amount of high-quality signal monitoring solutions that measure values such as frequency accuracy, in-band emissions, and adjacent channel leakage ratio. However, Keysight's device lacks the post-processing of data that this project provides.

Another related product is the Qualcomm 9150 C-V2X ASIC, which is mainly used for preliminary automotive applications, but again does not have any ability to detect an attack, much less differentiate from normal operating procedures. There is also CV2XinFIRE experiment, from the 5GinFIRE project. The experiment focused on comparing another high-frequency standard called ITS-G5 to CV2X in a variety of real-world scenarios. A large amount of transmission data was collected and statistically analyzed, but the experiment did not focus on security, nor did it use machine learning in its results.

There have been a handful of academic studies focused on security within C-V2X, such as An MEC-based DoS Attack Detection Mechanism for C-V2X Networks from Li et al. [1]. Their paper and experiment theorized a statistical approach to monitor malicious actors on a V2X network. Our project differs in that it utilizes classification-based machine learning from both normal and DoS corrupted data streams to warn of attacks. By using various machine learning approaches, we aim to streamline the process of detecting malicious attacks in real-time.

Additionally, there is no existing internal capability in terms of tools or previous implementation that our team is using as a jumping off point.

2 First Semester Summary

2.1 Concept Development

In the first semester, we focused on the research aspect of our project. Most of the research done has been through the developments of understanding C-V2X communication protocols and working with software defined radios (SDRs). Recently, we began to use signal analysis, software libraries such as srsRAN, and tools like iperf to better understand and use C-V2X. This allowed us to further our development and create foundations for our next objective: machine learning detection.

To build on a neural network for C-V2X and maintain an adequate level of accuracy - noise must be accounted for. Our motivation for this comes from radios developing large amounts of signals on a 5Ghz band, and thus creating noise. Because there is a massive amount of data every second, our model must be able to also withstand and process any signal correctly to be useful. Since SDRs have an immense range of signals that will cause noise to our model, we will need to utilize a separate algorithm or method to combat and preprocess our noisy data generated from the radios effectively.

Before developing a machine learning model and accounting for noise in the data, we needed to first realize what type of C-V2X packet layer we would need. With this in mind, we were able to begin implementing machine learning to detect misbehavior. Conceptually, detecting misbehavior attackers will utilize our signal analysis tools and methods to begin data collection. In the second semester, we aimed to look more into signal processing using the SDRs, and learn what packets and information C-V2X sends over the air. We decided early on to use a neural network such as PyTorch, since Python is the most commonly used language for signal analysis and data collection.

By studying this information, we were able to start planning a system and a general flow of data.

2.2 System Description

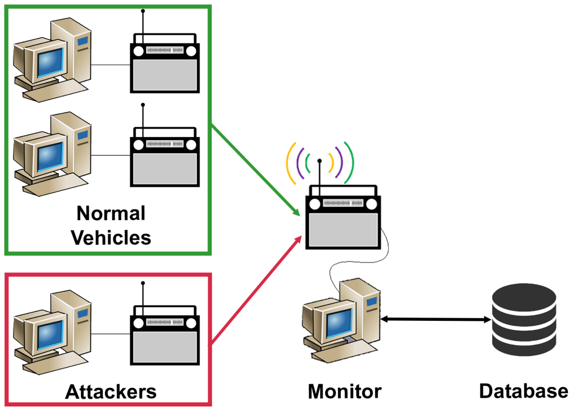

Fig. 4. The hardware setup used to model V2X behavior and attacks. The monitor will listen in on all V2X frequency blocks and save collected data to a file output. This data can then be visualized and fed into a machine learning algorithm to find attacks.

In the above figure, a few aspects should be considered. First, the monitor. This radio listens across the entire range of the V2X spectrum, recording activity every few ticks. All compiled information is sent to a PCAP file, which can then easily be parsed by a Python script. The attackers attempt to disrupt service of the "normal" vehicles, radios set to transmit simple Basic Safety Messages every few seconds. Other data points, such as the throughput are taken through occasional executions of the iperf command. As an additional visualization, we have GNU Radio to help us spot irregularities.

Only three radios were available to our team to model V2X communications. Thankfully, the way in which vehicles divide up the frequency band in V2X allows one radio to simulate multiple vehicles at a time.

2.3 First Semester Progress

In the first semester, our primary focus was on setting up the infrastructure for the radios, doing background research on the C-V2X protocol, and developing a design plan for our machine learning model. Over the course of two and a half months, the team met almost every Friday at 1 PM with Stefan Gvozdenovic, our graduate student advisor who also serves as a client, at our lab workbench. At these meetings, Stefan would provide us with hands-on instruction using the supporting hardware, i.e., an oscilloscope and two signal generators, necessary for synchronizing the radios and performing tests to verify that the radios were synchronized. Because radio synchronization is a prerequisite for virtually all cellular functionality, about a month was spent on setting up and verifying synchronization. This included troubleshooting issues with creating an over-the-air (OTA) srsRAN cellular client connection to an accompanying base station. While it was unclear what exactly was preventing an OTA connection from happening, a number of combined factors, like synchronization and host performance tuning errors (e.g., CPU cores not being in performance mode), likely contributed to these issues.

Once the cellular base station proof of concept was successful, we spent approximately two weeks performing network connectivity and bitrate tests. The first of these tests was a simple ICMP ping sent from the user end to the base station. This was verified as being successful by having Wireshark open on the base station and pinging the IP address of the base station from the cellular client. Once it was established that the basic network connection was established, a series of iperf tests were run. The purpose of these tests was twofold; first, in addition to demonstrating that the radios could transmit and receive meaningful amounts of traffic over-the-air (i.e., more than a ping), it also allowed us to observe how the frequency spectrum would operate under variable bitrate scenarios. Because iperf can ramp the amount of network traffic up or down depending on what flags are used, it was possible to observe how bitrate (in megabits per second) would affect bandwidth (usually in hundreds of kilohertz to megahertz).

In our first prototype test, our main deliverables were to show proof of an established OTA connection between two USRPs using srsRan and a third USRP that functioned as a monitor. The majority of tests passed with no issue. When executing ping and singly-connected iperf, in which three is only one connection to an endpoint, both the uplink and downlink tests exhibited the expected behaviors, with the UE and ENB issuing "connected" status messages, the USRP transmission and reception LEDs turning on, and both ends recording nonzero bitrates aligned in the direction of traffic. The monitor was able to see communications between the two radios in both packet form via Wireshark and as radio activity on dedicated frequencies via GQRX. The same can be said for multiply-connected iperf in the downlink connection, though the activity on GQRX exceeded the range of the expected frequency band. The only test that deviated from expectations was multiply-connected iperf, in which 20 parallel connections were created in the uplink direction, in which a disconnection during transmission occurred due to the flood of data from the UE. We suspect this is due to a buffer overflow on the base station end. Our setup also changed post-testing in reaction to feedback from Professor Hirsch, who is one of our professors for the Senior Design course we are taking in parallel with this project. The signal generators used to synchronize the UE and ENB previously used a square wave and a pulse with sine modulation. Following the test battery, the square wave was changed to a sine wave, and modulation was removed from both signal generators. After further testing, this has not appeared to have an impact on reliability or clarity of transmission. The disconnection error described above persists, but appears to be relatively rare and is not exclusive to the case of downlink communication. Fortunately, the volume of transmission demanded in multiply-connected iperf exceeds the demands of C-V2X, so this error should not occur in the final design.

2.4 Budget Estimate

No.	Project-related Device & Materials	Specs of Device & Material		Total cost
		Item number	Item price ($ per piece)	
1	Desktop with Dragon OS	2	Around 500	1000
2	NI 2901 USRPs (AFRL provided)	3	2010	6030
3	Keysight 33500B signal generators	2	1979	3958
4	LeCroy Wavesurfer 422 oscilloscope	1	5141	5141

TOTAL 16129

3 Second Semester Summary

To complete this project, we broke up our work into three parts. The first was to implement a frequency-hopping jammer in GNU Radio that would simulate an attacker. The second was to reverse engineer the srsRan source code to build a data extraction pipeline to generate IQ data. And finally, the third was to build a machine learning framework using a convolution neural net (CNN), TensorFlow, and deep learning using simulated data.

3.1 Jammer Implementation

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 [3]. The WPI jammer was written using gnuradio-companion version 3.7.11 and contains blocks not found in the latest iteration of gnuradio-companion, version 3.8.1.0. In particular, while version 3.7.11 has a standalone OFDM Mod block capable of generating OFDM signals, version 3.8.1.0 utilizes three separate blocks to generate that signal.

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 from 2.6775e9 < center_freq < 2.6825e9 to instead be 5.9165e9 < center_freq < 5.917e9. Additionally, the interval the jammer used was changed from 25 milliseconds to one second to reflect the interval in the C-V2X protocol by which the target reselects its resources and the jammer may need to hop frequencies .

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 three physical radios were configured to be in close-proximity with each other in order to minimize issues associated with variable gain, and we stacked our three NI USRP-2901 devices on top of each other. Early on in the project we discussed the possibility of setting up the radios on rolling carts in order to simulate them as being mounted on vehicles. While this was only briefly considered, it was eventually ruled out due to the need for the radios to be connected to a signal generator and, more significantly, the fact that they were Kensington-locked to our lab desk.

The jammer uses standard noise jamming to maximize ease of use. One trajectory that was considered and may be implemented in later iterations of the project is a pattern jammer, using similar bands but sending noise in intervals similar to V2X packets. The jammer was shifted to the 5.918 GHz range to match the V2X packets sent in the CV2X_traffic_generator. 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. Due to the complete randomness of this jammer, this is considered an “oblivious” jammer, as it does not actively search for occupied channels [2]. 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.

3.2 Data Extraction in srsRAN

As our primary goal is to distinguish jammed states from unjammed states, we honed in on 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 far more difficult and the data itself has limited applications.

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++. While READMEs and introductory documentation exist for both libraries, in-code comments and automated documents are almost entirely absent. Moreover, these libraries are built with ease of communication in mind, but data interfaces for extracting and analyzing metrics are not built-in. This meant that we not only had to implement our own methods for extracting data - we also had to find out exactly where to extract it from.

Our first target was IQ data, as we were assured that it existed in the code after enabling real time graphs plotting IQ data that ship with SrsRAN. By grepping for the headers on the graph and following lines of class inheritance, we were able to track down cc worker, which we assumed to be shorthand for carrier component worker. We realized that the complex numbers in this file were IQ data points, figured out how to print them, and, after a few different printing schemes, eventually matched the constellations on the natively generated graph. Printing these values as complex numbers, however, proved inefficient, so we created a function to encode IQ data in hexadecimal and printed out the data accordingly.

Next we tried to extract resource block allocation information. Our intent at the time was to detect jamming via discrepancies between resource blocks allocated by the transmitter and perceived resource block allocation on the receiver side. This approach began after seeing variables with the prefix "prb" in the traffic generator code, but we realized that PRBs, or physical resource blocks, exist on a layer below RBGs, or resource block groups, which is the layer at which allocation actually occurs.

The task of extracting the RBG data from the srsRAN source code was not straightforward due to insufficient documentation. Therefore, determining whether the data extracted corresponds to RGB blocks required guesswork and trial and error. The only seemingly relevant data type we could find was SrsRAN's rbgmask_t opaque type. By grepping the source files for the rbgmask keyword, we narrowed down the number of files to search through. Our next step was to determine which files were reached during runtime. Unfortunately, we could not print any indicators directly, as printing is disabled throughout large swaths of the SrsRAN codebase. To extract the data we inserted probes in functions of SrsRAN source files located in the srsenb/src and srsenb/hdr directories that printed the path of the function to a file as opposed to standard output. Using automation scripts to start the LTE network, we were able to see which files were being touched during transmission, and therefore where to add further probes to print the values of rbgmask variables. This output, however, was suspiciously static, consisting of either seventeen 0s or sixteen 0s and one 1. Since the output did not match what we anticipated, which are periodic and abundant vectors, we concluded that this was not the data we're looking for.

Due to the wider applicability of IQ data and the relatively limited usefulness of our extracted sparse resource block allocation vectors, we decided to proceed with IQ data for our machine learning model.

3.3 Machine Learning

We incorporated a series of python scripts that would help with automating and training machine learning models. We began with a simple CNN model adapted by using the MNIST database used by TensorFlow. The model's goal was to identify and relate the data between jammed and unjammed signals via resource pools. Although the data needed is still being developed, our first approach was to create a simulator that would randomly generate a binary map to represent the signals.

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

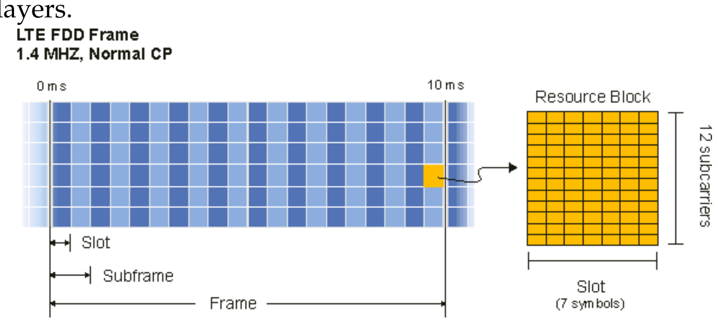

Fig. 5. A visualization of resource block allocation on the LTE physical layer. Each frame spans 10 ms and is subdivided into subframes each of which is capable of carrying at most one transmission. A transmission can use one or more resource blocks within a subframe. [6]

The following represents the final product of our simulator and how we thought of implementing it to match our criterias for training the simple 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.

Although the theory was simple, the output of the model followed a very probabilistic approach and resulted in a high accuracy of 67%, low accuracy of 50%, and a loss of 286.55%. This was unsatisfactory as the model is essentially flipping a coin on figuring out if the signal is jammed or not jammed. Furthermore, extracting real resource pool allocation data as opposed to simulated data from srsRAN in order to improve the model proved unfeasible, as SrsRAN allocates resources at a level above the granularity of resource blocks. 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.

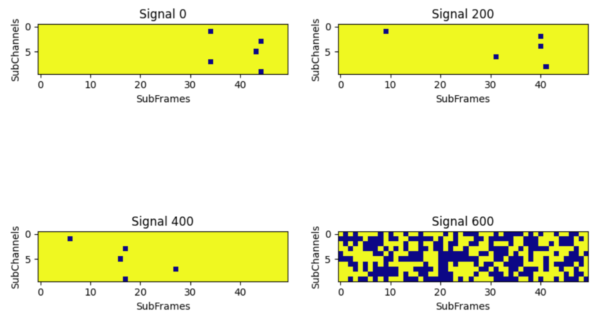

Fig. 6. Resource pool visualization created using simulated data for the CNN model. The resource pool represented in Signals 0 through 400 are not jammed. Signal 600 includes jamming.

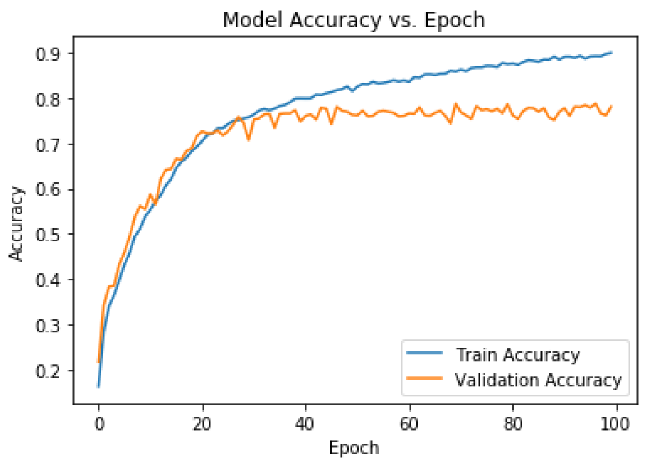

Fig. 7. Accuracy vs. Epoch plot for the Robust CNN deep learning model trained on the DEEPSIG's RADIOML 2016.10A dataset.

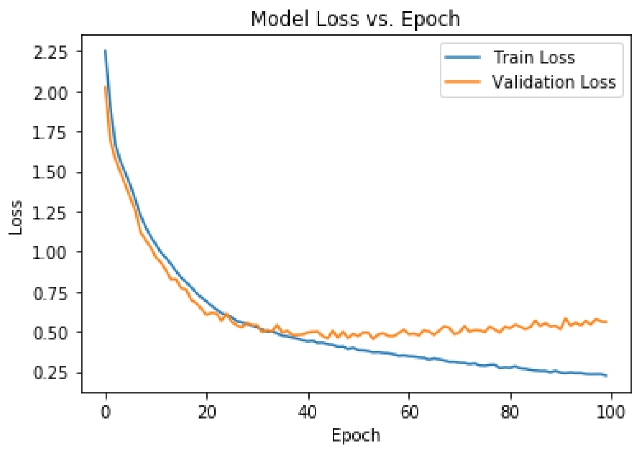

Fig. 8. Loss vs. Epoch plot for the Robust CNN deep learning model trained on the DEEPSIG's RADIOML 2016.10A dataset.

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. Either of these two would suffice, and should be able to detect features that we are not able to detect with a CNN model.

We firstly choose 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. Those trial models are separately listed for better clarity:

CNN-DL (Convolutional Neural Network):
- Data has plotting issues
- The layer construction is similar or identical to the approaches below, so we will delay testing this model for now

ResNet (Residual Network):
- Allows for tuning the model for our input dataset
- Currently looks good, but we are not sure if this model will suit our dataset

CLDNN (Convolutional Long Short Term Deep Neural Network):
- Currently has dataset import issues
- In progress of updating the Pickle data loading technique
- We are trying to load with a preprocessed dataset to make the model more responsive

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 [8]. Figures 7 and 8 show the results of training the Robust CNN model on the dataset with promising accuracy and loss values. 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.

However, those normalization layers are also expensive for our limited computing resources, therefore we plan to run them on BUSCC later if we want to improve our training accuracy.

4 Code

4.1 GitHub Organization

- **GitHub:** https://github.com/C-V2X-Senior-Design

All code referenced in this report can be viewed at the C-V2X-Senior-Design GitHub organization. It contains the TrackTasks repository which details the progress of the project through issues. It also contains the modSrsRan repository which tracks the changes made to the srsRan suite, i.e. file probing IQ and RBG data extraction.

The "CV2X_MachineLearning" folder contains our simple CNN model with the MNIST database. This pack of code includes files for data preprocessing, training and evaluation. For this simple CNN model, we utilized a 3 layer configuration (Flatten, Dense, Dense) for a simple demo purpose.

The "MachineLearning_Yixiu" folder contains three other deep learning models. The CLDNN Model, Resnet Model and Robust_CNN Model. Those models are still in the testing phase and we will select the best fitting model for our datasets once they are ready.

The "gnuradioRX-Jamming" folder contains our receiver code we used to do data collection and visualization. It also contains the modified jammer code written by the WPI authors. Included in the readme are links to the docker environment needed to get the jammer working and an explanation of what modifications were made to the jammer code.

5 Acknowledgements

The authors wish to thank Professor David Starobinski, Stefan Gvozdenovic, and Tony Lizza (Air Force Research Laboratory) for their continued support of this project.

6 References

[1] Li, Yang & Hou, Ronghui & Lui, King-Shan & Li, Hui. (2018), "An MEC-Based DoS Attack Detection Mechanism for C-V2X Networks," 1-6. 10.1109/GLOCOM.2018.8647323.

[2] Nataša Trkulja, David Starobinski, and Randall Berry, "Denial-of-Service Attacks on C-V2X Networks," AutoSec 2021, February 2021.

[3] Brown, YaYa Mao, and Cynthia Teng. Lte Frequency Hopping Jammer. : Worcester Polytechnic Institute, 2019.
https://digital.wpi.edu/concern/student_works/hm50tv580?locale=en

[4] R. Lindstedt, M. Kasparick, J. Pilz and S. Jaeckel, "An Open Software-Defined-Radio Platform for LTE-V2X And Beyond," 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), 2020, pp. 1-5, doi: 10.1109/VTC2020-Fall49728.2020.9348771.

[5] Twardokus, Geoff, "Intelligent Lower-Layer Denial-of-Service Attacks Against Cellular Vehicle-to- Everything" (2021). Thesis. Rochester Institute of Technology.

[6] "LTE Physical Layer Overview," *LTE physical layer overview*. [Online]. Available: https://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/lte/content/lte_overview.htm. [Accessed: 13-Mar-2022].

[7] A. Emam, M. Shalaby, H. A. A. Mansour, H. E. A. Bakr and M. A. Aboelazm, "An optimized Radio Modulation Classifier Using Deep Neural Network," 2020 12th International Conference on Electrical Engineering (ICEENG), 2020, pp. 175-180, doi: 10.1109/ICEENG45378.2020.9171707.

[8] DEEPSIG DATASET RADIOML 2016.10A, Arlington, VA, DeepSig Inc. , 2016. [Dataset]. Available: https://www.deepsig.ai/datasets. [Accessed: March 1, 2022]