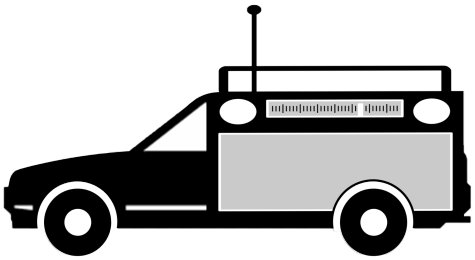
Boston University
Electrical & Computer Engineering
EC 464 Senior Design Project

Final Prototype Testing Plan

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: April 7th, 2022

Final Prototype Testing Plan

Table of Contents

1 Required Materials

1.1 Hardware

- 2 x DragonOS (Lubuntu) PCs
    - 8 Cores
    - 16 gb RAM
- 3 x NI 2901 USRP
- 1 x HackRF One
- 2 x Wall power adapters
- 5 x Vert 2450 Radio Antena
- 2 x Keysight 33500B Waveform Generator
    - 1 x 1 MHz, Pulse Wave
    - 1 x 10 MHz, Square Wave
- Considerable computing resources for ML (GPUs)

1.2 Software

- SrsRAN Library
    - SrsEPC
    - SrsENB
    - SrsUE
- GQRX
- Bash commands
    - ping
- GNU Radio
- Tensorflow
- Various python libraries

2 Setup

The lab setup consists of three NI 2901 USRPs, which are connected to a variety of devices for efficient, stable communication. Two of the USRPs model end-to-end communication, and the third monitors network traffic. Each of the communicating USRPs is plugged into the wall to get a consistent 1A, as propagating a signal in the 5.9 GHz range requires a high, consistent flow of power. All of the USRPs are connected to desktop computers running DragonOS, an Lubuntu operating system with pre-installed radio software such as GNURadio and SrsRAN. This USB connection enables us to use the SrsRAN library to configure the radios through software. From here, everything from the frequency band to the envelope itself can be digitally sent to the SDR. The base station and user end USRPs are connected to both a signal generator at 10 MHz and a signal generator at 1 Hz to synchronize communications. The base station SDR runs SrsENB and SrsEPC to serve as the network base for the other USRP device to connect to. The user end will run SrsUE, functioning as a communicating node to the base station. The third USRP will currently function as our data collecting node, monitoring the frequency on which the downlink and uplink are situated.

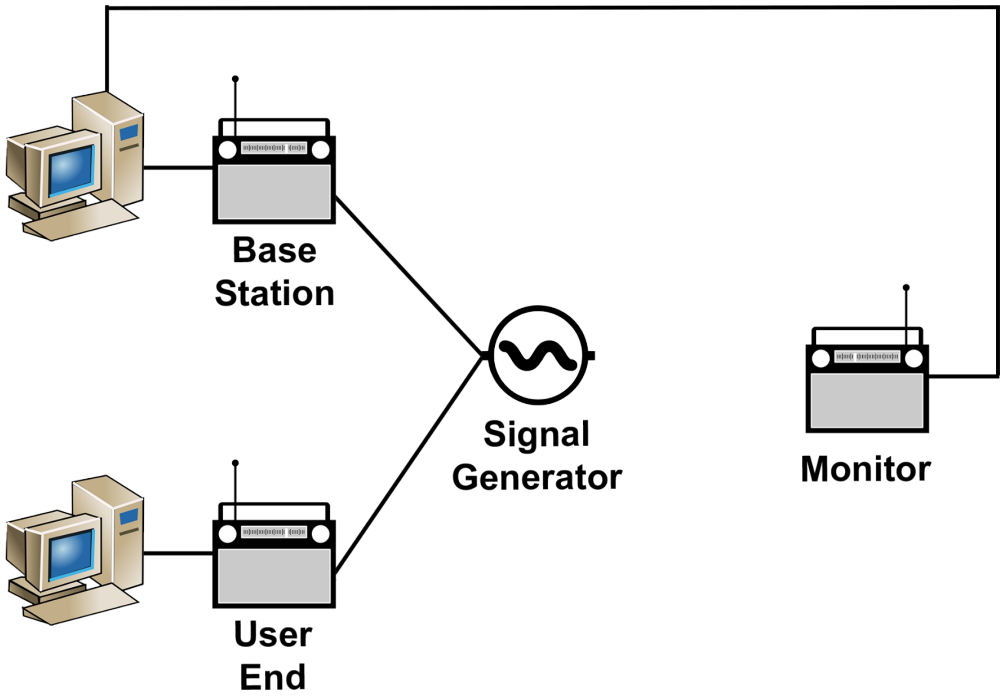


Figure 2.1 Illustration of our setup. Two PCs control communication between three SDRs, which are in-turn synchronized by a pair of signal generators.

3 Pre-testing Procedure

3.1 Basic Communication: Pre-testing Procedure

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. Connect the monitor USRP to the computer hosting the Base Station.
6. 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."
7. Open up a terminal on the base station; run `srsepc` and `srsenb`.
8. Open a terminal on the user end and run `srsue`.
9. Activate trace on the User End and Base Station.

3.2 C-V2X Traffic: Pre-testing Procedure

1. Perform steps 1-6 of the Basic Communication Pre-testing Procedure if not done already.
2. Open a terminal on the transmitter end and run `cv2x_traffic_generator.`
3. On the user end, run `pssch_ue` from the `modsrsRAN` directory.

3.3 MNIST Model: Pre-testing Procedure

1. Creating the environment:
    a. Activate virtual environment.
    b. Install required packages (pip install -r requirements.txt)
    c. Creating datasets. (createModel.py)

i. Run createModel.py by python createModel.py to create data to train and test the models from models.py.

d. Data preprocessing. (Preprocess.py)

i. Preprocessing library that matches the required matrices to match input shapes for TensorFlow models.

2. Creating models:

a. Launches simulator, trains and tests the data from simulator, and saves the newly tested model under the directory (models/)

b. Try sets of different models. (Models.py) has a different ML model for ease of training and testing our models.

3.4 DL Models: Pre-testing Procedure

1. Creating the environment:

a. Activate following environment and packages:

Python == 3.6.0

Tensorflow == 2.0.0

Tensorflow_gpu == 2.0.0

CUDA == 11.6

keras == 2.3.1

b. Preparing datasets. (download RADIOML 2016.10A dataset)

i. Run "extract_tarfile.py" once you have the dataset ready as a tar file. This step converts downloaded dataset into pickle format file (read only).

c. Data preprocessing. (revised_data_preprocessing.py)

i. Two files will be generated: 'new_model_SNR_test_samples' which includes all the test data for each SNR value, and 'combined_SNR_data' which contains the training set.

2. Picking models:

a. Try different models such as Robust_CNN_Model, Resnet Model or CLDNN Model.

4 Testing Procedure

4.1 Basic Communication: Testing Procedure

1. Verify uplink OTA communication between USRPS using the SrsRAN radio suite.
    a. Run ifconfig on the Base Station and record the ENB IP.
    b. Run ping `{ENB IP}` on the User End.
    c. Record whether or not a connection is made, PCAP file generation, uplink versus downlink bitrate, and radio connection light status.
2. Verify downlink OTA communication between USRPS
    a. Repeat steps in 1a-1c, but reverse the roles of the User End and Base Station
3. Perform the above tests with the jammer active to assess jammer effectiveness.

4.2 C-V2X Traffic: Testing Procedure

1. Open GQRX at 5.916 GHZ with bandwidth set to 10 MHz, and sample rate set to 8 million. Executed on NI USRP 2901
2. Enter Docker image for jammer, preparing jammer to be run with correct software versions.
3. Prepare `cv2x_traffic_generator`.
4. Execute `receive.grc` to show that minimal IQ points are populating the plot while neither the transmitter nor jammer are on.
5. Execute `cv2x_traffic_generator` and the jammer individually/in turn to generate respective `.pcap` files.
6. Execute `cv2x_traffic_generator` and jammer simultaneously.
7. Explain issues associated with getting the `pssch_ue` working and steps taken to remediate it.

4.3 CNN Model: Testing Procedure

Evaluating Models:

1. Uses data located in (data/) and models in (models/) to run an evaluation on all models.
2. Evaluation may include accuracy and loss relation graphs, matplotlib images to show resource pools being used, etc.

4.4 DL Models : Testing Procedure

Training Models:

Run 'revised_training.py' to start training on models specified.

Evaluating Models:

Evaluation may also include accuracy and loss relation graphs. Use "Plots.ipynb" to generate the plots and confusion matrix.

5 Troubleshooting Procedure

There are errors that can arise with connectivity when running the demo. A few are highlighted below.

1. "/usr/src/srsRAN/lib/src/phy/rf/rf_uhd_imp.cc:1335: USRP reported the following error: EnvironmentError: IOError: usb rx6 transfer status: LIBUSB_TRANSFER_ERROR" when running the base station command.
2. "Connect failed: Operation now in progress" when running iperf.
3. The/ping statistics are returning 0 for the bitrate (`brate`). This indicates that the UE (user equipment) and ENB (base station) are not connected. This can occur if the system is left idle for a period of time, e.g. no ping program running.
4. If not terminated through the GNURadio popup, the jammer in `top_block.py` will not run again, claiming that the radio is still in use.

To resolve these issues, try the following:

1. Teardown: Kill the terminal processes in the order of `srsue`, `srsenb`, `srsepc` and restart them in reverse order.
2. If step (a) does not work, power cycle the USRPs (turn them off and unplug the USB and power cables).

6 Measurable Criteria

6.1 Basic Communication: Measurable Criteria

1. The Base Station and User End USRPs should be able to communicate both uplink and downlink.
    a. When running ping
        i. Connection should begin between the ENB and UE.
        ii. Valid PCAP files should be generated.
        iii. There should be nonzero bitrates in the ENB and UE trace output.
        iv. Uplink bitrate should be higher for uplink communication and vice versa.
        v. Both the red and green LEDs should be on for both USRPs.
        vi. Nonzero IQ Data should be generated.

6.2 C-V2X Traffic: Measurable Criteria

1. When running ping and jammer
    a. Demonstrate that receiver is collecting data that is distinct from the protocol and/or jammer as opposed to random noise.
    b. Jammer IQ data primarily is concentrated in lower quadrature, while V2X packets are more evenly spaced, as well as clearly sent in identifiable bursts/packets. Jammer is sent in solid moving blocks due to the OFDM modulator sending constant, modulated noise.
    c. Demonstrate clear difference in packet decoding in Wireshark (different headers, data anomalies, etc) (ONLY IF WE CAN GET PCAP WORKING)
2. The receiver should be able to show the following four OTA radio states with IQ plots (`receiver.grc`):
    a. No data being sent from either the jammer or the C-V2X transmitter
    b. The standalone C-V2X transmitter (`cv2x_traffic_generator`)
    c. The jammer by itself (`./top_block.py`)
    d. The transmitter overlaid with the jammer
3. The .pcap packet captures from `pssch_ue` for the following states:
    a. No C-V2X traffic being sent

b. C-V2X traffic being sent (`cv2x_traffic_generator`)

c. Jammed C-V2X (`./top_block.py`)

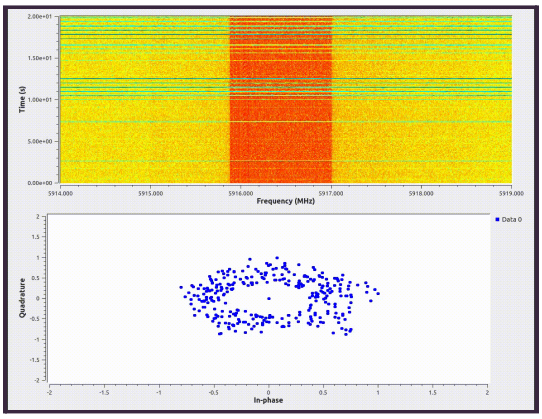


Figure 2: Standalone C-V2X transmitter

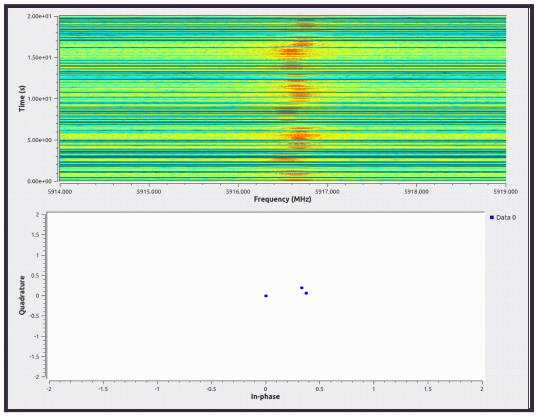


Figure 3: Standalone OFDM jammer

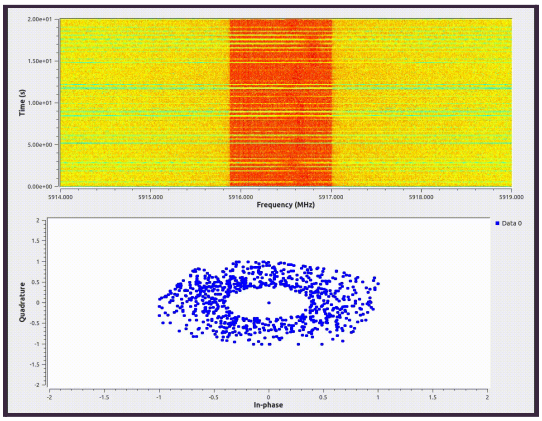


Figure 4: Jammer overlaid with C-V2X transmitter

6.3 CNN Model: Measurable Criteria

1. Dataset Generation:
    a. Dataset size matches the radio signal we collected.
2. Data Preprocessing:
    a. Check if input dataset shapes match the TensorFlow models.
    b. Level of Feature extraction.
3. Building the ML model:
    a. Load different ML models.
    b. Try the model with our own trial layers.
4. Model Training:
    a. Record the epochs and how the loss function varies over time.
5. Performance Evaluation:
    a. Monitor whether the training loss converged.
    b. Record the training accuracy (if converged.)

6.4 DL Models : Measurable Criteria

1. Dataset Loading:
    c. Dataset can be successfully loaded into the file format we specified.
6. Data Preprocessing:
    a. Check test samples are split as we wanted.
7. Training the ML model:
    a. Load different ML models.
    b. Try the model with different trial layers.
8. Model Training:
    a. Record the epochs and how the loss function varies over time.
9. Performance Evaluation:
    a. Monitor whether the training loss converged.
    b. Record the training accuracy (if converged.)

7 Score Sheet

7.1 Basic Communication and Jamming: Scoring

Test	Connected	IQ Data	PCAP	Bitrate Order	Red LED	Green LED
Ping, UL	Yes	Yes	Yes	Yes	Yes	Yes
Ping, DL	Yes	Yes	Yes	Yes	Yes	Yes

7.2 C-V2X Communication: Scoring

Test	Resource Blocks	Payload Validity (PCAP)
Unjammed	Yes	No
Jammed	Yes	No

7.3 CNN Model: Scoring

Test	Dataset Generation	Data Preprocessing	Building ML Model	Model Training	Performance Evaluation	Data & Result Visualization
Machine Learning: Custom + SciKit Algorithm	Yes	Yes	Yes	Yes	49% Accuracy	Yes

7.4 DL Models: Scoring

Test	Dataset Loading	Data Preprocessing	Model Training	Performance Evaluation	Data & Result Visualization
Robust-CNN Model	Yes	Yes	Yes	87% Accuracy	Yes
ResNet Model	Yes	Yes	Yes	81% Accuracy	Not Yet
CLDNN Model	Yes	Yes	Yes	84% Accuracy	Not Yet

8 Conclusions

8.1 Basic Communication and Jamming

We were able to demonstrate a couple of things to show good progress towards our final project. The first of which is successful over-the-air reception of C-V2X traffic in GNU Radio and successful jamming as indicated by the decreased power levels in the IQ plot. Another significant development is the generation of PCAP files on the receiver end (pssch_ue), which demonstrates successful debugging of the SrsRan code and discovery of potential causes of the decoding issue, i.e. GPS synchronization, duty cycle, power levels, pps scale. Using ping tests and receiver python script in GNU Radio, we are able to observe that traffic is being transmitted, but it cannot be decoded because of a checksum error that is likely due to synchronization issues. We are at a standstill with the resource block extraction until we can resolve the decoding issue. Our next steps will be to try to resolve the synchronization issue with an OctoClock distribution module.

8.2 Machine Learning

We were also able to demonstrate a proof-of-concept for our machine learning solution: we successfully deployed our models with two methods of data input.

For resource block data input, we have self-simulated data into a CNN model for around 50% of accuracy. To be specific, we currently have the resource block allocation algorithm to generate resource block data within the C-V2X traffic generator and we are still working on troubleshooting the software suite to see if we can further improve our training efficiency as well as eliminating the training loss.

For IQ data input, we have labeled IQ data from DeepSig Company (also from a radio research lab in UCSD) into CNN, CLDNN and ResNet Models. With the labeled dataset, we are able to show a decent accuracy around 80%. Due to the time constraints, we won't be able to label our own IQ dataset, so we may further make more trials on improving the layer configurations on three of the models shown above.