Boston University
Electrical & Computer Engineering
EC 464 Senior Design Project

Second Prototype Testing Report

Over-the-Air C-V2X Communications on Software Defined Radios

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

Required Materials

Hardware:

- 3 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)

Software:

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

Set Up

At the core of this project are three NI 2901 USRPs, which are connected to a variety of devices for efficient, stable communication. Two of the USRPs will model end-to-end communication, and the third will monitor 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 our desktop computers running DragonOS, an Lubuntu installation with pre-installed radio software. 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 will be connected to both a signal generator at 10 MHz and a signal generator at 1 Hz to synchronize communications. The base station SDR will run SrsENB and SrsEPC to serve as the virtual network base for the other USRP devices 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.

Pre-testing Setup 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. Open up a terminal on the base station; run `srsepc` and `srsenb`.
6. Open a terminal on the user end and run `srsue`.
7. Activate the trace on the User End and Base Station.

8. Connect the monitor USRP to the computer hosting the Base Station.
9. 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.”

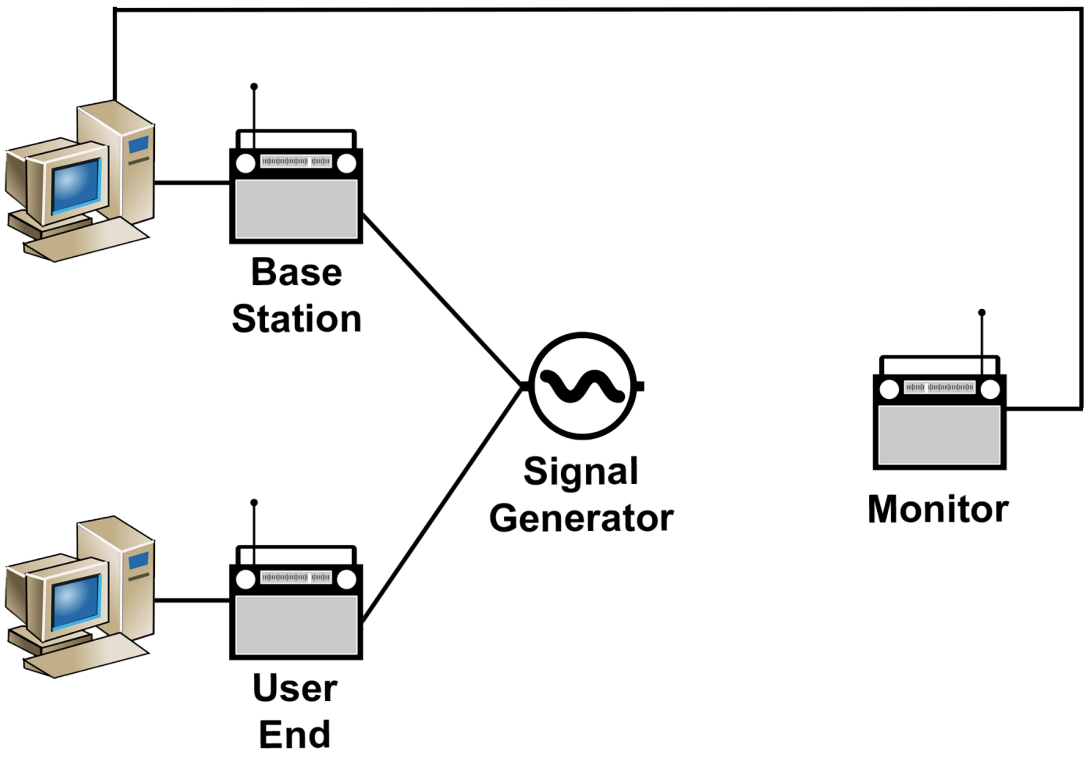

Figure 1: Illustration of Setup

Testing Procedure

1.
    a. Open GQRX at 5.916 GHZ with bandwidth set to 10 MHz, and sample rate set to 8 million. Executed on HackRF One.
    b. Enter Docker image for jammer, preparing jammer to be run with correct software versions
    c. Prepare `cv2x_traffic_generator` with gain set to 200.
    d. Execute `receive.grc` to show that minimal IQ points are populating the plot while neither the transmitter nor jammer are on.
    e. Execute `cv2x_traffic_generator` and the jammer individually/in turn.
    f. Execute `cv2x_traffic_generator` and jammer simultaneously.

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).

Measurable Criteria

Basic Communication and Jamming:

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. There should be nonzero bitrates in the ENB and UE trace output.
        iii. Uplink bitrate should be higher for uplink communication and vice versa.
        iv. Both the red and green LEDs should be on for both USRPs.
        v. Nonzero IQ Data should be generated.
    b. When running ping and jammer

i. Demonstrate that receiver is collecting data that is distinct from the protocol and/or jammer as opposed to random noise.
ii. 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.

Miscellaneous:

1. Probes should generate data providing insight into SrsRAN functionality.
    a. Positional probe `probes/rgbmask_t.txt` should contain 1+ filenames specifying files reached which contain `rgbmask_t` type instantiations.
    b. Value probe `probes/sched_grid.txt` should contain human readable arrays of 1s and 0s specifying resource block usage.
    c. Rebuild Process should be automated for ease of adding probes.
2. Automate LTE connection with shell script.

Machine Learning:

Set up for MNIST model method:

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.

Note: createModel.py uses the models located in models.py. To create a custom ML model, create a class in models.py.

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.

Testing for MNIST model method:

Evaluating Models:

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

Measurable Criteria for MNIST model method:

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.)

Score Sheet

Basic Communication and Jamming:

Test	Connected	IQ Data	Nonzero Trace	Bitrate Order	Red LED	Green LED
Ping (UL), Unjammed	Yes	Yes	Yes	Yes	Yes	Yes
Ping (UL), Jammed	Yes	Yes	Yes	Yes	Yes	Yes
Ping (DL), Unjammed	Yes	Yes	Yes	Yes	Yes	Yes
Ping (DL), Jammed	Yes	Yes	Yes	Yes	Yes	Yes

Machine Learning:

Test	Dataset Generation	Data Preprocessing	Building ML Model	Model Training	Performance Evaluation	Data & Result Visualization
MNIST Model	100,000	Yes	Yes	Yes	49.88% accuracy	Yes

Conclusion

We were able to demonstrate successful jamming of the C-V2X traffic generator using the WPI jammer from identifiable bursts of packets moving across the C-V2X frequency spectrum. We were able to confirm that changing the sleep parameter in the WPI jammer modified the interval at which the jammer hopped frequencies given the change in packet size when doing so. We were able to successfully demonstrate automation of probing scripts for providing insight into SrsRAN functionality. Furthermore, we have demonstrated a good baseline for our machine learning methods with CNN based on the MNIST dataset with our evaluator.py script which built and ran the already trained model on the simulated dataset. It yielded an accuracy of 50% which was unsatisfactory, but the difficulty of simulating a high-fidelity dataset and the lack of documentation provided for extracting resource block data

from SrsRan provide motivation for using a deep learning model for feature extraction on IQ data. Our goals moving forward are two-fold, (1) to get the C-V2X receiver working so that we can extract resource block data from there, (2) extract IQ data from SrsRan at the packet-level to use with a deep learning model trained on a third-party dataset by DeepSig.