**Terminal**

The way srsRAN works is you type the following into the terminal

sudo [EXECUTABLE] [CONFIGURATIONS]

For our experiments, we need to run the executable program found in the build files. This will run the modified source code on the computer and not run code with default values. For me, this is what it looked like

[EXECUTABLE] = ~/srsRAN/build/srsue/src/srsue

Configuration can be changed in the config files. You can also change one specific variable if you do --(variable you want to change) in the terminal. For DPASS, I would modify the config files. Modifying the config files is pretty easy. All it requires is using a text editor and it's pretty self explanatory.

For our experiments it looked like

[CONFIGURATIONS] = ~/.config/srsran/ue.conf

So all together it looked like this,

sudo ~/srsRAN/build/srsue/src/srsue ~/.config/srsran/ue.conf

I think that you will only need to type out this long command in the terminal when running the UE. I believe that the only changes we made in the source code affects the UE. The python script should do all of the terminal commands for UE, so it shouldn’t be anything crazy. You will probably need to modify it though to have the correct path for your computer. For everything else you can just run

sudo srsepc ~/.config/srsran/epc.conf

sudo srsenb ~/.config/srsran/enb.conf

When running the UE with modified source code, make sure that you type in srsue where it is needed.

When running the eNodeB with modified source code, make sure that you type in srsenb where it is needed

When running the EPC with modified source code, make sure that you type in srsepc where it is needed.

**Operation**

Make sure that you run on separate computers for the srsenb/srsepc and srsue. Also srsRAN only supports the b200 minis.

First thing that you want to run is srsepc. This is pretty straightforward and should work every time you run it. After that you want to run the srsenb. It will create an interface with the EPC and set itself up.

When it says “eNodeB started”, then you are good to start the srsue on your second computer. As I said earlier, running the python script will also start the srsue. You will know when the two devices are connected when at the bottom it prints “Software Radio Systems” or something along that line in the UE terminal. If you have a bunch of other random stuff pop up on the terminal, you are most likely disconnected from the eNodeB. srsRAN, after a few seconds, will try to reconnect but I have found that it is rarely successful at reconnecting. I usually just turn off the UE and try again.

When the two devices are connected, the IP address for the eNodeB is 172.16.0.1. The IP address for the UE is usually 172.16.0.2, but sometimes it isn’t. You can print traces on either terminal by typing “t”. There is a lot of info that will pop up (most of it I don’t know), but if you see non-zero numbers on the mcs column, then you know communication between the two devices is happening. Make sure that when you run the python code, that you use sudo or else it will not correctly terminate the UE. That’s pretty much it. After that it is pretty much up to DPASS what happens next with the LTE network.

**Code**

The only code that we changed from the srsRAN source code is found in lib>src>radio>radio.cc

After line 58 we included this code:

// Create a new publishing socket

zmq\_ctx = zmq\_ctx\_new();

zmq\_sock = zmq\_socket(zmq\_ctx, ZMQ\_PUB);

if (!zmq\_sock) {

fprintf(stderr, "Error: creating transmitter socket\n");

}

// The transmitter starts first and creates the socket

if (zmq\_bind(zmq\_sock, "tcp://\*:2003")) {

fprintf(stderr, "Error: connecting transmitter socket: %s\n", zmq\_strerror(zmq\_errno()));

}

After line 306 we included this code:

// Transmits RSSI values to zmq socket

int radio::tx\_zmq(cf\_t\* buff, uint32\_t buff\_length)

{

// Make a message zmq socket

zmq\_msg\_t envelope;

const int rmi = zmq\_msg\_init\_size(&envelope, sizeof(int16\_t));

// Check that the socket is already empty

uint8\_t dummy;

zmq\_recv(zmq\_sock, &dummy, sizeof(dummy), 0);

// Will change the IQ values to be RSSI values and cast them to the correct type for the socket

rssi\_buff[0] = (int16\_t)srsran\_convert\_power\_to\_dB(srsran\_vec\_avg\_power\_cf(buff, buff\_length));

memcpy((void\*)((char\*)zmq\_msg\_data(&envelope)), rssi\_buff, sizeof(int16\_t));

// Sends the RSSI values to our socket

const size\_t rs = zmq\_msg\_send(&envelope, zmq\_sock, 0);

if (rs != sizeof(int16\_t)) {

printf("RS Value: %ld\n", rs);

printf("ERROR: ZeroMQ error occurred during zmq\_msg\_send(): %s\n", zmq\_strerror(errno));

zmq\_msg\_close(&envelope);

return -1;

}

return 0;

}

After line 449 we included this code:

// Send the values from the buffer to the socket before it passes through the automatic gain controller

tx\_zmq((cf\_t\*)radio\_buffers[0], nof\_samples);