**ECE 430**

**Laura and Joshua**

**Radio Wars E3**

**Introduction**:

For Radio wars #3 we were tasked with creating a radio that can avoid a randomly dynamic hopping jammer while receiving a high transfer within a limited 10 MHz spectrum bandwidth. Our Team worked in a sdr-class-radio.py file to cycle through a list of frequency within our required Frequency Band. We also created a mechanism so that the change in frequencies while the hopping algorithm was being implemented would synchronize perfectly between Tx and Rx.

**Summary:**

The logistics for the competition was that there would be a single trial in the competition where the Jammer would jam a 2.5 MHz spectrum band (of the 10 MHz total band) randomly. We came up with the idea of creating a frequency hopper that would hop automatically in a randomized manner with 10% bandwidth of our total 10 Mhz bandwidth (that is 1 MHz band size).

Using this configuration in the pre-competition phase, we got an average data rate of **557 Kbits/sec**. However, in the competition, our configuration performed very poor due to unknown reasons.

**Procedure:**

We first needed to implement frequency hopping in the sdr-class-radio.py file.

* For this, the first thing we did was to create a common seed for both Tx and Rx using the clock.

random.seed(round(time.time(),-5))

The round(time.time(),-5) function basically rounded off the time.time() value to the nearest date. Since we are running both Tx and Rx on the same day (even though it is apart for a few seconds), this way, we ensure that the seed variable is same for both Tx and Rx.

* We next created a shell variable called the center\_freq so as to avoid the dilemma of adjusting/changing all value of the band whenever the center frequency of use in the radio were implemented. This way, all we needed was to change the variable value, and the bands got updated automatically.

Graphical user interface, text

Description automatically generated

* Then we equally separated range of frequencies with numpy.arrange. After creation, we adjusted the bands starting and ending frequencies. We then made a mechanism for selecting the band by creating a random variable (using the seed assigned previously which was same for the Tx and Rx) indexed through 0-9.
* Synchronization: For this, we put our whole code into a while True: loop (as shown in the image below).

Text

Description automatically generated

We know that when we start the Tx and Rx radios, there would be a delay of 1-2 secs. So, the main goal of synchronization was that we start the radios of Tx and Rx at the exact same time. For this, we made a mechanism where the radios would start at the moment the current time seconds reached either 10, 20, 30, 40, 50 or 60 seconds (or in other words, the remainder of the current seconds when divided by 10 would give a zero). This mechanism was a good idea as the radios won’t take a long time to boot up, as the maximum waiting time would only be 10 seconds.

For this reason in the screenshot, we can see the code if seconds%10==0.

A screenshot of this synchronization is shown below. We can see that the Tx and Rx is only off by a few milliseconds (10 ms).

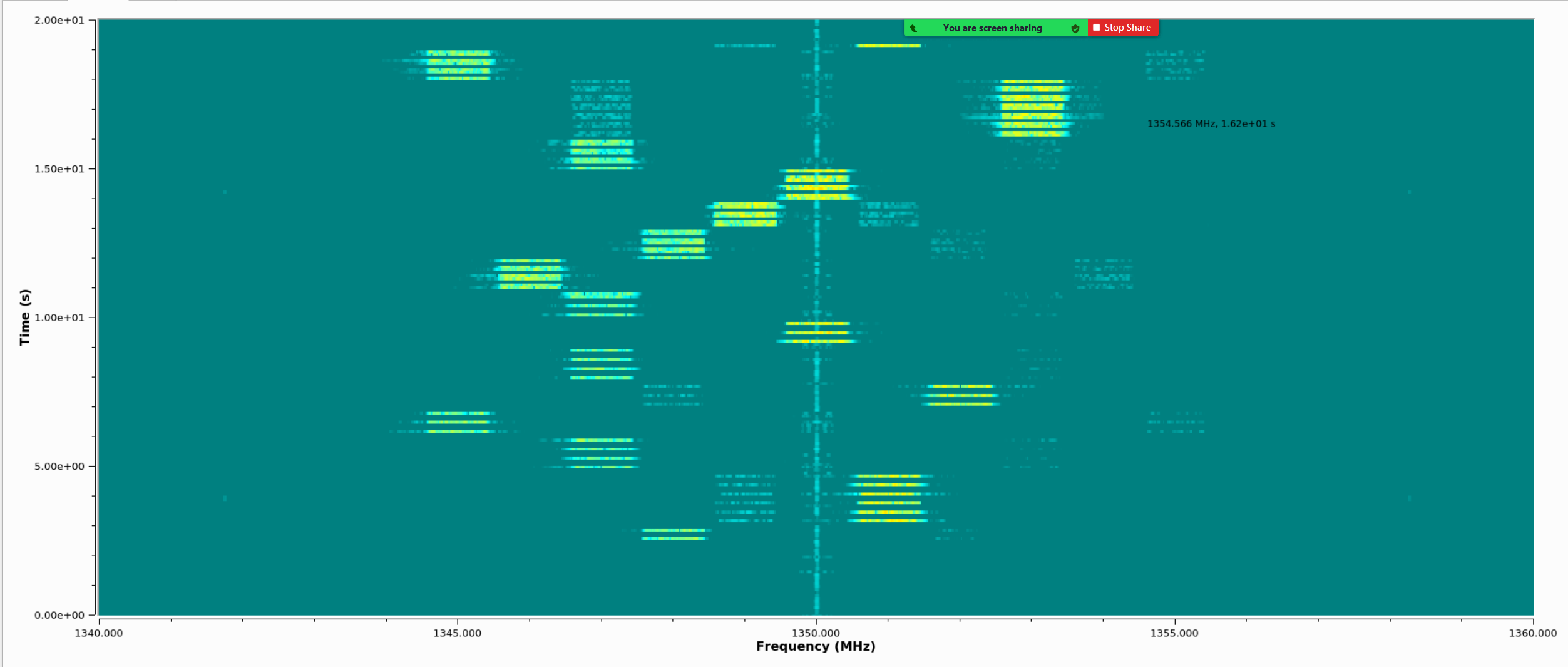
Graphical user interface, application

Description automatically generated

In this screenshot, the print(self.frequency) function is running. On the left, it is the Tx, and on the right, the Rx. It can be seen that in the first allocation, they are apart by 1 ms and in the second, by 2 ms. This is really good synchronization.

**Results:**

**Before the RadioWars (testing phase – without jammer)**



**Fig:** Frequency hopping transmission observed from a 3rd node using uhd\_fft

Table

Description automatically generated

**Fig:** Results when using **qam8**

Graphical user interface, text, table

Description automatically generated

**Fig:** Results when using **qam16**

Text, table

Description automatically generated

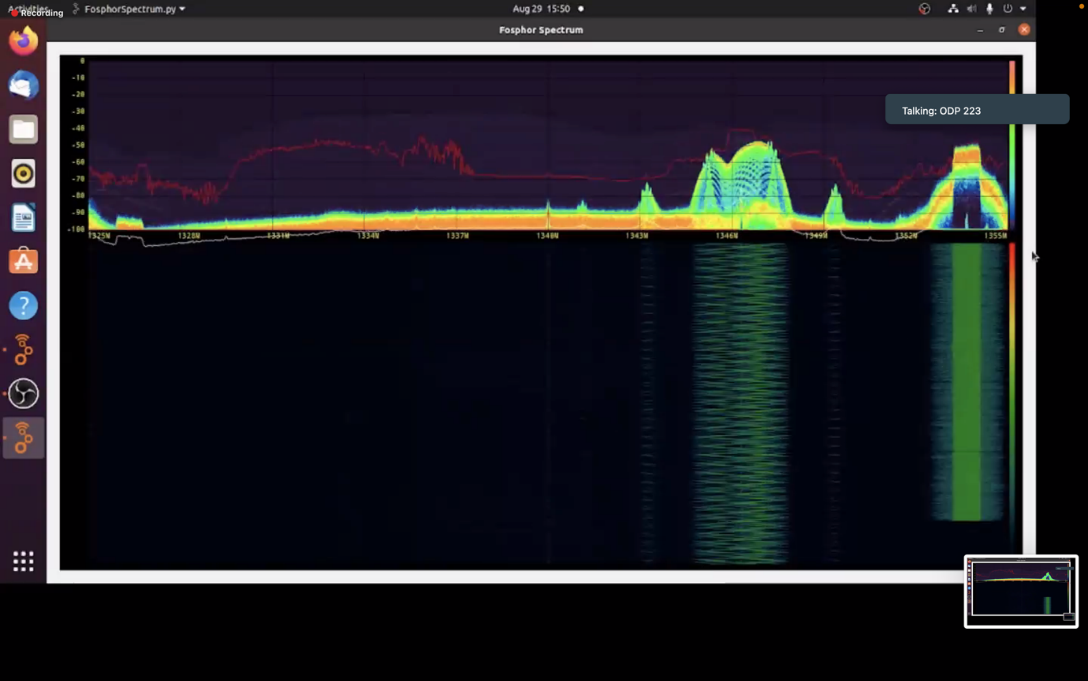
**Fig:** Results when using **qam32** *(the data transfer is not stable due to higher modulation)*

**During RadioWars Session**

A screenshot of a computer

Description automatically generated with medium confidence

**Fig:** Transmission with no jammer

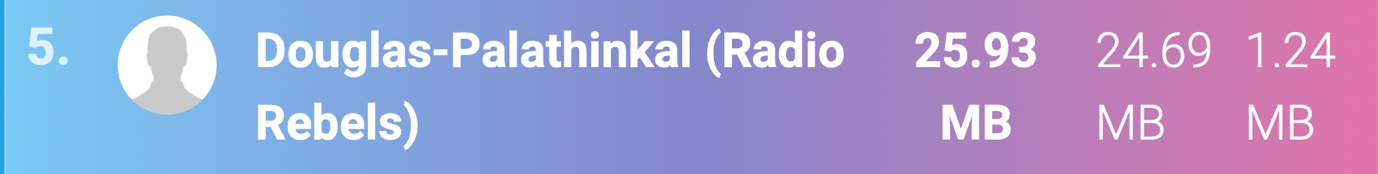


**Fig:** Transmission with jammer (but not being overlapped)

Chart

Description automatically generated

**Fig:** Transmission being interrupted by jammer (we can see the previously used band’s shadow)



**Fig:** Radio wars submission