

ECE430.
Laura and Joshua
Radio Wars Ep 2

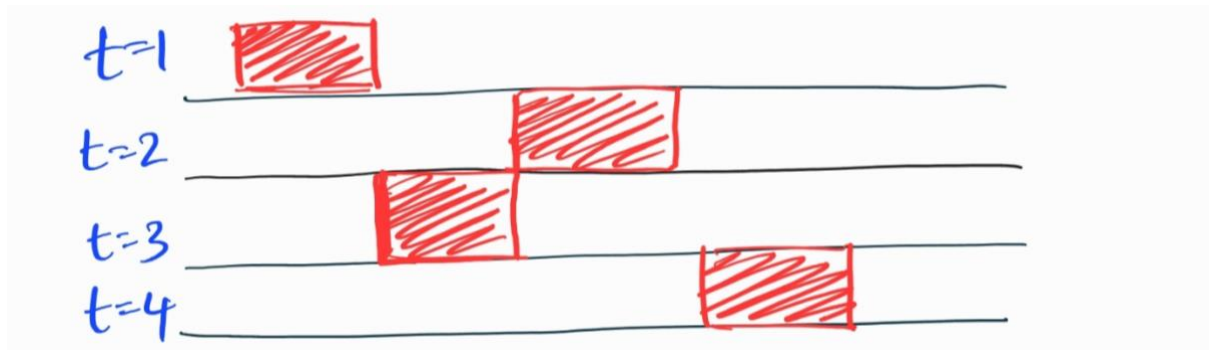


Figure 1: frequency hopping procedure explained

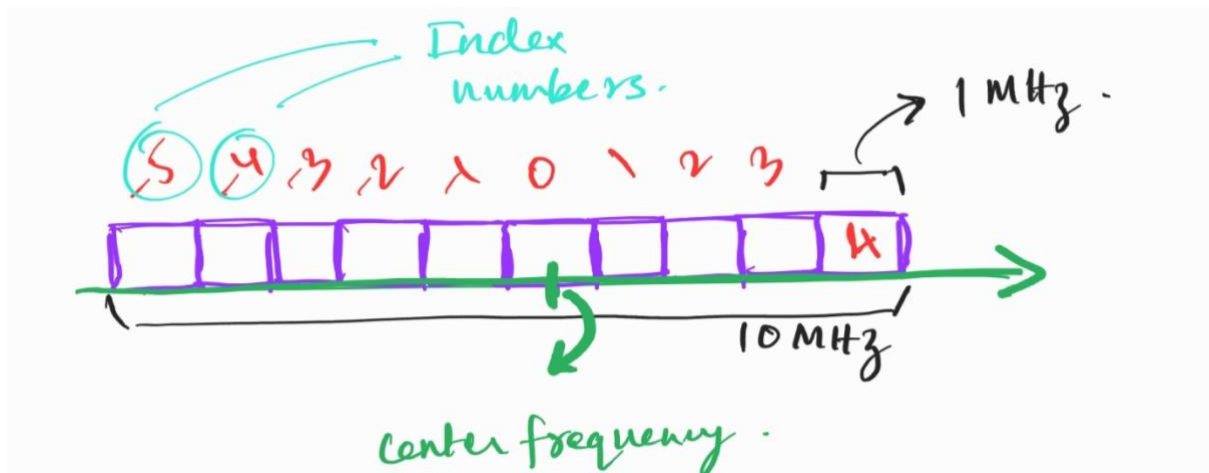


Figure 2: frequency hopping design used for Radio Wars

Our Implementation:

- i. Initialize center frequency
- ii. Create 10 slots (-5 to 4)
- iii. Make the center frequency adjust itself using the generated slots
- iv. Select one random carrier frequency each time and then repeat this procedure to get a unique carrier frequency each time (after a 1 second wait period).

This producer allows for our carrier frequency to be constantly changing; therefore, reducing the probability of our transmission being blocked by a jammer.

Seed used: `random.seed(round(time.time()),-5))`

This is because, as seeds both at the transmitter and receiver needs to be same for synchronization between the frequency hopping bands, we have rounded the `time.time()` so that we get the date, which is bound to be same at both the transmitter and receiver.

```

return sched

async def frequencyHop(self):
    try:
        while True:
            # Uncomment to set frequency to 1.31 GHz every second. Please do
            # something more intelligent!
            #self.frequency = 1.34e9

            center_freq = 1.34e9
            slots = np.arange(-5,5)
            freq_slots = (slots*1e6) + center_freq
            self.frequency = freq_slots[random.randint(0,9)]
            print(self.frequency)

            # Sleep for 1 second
            await asyncio.sleep(1)
        except CancelledError:
            pass

main():

```

Figure 3: frequency hopping function

```

jrp3940@r1001:~/gnu_files/lab5$ dragonradio sdr<-class-radio tx.py -i 2 -m qm16 --guard-size 0.001
Linux: GNU C++ version 9.4.0; Boost_107100; UHD_003.009.005-3-g51e5fd1

/usr/local/dragonradio/venv/lib/python3.8/site-packages/numpy/core/getlimits.py:499: UserWarning: The value of the smallest
subnormal for <class 'numpy.float64'> type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/dragonradio/venv/lib/python3.8/site-packages/numpy/core/getlimits.py:499: UserWarning: The value of the smallest
subnormal for <class 'numpy.float64'> type is zero.

jrp3940@r1002:~/gnu_files/lab5$ dragonradio sdr<-class-radio rx.py -i 1 -m qm16 --guard-size 0.001
Linux: GNU C++ version 9.4.0; Boost_107100; UHD_003.009.005-3-g51e5fd1

/usr/local/dragonradio/venv/lib/python3.8/site-packages/numpy/core/getlimits.py:499: UserWarning: The value of the smallest
subnormal for <class 'numpy.float64'> type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/dragonradio/venv/lib/python3.8/site-packages/numpy/core/getlimits.py:499: UserWarning: The value of the smallest
subnormal for <class 'numpy.float64'> type is zero.

```

Figure 4: Tx and Rx commands

```

jrp3940@r1001:~$ iperf -s -p 5001 -i 10 -t 10 -b 200M -t 10
Client connecting to 10.10.10.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 50000.00 us (kcalan adjust)
UDP buffer size: 47.7 Mbyte (default)
[ 3] local 10.10.10.2 port 3544 connected with 10.10.10.1 port 5001
[ 0] 0.0-1.0 sec 27.3 Kbytes 221 Kbits/sec
[ 1] 1.0-2.0 sec 24.4 Kbytes 200 Kbits/sec
[ 2] 2.0-3.0 sec 24.4 Kbytes 200 Kbits/sec
[ 3] 3.0-4.0 sec 24.4 Kbytes 200 Kbits/sec
[ 4] 4.0-5.0 sec 24.4 Kbytes 200 Kbits/sec
[ 5] 5.0-6.0 sec 24.4 Kbytes 200 Kbits/sec
[ 6] 6.0-7.0 sec 24.4 Kbytes 200 Kbits/sec
[ 7] 7.0-8.0 sec 24.4 Kbytes 200 Kbits/sec
[ 8] 8.0-9.0 sec 24.4 Kbytes 200 Kbits/sec
[ 9] 9.0-10.0 sec 24.4 Kbytes 200 Kbits/sec
WARNING: did not receive ack of last datagram after 10 tries.
[ 0] 0.0-1.0 sec 245 Kbytes 200 Kbits/sec
[ 1] Sent 271 datagrams
jrp3940@r1001:~$

jrp3940@r1002:~$ iperf -s -p 5001 -i 10 -t 10 -b 200M -t 10
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 47.7 Mbyte (default)
[ 3] local 10.10.10.1 port 5001 connected with 10.10.10.2 port 3544
[ 0] 0.0-1.0 sec 15.8 Kbytes 129 Kbits/sec 10.139 ms 0/ 10 (0%)
[ 1] 1.0-2.0 sec 14.4 Kbytes 118 Kbits/sec 29.138 ms 0/ 10 (0%)
[ 2] 2.0-3.0 sec 14.4 Kbytes 118 Kbits/sec 34.288 ms 0/ 10 (0%)
[ 3] 3.0-4.0 sec 14.4 Kbytes 118 Kbits/sec 36.981 ms 0/ 10 (0%)
[ 4] 4.0-5.0 sec 14.4 Kbytes 118 Kbits/sec 38.424 ms 0/ 10 (0%)
[ 5] 5.0-6.0 sec 14.4 Kbytes 118 Kbits/sec 39.199 ms 0/ 10 (0%)
[ 6] 6.0-7.0 sec 14.4 Kbytes 118 Kbits/sec 39.263 ms 0/ 10 (0%)
[ 7] 7.0-8.0 sec 14.4 Kbytes 118 Kbits/sec 39.386 ms 0/ 10 (0%)
[ 8] 8.0-9.0 sec 14.4 Kbytes 118 Kbits/sec 39.342 ms 0/ 10 (0%)
[ 9] 9.0-10.0 sec 14.4 Kbytes 118 Kbits/sec 39.346 ms 0/ 10 (0%)
[ 0] 10.0-11.0 sec 14.4 Kbytes 118 Kbits/sec 39.368 ms 0/ 10 (0%)
[ 1] 11.0-12.0 sec 14.4 Kbytes 118 Kbits/sec 39.345 ms 0/ 10 (0%)
[ 2] 12.0-13.0 sec 14.4 Kbytes 118 Kbits/sec 39.038 ms 0/ 10 (0%)

```

Figure 5: Tx and Rx are communicating

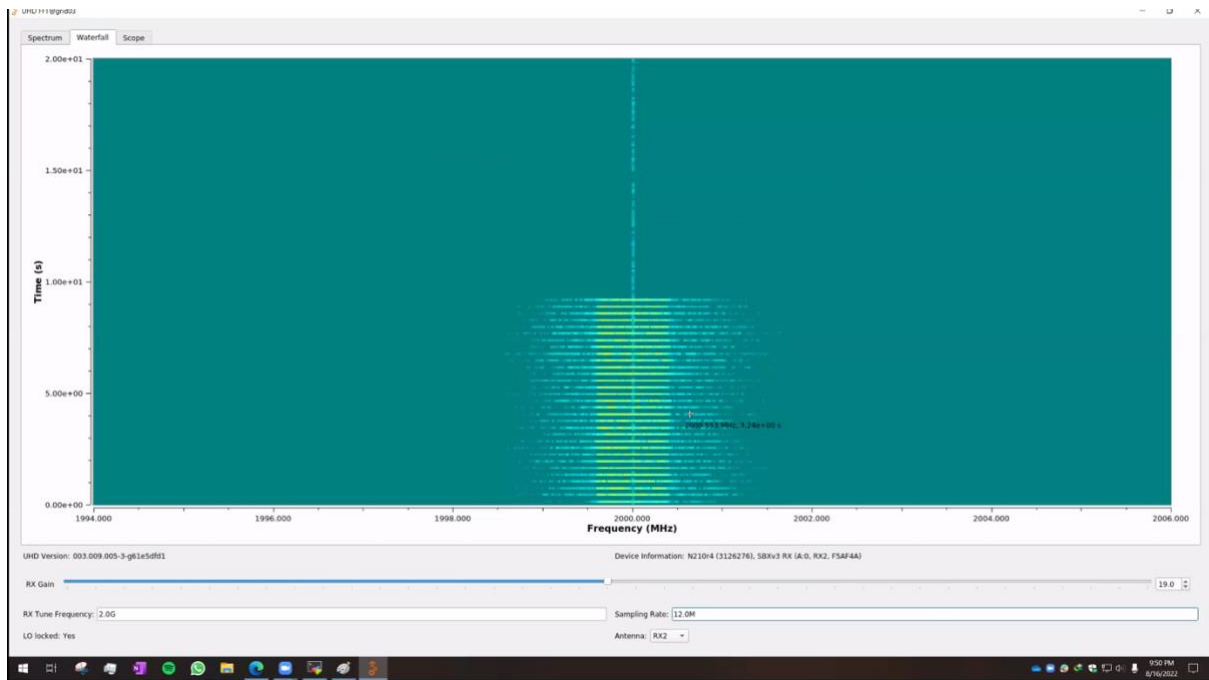


Figure 6: Transmission without frequency hopping enabled

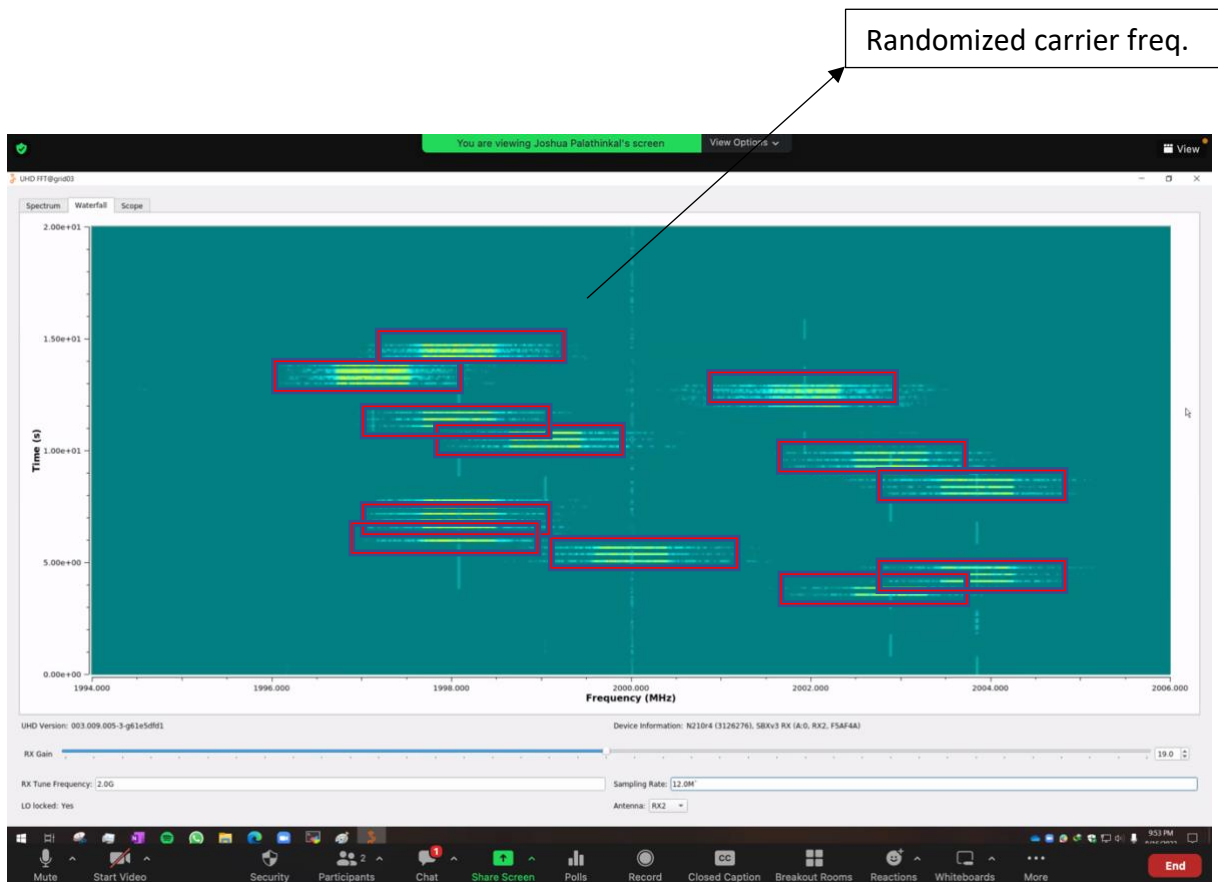


Figure 7: transmission with frequency hopping enabled

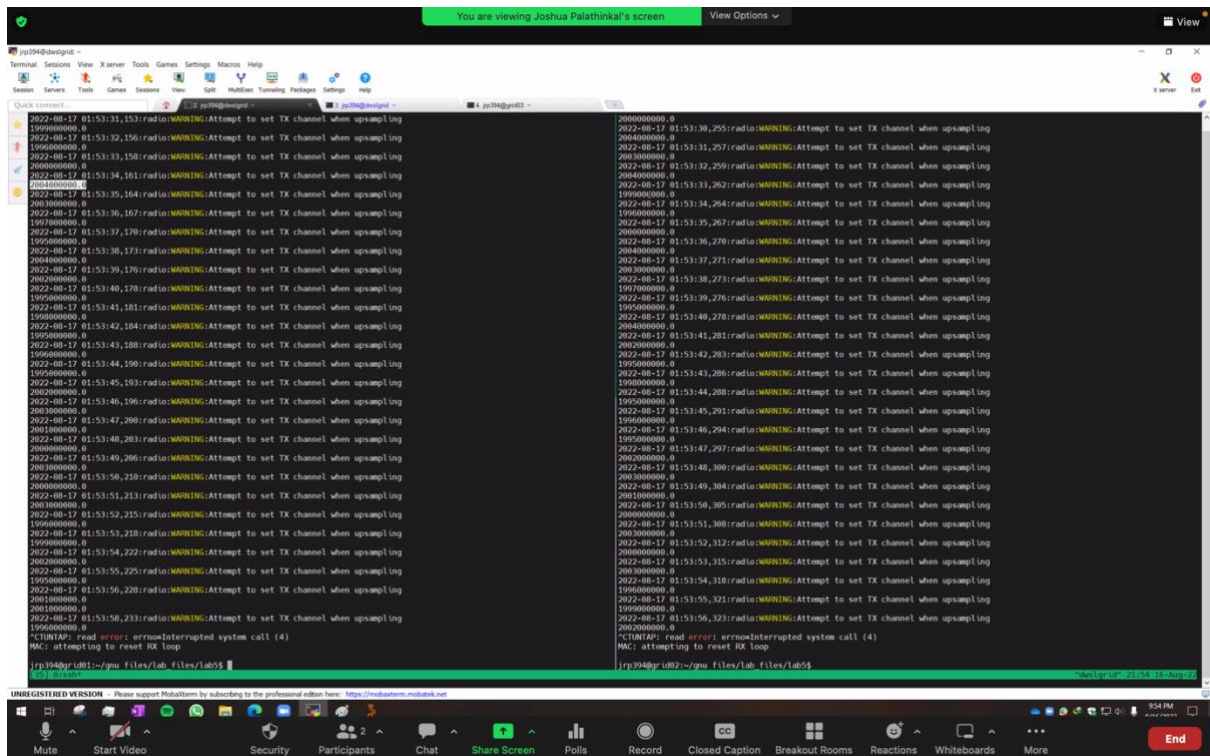


Figure 8: Frequencies taken at Tx and Rx

It can be observed that the frequencies assigned at both Tx and Rx are not in sync despite using the seed intelligently. **One of the ways in which this can be mitigated is to start the WHILE loop for the FrequencyHop function after both Tx and Rx reached the next minute (seconds = 00) from when it was started. This'll be implemented for the next Radio Wars.**