2017

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

A prototype White Space Network

Capstone Project

Andrew Keener

Kerrick1010@gmail.com

MS CSIS Candidate

University of North Carolina Wilmington

Table of Contents

1. Abstract

The purpose of this project was the development of a prototype white space network operating on the TV band channels freed up during the transition to digital cable broadcasts. This prototype network consists of two computers and two software defined radios. The network was developed with the use of Gnu radio in a linux environment and utilizes Orthogonal Frequency Domain Multiplexing to transmit data over a wide band. Relatively high data rates were achieved with this network and the prototype was proven on several data rates and channels.

2. Project Motivation

Today's society relies heavily on the internet for many aspects of modern life from commerce to social interaction. With the advent of cellular networks and WiFi hot-spots, much of the data and bandwidth that facilitates these exchanges is wireless. Current Wireless technologies are limited to a narrow spectrum that is already reaching its limits. In areas of high congestion like cities and universities people are experiencing a degradation in quality of service.

Opening different parts of the spectrum would allow for different network types like Wide area network that span miles instead of meters, Peer to peer networks to enable city spanning, free internet access, and would reduce reliance on already congested data/cellular providers.

3. Objective

The objective for this Capstone was to setup software defined radios on laptop computers to act as the physical access layer for a wireless network that operates on non-traditional frequencies (not WiFi). The physical layer is modeled on 802.11 standards to facilitate data transmissions on TV band white space spectrum. Because of FCC regulations (1) for this spectrum a further objective was to investigate spectrum sensing and integration of Google's Spectrum database.

4. Background

During the transition to digital television broadcasts from analog a sizable portion of the frequencies historically used for television broadcasts were opened up for public use (the original band was 50mhz to 806 mhz but some of the spectrum has since been auctioned off (2). The available frequencies that were the focus of this capstone are 150 to 250 mHz. The usable frequencies can possibly be expanded by using "Greyspace", or frequencies

designated for other tasks but not in use locally (1). The 100mhz wide band used in this Capstone translates to 16 channels each roughly 6mhz wide with additional space guarding each channel. Because of FCC regulations and for conformity's sake I will treat these channels as separate and distinct.

The benefits of these longer wavelength (longer than the traditional 2.4ghz wifi signals) transmissions are that they can travel farther and are hindered less by obstacles like trees/buildings. This decreased signal degradation makes them suitable for both rural and urban areas.

Accessing these frequencies will involve the use of a Software-Defined Radio. The concept of the software defined radio has been around for over 40 years with the actual term being coined in 1984(3) and an example implementation in 1985(5). Modern software-defined radios rely on radio communication where the signal processing has been implemented via software and a computer or embedded system instead of in hardware with filters, modulators, or amplifiers (4) (6).

Recent implementations of whitespace networks include:

1. Wilmington NC is the location of the first commercial/public white space (7). Two parks, Hugh MacRae and Arlie Gardens were networked to provide backbone data services which feed data to traditional wireless networks. These networks also transmit live video feeds from several cameras located around the parks and have a range of 1.5 miles point to point.

2. Google has a spectrum database with an API that allows developers to visualize and check the available spectrum in their area. This database is part of the requirements set up by the FCC when the spectrum became available (8).
3. Trial in South Africa through The Tertiary Education and Research Network of South Africa (TENET) which Networked 10 universities and high schools in Cape Town using multiple white space networks to provide a data backbone and deliver service to traditional WiFi networks(9).

See Appendix C. Related Work for further informaiton

5. Hypothesis

Using off the shelf components and open source software (like Gnu radio and TCP/IP and 802.11af) it should be possible to make a cheap and reliable white space network. These technologies should be available to the public and completely open source.

6. Requirements and Design

SDRs

The software defined radio used in this project is the HackRF one. It is broad spectrum (can operate beetween 6mhz – 6ghz tx/rx, is half duplex (life traditional WiFi) and has high sample rates (20 million+). There is also a cap on broadcast power of 50mA x 3.3V which puts us well below the 1 watt limit required by the FCC (**https://greatscottgadgets.com/hackrf/)**

GNU Radio

Gnu radio is a software developmnt toolkit that offers implementation of Software defined radios in rapid fashion using off the shelf components. The API consists of many prebuild code blocks with functionality like 802.11 and TCP/IP. Coding is done in Python and C++ for custom blocks and development can be done in Windows or Linux (with Ubuntu 14.04 used for this project).

See the list for Materials and methods in Appendix C for a more detailed list of components

GNU radio companion is the companion IDE that works with the GNU radio toolkit and it features a graphical user interface, source code represented visually using flow-graphs, and straightforward development of new code blocks.

Code blocks have functions usually including an input and output. Code blocks can implement almost anything. But in relation to this project most of the code blocks that will be discussed implement a part of a software defined radio, a digital signal processing algorithm, a packet/header, or a way to interface all of the above.

See Fig 1, Appendix C. Screen capture of gnuradio companion and its corresponding parts.

In the above figure the blue boxes correspond to different blocks. As you can see, blocks can have many different types. There are variables, parameters, operations, and complex algorithm implementations. The pane on the right which is highlighted in red shows the different blocks available. Blocks can be added to gnuradio by downloading out of tree implementations. Out of tree blocks can be developed by anyone and are available under commons licenses.

The orange and the green boxes highlight the RUN buttons and the debug pane respectfully. When a flow graph is complete and error free, pressing run will execute all of the code blocks and any output will show up in the debug pane.

See Appendix A. For a Quick Start Guide which can also be found on the Git Repo.

7. Project Timeline

November 11th 2015 Proposal

January – May 2016: Setup of Dev. Environments

Installed Ubuntu 14.04 on laptops (February)

- Fresh install of Ubuntu

- Updated all packages
- Installed Dependencies

Installed Hackrf Ones on Dev. Laptops (March - May)

Installed radios and dependencies

- Tested radios
- Ran into issues with one of the Hackrf installs and ubuntu
- Had to blow away ubuntu and clean install start over for one of the dev laptops

June to September 2016

Installed gnu radio and dependencies several times before getting a working dev. environment with gnu radio talking to the Hackrf Ones

Had to reinstall ubuntu fresh on one of the dev. Laptops

October 2016 – February 2017

Developed first implementation of flow graph for TX and RX

- One of the dev. Laptops auto updated to ubuntu 16.04 and broke gnuradio.
- Had to reinstall ubuntu fresh and reload everything
- Started testing different network metrics and configurations
- Wasn't getting very many packets over correctly

March 2017

Finally got configuration down and packets working with few to no dropped packets during several minute runs

April 2017

First week of April I started range testing… Blew out the amplifiers on the TX rails of both radios.

April 20th Defended Capstone

8. Project Deliverables

The goal of this Capstone as stated earlier is the delivery of:

- End to end communication over white space frequencies with custom packets and reasonably high data rates.
- Open source – Repository with code and flow graphs on GitHub
- Quick start guide/tutorial for easy setup
- Examinaion of Google Api and Spectrum sensing

9. Implementation

The following descriptions correspond to diagrams that can be found in Appendix C of this paper and detail the implementation of the prototype network in gnu radio.

a. Variables

There were a couple of main variables that configured the prototype network and were readily tweaked to adjust performance. They were Default Frequency, Sample Rate, gain, fft_len, payload_mod, and packet_len.

Default frequency is the center frequency at which the software defined radio transmits or receives. Sample rate is the number of samples per second tx/rx. Given a frequency of 100mhz and a sample rate of 1 million, the radio will either receive or transmit a signal from 99.5mhz to 100.5mhz (or roughly half the sample rate on either side of the center frequency. FFT_LEN stands for the Fast Fourier algorithm length. FFT is a standard algorithm used in digital signal processing to convert a signal from a time domain into a signal represented in the frequency domain. This is important because you have to translate a digital signal composed of 1s and 0s into a spectrum which can be broadcast.

QSPK or quadrature Phase shift keying is the modulation of bits from a carrier wave into four possible phase shifts. This allows for the data to be transmitted over our physical layer. PACKET_LEN is the packet size that we transmit over our network. Instead of using a complicated scheme like dynamic MTU size, both the receiver and transceiver know the packet size ahead of time.

See Fig 2. Appendix C. for the variable blocks

b. Transmit side of Network

See Fig 3. Appendix C. Generator Packets

In this stage we generate a message, in this case PDU test. This protocol data unit is our basic packet and contains all of our messages. This block is versatile and could encapsulate higher levels (for example TCP/IP packets from the network layer or data-link layer. After the message is generated it is tagged and a Cyclic redundancy check is added to let us know if the data or packet is mal-formed after transmission. A packet header for our physical layer is then generated and the header and payload are passed to the next stage of our Flow graph.

Next, we convert the bits of our packet to something OFDM can use(20)(21). Virtual source is just the continuation from above with header and payload bits. Chunks to symbols converts our bits into symbols that can be transmitted via OFDM (described in the OFDM section). The Tagged stream Mux is a multiplexer that takes two streams and combines them into one ($). At this point we have a stream of symbols we can run in the OFDM blocks.

See Fig 4. Appendix C. Modulation

OFDM is a method for spreading our data over a broader channel (%). This is done by modulating the signal into multiple sub-carrier bands that can be broadcast simultaneously. The modulation for our sub-carriers was done in previous steps. In our implementation, QSPK

and DSPK modulation is used for the payload and packet respectively. The Carrier allocator block also adds pilot carriers and symbols to help the receiver find our channel. In these blocks the tagged complex symbols from our previous blocks are mapped onto corresponding OFDM symbols and added to their corresponding carrier. An FFT is then performed on the OFDM Carrier Allocator stream to convert it to the Frequency Domain. The Cyclic prefixer adds a guard interval to the signal and facilitates channel estimation and equalization. At the end of this process we now have a signal spread across a spectrum that we can broadcast on.

See Fig 5. Appendix C. TX Orthogonal Frequency Domain Multiplexing

Finally the signal is ready for transmission. It is piped to the Osmocom sink which is the interface to our HackRF On.

See Fig 6. Appendix C. Transmit

c. Receive side of Network

These blocks receive a signal, synchronize it, and then demux (de-multiplex) our header from our payload.. This means we separate our header and payload signals upfront. We can do this because we used different symbol sets to modulate them.

See Fig 7. Appendix C. OFDM sync. Packet header and payload demux

See Fig 8. Appendix C.RX Header demodulation

After we have separated our header from our payload we can convert it back to a series of bits that we can read. First we have to convert it from the frequency domain to the time domain. We can then estimate the channel by identifying the same sync words used in the carrier allocator from our transmit side. Equalization is then performed on the stream. This helps reduce distortion and background noise from the signal. The OFDM Serializer removes

the pilot symbols and outputs the complex symbol stream. The constellation decoder converts the complex symbols from the transmitted signal back to the unpacked bits of the header. The packet header parser does exactly what it says. Reads in the stream of bits and parses it based on the same header formatter used to generate it on the transmit side.

Much the same as with the header; we perform FFT on the payload stream, equalize the signal, and remove the pilot symbols.

See Fig 9. RX Payload demodulation

As with the header, the payload is unpacked from complex symbols to bits and translated to our PDU. This is ten passed to a message debug block that outputs the PDU to console and the debug pane. The top input is from the header so that the header is output first, then the PDU.

See Fig 10. RX Complex to PDU

d. TX and RX simultaneoulsy (Transceiver):

Building a transceiver in gnuradio with the hackrf is as simple as combining the TX and RX into one flow-graph. Since the Hackrf one is capable of half duplex transmission, gnuradio and the Osmocom blocks will alternate transmission and receiving to achieve this.

10. Results and Network Metrics

This section details the results achieved with the implemented physical layer topology described above and details some of its features.

a. Spectrum

See Fig 11. Broadcast at 145.25mhz 2mhx bandwidth OFDM low gain

This spectrum illustrates our OFDM broadcast. 145.25mhz was the center frequency with 1mhz on either side composing 2mhz of bandwidth. The gain was set to low (2db on the transmitter). Even though we were transmitting on a lower gain there is still a nice OFDM signal from our network.

You can see the same signal broadcast at 10db gain. The OFDM shape and curve are more pronounced. See Fig 12. 145.25mhz OFDM 2MSPS high gain

Below is the same signal with a bandwidth of 6mhz and a gain set to 10db. What is interesting in this graph is that you can start seeing the individual peaks and channels of the sub-carriers in the signal (i.e. the striation seen across the band).

See Fig 13. 145.25mhz OFDM 6MSPS high gain

These results are representative of the spectrums produced at other frequencies. Frequencies tested include 109mhz, 211.25 mhz, and many more.

c. Data

See Fig 14. 211mhz 1MSMP DPU

The above figure is an example of a packet we received over our network transmitting and receiving at 1MSPS. In this example you can see that the header is empty. It was just tagged with #f. The PDU has a length of 280 bits and is a randomized vector.

Fig 15 shows samples transmitted and received at 2MSPS and is illustrative of a problem encountered during testing. At higher samples per second and data rates there was increased

packet loss. This may be due to limitations in processing or transmitting. This could be narrowed down by using more powerful computers than the two elderly laptops used to implement the network. See Fig 15 211mhz 2MSMP DPU

Fig 16 shows the last packet received in a test run at 1MSPS. Packets were limited to 1400 bits and most packets approached this length. The duration of the network run was 1 minutes and the calculated bandwidth was approximately 1.1 Mbps. See Fig 16 211mhz PDU 1400 packet length.

11. Lessons learned

1. Projects of any complexity need to be thoroughly planned and scoped before initiation. One of the things I struggled with during the execution of this project was scoping it down to something that was actionable and achievable. When you select a project like this that has seemingly endless possibilities and interesting avenues to pursue it is important to identify a goal that is accomplish-able in a given time frame.

2. Starting and stopping a project every couple of weeks or months is not a good way to make forward progress.

For me at least, it is very difficult to drop work for several weeks and then come back to it. This is a key reason that the project took as long as it did (over 2 years). The way I combated this towards the end was to have mini “deliverables” and accomplishments such that I would not stop work or take a break until I had accomplished a certain task. Eventually these all add up and you don’t have to worry about where you were and what steps you were in the middle of when you last looked at it.

3. Gnuradio, Ubuntu, and the Hackrf One are not professional tools.

These open source tools and platforms are all a work in progress. Even gnuradio and ubuntu, which are projects that have been running for over a decade, have their bugs and dependencies that are difficult to track down. When you combine several of these open source tools for a project like this there are going to be times where you spend a week on fixing an issue, reinstalling a library, or completely blowing away your OS install and starting over. In point of fact, I reinstalled Ubuntu at least 8 times on two machines and had to reinstall gnuradio and hackrf.

12. Acknowledgments

I'd like to thank Dr. Kline, the chair of my Capstone for sticking with me even as this project has taken longer than expected. Dr. Kline is one of the main reasons I picked UNCW for graduate school. He was very welcoming and I really enjoyed the courses he taught.

I'd also like to thank Dr. Vetter. Besides being a member of my Capstone Committee Dr. Vetter helped me innumerable times throughout my graduate experience. From graduate assistance-ships to research ideas and generally just always being there to listen I can't thank Dr. Vetter enough.

The same goes for Dr. Cummings. Thank you for being on my Committee. I really appreciated your input and guidance.

Nancy Holland at the Graduate school must also be mentioned. I can't count the number of times she has helped me we scheduling and getting on to Dr. Vetter's Calendar.

Finally I'd like to thank the Computer Science and Information Systems Master's program at UNCW and everyone that makes it work. I learned very many things while seeking my master's and became a Computer Scientist along the way.

13. Conclusion and Other Thoughts

Whitespace networking is the future of wireless networking. Implementation of Whitespace networks offers the possibility of cheap, reliable, and equitable data communications for society as a whole. Possible benefits include free or close to free wireless data communications/VOIP, and less reliance on the last mile providers (which would have positive impacts in the Net Neutrality debate)(15). It is not limited to rural or urban settings and if implemented correctly can offer free and ubiquitous internet access around the country.

The implementation of white space networks also has broad implications for current technologies and the businesses that rely on them. Current Cellular and wireless data service may become superannuated depending on how widespread and how quickly this technology is adopted. This has been highlighted recently by increased lobbying activities by the major Telecoms in the United States in an attempt to restrict (19) the use of these frequencies set aside for the public. Fortunately, there are other technology companies on the opposite side of this issue. This means that the political climate will have to be monitored throughout the course of this project.

Other thoughts:

The implementation of this functionality via SDRs in smart phones and other portable devices offers an additional and cutting edge area of study involving signal processing and parallel computing research. Smart phones have been trending towards parallelism with recent

advances like nVidia's tegra chips that include 72 custom cores and are being implemented in smart phones. These advances in technology make possible the implementation of true Software-defined radios in most any portable device and this convergence of multiple technologies including mobile computing and digital signal processing will change how our devices communicate and bring greater connectivity to the world.

14. Cited Works:

1. ET Docket No. 04-186, Office of Engineering and Technology Authorizes TV White Space Database Administrators to Provide Service to Unlicensed Devices Operating on Unused TV Spectrum Nationwide.
2. FCC Encyclopedia - Whitespace Database Administration – FCC http://www.fcc.gov/encyclopedia/white-space-database-administration
3. Various authors, Cognitive Network, http://en.wikipedia.org/wiki/Cognitive_network
4. Various Authors - White space (radio) - http://en.wikipedia.org/wiki/White_spaces_(radio)
5. P. Johnson, "New Research Lab Leads to Unique Radio Receiver," E-Systems Team, May 1985, Vol. 5, No. 4, pp 6-7 http://chordite.com/team.pdf
6. Fette, Bruce Alan. "Software-defined radio." U.S. Patent No. 20,040,242,261. 2 Dec. 2004.
7. MB - New Hanover County Deploys New TV White Space Network http://www.nhcgov.com/News/Lists/Posts/Post.aspx?ID=190
8. Google Spectrum Database and API - https://www.google.com/get/spectrumdatabase/
9. The Tertiary Education and Research Network of South Africa - Cape Town TV White Space Trial - http://www.tenet.ac.za/tvws/cape-town-tv-white-spaces-trial-field-measurements-report-1

10. Hillenbrand, J. ; Daimler-Chrysler AG, Sindelfingen, Germany ; Weiss, T.A. ; Jondral, F.K -Calculation of detection and false alarm probabilities in spectrum pooling systems. - Communications Letters, IEEE (Volume:9 , Issue: 4) 2005
11. Joseph Mitola - Cognitive Radio --- An Integrated Agent Architecture for Software Defined Radio (8 May 2000)
12. Bahl, Paramvir, et al. "Enabling white space networks independent of low-threshold sensing." U.S. Patent No. 8,473,989. 25 Jun. 2013.
13. SPACHOS, P. ; Dept. of Electr. & Comput. Eng., Univ. of Toronto, Toronto, ON, Canada ; Chatzimisios, P. ; Hatzinakos, D. - Cognitive networking with opportunistic routing in Wireless Sensor Networks - Communications (ICC), 2013 IEEE International Conference on. 9-13 June 2013
14. Liang Quan-quan - Energy-aware cooperative spectrum sensing for underground cognitive sensor networks - http://xuebao.nuc.edu.cn/new/english.php?id=90&tid=791
15. Brew, M., et al. "UHF white space network for rural smart grid communications." Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on. IEEE, 2011.
16. Chen, Shaxun, Kai Zeng, and Prasant Mohapatra. "Hearing Is Believing: Detecting Wireless Microphone Emulation Attacks in White Space." *Mobile Computing, IEEE Transactions on* 12.3 (2013): 401-411.
17. Nathan White – Net Neutrality and the Future of the Internet - http://www.huffingtonpost.com/nathan-white/net-neutrality_b_5256423.html
18. Thomas, R.W. et al. (2005), "Cognitive networks", *Proceedings of the First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, Baltimore(registration required)

19. Cecilia Kang, Washington Post, http://www.washingtonpost.com/business/technology/tech-telecom-giants-take-sides-as-fcc-proposes-large-public-wifi-networks/2013/02/03/eb27d3e0-698b-11e2-ada3-d86a4806d5ee_story.html
20. Multiplexer, Wikipedia.org, https://en.wikipedia.org/wiki/Multiplexer
21. OFDM, Wikipedia.org, https://en.wikipedia.org/wiki/Orthogonal_frequency-division_multiplexing

Appendix A. Quick Start Guide (http://github.com/amkeener/Whitespace/README.md)

A prototype White Space Network

A simple OFDM based network operating on TV band frequencies

Contributors

Andrew Keener - Author

Stages

1.Test (test) - Experimental and initial commits

2.Stable (Master) - TODO

Technical Details

App

Platform - GNU Radio Companion

Development - Flow chart with custom blocks in C++

Background

TODO

Tutorial/Quick start guide

Notes:

all commands are ran as root (sudo is ommitted)

install instructions for ubuntu 14.04

During the execution of this project I found that the only reliable version of Ubuntu that played well with the Hackrf and Gnuradio was 14.04. I spent many an hour uninstalling, reinstalling, upgrading, and downgrading to figure this out. Use 14.04 if you can.

Install Hackrf one, if using another SDR you can skip Also see: installing hackrf as per: https://mborgerson.com/getting-started-with-the-hackrf-one-on-ubuntu-14-04

1.$ apt-get update

2.$ apt-get upgrade

3.$ apt-get install hackrf

Alternative I had trouble with different versions of hackrf including the one from cannonical Try:

1.$ git clone https://github.com/mossmann/hackrf.git

2.$ cd hackrf/host

3.$ mkdir build && cd build

4.$ cmake ../ -DINSTALL_UDEV_RULES=ON

5.$ make

6.$ make install

7.$ ldconfig

At this point you should be able to test your hackrf

1.$ hackrf_info This command will display the firmware version, software, and serial. If this command doesn't work you will have to try and install another version of the software or troubleshoot your USB ports/drivers.

Install Pybombs see: https://github.com/gnuradio/pybombs/ for troubleshooting

1.$ pip install pybombs

2.$ pip install [--upgrade] git+https://github.com/gnuradio/pybombs.git

Add recipes for pybombs

1.$ pybombs recipes add gr-recipes git+https://github.com/gnuradio/gr-recipes.git

2.$ pybombs recipes add gr-etcetera git+https://github.com/gnuradio/gr-etcetera.git

Install GNU Radio with pybombs

1.$ pybombs prefix init ~/prefix -a myprefix -R gnuradio-default

Setup your prefix and run GRC

1.$ source ~/prefix/setup_env.sh

2.$ gnuradio-companion

Other tools:

GQRX is helpful for visualizing spectrum and has a digital radio tuner for radio stations as well To Install:

1.$ add-apt-repository -y ppa:gqrx/gqrx-sdr

2.$ apt-get update

3.$ apt-get upgrade

4.$ apt-get install gqrx

If everything works and you can pick up radio stations you are good to go

Running the flow graphs with HackRF

If you have two HackRF One boards and multiple machines you can simply run RX_PDU_OFDM.grc via gnuradio-companion on one machine, and TX_PDU_OFDM.grc on the other.

If you want to run a transceiver you will have to combine the RX and TX graphs. You can simply drag and drop all the blocks from Rx to Tx. This will not leave much room for both RX and TX to be displayed and Blocks will end up on top of each other.

With the HackRF One, the transceiver will work in half duplex mode as per the Hackrf one spec. You may need to add some synchronization blocks or schemes to sync one transceiver to another.

I recommend getting RX and TX working separately before trying a transceiver.

Appendix B. Google Spectrum API and Spectrum sensing overview

The Google Spectrum API facilitates identifying available channels in your local area. There are a couple of prerequisites for using the API. You must have a google account. You will also need an FCC ID but there is a default FCC ID that you can see in the code examples below.

To pull back spectrum info you will have to first generate a SHA1 key with a keygen like Keytool Utility. You'll need one for your application or code block if it interfaces with the API. You'll also need separate keys if you implement a server or another client. After generating keys, simple JSON API requests to the interface will hit their database and pull back the requested information (example below).

```
curl -XPOST https://www.googleapis.com/rpc -H "Content-Type: application/json" --data '{
  "jsonrpc": "2.0",
  "method": "spectrum.paws.getSpectrum",
  "apiVersion": "v1explorer",
  "params": {
    "type": "AVAIL_SPECTRUM_REQ",
    "version": "1.0",
    "deviceDesc": { "serialNumber": "your_serial_number", "fccId": "TEST", "fccTvbdDeviceType":
    "location": { "point": { "center": {"latitude": 42.0986, "longitude": -75.9183} } },
    "antenna": { "height": 30.0, "heightType": "AGL" },
    "owner": { "owner": { } },
    "capabilities": { "frequencyRanges": [{ "startHz": 800000000, "stopHz": 850000000 }, { "sta
    "key": "your_API_key"
  },
  "id": "any_string"
}'
```

The response to that request contains Bandwidth, Frequency ranges, and Transmit decibel limits.

```
{
 "jsonrpc": "2.0",
 "id": "any_string",
 "result": {
  "type": "AVAIL_SPECTRUM_RESP",
  "version": "1.0",
  "timestamp": "2013-08-31T03:28:08Z",
  "deviceDesc": {
   "serialNumber": "your_serial_number",
   "fccId": "TEST",
   "fccTvbdDeviceType": "MODE_1"
  },
  "spectrumSchedules": [
   {
    "eventTime": {
     "startTime": "2013-08-31T03:28:08Z",
     "stopTime": "2013-09-02T03:28:08Z"
    },
    "spectra": [
     {
      "bandwidth": 6000000.0,
      "frequencyRanges": [
       {
        "startHz": 5.4E7,
        "stopHz": 5.12E8,
        "maxPowerDBm": -56.799999947335436
```


Spectrum Sensing

This can be accomplished with gnu radio and code blocks. A simple spectrum analyzer would:

- Sample portion of spectrum you are interested in
- Run FFT on samples to visualize/turn into frequencies and amplitudes
- Sum all amplitudes and divide by total number of FFT series.

This would give you an average amplitude over the whole sample.
With this you could develop an average amplitude for a cutoff. If a scanned frequency range had a higher amplitude than the cutoff, you would simply look for another channel.

The amplitude cutoff would have to take into account the types of signals you might expect to interfere at particular frequencies. You'd also need to know something about ambient background noise for each channel.

Spectrum sensing is important because it is part of the FCCs conditions for using the spectrum. The Google API fulfills this obligation in most cases, but it is advisable to implement spectrum sensing in any White space application to prevent interference with other devices.

Appendix C. Related Work, Materials and Methods, Figures and diagrams

Related Work

Spectrum Sensing and Switching

Spectrum sensing and switching is an integral feature of any cognitive network. Not only is it important to detect which channels are open or closed, this information must be shared between nodes in order to increase the probability that no channels are falsely labeled and used (10)/

As far back as 2000 Joseph Mitola developed an outline for using cognitive radio as the basis for personal wireless networks (11). Most importantly, his work on real-time radio spectrum "renting" is the basis for the ideas of spectrum sensing and switching that I plan to use in the implementation of this project. More recently, work has been done by several teams to develop protocols and schemes for the implementation of spectrum sensing and switching on base stations (12) which would then service local devices. They would then free up channels for wireless microphones and other devices (as per FCC regulations) as needed.

Peer to Peer functionality

Recent work in this area has focused on opportunistic routing protocols for cognitive radio networks (13). These opportunistic routing protocols have not only been shown to deliver higher performance than traditional wireless routing techniques, they also have the added benefits of reduced energy consumption (14), and network utilization. Peer to peer functionality will be an important addition to any whitespace enabled cognitive network developed through this project.

Other Whitespace Network Applications

There are many other projects and technologies being developed using the spectrum made available during the switch to digital television. Some of the more interesting technologies deal with the "internet of things" idea and with building a smarter power grid. An example of this is a project in the highlands of Scotland that is designed to bring a whitespace network to

a region with multiple power generation resources like wind and tidal power, but without the necessary connectivity infrastructure to implement meters and load balancing (15).

An interesting side note is the development of countermeasures for wireless microphone spoofing which is aimed at kicking whitespace devices off of channels and reducing available bandwidth (16). Whitespace networks are in their infancy with very few actually seen in the wild and attacks are already being developed and countered. Any implementation of spectrum sensing and switching for this capstone will need to take into account methods for detecting microphone spoofing.

Materials and methods

The white space network prototype was built around readily available technology and open source software.

a. Materials:

- HackRF One (2x) – Software Defined Radio Transceiver. This device operates on frequencies ranging from 10 Mhz to 6 Ghz. They are half-duplex transceivers that can process up to 20 million samples per second. TX is limited to 50 mA at 3.3v which caps transmissions slightly above FCC rules.
- Linux boxes running Ubuntu Linux 14.04
- GNU Radio API and GNU Radio Companion
- 802.11(af) and 802.22 (still being developed) protocols.

b. The GNU Radio API:

GNU Radio is a software development toolkit that offers the ability to implement Software defined radios in a rapid fashion using off the shelf components.

Features include:

- Pre-built code blocks
- Coding in python or C for custom blocks
- The ability to develop windows or linux applications.

c. GNU Radio Companion

Features:

- Graphical user interface
- Source Code is represented visually using flow-graphs
- Straightforward development of new code blocks

Figures and Diagrams

Fig 1: Screen capture of gnuradio companion and its corresponding parts.

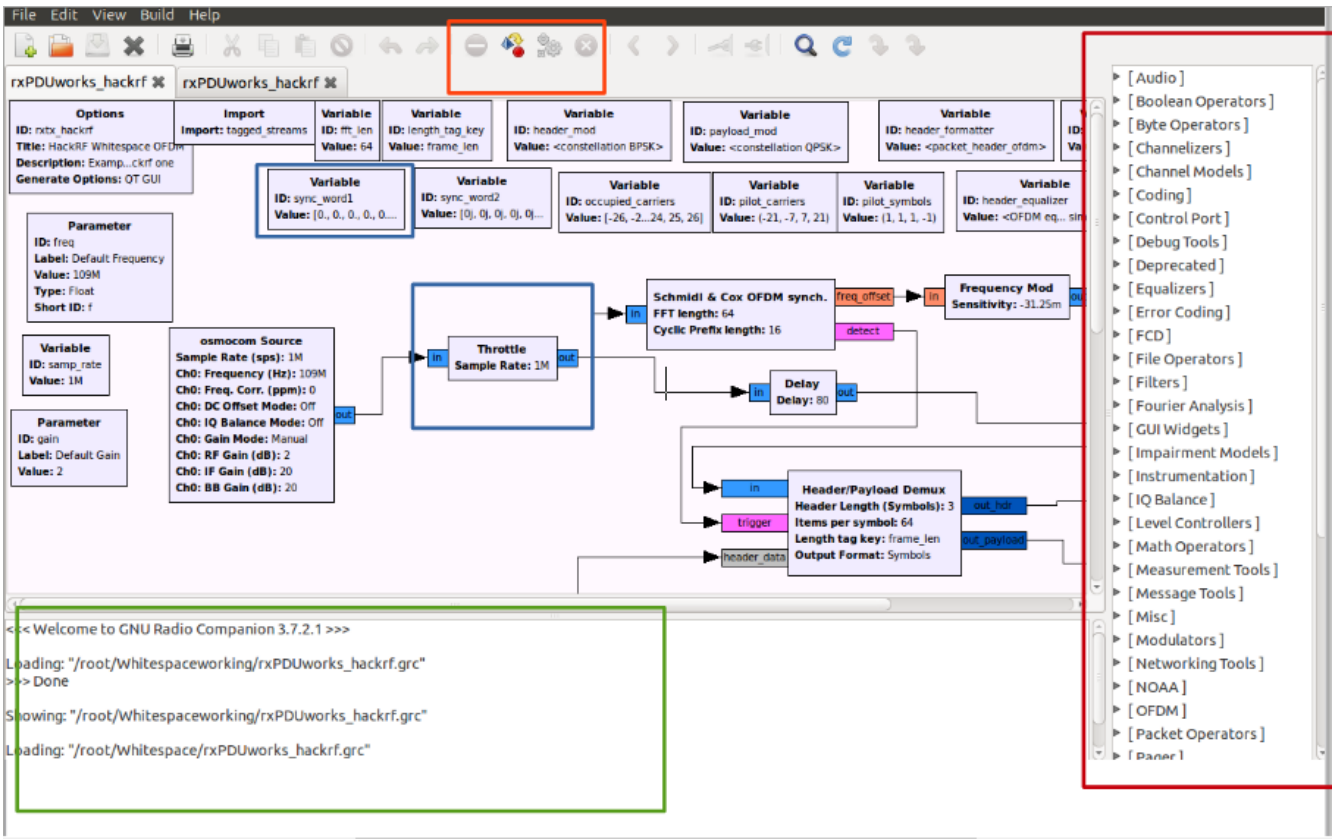

Fig 2: Variables

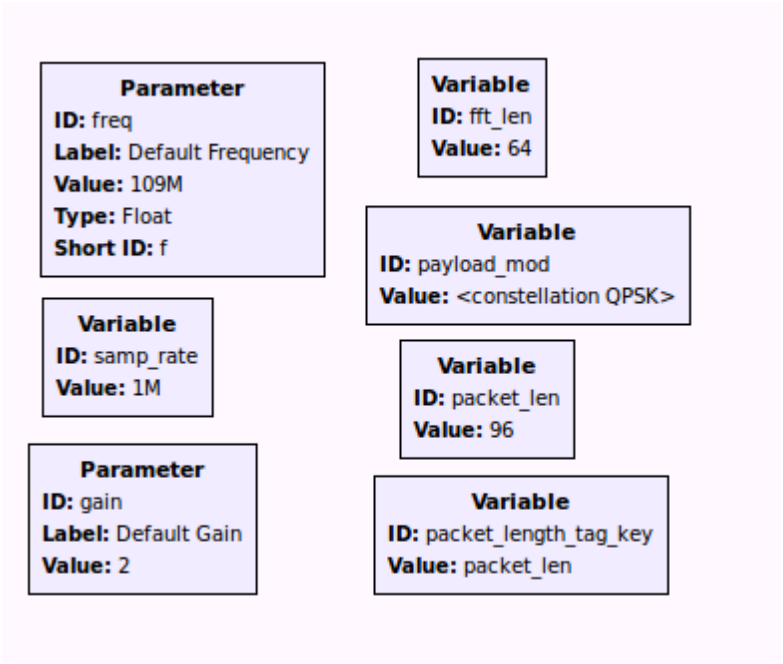

Fig 3: Generating our packets

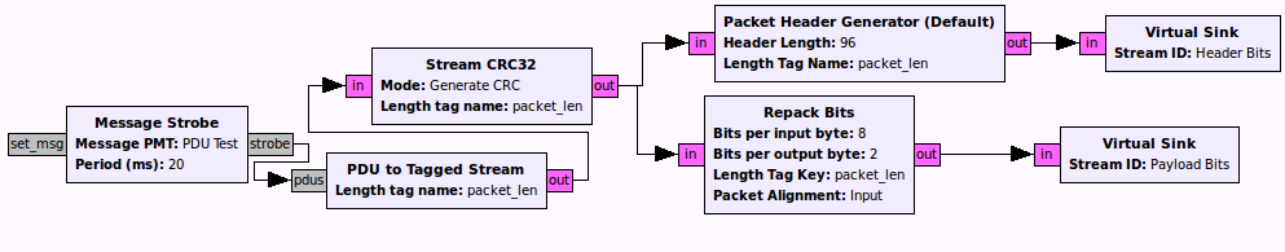

Fig 4: Modulation

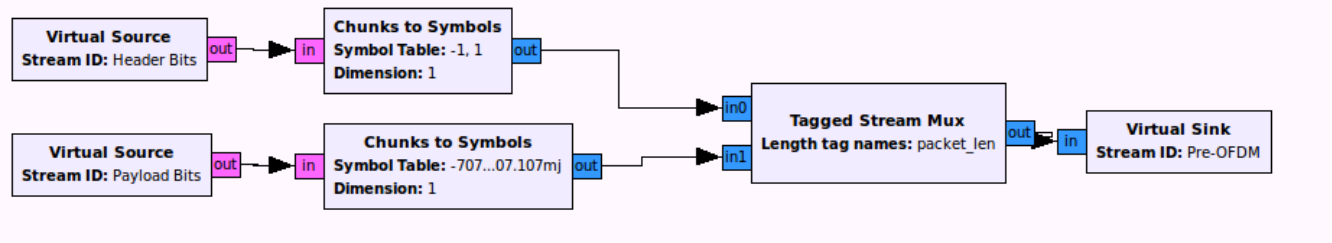

Fig 5: Tx Orthogonal Frequency Domain Multiplexing

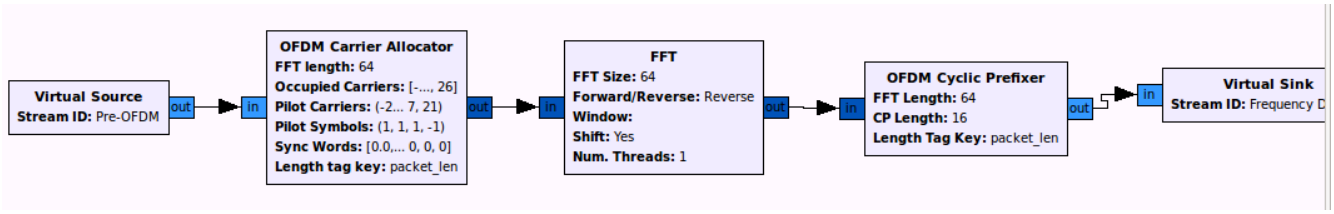

Fig 6: Tx Transmit

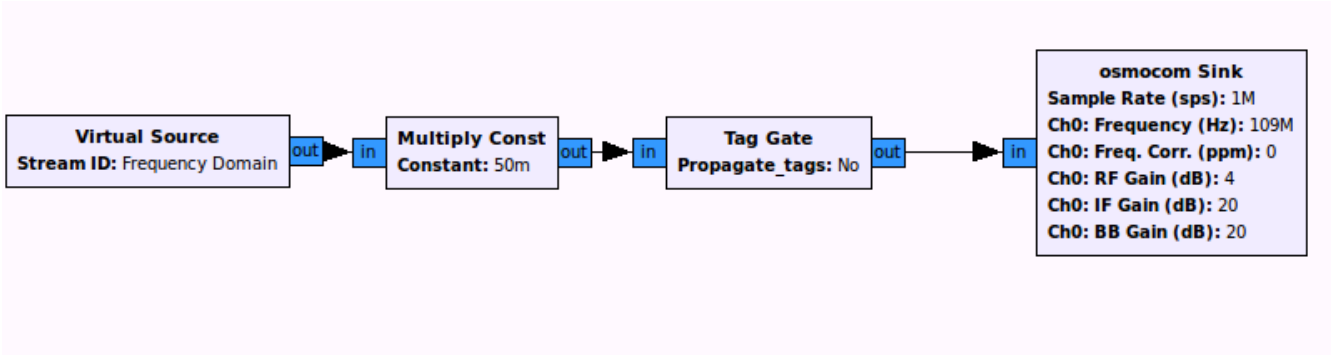

Fig 7: OFDM sync, Packet header and payload demux

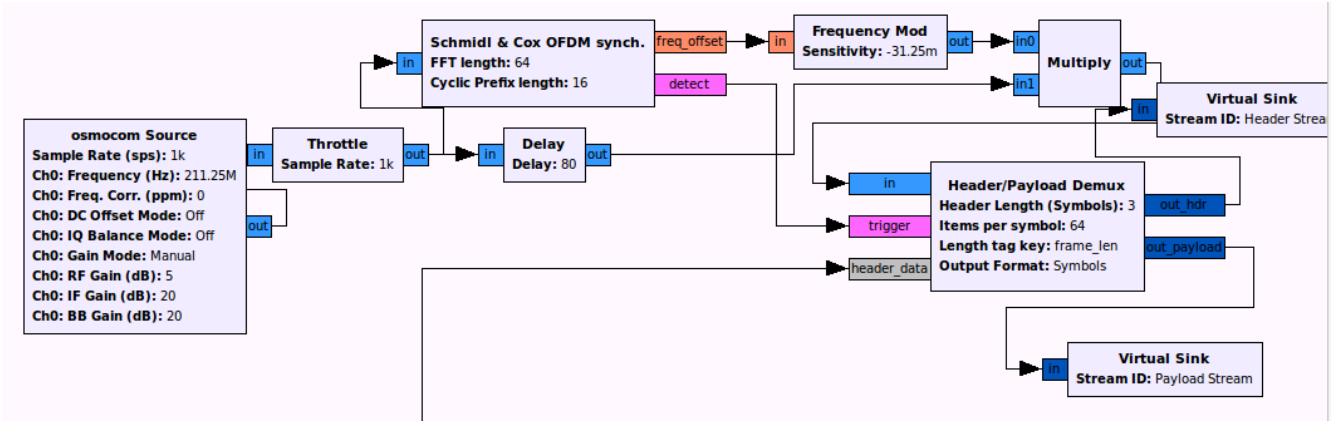

Fig 8: RX Header Demodulation

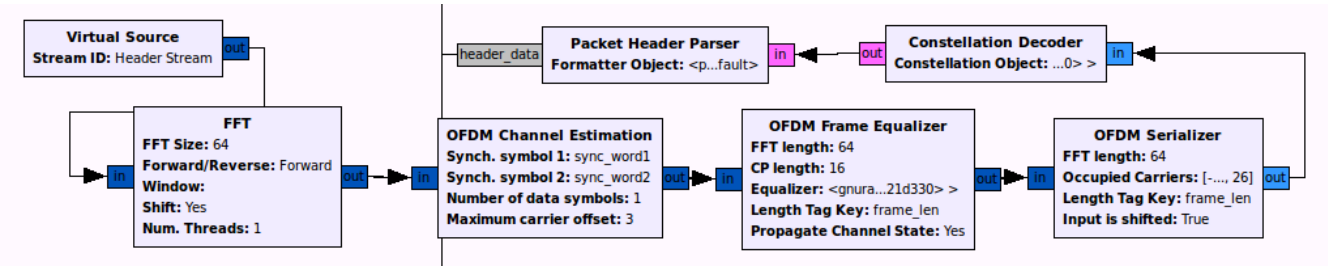

Fig 9: RX Payload Demodulation

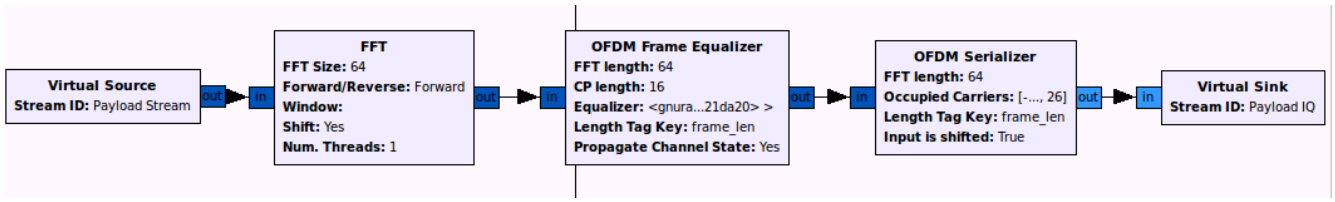

Fig 10: RX Complex to PDU

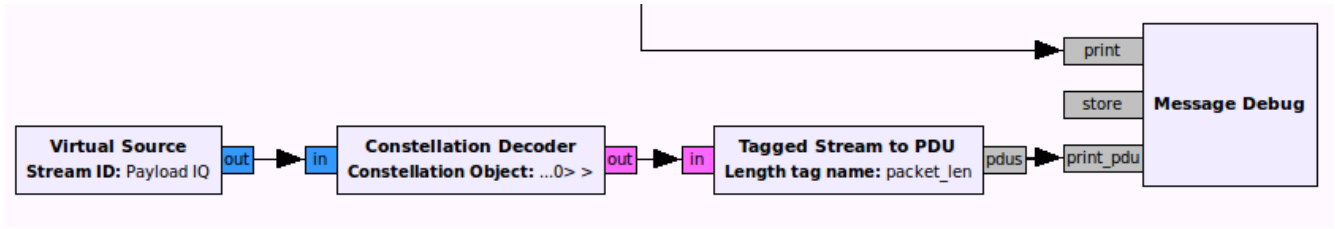

Fig 11: Broadcast at 145.25 mhz 2mhz bandwidth OFDM low gain

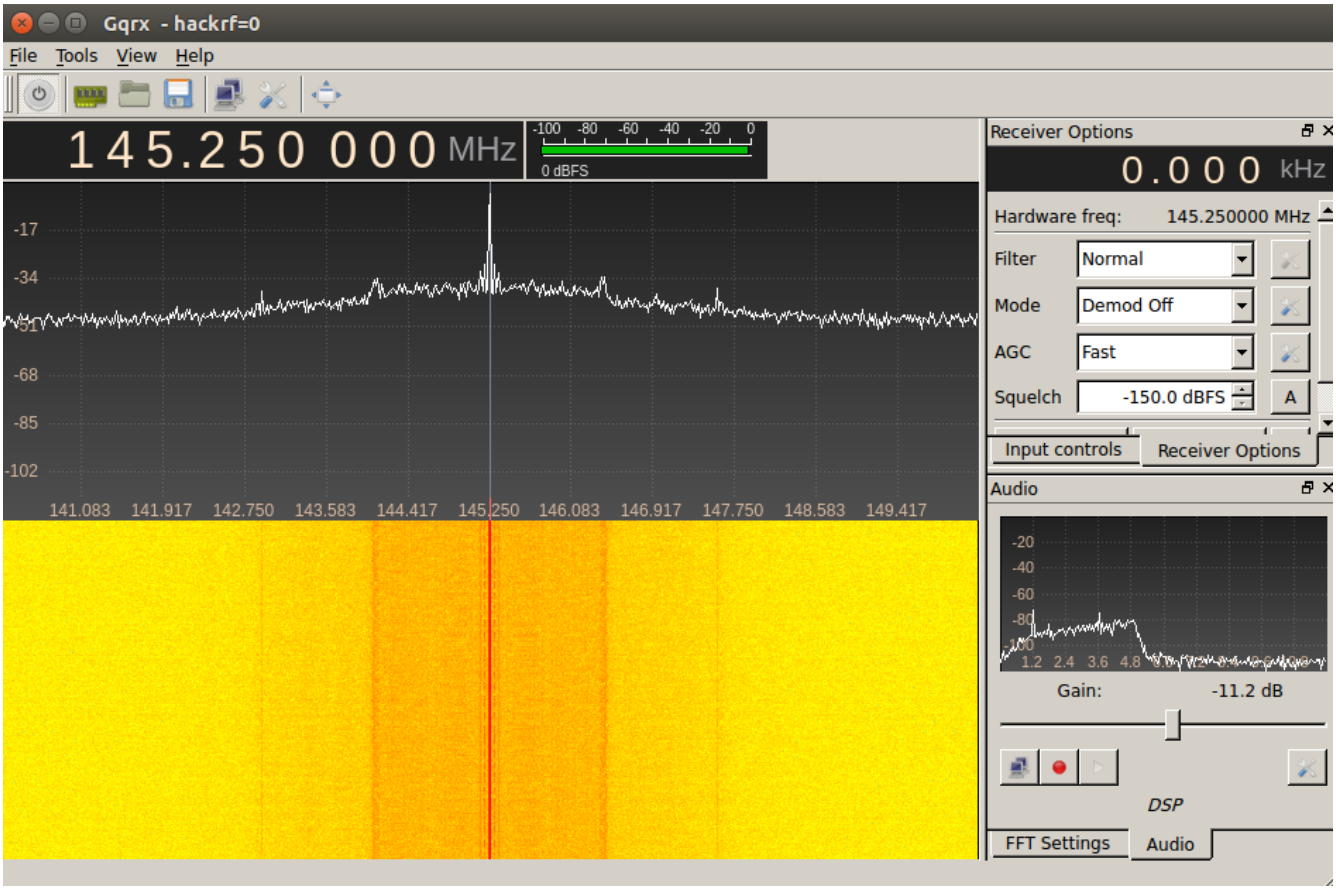

Fig 12: 145.25mhz OFDM 2MSPS high gain

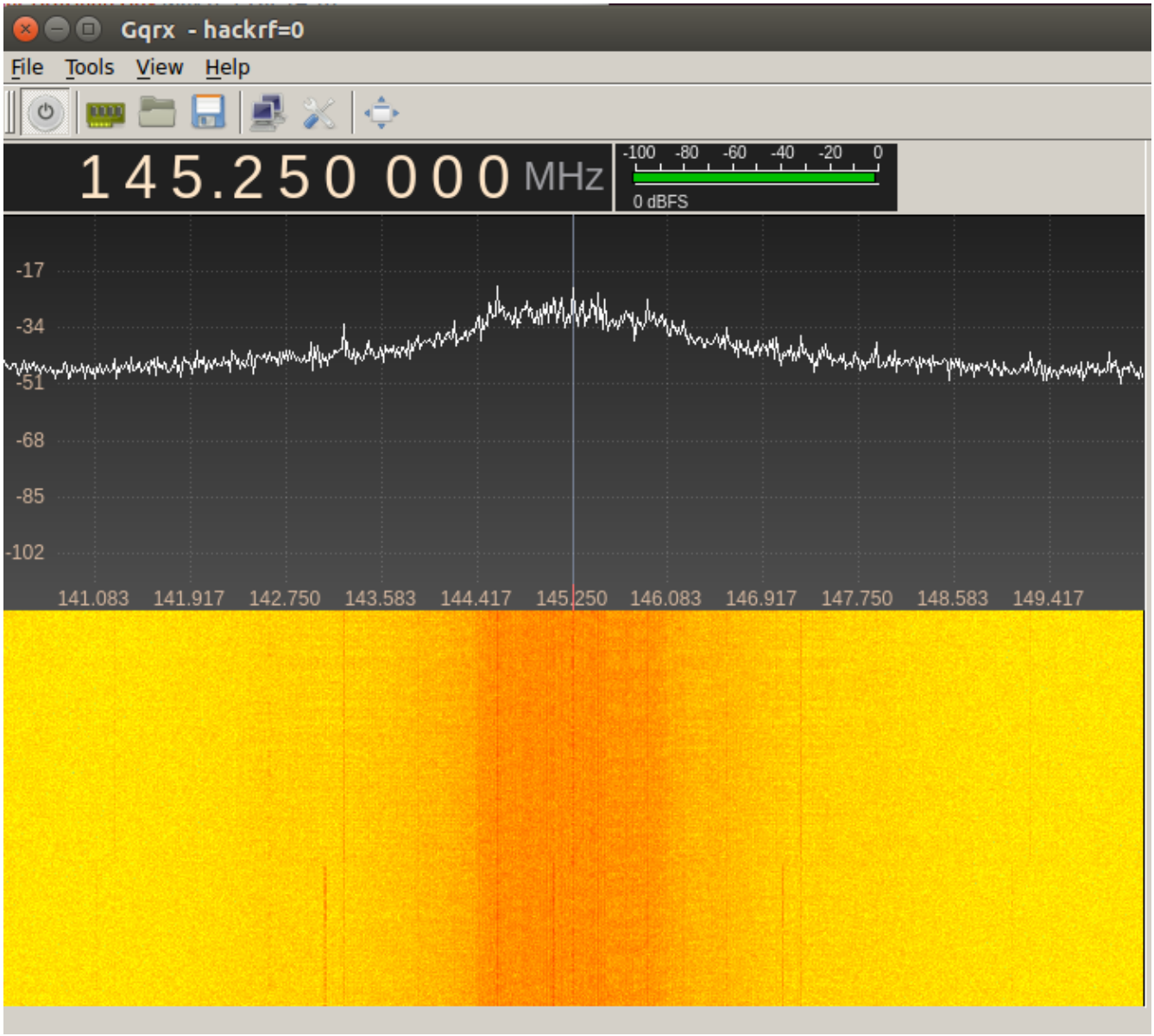

Fig 13. 145.25mhz OFDM 6MSPS high gain

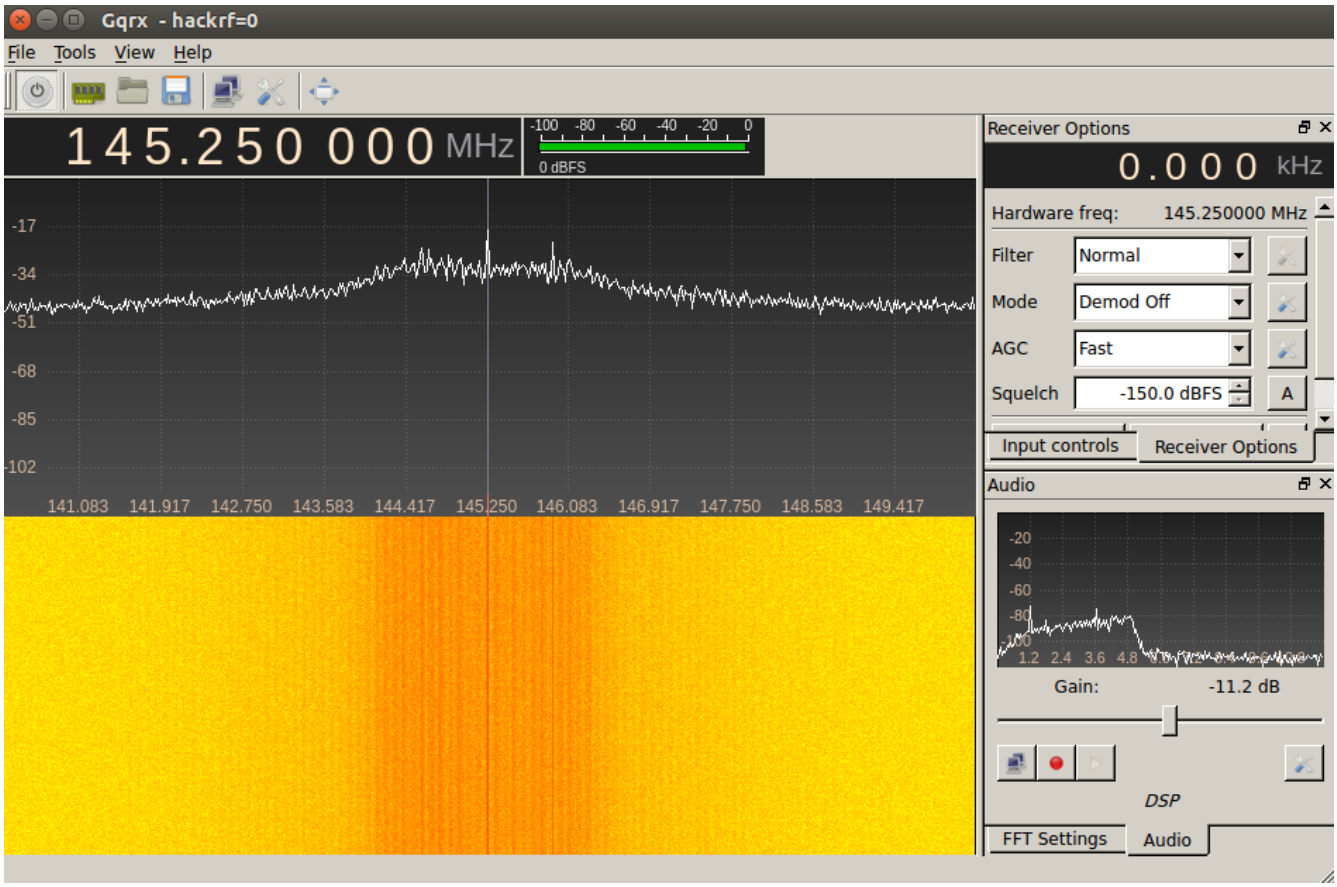

Fig 14: 211mhz 1MSPS PDU

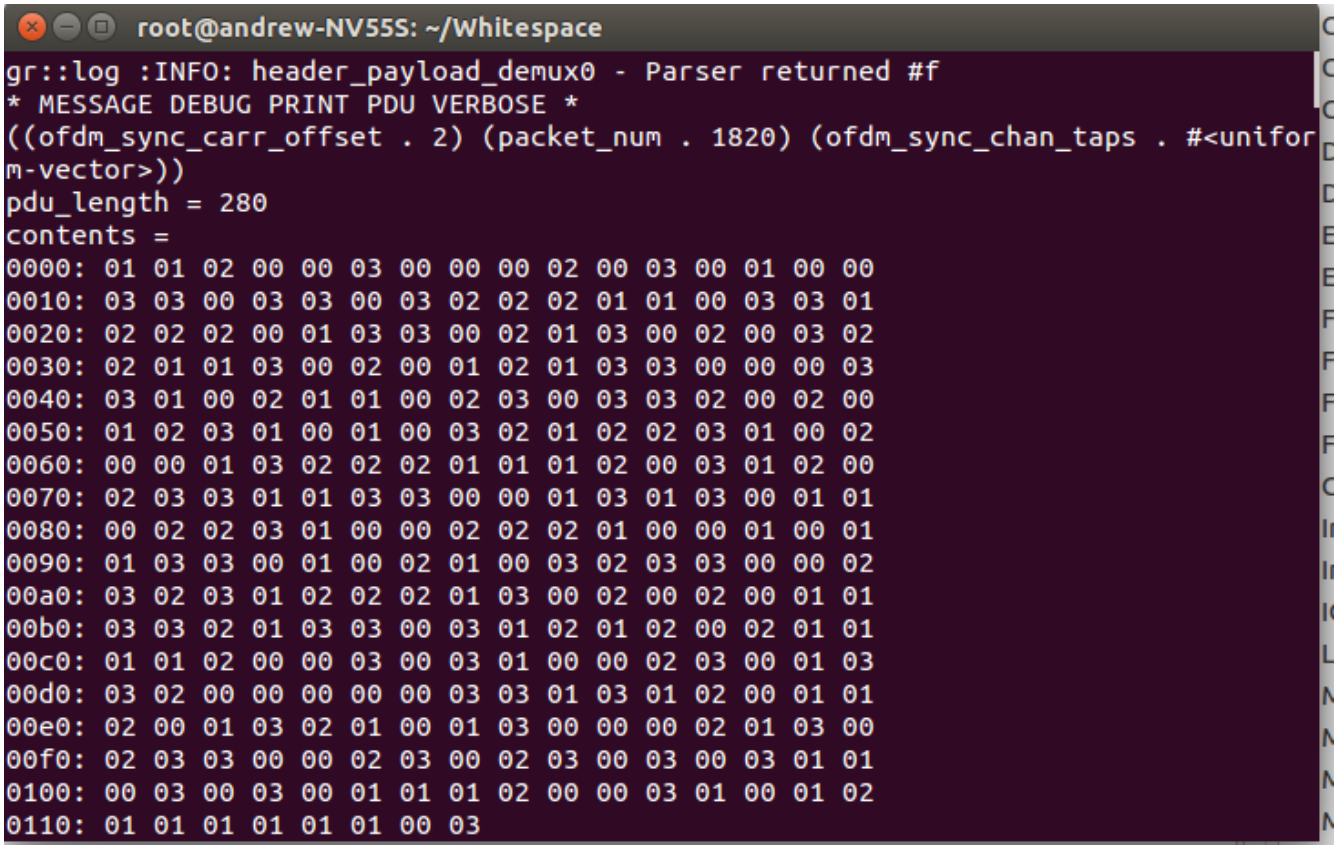

Fig. 15: 211mhz 2MSPS PDU

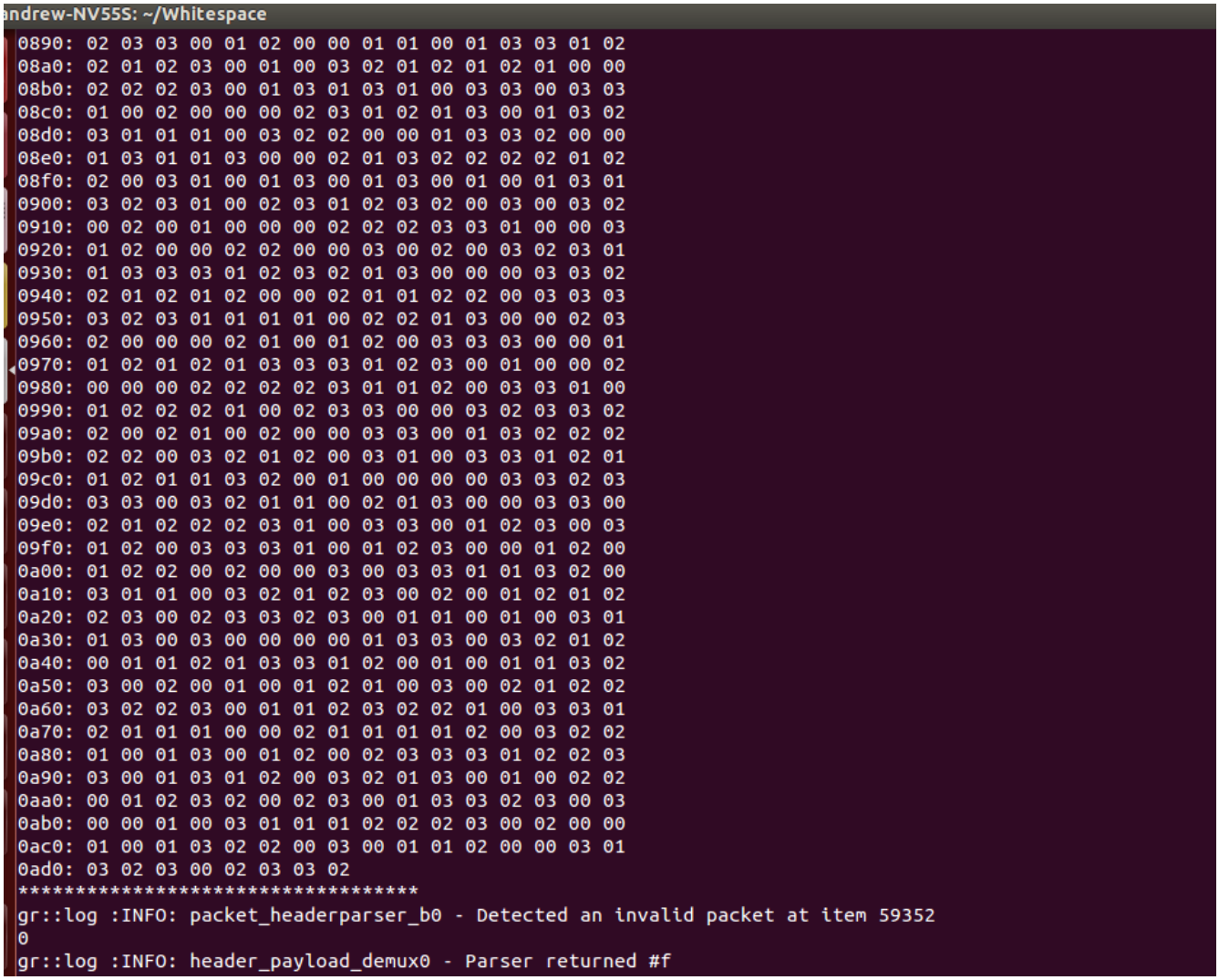

Fig 16: 211mhz PDU 1400 packet length

```
gr::log :INFO: packet_headerparser_b0 - Detected an invalid packet at item 10560
gr::log :INFO: header_payload_demux0 - Parser returned #f
* MESSAGE DEBUG PRINT PDU VERBOSE *
((ofdm_sync_carr_offset . -2) (packet_num . 3759) (ofdm_sync_chan_taps . #<unifo
rm-vector>))
pdu_length = 1340
contents =
0000: 01 01 00 01 00 02 01 03 02 01 02 01 00 03 03 02
0010: 00 02 00 01 02 03 01 02 00 01 02 01 02 03 00 02
0020: 02 00 03 02 00 00 01 03 03 03 03 01 03 03 01 00
0030: 02 03 00 02 01 02 01 01 03 03 02 03 02 02 03 01
0040: 01 01 00 03 00 03 00 03 01 03 03 02 01 00 02 03
0050: 01 01 00 01 02 00 01 02 03 03 00 03 02 02 00 02
0060: 01 01 00 02 01 01 03 00 03 03 03 03 01 03 03 00
0070: 01 00 03 01 01 02 00 02 00 00 01 02 01 03 01 01
0080: 02 03 02 02 01 03 01 03 02 03 00 02 00 02 02 00
0090: 01 03 02 03 03 00 00 02 02 03 03 01 00 00 01 01
00a0: 00 03 03 02 02 00 02 01 02 02 00 01 02 02 00 01
00b0: 01 01 00 03 01 00 01 03 00 02 02 00 01 03 02 02
00c0: 03 00 02 00 03 01 03 00 03 01 01 01 01 00 01 01
00d0: 03 03 01 02 03 01 02 00 03 01 02 00 02 01 01 00
00e0: 03 03 03 00 00 00 01 00 00 03 01 01 00 02 01 03
00f0: 00 03 01 02 00 01 01 01 03 03 01 03 03 01 01 03
0100: 03 01 03 01 02 03 02 01 01 02 03 00 01 02 03 00
0110: 00 02 01 02 03 00 02 02 02 02 02 03 02 03 03 02
0120: 03 03 02 02 03 02 02 00 01 03 03 01 03 02 01 03
0130: 01 00 03 02 00 02 02 02 02 03 01 00 00 01 00 00
0140: 02 03 01 02 01 00 02 03 03 01 02 03 02 03 01 02
0150: 02 02 02 03 00 03 00 03 03 02 01 01 00 02 00 00
0160: 02 00 00 02 02 03 03 00 01 02 00 02 02 03 02 00
0170: 02 02 03 03 00 02 01 02 03 00 03 03 02 00 02 03
0180: 03 03 01 00 00 03 00 02 01 01 01 01 01 01 01 01
0190: 03 00 02 00 03 02 02 02 03 00 02 03 01 02 02 01
01a0: 01 00 02 01 00 01 00 02 02 03 02 03 02 00 01 03
01b0: 01 03 02 02 02 00 02 03 00 03 01 02 02 02 01 01
01c0: 00 01 00 00 02 03 02 03 03 02 03 02 03 01 02 02
01d0: 01 00 03 00 01 00 02 02 03 01 02 02 02 01 01 03
01e0: 03 00 02 00 02 01 00 02 00 01 01 02 03 02 03 00
01f0: 02 00 00 02 00 03 02 03 00 00 01 02 02 02 03 01
0200: 02 00 00 01 00 02 00 03 03 01 03 02 02 00 00 03
0210: 02 02 02 01 03 03 01 00 02 03 00 00 01 03 01 02
```