FLORIDA POLYTECHNIC UNIVERSITY

MASTER'S THESIS

ARCAM-NET: A Software Defined Radio Network Testbed

Author:
John MCCORMACK

Supervisor:
Dr. Ryan INTEGLIA

A thesis submitted in fulfillment of the requirements for the degree of Masters of Engineering

in the

College of Engineering

March 19, 2016

Declaration of Authorship

I, John McCormack, declare that this thesis titled, "ARCAM-NET: A Software Defined Radio Network Testbed" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

FLORIDA POLYTECHNIC UNIVERSITY

Abstract

Dr. Ryan Integlia
College of Engineering

Masters of Engineering

ARCAM-NET: A Software Defined Radio Network Testbed

by John McCormack

ARCAM-Net is a Software Defined Radio Network (SDRN) testbed and platform. Nearly every component except the Software Defined Radios (SDRs) themselves are open source components. The goal of ARCAM-Net is to establish a low cost platform that can be quickly implemented by anyone interested in experimenting with SDRNs. This document presents the network itself and also acts as a manual for working with ARCAM's first implementation.

Acknowledgements

I would like to thank Dr. Ryan Integlia for all of his support throughout this process. I would also like to acknolwedge Joseph Prine, Bradley Trowbrdige, and R. Cody Maden for all of their hardwork.

Contents

Declaration of Authorship **iii**

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**
1.1 Mesh Networks . . . 1
1.2 Software Defined Radio Networks . . . 1
1.3 Trends Towards Cognitive Radio Environments . . . 2
1.4 ARCAM-Net . . . 2

2 Literature Review **5**
2.1 SDR Fundamentals . . . 5
2.2 GNU Radio and MATLAB . . . 5
2.2.1 GNU Radio . . . 5
2.2.2 MATLAB . . . 5
2.2.3 RFNoc . . . 5
2.3 Network Concepts . . . 5
2.3.1 What is a Network . . . 5
2.3.2 Mesh Network . . . 5
2.3.3 GNU Radio Mesh Networks . . . 5
2.3.4 Mesh Extension into SDR . . . 5
2.3.5 What is a network protocol . . . 5
OSI Model . . . 5
Common Protocols . . . 5
Batman . . . 5
OLSRD . . . 5
2.4 SDR and Cognitive Radios . . . 5
2.5 State of the Practice . . . 5
2.6 State of the Art . . . 5

3 Methodology **7**
3.1 Design . . . 7
3.1.1 SDR . . . 7
3.1.2 GNU Radio Flowgraph . . . 7
3.1.3 Batman-adv . . . 9
3.1.4 Flask Web Server and Socket.IO . . . 9
3.1.5 A.L.F.R.E.D. . . . 10
3.2 Fabrication and Build Process . . . 11

3.3 Test - SDR (Environment/Setup) 11
3.4 Test Procedures 11
3.4.1 Network Benchmarks 12
3.4.2 Route Changes 12
3.4.3 Frequency Distribution 12
3.4.4 Error (inaccuracy) 13
3.4.5 Energy Use 13
3.5 Test - Network 13
3.5.1 Signal Strength 13
3.5.2 Range 13
3.5.3 Noise 13
3.5.4 Loss 13
3.5.5 Error 13
3.5.6 Energy Efficiency 13
3.5.7 Latency 13
3.5.8 Throughput 13
3.5.9 Topology Effects 13
3.5.10 Protocol 13
3.5.11 Interference 13
3.5.12 Crowding 13
3.5.13 Crosstalk 13

4 Findings **15**
4.1 Results 15
4.1.1 Network Benchmarks 15
4.1.2 Route Changes 15
4.1.3 Frequency Changes 15
4.2 Data Analysis 18
4.3 Assumptions 18
4.4 Discussion of Error 18
4.5 Error Bars 18

5 Discussion **19**
5.1 Constraints and Limitations 19
5.2 SDR Network 19
5.3 Cognitive Networks 19

6 Conclusion **21**
6.1 Summary 21
6.2 Conclusion 21
6.3 Recommendation 21
6.4 Future Work 21

A Running ARCAM-Net **23**
A.1 Configuring the Computer 23
A.1.1 Installing Natively 23
A.1.2 Installing Through Vagrant 23
A.1.3 Manual Install 23

A.2 Getting The Flowgraphs 26
A.3 Running The Flowgraphs 27
A.3.1 Just the Mesh Network 27
Raising the Bat Signal 27
A.3.2 TMux Script 27
A.3.3 Full Web Server 27

List of Figures

3.1 An overview of the components of the system, including the OSI Layers they interact with. [1] [2] [3] [4] [5] [6] 8
3.2 The web interface that lets the user initiate the network to hop to a new frequency. 10
3.3 The configuration used for the first set of tests. 11
3.4 The configuration used for the second set of tests. 11

4.1 The data received from operating at 902/915 MHz. 16
4.2 The data received from operating at 2.4/2.5 GHz. 16
4.3 The Averages from all three tests. 17
4.4 The initial condition from the test presented in Section 4 B, where there are two possible routes the packet can take. The output is from running batctl. The link quality is listed under the column labeld (#/255). A higher number represents a better quality connection. 17
4.5 After the gain is reduced in the test presented in Section 4 B, the packets are now routing through a different node. 17

5.1 The result of using ALFRED to shift frequencies. One node is left behind as the others move to the new channel. Image collected using the Tektronix RSA306 Spectrum Analyzer 20

List of Tables

List of Abbreviations

SDR	**S**oftware **D**efined **R**adio
SDRN	**S**oftware **D**efined **R**adio **N**etwork
CR	**C**ognitive **R**adio
CRN	**C**ognitive **R**adio **N**etwork
CRAHN	**C**ognitive **R**adio **A**d-**H**oc **N**etwork
BATMAN	**B**etter **A**pproach **T**o **Mobile** **A**d-hoc **N**etworking
ALFRED	**A**lmighty **L**ightweight **F**act **R**emote **E**xchange **D**aemon
GMSK	**G**aussian **M**inimum-**Shift** **K**eying
OFDM	**O**rthoganal **F**requency **D**ivision **M**ultiplexing
WAN	**W**ide **A**rea **N**etwork
LAN	**L**ocal **A**rea **N**etwork
ISP	**I**neternet **S**ervice **P**rovider
FCC	**F**ederal **C**ommunications **C**ommission
OSI	**O**pen **Systems** **Interconnection**
ISM	**I**ndustrial **S**cientific **M**edical

To my parents, Joe and Kathy McCormack for supporting me in all my endeavors.

Chapter 1

Introduction

1.1 Mesh Networks

In a traditional Wide Area Network (WAN), a user is able to connect to the internet through an Internet Service Provide (ISP). The user will almost always pay the ISP in exchange for the ability to connect to the rest of the internet. This is considered a centralized way to connect to the internet, where the users all connect through a few central points in the ISP's infrastructure. An alternative to this type of networking is multi-hop, ad-hoc, mesh networking. Mesh networks are decentralized networks. This means there is no single point of failure.

In an ad-hoc network, each radio is able to communicate directly to any other radio within its transmission range. There is no need to connect to a central router. Ad-hoc networks are defined at the physical layer (PHY), or layer 1 in the Open Systems Interconnect (OSI) model. Mesh routing takes place as part of layer 2 or 3 of the OSI model depending on the routing protocol chosen. A mesh network builds upon an adhoc network by allowing radios to retransmit any packets they receive. This allows two radios to communicate over a larger distance by leveraging other radios located inbetween the sender and receiver.

In simple mesh networking protocols, any packet sent may flood through the network to every other radio. However, with more advanced protocols an algorithm is used to ensure that a packet follows a direct path from sender and receiver and only uses a hop if necessary. The distributed nature of a mesh network creates many unique features. The decentralized nature of a mesh network prevents issues related to single points of failure. If a node goes down, the network can reconfigure and find a new path to the target.

1.2 Software Defined Radio Networks

Software Defined Radios (SDRs) are radio communication systems that utilize software to process radio frequency information in place of traditional hardware. A radio frequency frontend is able to capture and transmit signals, while the actual processing of the signal is taken care of by a digital system like general purpose processor on a traditional computer. This allows for a single piece of hardware to replace the need for multiple types of radios.

A typical cell phone can have a bluetooth, wifi, gps, and cellular radio all in a very small package. In the future, these systems could be replaced by a single SDR. SDRs are capable of using both digital and analog transmission protocols. They can use general purpose processors, digital signal processors, or FPGAs to process the RF information. Analog to Digital Converters (ADCs) are used to receive data from the antenna while Digital to Analog Converters (DACs) are used to transmit the processed signals.

As the name suggests, a Software Defined Radio Network (SDRN) is a network made up of SDRs. The networks can operate on a nearly infinite combination of center frequencies, amplitudes, bandwidths, and protocols. The flexibility of an SDRN is limited by the physical hardware capabilities of the SDR, the computation speed of the processing unit, and regulations from governing bodies like the Federal Communications Commission (FCC). Still, the flexibility of recofigurable radios leads to opportunities for advancing communications infrastructure beyond traditional protocols.

1.3 Trends Towards Cognitive Radio Environments

Cognitive Radio Networks (CRNs) are systems of SDRs that are capable of utilizing artifical intelligence and machine learning to dynamically alter transmission patterns in real time. These decisions can be made by a central server that oversees all nodes on the network, but modern systems strive to make each node capable of independent decisions.

Cognitive Radio Adhoc Networks (CRAHNs) combine software defined radios to form mesh networks. Cognitive features of the radios can allow them to change transmission parameters in accordance with link quality to ensure packets are routed properly in the mesh network. The CRs could make small changes, like increasing or decreasing their gains, in order to continue to transmit to moving nodes. They could also make larger changes, like switching entire protocols, to adapt to the needs of the network in near real time.

Beyond ensuring good throughput in a network, CRNs also solve a major issue facing the wireless world. Frequency spectrum is a finite resource, and the available bandwidth is being quickly used up. CRNs can utilize frequency hopping to share frequency resources with traditional wireless communication systems. A CRN can begin its operation on a specified band. If a traditional transmitter, usually called the primary user (PU), begins to operate on that frequency then the CRN will "hop" by switching to a different frequency and continuing operation.

1.4 ARCAM-Net

In order to begin work on SDRNs and CRAHNs, a testbed and research platform needed to be established. This thesis serves to describe the Advanced Radio Communication Ad-hoc Mesh Network, or ARCAM-Net, test bed and platform. The goal of this project is to create a open source, low cost, research platform that can serve as the basis for future work at Florida Polytechnic University. By combining well established open source tools, ARCAM-Net could become a great tool for other research groups to break into SDR and CR and get up and running in minimal time. All software used by ARCAM-Net is freely available. The only costs come from the hardware.

ARCAM-Net's architecture will be throughly discussed in this document, but the two major components are GNU Radio and Batman-adv. GNU Radio is an open source toolchain for creating digital signal processing tools to be used with SDRs. Batman-adv is a layer 2 mesh networking protocol. Both projects are community driven, but exist as separate entities. I hope that with ARCAM-Net the two projects can begin collaborating to help create next generation wireless network resources.

ARCAM-Net is designed to be a platform. Therefore all of the code will be released on github. I encourage interested users to fork the repository and begin experimenting on their own. I will also be releasing and documenting code I used to become familar with SDRs on github and in the Appendix of this document.

Chapter 2

Literature Review

2.1 SDR Fundamentals

2.2 GNU Radio and MATLAB

2.2.1 GNU Radio

2.2.2 MATLAB

2.2.3 RFNoc

2.3 Network Concepts

2.3.1 What is a Network

2.3.2 Mesh Network

2.3.3 GNU Radio Mesh Networks

2.3.4 Mesh Extension into SDR

2.3.5 What is a network protocol

OSI Model

Common Protocols

Batman

OLSRD

2.4 SDR and Cognitive Radios

2.5 State of the Practice

2.6 State of the Art

Chapter 3

Methodology

3.1 Design

The Design of the test bed can be broken down into the following parts:

- USRP Software Defined Radio
- GNU Radio Flowgraph
- Batman-adv
- Flask Web Server
- SocketIO Web Sockets
- A.L.F.R.E.D.

This configuration is shown in Figure 3.1

3.1.1 SDR

For ARCAM-Net, we utilized a combination of Ettus Research USRP B200 and USRP B210 SDRs. These radios are able to communicate from 70 MHz to 6 GHz and are well supported in GNU Radio using the open-source USRP Hardware Driver (UHD) provided by Ettus [2]. Their relatively low cost makes them ideal for building out larger testbeds. These serve as the radio transceiver for the current version of our platform. However, thanks to the UHD support in GNU Radio, any other USRP device will be compatible with the rest of the system, with little to no changes made to the development environment.

3.1.2 GNU Radio Flowgraph

GNU Radio utilizes programs called “Flowgraphs" to allow for graphical programming of SDR software. To implement the physical and link layers on the SDR, we utilize the Out of Tree (OOT) module gr-mac created by John Malsbury [4]. This flowgraph is an implementation of a Gaussian Minimum-Shift Keying (GMSK) or Orthogonal Freqency-Division Multiplexing (OFDM) transceiver with a mac layer protocol called “simple mac". There are two main blocks in the flowgraph. The first sets up the GMSK or OFDM radio. This hierarchical block is built by running a separate flowgraph which contains the

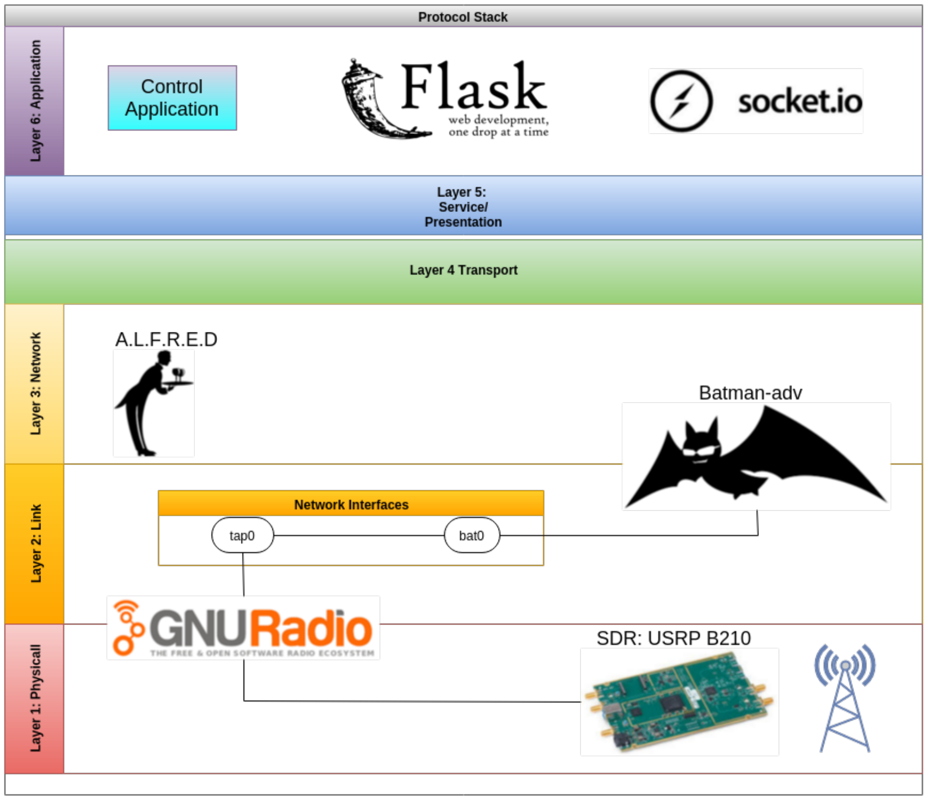


FIGURE 3.1: An overview of the components of the system, including the OSI Layers they interact with. [1] [2] [3] [4] [5] [6]

UHD blocks to interface into the USRP as well as the modulation and demodulation blocks for the waveform. One of the more important aspects of the two radio blocks, is that they convert from streaming data to message data.

Most features of GNU Radio work on streaming data where there is constant data transmission. However, packets are not sent continuously, therefore separate logic is needed to convert streams to messages. These messages are passed into and out of the GMSK and OFDM heirarchical blocks, so the remainder of the flowgraph deals with passing messages only.

We use the GMSK block to convert from streaming data to message data and then connect this block to a tunnel (TUN) or network tap (TAP) interface block. TUN/TAP devices are virtual network kernel devices supported entirely in software. TUNs are used to simulate layer 3 devices and TAPs simulate layer 2 [7]. Either of these could be selected to suit the users purpose, but as batman-adv is a layer 2 protocol, we will use the TAP protocol.

3.1.3 Batman-adv

Batman-adv was chosen based on its large community and documented success as a mesh routing protocol [8]. It is already included as part of the Linux kernel, and additional software can be downloaded from most distributions repositories [3]. Configuring batman-adv to work on the SDR involves running the program batctl and selecting the recently generated TAP interface created by GNU Radio. The Maximum Transmission Unit (MTU) of the TAP interface must also be changed to 1532 from 1500 in order to incorporate the additional header batman-adv uses when sending data. With just batman-adv and GNU Radio, we are able to create a Software Defined Radio based mesh network. The remainder of the test bed was implemented to leverage features unique to GNU Radio and batman-adv to create a method of sharing frequency and other data.

3.1.4 Flask Web Server and Socket.IO

Flask is a lightweight, open source, web framework for the Python programming language [6]. Flask was used to act as a broker between GNU Radio and any other user space applications or control systems we wished to implement. The Flask server runs the GNU Radio flowgraph in a background thread, while simultaneously configuring the TAP interface, setting up batman-adv, and starting A.L.F.R.E.D. as a background process.

Socket.IO is a JavaScript library that enables real-time bidirectional event-based communication [5]. SocketIO was chosen as a means of relaying data between the Flask server and other components of the system due to its speed, flexibility, and ability to broadcast messages to any connected client. Socket.IO also integrates into Flask [9] and can be used in stock Python with a client library [10]. In Flask, we create wrappers to all the necessary GNU Radio parameters so that external tools can relay data to and from GNU Radio over web sockets.

We also use Flask to host a single webpage that displays various settings about the radio, and allows for the user to change parameters. The interface is

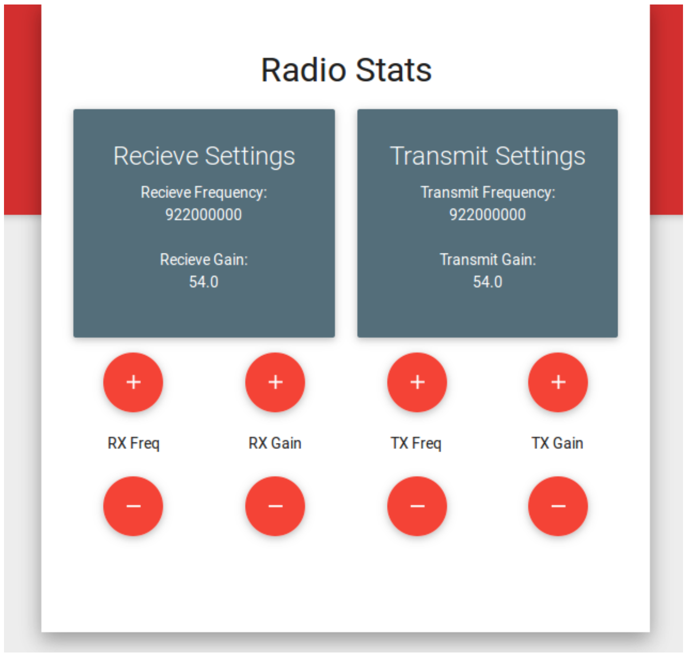


FIGURE 3.2: The web interface that lets the user initiate the network to hop to a new frequency.

shown in Figure 3.2. Since our platform does not yet include logic for automatic detection of primary users, we simulate this by allowing a person to click a button to change to a new frequency. This frequency will then be sent to the Flask server using web sockets.

3.1.5 A.L.F.R.E.D.

The “Almighty Lightweight Fact Remote Exchange Daemon," or A.L.F.R.E.D., is a system for distributing data to all nodes on a mesh network [4]. Whenever a node writes data to a channel on A.L.F.R.E.D., that data is passed between each node so that all members of the network receive the data. Typical uses for A.L.F.R.E.D. include keeping track of sensor data or building a visual map of the network.

An additional feature of A.L.F.R.E.D. is its ability to pass a command to the command line whenever new data is added. When the transmission frequency of the USRP is changed on the Flask server, Flask sends this information along with a UTC timestamp to A.L.F.R.E.D. before changing frequencies. A small delay is created so that we can be sure the information was sent to the other nodes before the node changes its broadcast frequency.

When the other nodes receive the updated data table, A.L.F.R.E.D.'s callback function will run. This is a short program that parses the A.L.F.R.E.D. data table and looks for the most recent data it received. The callback function then sends the new frequency to Flask using Socket.IO which causes Flask to change the frequency in the GNU Radio flowgraph.

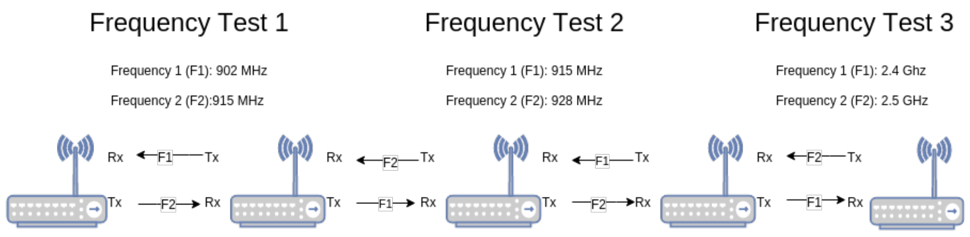

Frequency Test 1
Frequency 1 (F1): 902 MHz
Frequency 2 (F2):915 MHz
Frequency Test 2
Frequency 1 (F1): 915 MHz
Frequency 2 (F2): 928 MHz
Frequency Test 3
Frequency 1 (F1): 2.4 Ghz
Frequency 2 (F2): 2.5 GHz
Rx
Tx
F1
F2

FIGURE 3.3: The configuration used for the first set of tests.

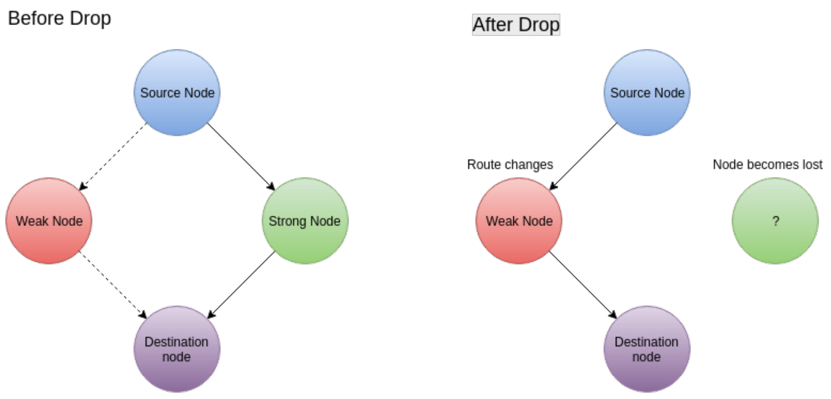


FIGURE 3.4: The configuration used for the second set of tests.

3.2 Fabrication and Build Process

The network was setup and run on a series of Lenovo S30 Computers. All computers features an Intel Xeon E5 processor and 32 GB of RAM. Each computer was running Ubuntu 14.04 LTS. The install process for GNU Radio, Batman-adv, and the rest of the technology stack is detailed in the appendix. Each computer was connected to one Ettus B200/B210 SDR using USB 3.0.

All of the computers were connected together on a 10/100/1000 Ethernet router. This allowed for the control of the entire mesh to be handled by one computer. The popular terminal multiplexer Tmux was used in conjunction with SSH to send commands to all of the nodes in parallel.

An additional Windows 10 laptop computer was used to control a Tektronix RSA306 spectrum analyzer. This computer had an Intel i7 processor, and 8 GB of RAM. It is important to note that the RSA 306 requires a Windows machine, and therefore cannot be run on the same computer as the ones being used to run ARCAM-Net.

3.3 Test - SDR (Environment/Setup)

3.4 Test Procedures

In order to characterize the platform we ran three sets of tests presented below. The first test characterizes data hopping from one node to the next. The second test demonstrates batman-adv's ability to switch routes based on the quality of each node. The final tests were used to examine A.L.F.R.E.D.'s ability to be used for exchanging frequency information from node to node.

3.4.1 Network Benchmarks

The first test was used to investigate the overhead each node adds to the network. To examine how adding hops affects the network, we arranged the USRPs in a line so that each node only had one route to the following nodes. We used a total of 5 nodes as shown in Figure 3.3. We staggered the transmit and receive frequencies of the nodes to ensure that nodes could only talk to their immediate neighbors, forcing the network into the proper configuration. The staggering of frequencies was needed to ensure each node would be unable to communicate to nodes other than its neighbors.

We then tested the setup at three different sets of frequencies, all within the Industrial, Scientific, and Medical (ISM) Band. We used different sets of ping tests in order to determine the number of dropped packets and the time it took to send the packets. We ran a standard ping test, one with reduced packet sizes, and one with increased time to live (TTL) settings. For a control group, two USRPs were connected together without batman-adv running.

3.4.2 Route Changes

In a typical mesh environment, there will usually be more than one route from a source to a destination [11]. In a traditional network, batman-adv switches routes based on the quality of each available link. The test was designed to see if the same features would work in an SDRN. We initially setup four SDRs: a source, a destination, and two nodes to connect them. We give a significantly larger gain to the first node, in order to see if batman-adv will recognize that this is a stronger path to the destination. Then, once the route is placed into the routing table, we lower the gain to 0 in order to force a transition to the other node. We then see if batman-adv is still able to find the new route. This setup is shown in Figure 3.4.

3.4.3 Frequency Distribution

In the final test, we tested to see if A.L.F.R.E.D. would properly relay frequency changes over the mesh to other nodes. The user would increase or decrease the frequency using the web interface in order to simulate a cognitive radio making a decision to change to a new frequency. If A.L.F.R.E.D. was able to exchange the information properly, then the routing table would still show all connected nodes. We also used a Tektronix RSA306 Spectrum Analyzer in order to see that the transmission was occurring on the new frequency. If A.L.F.R.E.D. was not able to relay the information to all nodes, then some would change to the new frequency while others remain. This would be reflected in the output from the spectrum analyzer.

3.4.4 Error (inaccuracy)

3.4.5 Energy Use

3.5 Test - Network

3.5.1 Signal Strength

3.5.2 Range

3.5.3 Noise

3.5.4 Loss

3.5.5 Error

3.5.6 Energy Efficiency

3.5.7 Latency

3.5.8 Throughput

3.5.9 Topology Effects

3.5.10 Protocol

3.5.11 Interference

3.5.12 Crowding

3.5.13 Crosstalk

Chapter 4

Findings

4.1 Results

4.1.1 Network Benchmarks

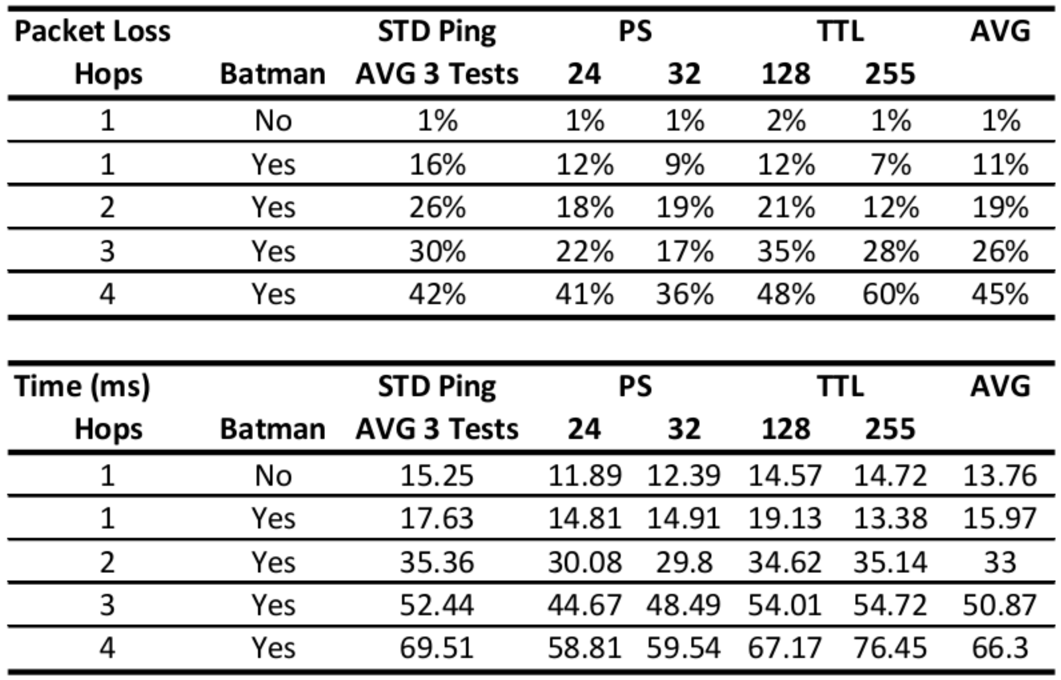
The results of the Network Benchmark tests from section 4 part A are summarized in Figure 4.3. In all cases, the single point to point communication, without the batman-adv protocol running, resulted in a much lower packet loss. This served as our control group. However, in the two sets of lower frequency ratings, the packet loss remained below 50%. Also, the increase in time as hops were added has a roughly linear change. This means the overhead of adding more hops is not unmanageable. A full listing of tests run for the 902/915 MHz, and 2.4/2.5 GHz cases are provided in Figure 4.1 and Figure 4.2 respectively. These tables show that running at the higher frequencies causes the SDRN to drop a lot more packets, especially when moving through the full four hops.

4.1.2 Route Changes

The route changing feature of batman-adv showed success with our test setup from section 4 part B. Initially, batctl reported two possible links, one with a link quality of 55, and the other with a link quality of 18. As we decreased the gain of the intermediate node, the link quality reported by batctl also decreased. Eventually, batman-adv switched and began using the other node. At this point, it no longer saw the original node, and reported a link quality of 75 on the alternate one. The initial setup can be seen in Figure 4.4. After the change, the routing table appeared as it does in Figure 4.5. This feature works in the SDR system, and can continue to be used without significant changes.

4.1.3 Frequency Changes

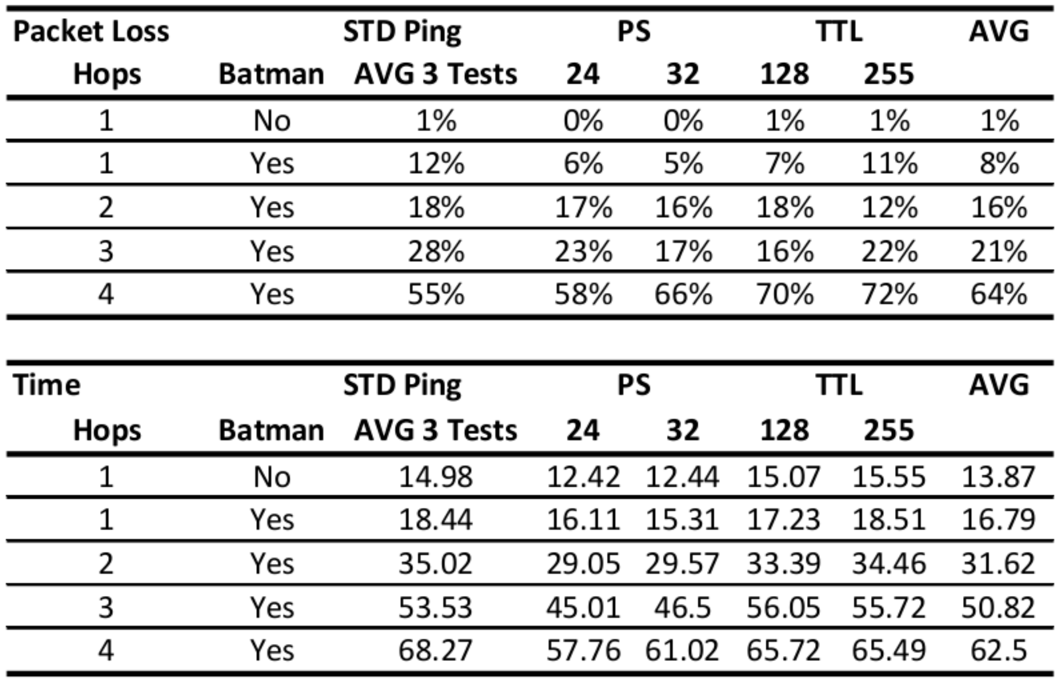
Using A.L.F.R.E.D. to distribute frequency hopping showed mixed results. Using the setup from section 4 part C, we were able to get the nodes to change frequency in unison, but not reliably. A.L.F.R.E.D. itself is designed for a traditional Wi-Fi environment, and therefore does not have an expectation that the other nodes will become completely unreachable due to changes in operating frequency [4]. We were finding that some nodes would switch before A.L.F.R.E.D. had propagated the data table to the other nodes. This would



Packet Loss		STD Ping	PS		TTL		AVG
Hops	Batman	AVG 3 Tests	24	32	128	255	
1	No	1%	1%	1%	2%	1%	1%
1	Yes	16%	12%	9%	12%	7%	11%
2	Yes	26%	18%	19%	21%	12%	19%
3	Yes	30%	22%	17%	35%	28%	26%
4	Yes	42%	41%	36%	48%	60%	45%

Time (ms)		STD Ping	PS		TTL		AVG
Hops	Batman	AVG 3 Tests	24	32	128	255	
1	No	15.25	11.89	12.39	14.57	14.72	13.76
1	Yes	17.63	14.81	14.91	19.13	13.38	15.97
2	Yes	35.36	30.08	29.8	34.62	35.14	33
3	Yes	52.44	44.67	48.49	54.01	54.72	50.87
4	Yes	69.51	58.81	59.54	67.17	76.45	66.3

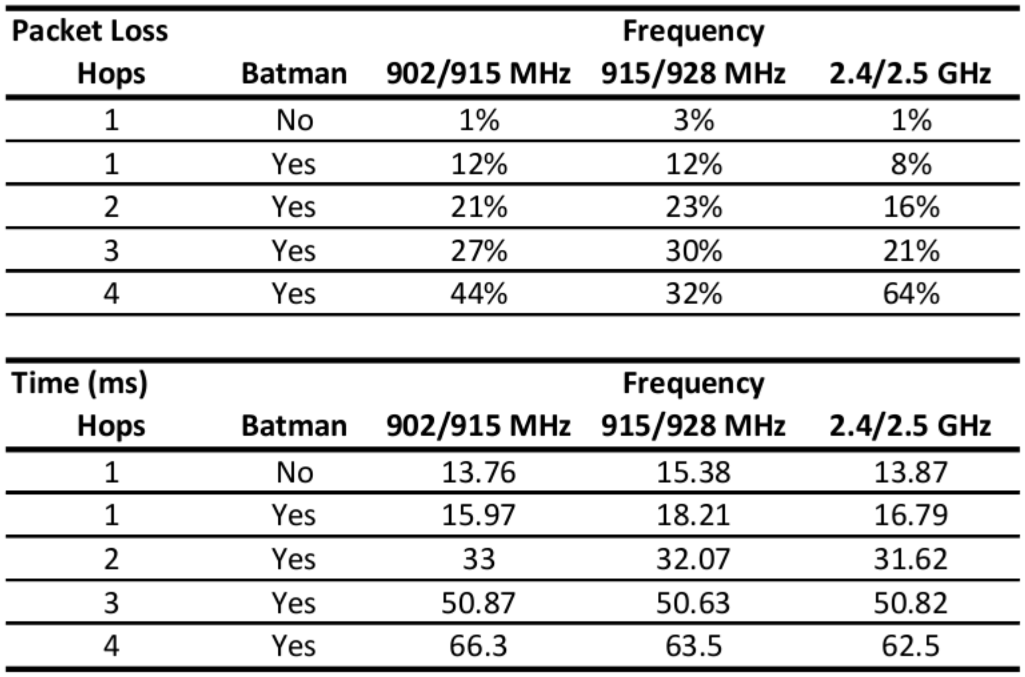
FIGURE 4.1: The data received from operating at 902/915 MHz.



Packet Loss		STD Ping	PS		TTL		AVG
Hops	Batman	AVG 3 Tests	24	32	128	255	
1	No	1%	0%	0%	1%	1%	1%
1	Yes	12%	6%	5%	7%	11%	8%
2	Yes	18%	17%	16%	18%	12%	16%
3	Yes	28%	23%	17%	16%	22%	21%
4	Yes	55%	58%	66%	70%	72%	64%

Time		STD Ping	PS		TTL		AVG
Hops	Batman	AVG 3 Tests	24	32	128	255	
1	No	14.98	12.42	12.44	15.07	15.55	13.87
1	Yes	18.44	16.11	15.31	17.23	18.51	16.79
2	Yes	35.02	29.05	29.57	33.39	34.46	31.62
3	Yes	53.53	45.01	46.5	56.05	55.72	50.82
4	Yes	68.27	57.76	61.02	65.72	65.49	62.5

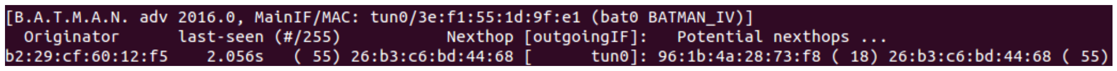
FIGURE 4.2: The data received from operating at 2.4/2.5 GHz.



Packet Loss		Frequency		
Hops	Batman	902/915 MHz	915/928 MHz	2.4/2.5 GHz
1	No	1%	3%	1%
1	Yes	12%	12%	8%
2	Yes	21%	23%	16%
3	Yes	27%	30%	21%
4	Yes	44%	32%	64%

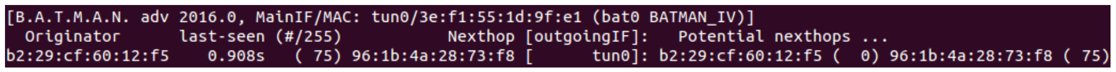
Time (ms)		Frequency		
Hops	Batman	902/915 MHz	915/928 MHz	2.4/2.5 GHz
1	No	13.76	15.38	13.87
1	Yes	15.97	18.21	16.79
2	Yes	33	32.07	31.62
3	Yes	50.87	50.63	50.82
4	Yes	66.3	63.5	62.5

FIGURE 4.3: The Averages from all three tests.



```
[B.A.T.M.A.N. adv 2016.0, MainIF/MAC: tun0/3e:f1:55:1d:9f:e1 (bat0 BATMAN_IV)]
  Originator      last-seen (#/255)           Nexthop [outgoingIF]:   Potential nexthops ...
b2:29:cf:60:12:f5    2.056s   ( 55) 26:b3:c6:bd:44:68 [      tun0]: 96:1b:4a:28:73:f8 ( 18) 26:b3:c6:bd:44:68 ( 55)
```

FIGURE 4.4: The initial condition from the test presented in Section 4 B, where there are two possible routes the packet can take. The output is from running batctl. The link quality is listed under the column labeld (#/255). A higher number represents a better quality connection.



```
[B.A.T.M.A.N. adv 2016.0, MainIF/MAC: tun0/3e:f1:55:1d:9f:e1 (bat0 BATMAN_IV)]
  Originator      last-seen (#/255)           Nexthop [outgoingIF]:   Potential nexthops ...
b2:29:cf:60:12:f5    0.908s   ( 75) 96:1b:4a:28:73:f8 [      tun0]: b2:29:cf:60:12:f5 (  0) 96:1b:4a:28:73:f8 ( 75)
```

FIGURE 4.5: After the gain is reduced in the test presented in Section 4 B, the packets are now routing through a different node.

leave one node with an out-of-date table, meaning it would not make the frequency change. In the current iteration of the project, there is no way for these orphaned nodes to find the rest of the network again. Figure 5.1 shows a situation in which four out of five nodes were able to make the jump, with one node remaining at the original frequency.

4.2 Data Analysis

4.3 Assumptions

4.4 Discussion of Error

4.5 Error Bars

Chapter 5

Discussion

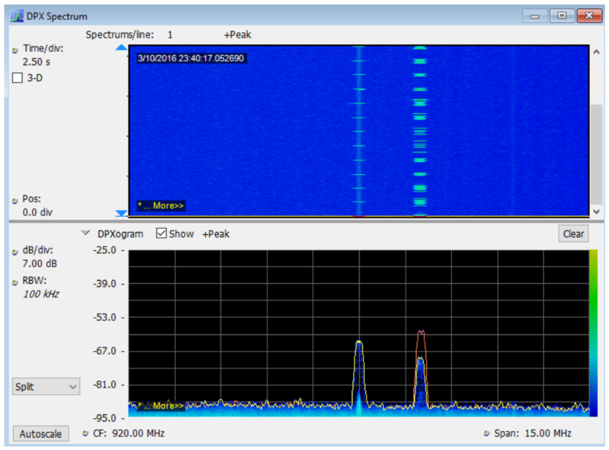
5.1 Constraints and Limitations

The results from our experiments show that the network is functioning as a multi hop SDRN. In addition to the experiments performed, we were also able to use Secure Shell (SSH) and Secure Copy (SCP) over multiple hops of the SDRN. However, it is clear that more work needs to be done. In a deployed network, packet loss as high as is seen in this network is not optimal. Therefore, it is important to examine ways to mitigate packet loss and increase throughput. For example, machine learning or artificial intelligence algorithms could be used to adjust transmission parameters as issues are detected. It is likely that a change in frequency or amplitude could mitigate some of the packet loss. For example, if the loss is due to crowding near one node, frequency hopping could be employed to shift to better operating conditions. Batctl's link quality metric, as shown in Figures 4.4 and 4.5 could help with finding weak nodes and making decisions. This would be the beginning of ARCAM-Net's transition from a SDRN to a Cognitive Radio Network (CRN).

Furthermore, it would be beneficial to either improve upon A.L.F.R.E.D. or implement certain features in a new way in order to handle the frequency changing. If we have each node wait for an acknowledgment from its immediate neighbors before changing frequency, that node could then change its frequency knowing that the data will propagate to the rest of the network. Batctl is able to report the immediate next hop neighbors, so the program could use this information to only wait for acknowledgment from neighbors instead of waiting for the entire network to be ready to change. In order for the current A.L.F.R.E.D. setup to function, a delay was needed to give the network time to respond. Therefore, an asynchronous acknowledge would likely speed up the frequency change.

5.2 SDR Network

5.3 Cognitive Networks



Spectrums/line: 1

Time/div: 2.50 s

Pos: 0.0 div

3/10/2016 23:40:17.052690

dB/div: 7.00 dB

RBW: *100 kHz*

CF: 920.00 MHz

Span: 15.00 MHz

FIGURE 5.1: The result of using ALFRED to shift frequencies. One node is left behind as the others move to the new channel. Image collected using the Tektronix RSA306 Spectrum Analyzer

Chapter 6

Conclusion

6.1 Summary

6.2 Conclusion

This work represents a first step in a longer term project to create a fully functioning SDRN. The work demonstrates the potential for using GNU Radio in conjunction with batman-adv and other Open-Mesh solutions. As the work continues forward, we hope the testbed can serve as a collaboration point between GNU Radio and Open-Mesh. Both represent next generation, open source, wireless solutions and could likely benefit from collaboration between the two projects. Our work can be used as an excellent starting point for anyone looking to get an SDRN up and running quickly to begin prototyping other sections of the tool chain. We plan on releasing the code as well as a handful of tools to help other researchers get started. We hope that this will serve as the first step in creating an equivalent Cognitive Radio Network platform and testbed.

6.3 Recommendation

6.4 Future Work

Appendix A

Running ARCAM-Net

This appendix will serve as a "How-to" manual for getting started with ARCAM-Net.

A.1 Configuring the Computer

First, Ubuntu 14.04 LTS will need to be installed on each node computer. Installing Ubuntu is beyond the scope of this document. However, there are plenty of easy to follow tutorials available online.

Once installed, the remaining steps can be done manually, or you can use a shell script I created to automate most of the process. This shell script is available on Github. As the name of the repository suggests, you can also use this to setup and configure a virtual machine to run the SDR software using Vagrant and Virtual Box. If you are unable to install an OS on the computers you are using, this is a good alternative. Please be aware that both options involve a lot of downloading and compiling of software. This process can take up to 8 hours so be sure to have plenty of time to let this run. An internet connection is required.

A.1.1 Installing Natively

```
git clone https://github.com/jmccormack200/GnuRadio-Vagrant-Script.git
cd GnuRadio-Vagrant-Script sudo sh bootstrap.sh
```

A.1.2 Installing Through Vagrant

Download and install Vagrant and Virtualbox to your computer. Then run:

```
vagrant up
```

The rest should be configured automatically.

A.1.3 Manual Install

If you prefer to do things yourself just be sure to follow along with the "bootstrap.sh" file below:

LISTING A.1: Installer Shell Script

```
#!/usr/bin/env bash
```

```
apt-get update

#Everything below here is mentioned as being a prereq for
#GNURadio except Vim, Vim is just swell. Love Vim.
apt-get --yes --force-yes install python
apt-get --yes --force-yes install vim
apt-get --yes --force-yes install cmake
apt-get --yes --force-yes install python-pip
apt-get --yes --force-yes install libboost-all-dev
apt-get --yes --force-yes install libcppunit-dev
apt-get --yes --force-yes install git
apt-get --yes --force-yes install libfftw3-dev libfftw3-doc
apt-get --yes --force-yes install libgsl0ldbl
apt-get --yes --force-yes install swig
apt-get --yes --force-yes install libsdl2-2.0
apt-get --yes --force-yes install python-wxgtk2.8
apt-get --yes --force-yes install build-essential
apt-get --yes --force-yes install mesa-common-dev
apt-get --yes --force-yes install qt-sdk
apt-get --yes --force-yes install libqwt-dev
apt-get --yes --force-yes install python-numpy python-scipy python-matplotlib
apt-get --yes --force-yes install python-setuptools
apt-get --yes --force-yes install python-gtk2-dev
apt-get --yes --force-yes install doxygen
apt-get --yes --force-yes install doxygen-gui
apt-get --yes --force-yes install graphviz
apt-get --yes --force-yes install git-core
apt-get --yes --force-yes install g++
apt-get --yes --force-yes install python-dev
apt-get --yes --force-yes install pkg-config
apt-get --yes --force-yes install python-cheetah
apt-get --yes --force-yes install python-lxml
apt-get --yes --force-yes install libxi-dev
apt-get --yes --force-yes install python-sip
apt-get --yes --force-yes install libqt4-opengl-dev
apt-get --yes --force-yes install libfontconfig1-dev
apt-get --yes --force-yes install libxrender-dev
apt-get --yes --force-yes install python-sip
apt-get --yes --force-yes install python-sip-dev
apt-get --yes --force-yes install libboost-all-dev
apt-get --yes --force-yes install libusb-1.0-0-dev
apt-get --yes --force-yes install python-mako
apt-get --yes --force-yes install python-docutils
apt-get --yes --force-yes install xinitx
apt-get --yes --force-yes install batctl
apt-get --yes --force-yes install gnome
apt-get --yes --force-yes install xinit
```

```
sudo pip install pyside

#These were to allow for qt to work, but seem to be errors
export PYTHONPATH=$PYTHONPATH:/opt/qt/lib/python2.7/dist-package
export PYTHONPATH=$PYTHONPATH:/usr/bin/python


#remove standard packages of UHD if they are there
sudo apt-get --yes --force-yes remove libuhd-dev
sudo apt-get --yes --force-yes remove libuhd003
sudo apt-get --yes --force-yes remove uhd-host

#Install UHD Using Ettus Repos
sudo bash -c 'echo "deb␣http://files.ettus.com/binaries/uhd/repo/uhd/ubunt
sudo apt-get update
sudo apt-get --yes --force-yes install -t 'lsb_release -cs' uhd

git clone --recursive http://git.gnuradio.org/git/gnuradio.git

#Here we build gnuradio from sources
#This will be the longest process
cd gnuradio
mkdir build
cd build
cmake ..
sudo make
sudo make install

sudo ldconfig


##Now we will download gr-mac
cd ..
cd ..
git clone https://github.com/balint256/gr-mac.git
cd gr-mac
mkdir build
cd build
cmake ..
sudo make
sudo make install

sudo ldconfig

##Which also needs gr-foo
cd ..
cd ..
git clone https://github.com/bastibl/gr-foo.git
cd gr-foo
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig


#RTL Driver
cd ..
cd ..
git clone git://git.osmocom.org/rtl-sdr.git
cd rtl-sdr
mkdir build
cd build
cmake ../ -DINSTALL_UDEV_RULES=ON
sudo make
sudo make install
sudo ldconfig

#RTL-GNU Radio Pluging
cd ..
cd ..
git clone git://git.osmocom.org/gr-osmosdr
cd gr-osmosdr/
mkdir build
cd build/
cmake ../
make
sudo make install
sudo ldconfig


cd ..
cd ..
git clone https://github.com/btrowbridge/alfred-ubuntu.git
cd alfred-ubuntu
sudo bash ubuntuAlfredInstall.sh
```

A.2 Getting The Flowgraphs

TODO once the flowgraphs are in their own repo.

A.3 Running The Flowgraphs

TODO

A.3.1 Just the Mesh Network

A.3.2 Raising the Bat Signal

A.3.3 TMux Script

A.3.4 Full Web Server

Bibliography

[1] (2016) Welcome to gnu radio. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki

[2] (2016) Uhd (usrp hardware driver). [Online]. Available: https://www.ettus.com/sdr-software/detail/usrp-hardware-driver

[3] (2016) Open-mesh. [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki

[4] (2014) A.l.f.r.e.d - almighty lightweight fact remote exchange daemon. [Online]. Available: https://www.open-mesh.org/projects/alfred/wiki

[5] (2016) Socket.io. [Online]. Available: http://socket.io/

[6] (2016) Flask (a micro framework). [Online]. Available: http://flask.pocoo.org/

[7] (2015) Using openwrt as an openvpn server with a tap device (with bridging). [Online]. Available: https://wiki.openwrt.org/doc/howto/vpn.server.openvpn.tap

[8] M. Abolhasan, B. Hagelstein, and J. C. P. Wang, "Real-world performance of current proactive multi-hop mesh protocols," in *Communications, 2009. APCC 2009. 15th Asia-Pacific Conference on*, Oct 2009, pp. 44–47.

[9] (2016) Flask-socket.io. [Online]. Available: https://flask-socketio.readthedocs.org/en/latest/

[10] (2015) socketio-client. [Online]. Available: https://pypi.python.org/pypi/socketIO-client

[11] I. F. Akyildiz, W.-Y. Lee, and K. R. Chowdhury, "Crahns: Cognitive radio ad hoc networks," *Ad Hoc Networks*, vol. 7, no. 5, pp. 810 – 836, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S157087050900002X