ARCAM-Net: A Software Defined Radio Mesh Network Testbed Implemented in GNU Radio with Batman-adv

John McCormack*, Bradley Trowbridge†, Joseph Prine*, R. Cody Maden†, Ryan Integlia*†
*College of Engineering, † College of Information and Technology
Florida Polytechnic Univeristy
Lakeland, FL, U.S.A.
{johnmccormack2307, jprine2716, bradleytrowbridge06, randallmaden1497, rinteglia}@flpoly.org

***Abstract*—Software Defined Radio Networks (SDRNs) use systems of Software Defined Radios (SDRs) to establish networks with flexible physical and link layers. GNU Radio is an open source software tool set for working with SDRs and can be used as a basis for creating SDRNs. Mesh networks are designed to allow for flexible and distributed network architectures that are self forming and function without the need for centralized infrastructure. Batman-adv is a popular, open source, layer 2 mesh network protocol.**

In this paper we present the Advanced Radio Communication Adhoc Mesh Network (ARCAM-NET) Platform. Our work establishes an SDRN platform by combining GNU Radio with Batman-adv to create a fully open source software defined radio mesh network. The platform can work with USRP SDR devices to quickly prototype and potentially explore with SDR and Cognitive Radio (CR) Protocols. Due to the flexibility of Batman-adv and GNU Radio, programs acting above Layer 2 can utilize this network without any changes. In order to further increase the cognitive abilities of the platform, we explore using the A.L.F.R.E.D. tool chain within the batman-adv ecosystem to distribute information about frequency changes across the mesh network. This creates a novel method to globally change the frequency of the network in a completely decentralized way.

***Index Terms*—Software Defined Radio, ad-hoc network, mesh network, Software Defined Radio Network, SDR, SDRN, Batman-adv, USRP, GNU Radio**

I. Introduction

As the cost of SDRs continues to drop, the technology becomes much more accessible. Open source tools such as GNU Radio make developing for SDRs much easier [1]. GNU Radio provides a feature rich ecosystem that provides a wealth of signal processing blocks. Though hardware is not a direct component in GNU Radio, numerous other developers and projects have integrated hardware functionality into the system either natively or through additional out-of-tree modules [2] [3].

GNU Radio Companion allows for GUI development of PHY and MAC layer protocols within the GNU Radio environment.The project itself is implemented in a combination of Python and C++ modules [1]. Ettus research, a division of National Instruments [4], created the Universal Software Radio Peripheral (USRP) which is the SDR we chose for the project and a popular choice among many projects.Ettus also released the Universal Hardware Drivers (UHD) [5] which allow for the use of the USRP with GNU Radio.

Cognitive Radio Networks (CRNs) are networks made up of SDRs that are capable of sensing their environment, making decisions, and changing transmission parameters. Many Cognitive Radio scenarios are designed around the idea of an ad-hoc or mesh networks. In these networks, all of the associated radio components are able to talk to each other either directly or by "hopping" from one node to another until they reach their destination. The Open-Mesh project created the Batman-adv protocol which allows for the formation of multihop mesh networks [6]. This layer 2 protocol has a fairly large community and is integrated into the Linux Kernel.

Our goal with this project was to create a low cost, open source platform, that could serve as a basis for future projects in cognitive radio and mesh network research. Our hope is that by combining the GNU Radio and Batman-adv projects, more collaborative research can be done with a standarized toolset. Our platform will allow for students who are looking to create application layer products that leverage cognitive radio meshes to get started without having to worry about all the intracasies of these networks. We also hope to encourage both open source projects to work collaboratively in creating next generation wireless systems.

II. Related Work

A. Software Defined Radio Testbeds

There are several well known Software Defined Radio testbeds in use at different Universities. One major platform is the WARP platform from Rice University. This platform is made up of many custom components including the radio hardware itself [7]. This makes the platform very expensive and limits its adaptability for use in other research facilities.

Another platform is the Hydra platform developed at UT Austin. This platform uses GNU Radio to define PHY Layer parameters and the Click Modular Router to implement Layer 2 protocols [8]. The Hydra platform also uses USRP radios as the hardware frontend. However, Click is an older software which, according to their website, has not had an updated release since 2011 [9]. GNU Radio has since added the

Polymorphic Tree (PMT) and Message types to allow for more Layer 2 development to be done right inside GNU Radio [10].

The ADROIT project was another platform developed in conjunction with DARPA. This project relied heavily on Click and GNU Radio for much of its functionality. [11] Similarly, the University of California, Irvine and Boeing Corporation developed a testbed based off of USRP Radios and GNU Radio, but they implement custom MAC layers [12].

The platform that most closely resembles ours is presented in [13]. However, this platform uses OLSR which has been shown to perform poorly when compared to Batman-adv. OLSR is also not truly decentralized as only certain nodes relay network information [14].

B. GNU Radio and Mesh Networks

There has been work done in the past on establishing Mesh Networks using GNU Radio, however most of the research has had a different focus than ours. In [15] and [16] the authors use GNU Radio as a way to verify the succesful use of algorithms for mesh networking. However, they do not use SDRs,instead choosing to use GNU Radio for simulation.

The researchers in [17] created a simple multihop test bed using three USRP radios to relay data from one computer to another. A forth USRP acts as a primary user and attempts to block the signal. However, their work focuses on using reinforcement learning to allow for frequency hopping instead of focusing on a mesh routing protocol that would be able to scale to more radios.

Much of the existing work done using USRPs and GNU Radio for SDR Mesh Networks revolves around implementing different parts of the OSI protocol stack from the ground up. In some papers the authors focus on the physical or mac layer [18]. There has also been work in developing new higher layer protocols for cognitive radio mesh networks such as work done to replace TCP with a more robust protocol [19]. These systems will usually react to frequency changes but some also change their topology based on power use [20]. All of this research is extremely important, does not create a complete platform to serve as the backbone for future work.

C. Mesh Network Testbeds

The CONFINE platform uses Batman-adv as the routing protocol for their mesh network testbed. However,this testbed does not utilize GNU Radio or SDRs. [21] Batman-adv was also a key component of WiBed, a project to create a COTS mesh test bed using low cost wirelessrouters [22] [23]. Though they are not specific to SDR platforms, these still show the usefulness of Batman-adv as a component of a testbed.

III. 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

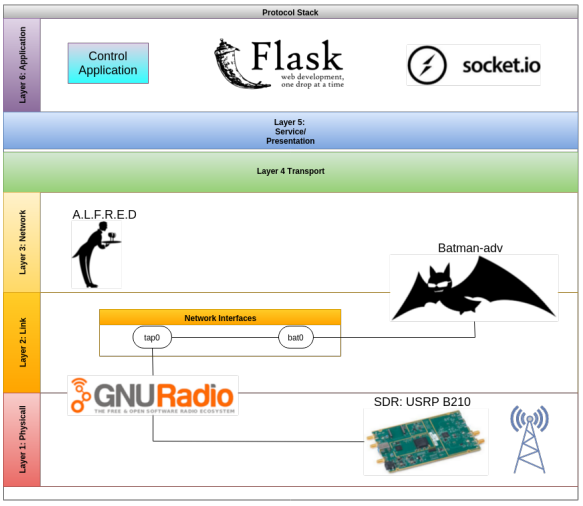

Fig. 1. An overview of the components of the system, including the OSI Layers they interact with. [1] [6] [26] [25] [24]

- SocketIO Web Sockets
- A.L.F.R.E.D.

This configuration is shown in Figure 1

A. USRP Software Defined Radio

For our project, 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. Their relatively low cost makes them ideal for building out larger testbeds. These serve as the radio transcievers 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.

B. GNU Radio Flowgraph

GNU Radio utilizes programs called “Flowgraphs” to allow for graphical programming of 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 [26]. This flowgraph is a very simplistic, but effective, implementation of a GMSK or OFDM transciever 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 heirarchical block is built by running a separate flowgraph which contains the 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 in that portion of the flowgraph. However, packets are not sent continuously so separate logic is needed to convert streams to messages. These

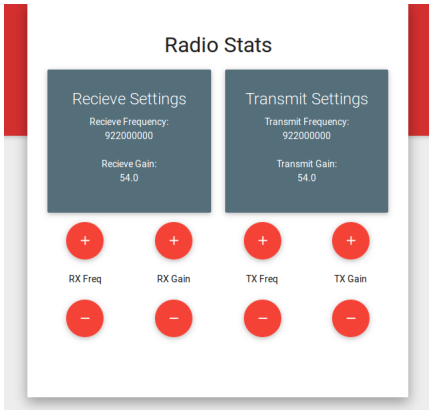

Fig. 2. The web interface that lets the user initiate the network to hop to a new frequency.

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. 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. This flowgraph also implements an 802.3 Tracker to build out a radio to address map of the network.

C. Batman-adv

Batman-adv was chosen based on its large community and documented success as a mesh routing protocol [14]. It is already included as part of the Linux Kernel, and additional software can be downloaded from most distributions repositories [6]. 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 in an effort to make cognitive radio testing much simplier.

D. Flask Web Server and Socket.IO

Flask is a lightweight, open source, web framework for the Python programming language [24]. 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. The only input needed from the user is an IP address for the TAP port, but this could later be replaced by running a DHCP server on the mesh network.

Socket.IO is a javascript library that enables real-time bidirectional event-based communication [25]. 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 easily into Flask [27] and can be used in stock python with a client library [28]. 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 verious settings about the radio, and allows for the user to change parameters. The interface is shown in Figure 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. Flask recieves the request, and then uses A.L.F.R.E.D. to manage the next step.

E. 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 [26]. A.L.F.R.E.D. is very simple to use, but still powerful. Whenver a node writes data to a channel on A.L.F.R.E.D., that data is passed from node to node to pass the data to all other members of the network. 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 recieved 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 recieved. 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.

IV. Test Procedures

In order to characterize the platform we run three sets of tests presented below. The first test characterizes data hopping from one node to the next. The second test demonstrates batman-adv 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.

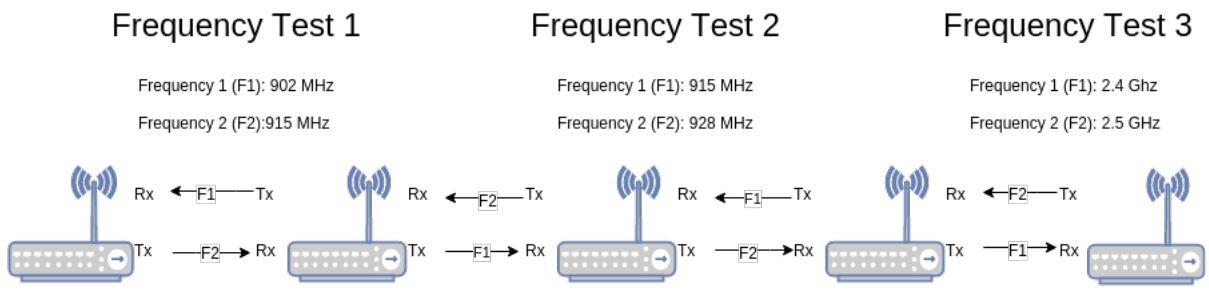

Fig. 3. The configuration used for the first set of tests.

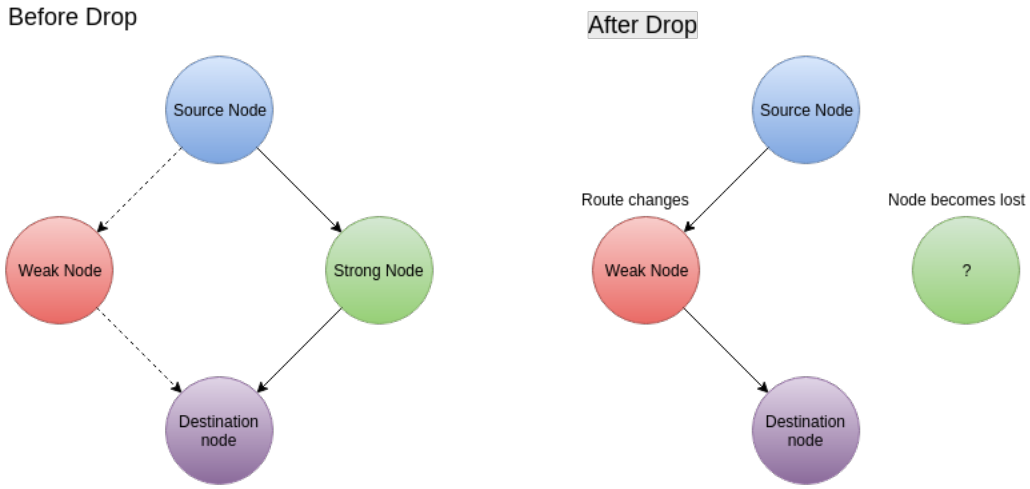

Fig. 4. The configuration used for the second set of tests.

A. Network Benchmarks

For the first test we wanted to investigate the overhead each node adds to to the network. To test this, we arranged the USRPs in a line so that each node only had one route to the following nodes. This setup is shown in Figure 3. To ensure that nodes could only talk to their immediate neighbors, we staggered the transmit and recieve frequencies in order to force the network into the configure we wanted. We then tested the setup at three different sets of frequenies, all within the 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 used a total of 5 nodes. To get a baseline for the test, we setup two usrps and did not configure batman-adv on these nodes. This would help to get a clearer understanding of what the hardware limitations are of the USRPs.

B. Route Changes

In a typical mesh environment, there will usually be more than one route from a source to a destination. In a traditional network, Batman-adv does a good job of switching routes based on the quality of links. This test was used to see if the same features would work in a cognitive environment. We initially setup four SDRs, a source, a destination, and two nodes that connect them. We give a much 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 place into the routing table, we lower the gain to 0 and make it disappear. We then see if batman-adv is still able to find the new route even though it is running on a USRP. This setup is shown in Figure 4.

C. Frequency Changes

In the final test, we looked to see if A.L.F.R.E.D. would properly relay frequency information in order to keep the network communicating as the frequency increased and decreased. 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 setting. If A.L.F.R.E.D. was able to exchange the information properly, then the network would still show all connected nodes, and the new frequency should be visible on the spectrum analyzer. If A.L.F.R.E.D. was not able to relay the information to all nodes, then we would lose nodes as some change to the new frequency while others remain. This would be reflected in the output from the spectrum Analyzer.

V. Results

A. Network Benchmarks

The results of the Network Benchmark tests are summarized in Figure 7. In all cases, a single point to point communication,

Packet Loss	STD Ping			PS		TTL		AVG
	1	2	3	24	32	128	255	
No Batman	1%	1%	1%	1%	1%	2%	1%	1%
1 Hop	12%	19%	16%	12%	9%	12%	7%	12%
2 Hop	27%	26%	24%	18%	19%	21%	12%	21%
3 Hop	32%	32%	26%	22%	17%	35%	28%	27%
4 Hop	43%	39%	44%	41%	36%	48%	60%	44%
Time	STD Ping			PS		TTL		AVG
	1	2	3	24	32	128	255	
No Batman	15.308	15.351	15.079	11.889	12.386	14.57	14.723	14.19
1 Hop	17.526	17.76	17.611	14.815	14.912	19.133	13.382	16.45
2 Hop	34.23	36.65	35.214	30.083	29.803	34.616	35.142	33.68
3 Hop	57.013	50.78	49.532	44.666	48.487	54.002	54.719	51.31
4 Hop	69.812	69.866	68.838	58.814	59.537	67.17	76.448	67.21

Fig. 5. The data received from operating at 902/915 MHz.

Packet Loss	STD Ping			PS		TTL		AVG
	0	2	3	24	32	128	255	
No Batman	0%	2%	1%	0%	0%	1%	1%	1%
1 Hop	22%	4%	10%	6%	5%	7%	11%	9%
2 Hop	18%	22%	14%	17%	16%	18%	12%	17%
3 Hop	29%	39%	16%	23%	17%	16%	22%	23%
4 Hop	41%	57%	67%	58%	66%	70%	72%	62%
Time	STD Ping			PS		TTL		AVG
	0	2	3	24	32	128	255	
No Batman	15.027	14.923		12.424	12.441	15.068	15.55	14.08
1 Hop	19.981	17.292	18.033	16.109	15.314	17.225	18.505	17.08
2 Hop	37.812	33.573	33.679	29.05	29.567	33.393	34.46	32.29
3 Hop	55.757	52.253	52.57	45.011	46.501	56.048	55.717	51.35
4 Hop	67.997	71.425	65.375	57.763	61.002	65.717	65.494	64.46

Fig. 6. The data received from operating at 2.4/2.5 GHz.

without the Batman-adv protocol running, resulted in a much lower packet loss. However, it is intersting to see that in the two sets of lower frequency raitings, the packet loss remained below 50%. Also, the increase in time as hops were added has a roughly linear change. This is good as it 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 5 and Figure 6 respectively. These tables show that running at the higher frequencies causes the SDRs to drop a lot more packets, especially when moving through the full four hops.

B. Route Changes

The route changing feature of Batman-adv worked very well with out test setup. 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. The initial setup can be seen in Figure 8. After the change, the routing table appeared as it does in Figure 9. This feature seems to work well even in the SDR system, and can likely be without much change.

C. Frequency Changes

Using A.F.R.E.D. for frequency hopping showed mixed results. We were able to get the Nodes to change frequency in unison, but not reliably. A.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. We were finding that often times the nodes would switch well before A.F.R.E.D. had propagated the datatable to

```
[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)
```

Fig. 8. The initial condition, where there are two possible routes the packet can take.

```
[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)
```

Fig. 9. After the gain is reduced, the packets are now routing through a different node.

Packet Loss		Frequency	
	902/915 MHz	915/928 MHz	2.4/2.5 GHz
No Batman	1%	3%	1%
1 Hop	12%	12%	12%
2 Hops	21%	23%	22%
3 Hops	27%	30%	23%
4 Hops	44%	32%	62%
Time		Frequency	
	902/915 MHz	915/928 MHz	2.4/2.5 GHz
No Batman	14.19	14.17	14.17
1 Hop	16.45	18.21	17.33
2 Hops	33.68	32.07	32.65
3 Hops	51.31	50.63	51.5
4 Hops	67.21	63.5	65.69

Fig. 7. The Averages from all three tests.

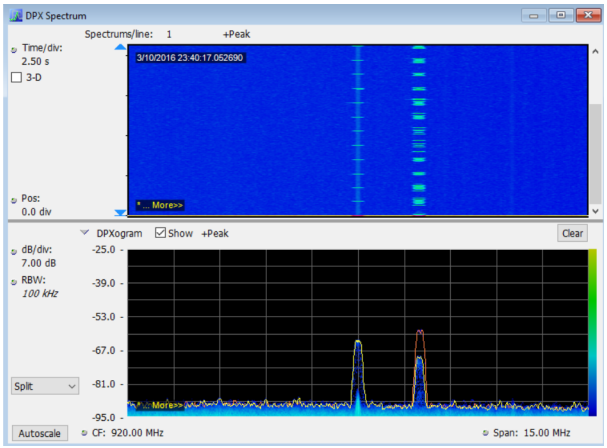

Fig. 10. The result of using ALFRED to shift frequencies. One Node is left behind as the others move to the new channel.

the other nodes. This would 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 was no way for these orphaned nodes to find the rest of the network again. Figure 10 shows a situation in which four out of five nodes were able to make the jump, with one node remaining at the original frequency.

VI. Limitations and Future Work

The results from our experiments show that the network is functioning as a multi hop 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 would not be tolerated. Therefore, an important next step would be to examine how we can utilize machine learning and artificial intelligence to adjust parameters when these large packet losses 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 too many nodes operating on the same frequency, that set of nodes could change to a unique, unused frequency for the duration of the transmission and then switch back. This would be the beginning of the networks 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 reimplement certain features in a new way in order to handle the frequency changing. If we have each node wait for an acknolwedge from its immediate neighbors before changing frequency, that node could then change its operation knowing that the data will make it to the rest of the network. Batctl is already able to report the immediate next hop neighbors, so the program could use this information to only wait for acknolwedgements from neighbors instead of waiting for the entire network to be ready to change. A frequency change under these conditions would lead to a much more robust network 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 this transition as well.

VII. Conclusions

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 excellent 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.

VIII. Acknowledgments

We would like to thank Dr. Harish Chintakunta, Dr. Anas Salah Eddin, and Dr. Jorge Vargas. We would also like to thank Jerome Sonnenberg and David Chester of Harris for their support.

References

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

[2] (2016) A quick guide to hardware and gnu radio. [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/Hardware

[3] (2016) The comprehensive gnu radio archive network. [Online]. Available: http://cgran.org/

[4] (2016) Ettus research. [Online]. Available: https://www.ettus.com/

[5] (2016) Uhd (usrp hardware driver). [Online]. Available: https://www.ettus.com/sdr-software/detail/usrp-hardware-driver

[6] (2016) Open-mesh. [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki

[7] P. Murphy, A. Sabharwal, and B. Aazhang, "Design of warp: A wireless open-access research platform," in *Signal Processing Conference, 2006 14th European*, Sept 2006, pp. 1–5.

[8] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. Daniels, W. Kim, R. Heath, and S. Nettles, "Early results on hydra: A flexible mac/phy multihop testbed," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, April 2007, pp. 1896–1900.

[9] (2011) Click modular router. [Online]. Available: http://read.cs.ucla.edu/click/click

[10] (2016) Polymorphic types. [Online]. Available: https://gnuradio.org/doc/doxygen/page_pmt.html

[11] G. Troxel, E. Blossom, S. Boswell, A. Caro, I. Castineyra, A. Colvin, T. Dreier, J. B. Evans, N. Goffee, K. Haigh, T. Hussain, V. Kawadia, D. Lapsley, C. Livadas, A. Medina, J. Mikkelson, G. J. Minden, R. Morris, C. Partridge, V. Raghunathan, R. Ramanathan, C. Santivanez, T. Schmid, D. Sumorok, M. Srivastava, R. S. Vincent, D. Wiggins, A. M. Wyglinski, and S. Zahedi, "Adaptive dynamic radio open-source intelligent team (adroit): Cognitively-controlled collaboration among sdr nodes," in *Networking Technologies for Software Defined Radio Networks, 2006. SDR '06.1st IEEE Workshop on*, Sept 2006, pp. 8–17.

[12] X. Li, W. Hu, H. Yousefi'zadeh, and A. Qureshi, "A case study of a mimo sdr implementation," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, Nov 2008, pp. 1–7.

[13] V. C. Jawad Seddar, Hicham Khalife and J. Leguay, "A dtn stack for cognitive radio ad hoc networks," in *8th Karlsruhe Workshop on Software Radios*.

[14] 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.

[15] R. Alimi, L. Li, R. Ramjee, H. Viswanathan, and Y. Yang, "ipack: in-network packet mixing for high throughput wireless mesh networks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, April 2008, pp. –.

[16] L. Li, R. Alimi, R. Ramjee, H. Viswanathan, and Y. Yang, "munet: Harnessing multiuser capacity in wireless mesh networks," in *INFOCOM 2009, IEEE*, April 2009, pp. 2876–2880.

[17] A. Syed, K. Yau, H. Mohamad, N. Ramli, and W. Hashim, "Channel selection in multi-hop cognitive radio network using reinforcement learning: An experimental study," in *Frontiers of Communications, Networks and Applications (ICFCNA 2014 - Malaysia), International Conference on*, Nov 2014, pp. 1–6.

[18] P. Nagaraju, L. Ding, T. Melodia, S. Batalama, D. Pados, and J. Matyjas, "Implementation of a distributed joint routing and dynamic spectrum allocation algorithm on usrp2 radios," in *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, June 2010, pp. 1–2.

[19] H. Khalife, J. Seddar, V. Conan, and J. Leguay, "Validation of a point to multipoint cognitive radio transport protocol over gnu radio testbed," in *Wireless Days (WD), 2013 IFIP*, Nov 2013, pp. 1–6.

[20] S. Kamruzzaman, A. Alghamdi, and S. Mizanur Rahman, "Spectrum and energy aware multipath routing for cognitive radio ad hoc networks," in *Information and Communication Technology Convergence (ICTC), 2014 International Conference on*, Oct 2014, pp. 341–346.

[21] (2015) Confine project. [Online]. Available: http://confine-project.eu/

[22] P. Escrich, R. Baig, A. Neumann, A. Fonseca, F. Freitag, and L. Navarro, "Wibed, a platform for commodity wireless testbeds," in *Wireless Days (WD), 2013 IFIP*, Nov 2013, pp. 1–3.

[23] P. Escrich, R. Baig, E. Dimogerontakis, E. Carbo, A. Neumann, A. Fonseca, F. Freitag, and L. Navarro, "Wibed, a platform for commodity wireless testbeds," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on*, Oct 2014, pp. 85–91.

[24] (2014) A.l.f.r.e.d - almighty lightweight fact remote exchange daemon. [Online]. Available: https://www.open-mesh.org/projects/alfred/wiki

[25] (2016) Socket.io. [Online]. Available: http://socket.io/

[26] (2016) Flask (a micro framework). [Online]. Available: http://flask.pocoo.org/

[27] (2016) Flask-socket.io. [Online]. Available: https://flask-socketio.readthedocs.org/en/latest/

[28] (2015) socketio-client. [Online]. Available: https://pypi.python.org/pypi/socketIO-client