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) utilize systems of Software Defined Radios (SDRs) to establish networks with flexible physical and link layers, often in the form of self-forming, multi-hop networks called mesh networks. 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 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 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, Open-Mesh, A.L.F.R.E.D., 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. Hardware is not a direct component of GNU Radio. However, numerous other developers and projects have integrated hardware functionality into GNU Radio, 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]. ARCAM-Net uses the Universal Software Radio Peripheral (USRP) created by Ettus Research, a division of National Instruments[4]. 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 [6]. Many Cognitive Radio scenarios are designed around the concept of ad-hoc or mesh network [7]. 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 [8]. ARCAM-NET uses batman-adv as its layer 2 protocol. Batman-adv has a fairly large community and is integrated into the Linux kernel [8].

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 standardized toolset. Our platform will allow for others to create application layer products that leverage software defined radio meshes to get started without having to worry about all the intricacies 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 [9]. 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 [10]. 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 [11]. 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 [12].

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. [13] Similarly, the University of California, Irvine and Boeing Corporation developed a testbed based on USRP Radios and GNU Radio, but they implement custom MAC layers [14].

A platform similar to ARCAM-Net is presented in [15]. However, this platform uses the Optimized Link State Routing Protocol (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 [16].

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 [17] and [18] the authors use GNU Radio as a way to verify the successful use of algorithms for mesh networking. However, they do not use SDRs, instead choosing to use GNU Radio for simulation.

The researchers in [19] created a simple multihop test bed using three USRP radios to relay data from one computer to another. A fourth 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 Open Systems Interconnection (OSI) protocol stack from the ground up. In some papers the authors focus on the physical or mac layer [20]. There has also been work in developing new higher layer protocols for cognitive radio mesh networks such as work done to replace Transmission Control Protocol (TCP) with a more robust protocol [21]. These systems will usually react to frequency changes but some also change their topology based on power use [22].

C. Mesh Network Testbeds

The CONFINE platform uses batman-adv as the routing protocol for their mesh network testbed. However, their testbed does not utilize GNU Radio or SDRs. [23] batman-adv was also a key component of WiBed, a project to create a Commercial off-the-shelf (COTS) mesh test bed using low cost wireless routers [24] [25]. 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
- SocketIO Web Sockets

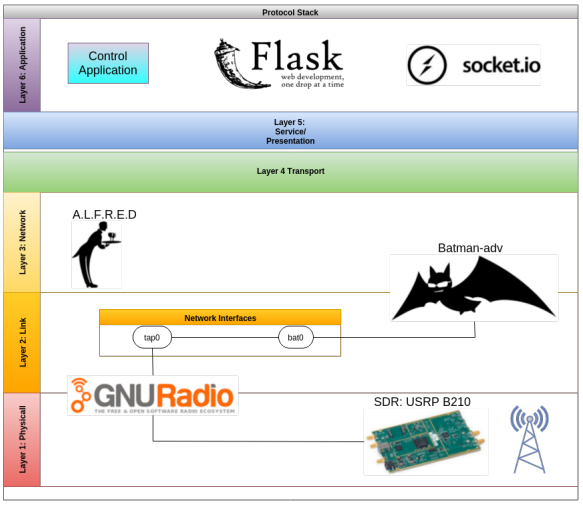

Fig. 1. An overview of the components of the system, including the OSI Layers they interact with. [1] [5] [8] [26] [27] [28]

- 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 [5]. 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.

B. 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 [26]. 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 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

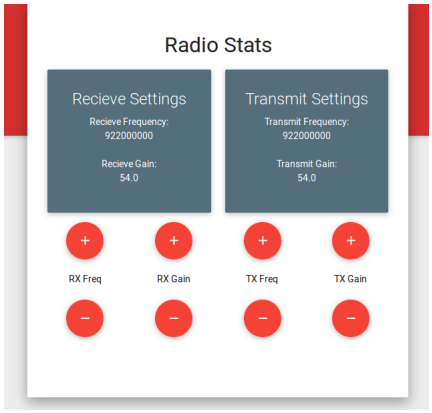

Fig. 2. The web interface that lets the user initiate the network to hop to a new frequency.

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 [29]. 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.

C. Batman-adv

Batman-adv was chosen based on its large community and documented success as a mesh routing protocol [16]. It is already included as part of the Linux kernel, and additional software can be downloaded from most distributions repositories [8]. 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.

D. Flask Web Server and Socket.IO

Flask is a lightweight, open source, web framework for the Python programming language [28]. 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 [27]. 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 [30] and can be used in stock Python with a client library [31]. 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 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.

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]. 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.

IV. 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.

A. 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. We staggered the transmit and receive frequencies of the nodes to ensure that

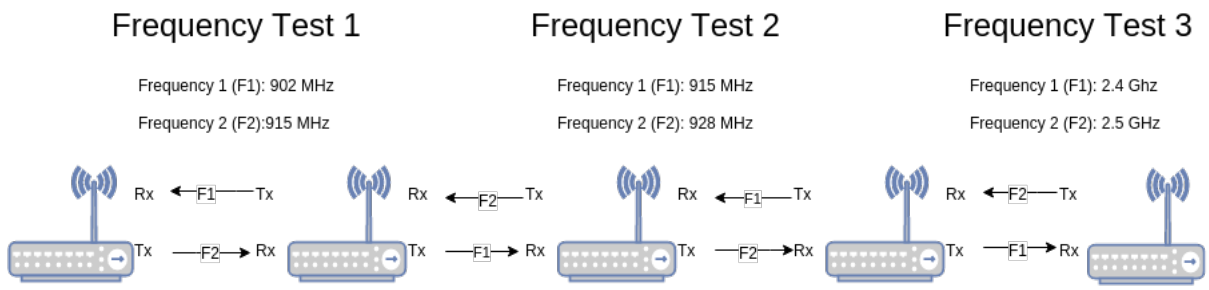

Fig. 3. The configuration used for the first set of tests.

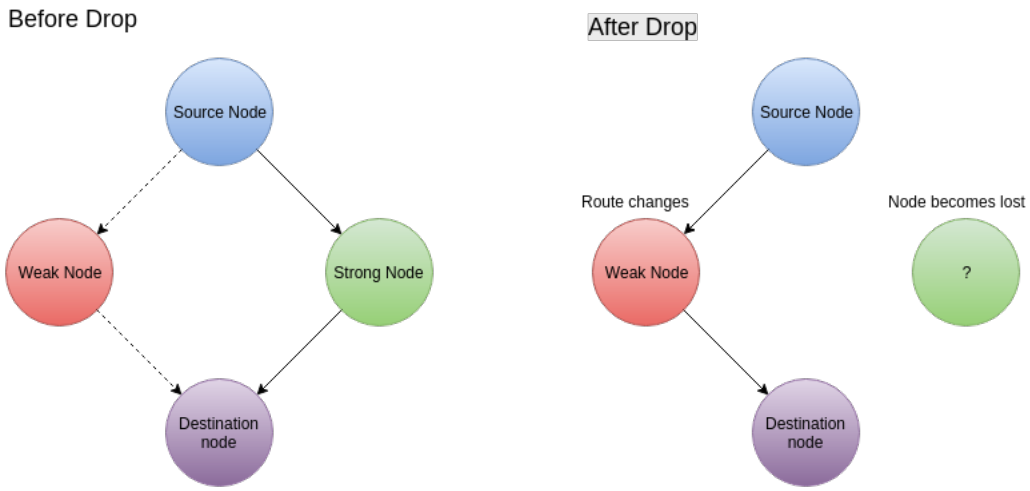

Fig. 4. The configuration used for the second set of tests.

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.

B. Route Changes

In a typical mesh environment, there will usually be more than one route from a source to a destination [7]. 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 4.

C. 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.

V. Results

A. Network Benchmarks

The results of the Network Benchmark tests from section 4 part A are summarized in Figure 7. 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

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

Fig. 5. 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

Fig. 6. 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

Fig. 7. The Averages from all three tests.

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 5 and Figure 6 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.

B. 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 8. After the change, the routing table appeared as it does in Figure 9. This feature works in the SDR system, and can continue to be used without significant changes.

C. 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 [26]. 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 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 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. 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 8 and 9 could help

```
[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 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)
```

Fig. 9. After the gain is reduced in the test presented in Section 4 B, the packets are now routing through a different node.

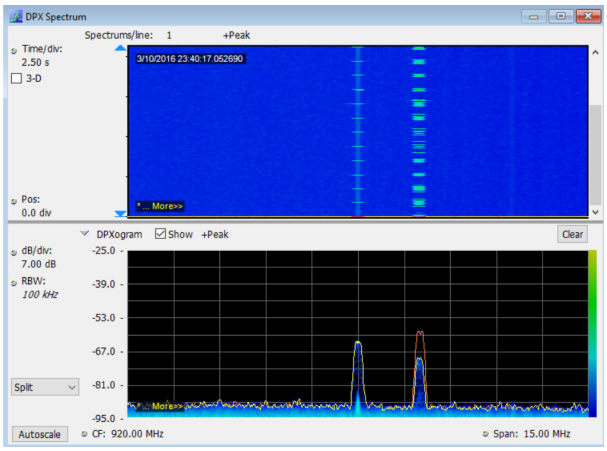

Fig. 10. The result of using ALFRED to shift frequencies. One node is left behind as the others move to the new channel. Image collected using the Tektronix RSA306 Spectrum Analyzer

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.

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 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 The Harris Corporation and Florida Polytechnic University 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] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Computer Networks*, vol. 51, no. 4, pp. 921 – 960, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128606002751

[7] 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

[8] (2016) Open-mesh. [Online]. Available: https://www.open-mesh.org/projects/open-mesh/wiki

[9] 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.

[10] 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.

[11] (2011) Click modular router. [Online]. Available: http://read.cs.ucla.edu/click/click

[12] (2016) Polymorphic types. [Online]. Available: https://gnuradio.org/doc/doxygen/page_pmt.html

[13] 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.

[14] 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.
[15] 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*.
[16] 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.
[17] 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. –.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] (2015) Confine project. [Online]. Available: http://confine-project.eu/
[24] 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.
[25] 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.
[26] (2014) A.l.f.r.e.d - almighty lightweight fact remote exchange daemon. [Online]. Available: https://www.open-mesh.org/projects/alfred/wiki
[27] (2016) Socket.io. [Online]. Available: http://socket.io/
[28] (2016) Flask (a micro framework). [Online]. Available: http://flask.pocoo.org/
[29] (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
[30] (2016) Flask-socket.io. [Online]. Available: https://flask-socketio.readthedocs.org/en/latest/
[31] (2015) socketio-client. [Online]. Available: https://pypi.python.org/pypi/socketIO-client