Totally-Ordered, Reliable Multicast over Cognitive Radio Networks

Matthew R. Tolson, Clark V. Dalton, Mark D. Silvius, Ethan S. Hennessey,
Curtis C. Medve, Jared J. Thompson, Kenneth M. Hopkinson, Seif Azghandi

Department of Electrical and Computer Engineering
Air Force Institute of Technology (AFIT)
2950 Hobson Way, Wright-Patterson AFB, Ohio 43433, USA

{mrtolson6, clark.dalton01}@gmail.com, {mark.silvius, ethan.hennessey,
curtis.medve, jared.thompson.6}@us.af.mil, {kenneth.hopkinson, sazghand}@afit.edu

Abstract

This paper summarizes the evaluation of a totally-ordered, reliable multicast protocol over a cognitive radio (CR) network. Multicast enables a group of nodes to subscribe to broadcast messages. Reliability ensures that all nodes receive messages correctly; totally-ordered ensures that all nodes receive messages in sequential order. In the proposed architecture, CR nodes employ multicast to exchange radio environment maps and to perform dynamic spectrum access. Emulation and wireless testbeds were constructed and baseline unicast quality of service (QoS) measured. The Spread open-source multicast software toolkit was installed on all nodes and the resulting multicast QoS measured. Channel effects were applied to the emulation testbed, so that its QoS modeled that of the wireless testbed. This paper demonstrates that it is feasible to characterize multicast QoS over a CR wireless network, through the use of an equivalent emulated network.

1. Introduction

Cognitive radios (CR) have the ability to change their frequencies and waveforms on-the-fly in a process called *dynamic spectrum access* (DSA). CRs also contain control logic which enable them to *observe* the surrounding radio frequency (RF) spectrum; *orient* themselves to the presence of competing secondary users and sources of RF interference; *decide* which frequencies, waveforms, and protocols to use to avoid interference and optimize network quality of service (QoS) metrics; and then *act* by configuring their underlying software defined radio (SDR) platform. This adaptation cycle is often referred to as the OODA loop and forms the foundation for a CR's operation [1, 2].

In previous work [3-5], the authors presented a more detailed OODA loop and introduced a new architecture to support a network of frequency-hopping (FH) CRs. The network is capable of co-existing with primary users, competing secondary users, and sources of interference in the band. The specific functions are detailed in Figure 1.

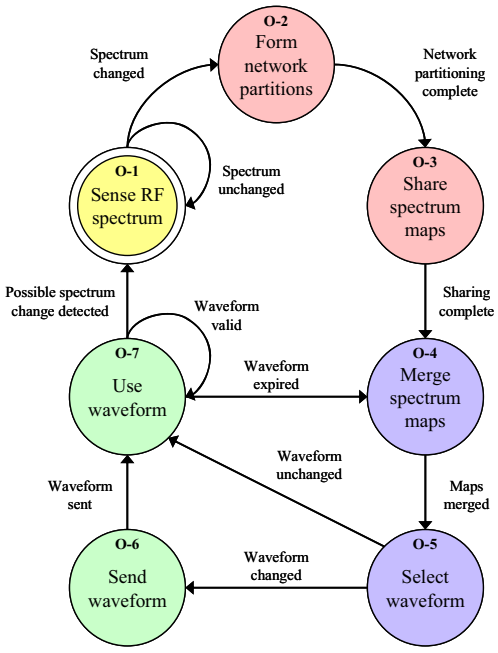

Figure 1. FH-CR System Functions

In the observe phase, the radios sense the RF spectrum (O-1). In the orient phase, radios form network partitions and share spectrum maps (O-2, O-3). In the decide phase, radios merge these spectrum maps, also known as *radio environment maps* (REMs) [6], and select a waveform (O-4, O-5). Finally, in the act phase, radios distribute the waveform selection and reconfigure their SDR platforms (O-6, O-7).

978-1-4799-2504-9 2014

DOI 10.1109/HICSS.2014.632

The authors' proposed radio architecture needs a network protocol which distributes REM data between nodes in Function O-3 of Figure 1. The goal is for nodes within the network to have an identical copy of the REM. Specifically, the REM distribution protocol has the following requirements:

1.1. Broadcast to Subscription Group

REM-update messages must be broadcast from the source node, with copies being sent to all other subscribed or participating receiver nodes. Nodes must be able to subscribe or unsubscribe to these broadcasts dynamically as cluster assignments change or nodes join or depart the network.

1.2. Reliability

REM update message must be transmitted in a reliable fashion from the source to all receiver nodes. Each node which wishes to publish a REM-update messages, must maintain knowledge of all the subscribed receiver nodes. After a REM-update broadcast, each subscribed receiver must acknowledge its successful receipt. If an acknowledgement is not received, a re-transmission must be sent to that receiver node.

1.3. Totally-Ordered

The broadcast message must occur in a synchronized fashion; specifically, only one source node may broadcast an REM-update at time. All subscribed nodes must acknowledge their receipt and subsequent processing of the data. This must occur before that source node is allowed to send out another update, or before any other nodes within the network are allowed to send updates.

Given all these requirements, the authors have selected the *Spread* open-source multicast software toolkit [7] as a candidate totally-ordered, reliable, multicast protocol for use in their CR network.

This paper makes the following contributions: first, it investigates suitability of the Spread totally-ordered reliable multicast for exchanging REMs within a CR network; second, it demonstrates feasibility of characterizing multicast QoS over a CR wireless network, by using an equivalent emulated network.

2. Previous Work

This section introduces the Spread open-source multicast software toolkit and summarizes related work using multicast over CR networks.

2.1. The Spread Toolkit

The Spread toolkit provides reliable, wide area group communications. It does so using two low-layer protocols—one for local area networks called *Ring*, and one for the wide area network connecting them, called *Hop*. It implements efficient reliability and ordering of existing available wide area multicast mechanisms. Spread is similar to existing protocols that provide reliability over existing IP-multicast, but it scales dissemination and flow control to wide-area networks. Spread does not scale with the number of nodes, but rather, scales with the number of participating groups. Spread also decouples the dissemination and local reliability mechanisms from the group order and stability protocols, and it supports the Extended Virtual Synchrony model [8].

Spread uses a daemon-client architecture. The daemons manage group membership, keeping track of the join and leave messages and processes from the participating nodes. The daemons establish the basic message dissemination network and provide basic membership and ordering services. They connect local area networks (LANs) over the wide area network (WAN) using unicast traffic. Clients within a LAN connect to the closest daemon to access the group communication services. In this way, not every client needs to provide member and order services; only a limited number of daemons must do this. Each LAN may have up to several tens of clients participating in group communications and connected to daemons [9].

Within the LAN, Spread relies on local distribution services such as IP-multicast, hardware multicast, or hardware broadcast. Spread's Ring protocol facilitates multicast within each LAN, and it provides reliability and flow control for packets. Spread's Hop protocol facilitates unicast point-to-point connections between the LAN, across the WAN. It operates over an unreliable datagram service such as UDP/IP. This provides the least possible latency when transferring packets across networks consisting of several hops. Lost packets are handled on a hop-by-hop basis instead of end-by-end basis, and packets are immediately

forwarded, even if they are not in order [8]. Spread is considered an overlay network, since it constructs a virtual network, where each link connects two edge nodes in an underlying physical network, such as the Internet. Each virtual link in the overlay network may consist of several hops in the underlying network [9].

Spread also provides an application program interface (API) which allows developers to incorporate the protocol into their network applications. The toolkit can be downloaded on the Internet, and its authors provide a substantial user's manual which explains its installation, configuration, and use [7, 10].

2.2. Multicast in CR Networks

Compared to legacy networks, less research has been accomplished on the performance of multicast protocols in wireless CR networks. CR networks, designed to operate in DSA scenarios, are unique in that the physical (PHY) and media access control (MAC) layers of the protocol stack must be adaptable. They may use a frequently changing center frequency, waveform, symbol rate, coding, etc, in response to a changing RF environment. Not only does the process of dynamically adapting waveforms incur latencies, but there may be instances where no available frequencies, called *whitespace*, exist for transmission and reception. The lower layers of the protocol stack in the CR may simply place the higher layers on-hold, while they search for available frequencies and coordinate new waveforms. Multicast protocols, running at either the network (NET) or application (APP) layers, typically require connection setup and handshaking. This overhead requires stability from the lower PHY/MAC layers. Without this stability, the QoS provided by the multicast protocol begins to break down.

In [11], the authors address the issue of QoS multicast routing and transmission scheduling in multi-hop CR networks. They propose a distributed protocol which uses the shortest path tree as the multicast tree. They design a protocol which sets up a multicast connection, which minimizes the total multicast bandwidth consumption, while satisfying the QoS requirements. Their design takes advantage of the broadcast nature of the wireless channel. They also use a time-division multiple-access (TDMA) waveform for their PHY/MAC layer, where each radio in the network must use time slots and frequency channels in the DSA environment, to maintain link connectivity and to support multicast. This TDMA waveform, and its enumeration of time slots and frequency channels, closely parallels this paper's authors desire to use multicast over an adaptable FH-CR network.

In [12], the authors study QoS routing in wireless mesh networks with CRs, and discuss route selection, channel allocation, and scheduling. They present a distributed routing protocol which can select a route and allocate resources for a connection request to satisfy end-to-end bandwidth requirements. In [13], authors study the issue of enabling multicast video in CR networks, and propose a cross-layer optimization approach. They optimize the overall received video quality, and achieve proportional fairness among multicast users, while keeping the interference to primary users below a subscribed level.

3. Evaluation Methodology

In their previous work [3-5], the authors proposed a new architecture for an adaptable FH-CR. The system contains novel design features, such as the use of a distributed REM, a clustering algorithm to group nodes into geographic regions of similar REM, as well as a FPGA-based circuit for merging REM data on each radio. The proposed system can best be viewed as a *middleware architecture*. Many of the novel design contributions implementing the FH-CR OODA loop reside in middleware module, as represented by the yellow-shaded block in Figure 2.

The architecture was originally targeted towards the stand-alone Rice University Wireless Open-Access Research Platform (WARP) [14]. The WARP uses the PowerPC processor on the Virtex-IV FPGA for communication processing. The PowerPC is also an ideal processor to run embedded Linux and network and transport layer protocols and applications. Rice University has tested and verified the FPGAs and PowerPC suitability on the WARP board in over-the-air tests.

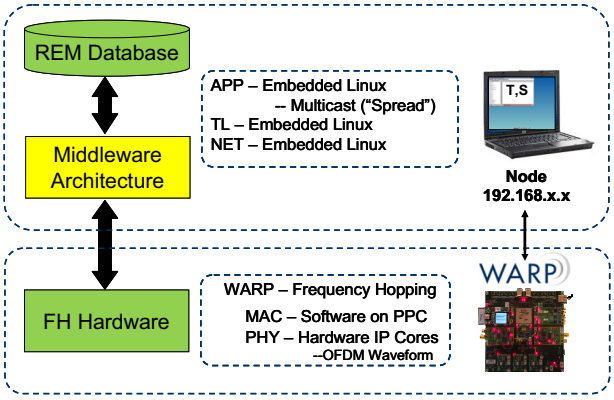

Figure 2. Middleware Architecture

In this paper, by contrast, the network stack of the wireless radio node is split, as show in Figure 2. The

PHY and MAC layers are handled by the WARP platform, and the APP, transport (TL), and NET layers are handled by a stand-alone Linux OS instance. This Linux is running within a virtual machine (VM), either on a laboratory workstation or laptop. The Linux OS runs the Spread multicast toolkit. The host computer, running the VM, is connected to the WARP platform via an Ethernet cable. The VM host, the Linux OS, the Spread toolkit, and the WARP platform, all together represent an entire network stack, and operate as a single wireless node instance. After completing the evaluation of Spread multicast and the WARP platform's QoS performance, the authors plan to collapse the entire system, and its components, to reside on a stand-alone WARP device.

In the following sections, we investigate the suitability of the Spread totally-ordered reliable multicast for exchanging REMs to satisfy the architecture detailed in Figures 2-3. Emulation and wireless testbeds are constructed and baseline QoS measured. Channel effects are applied to the emulation testbed, so that its QoS modeled that of the wireless testbed. In this way, this paper demonstrates that it is feasible to characterize multicast QoS over a CR wireless network, through the use of an equivalent emulated network.

4. Results: Emulation Network (EmuNet)

This section details the design and implementation of the emulation network. It then details both unicast and multicast baseline QoS results.

4.1. Architecture

The *EmuNet* emulation network resides on a *Dell Precision T7500 Workstation* with Intel Xeon X5650, 6-core 2.66 GHz CPU, 24 GB DDR3 1333 MHz RAM, with Windows 7 64-bit as the host operating system. The workstation provides the high performance and scalability that is needed to emulate the multicast over a large virtual network. We installed *VM Ware Workstation 8.0* as our virtual machine host [15]. VM Ware gives us full control of the networking of our VM clients, which allows us to create a virtual network with different subnets. We created five instances of a *Tiny Core Linux* [16]. Tiny Core Linux is a small, extremely fast, and highly modular operating system with many useful built-in extensions. We are running the *Core Plus* version of Tiny Core, which at only 64 MB could easily be installed on a compact flash and used with an FPGA embedded system. A diagram of the EmuNet is shown in Figure 3.

Each node in the EmuNet is assigned a hostname and an internet protocol (IP). Node 5 performs the role of a virtual network router, connecting two distant subnets, via the Linux *route* command. To simulate congestion, we use the *Traffic Control* (*tc*) and *Network Emulation* (*netem*) utilities. Their operation will be detailed in greater depth in Section 6. Within the network, end-to-end QoS is measured using the *Iperf* and *Ping* utilities. Iperf is a software tool, written, in C++, that measures TCP and UDP bandwidth performance and characteristics. It features many ways to control the data sent and the frequency at which these tests are performed [17]. Each node is preloaded with the Spread Toolkit, and is able to create, join, and publish messages to a multicast group.

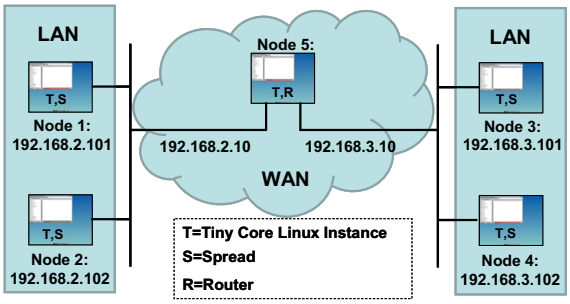

Figure 3. Emulation Network Testbed (EmuNet)

4.2. Unicast Performance

The system components are assembled in accordance with Figure 3. We have five instances of Tiny Core Linux running, each installed on the Dell Precision T7500 workstation. Multiple startup scripts are configured on each VM. On each machine running Spread, we have scripts that set up the IP address, start the Spread daemon, and start the Spread program that allows communication between VMs. The VM that represents our router has a startup script that sets up both IP addresses for each virtual network adapter. This script also enables packet forwarding. Without Spread or any channel effects programs running, we test the control QoS of our virtual networking using the Iperf and Ping tools to calculate goodput, latency, and packet loss. The results are summarized in Table 1.

Table 1. Unicast QoS: EmuNet

	Throughput [Mbps]	Latency [ms]	Packet Loss [%]
Average	89.726	0.443	6.385

We then expanded our Iperf and Ping tests to determine the QoS of Iperf with increasing offered loads and no channel effects. Iperf offers different

options to control the unicast tests. We ran the Iperf program, starting with an offered load of 25 Mbps, going up to 200 Mbps. One constraint that we had for the EmuNet was to enforce a 100 Mbps bandwidth on each of the Tiny Core virtual network adapters.

The results of these tests are given in Table 2. The max average throughput peaked at the max bandwidth set in each VM instance, 100 Mbps. The difference in throughput between an 100 Mbps and 150 Mbps offerered load is -3 Mbps. This 50% increase in the offered load, significantly increases the packet loss, and has no advantage in throughput.

Table 2. Unicast QoS: EmuNet

Offered Load [Mbps]	Average Throughput [Mbps]	Average Packet Loss [%]
25.0	23.837	0.000
50.0	47.782	0.004
100.0	89.726	6.373
150.0	86.843	38.683
200.0	82.332	55.367

Figure 4 depicts the graph of the data in Table 2. This plot compares the average throughput and the average packet loss over the same offered load tests.

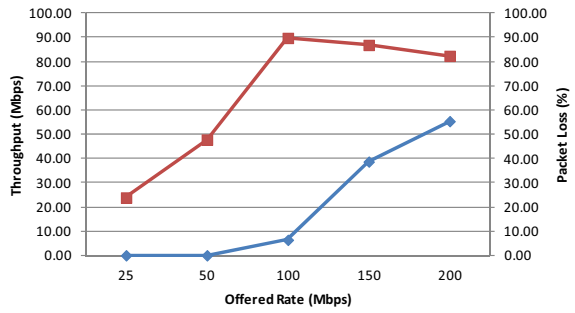

Figure 4. Average Throughput and Packet Loss

4.3. Multicast Performance

Multiple startup scripts are configured on each VM running Spread. The scripts set up the IP address, start the Spread daemon, and start the Spread program, which allows communication between VMs. The VM that represents our router also has a startup script that sets up both IP addresses for each virtual network adapter and packet forwarding. The Spread program was modified to send files, including binary files. Spread was also modified to send an acknowledge message back once the file has been received. This message is used to calculate throughput for messages sent, as well as the round trip latency.

QoS data was calculated with each machine running a modified Spread program which sends files to the other VMs. Once a VM has received the file, it sends a message to the sender stating that it had received the file. With the use of timestamps, we are able to calculate the throughput of the messages.

To determine the numbers for Table 3, a series of Spread message tests were conducted. Each message size was 1,472 bytes. Five messages were sent from each node to all of the other nodes in the virtual network with a total of 20 messages sent. When gathering information, we only used data from nodes that were across the subnet. An example of this would be data that was sent from Tiny 1 to Tiny 2, Tiny 3, and Tiny 4, would only be considered if the other nodes were on a separate subnet. In this case it would be Tiny 3 (x.x.3.101) and Tiny 4 (x.x.3.102) compared to Tiny 1 (x.x.2.101) and Tiny 2 (x.x.2.102). We chose to do this based on how Spread is implemented. Since we use a virtual network the packets never travel over a physical wire, which affects the data from nodes on the same subnet. From the Spread literature and data logs, we concluded that a message would be sent to the first node of the subnet, and then re-broadcasted to the rest of the subnet. Since Tiny 1 and Tiny 3 were the first nodes on their respective subnets, their average throughputs were measured to be much higher, since they did not have to wait for the message to be re-broadcasted from the first node of the subnet. The two metrics computed were: *throughput*—this was calculated by adding each throughput for a set of five tests, and then dividing by five; *latency*—this was calculated by adding each latency test, i.e. the raw difference in time, for a set of five tests, and then dividing by five. *Package loss* was not detected over the EmuNet. Average values were calculated by averaging all measurements between the combinations of four nodes. These results are summarized in Table 3.

Table 3. Multicast QoS: EmuNet

	Message Size [bits]	Throughput [Mbps]	Latency [ms]	Packet Loss [%]
Average	11,776	38.590	0.507	n/a

From Table 3, we can conclude that the Spread protocol is working as expected. Messages are sent from one node to the first node in the other subnet, and then this first node re-broadcasts the message out to the rest of the nodes on the subnet.

5. Results: Wireless Network (WARPnet)

This section details the design and implementation of the wireless network. It then details both unicast and multicast baseline QoS results.

5.1. Architecture

The WARPnet wireless network consists of four HP Compact 8510p laptops with Intel Core 2 Duo 2.4 GHz 2 GB DDR RAM with Windows XP 32-bit as the host operating system. Currently, each laptop runs VM Ware Workstation 8.0 with a single instance of Tiny Core Linux. Although the laptop is capable of running Tiny Core Linux natively, the use of a single standardized Tiny Core Linux image in both EmuNet and WARPnet greatly simplifies administration, deployment, and scripting. A diagram of the WARPnet is shown in Figure 5.

The WARPnet features four WARP boards. For our implementation, we use the *OFDM Reference Design* software available on the Rice software repository [18]. The package implements a real-time network stack and the ability to bridge the wired-to-wireless abilities of the WARP board to an Ethernet-connected PC. We also use the *CSMA MAC Project* extension to the OFDM Reference Design. This design creates a wired-to-wireless bridge while also implementing a CSMA MAC protocol. It offers support for only two nodes, but while doing so allows for packets to transfer seamlessly over the WARP board into the Ethernet-connected PC and vice versa. We later redesigned the CSMA MAC to allow for multiple wireless nodes. We refer to this updated design as our *Modified CSMA MAC Project*. This involved modifying the code to allow for the WARP board to set its source MAC address according to the header information contained in packets arriving from its Ethernet connection. The WARP board's wireless interface then takes on the same MAC source address as the laptop's Ethernet interface.

In the WARPnet, the WARP boards perform a similar function as an Ethernet bridge for the laptops. They simply encapsulate each 802.3 Ethernet frame received from the laptop into a wireless 802.11-like frame and transmit it over the air. Conversely, they take each received wireless 802.11-like frame, de-encapsulate it to recover the 802.3 Ethernet frame, and relay it to the laptop. The laptops are also not controlling operations on the WARPs boards; the boards run independently, controlled by the software running on PowerPC core and the circuitry running within the FPGA fabric.

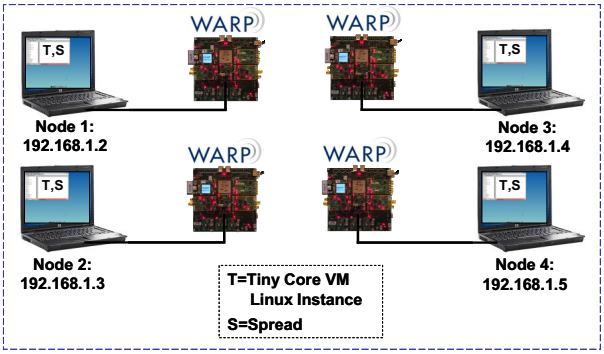

Figure 5. Wireless Network Testbed (WARPnet)

To program the WARP boards, we employed the Xilinx's EDK, SDK, and Impact tools [19]. This suite of software tools allows for easy development of software for the WARP board and changes to programmable hardware on the WARP board. Specifically, SDK develops and uploads software to the WARP board PowerPC processor. EDK develops and flashes hardware configurations for the WARP board FPGA. The Impact tool uploads and packages hardware and software configurations for the WARP Board. It can also create bootable compact Flash images for the WARP board. As in EmuNet, end-to-end QoS in WARPnet is measured using the Iperf and Ping utilities. Each node is preloaded with the Spread toolkit, and is able to create, join, and publish messages to a multicast group.

5.2. Unicast Performance

The system components are assembled in accordance with Figure 5. We start up the OFDM Reference Design with our Modified CSMA MAC Project running. Each board boots from its own external compact flash memory using a board image common to all four nodes. We test the steady-state QoS of our radio network using the Iperf program to calculate throughput, latency, and packet loss. We use Iperf running in Tiny Linux, within VM Ware Workstation, running on Windows XP on the laptops. The results are summarized in Table 4.

Table 4. Unicast QoS: WARPnet

	Throughput [Kbps]	Latency [ms]	Packet Loss [%]
Average	128.1	55.1	27.68

Data was collected with Iperf and Ping scripts running on Linux Tiny Core. Throughput and latency values are averaged over five separate tests. Packets lost and sent values are sums over five tests for each pair of nodes. Iperf tests were run with an outgoing UDP bandwidth of 200 Kbps. Both Iperf and Ping tests ran with 1,470 byte packets.

5.3. Multicast Performance

Table 5 details the results of the Spread testing with the WARPnet. These values were gathered using the same methods that were used to gather Spread values over the EmuNet in Section 4.3. System time across the Tiny Cores was synchronized using NTP in Linux over a LAN network, on a different subnet than the WARP subnet. All nodes sync to Node 4 (192.168.6.4) with a daemon and ignored the time values of the other nodes. The organization of the results in Table 5 was similar to that of Table 3.

Table 5. Multicast QoS: WARPnet

	Message Size [bits]	Throughput [Kbps]	Latency [ms]	Packet Loss [%]
Average	11,776	36.958	38.7	n/a

A summary of all QoS measures is shown in Table 6. For Unicast, the WARPnet had 89.598 Mbps (-99.9%) slower throughput, 54.657 ms (+99.2%) longer latency, and 21.30% (+76.93%) larger packet loss rate than the EmuNet. This is expected given the ideal network conditions of the emulated environment compared to the relatively noisy lab environment of the WARPnet.

Table 6. Summary of Average QoS Measurements

Experiment	Throughput [Kbps]	Latency [ms]	Packet Loss [%]
EmuNet - Unicast	89,726	0.443	6.385
EmuNet - Multicast	38,590	0.507	n/a
WARPnet - Unicast	128.1	55.1	27.68
WARPnet - Multicast	36.958	38.7	n/a

6. Results: Environment Matching

In this section, we model the delay of the WAN and make the emulation testbed model the wireless testbed. This will enable us to scale the emulation testbed in the future, so that it can approximate a larger wireless testbed.

6.1. Spread Protocol Design: WAN vs. LAN

Spread is designed with two major components, the WAN and LAN. During the operation of Spread, a message can be sent from a node on the LAN to another node in another LAN. Since the LANs are on different subnets, the message must cross the WAN. In the LAN, Spread uses rings to distribute the messages, and in the WAN, it use hops. Figure 6 depicts the setup of the EmuNet testbed. Node 5 is a router, and it models the WAN for messages traveling to different subnets. Nodes 1 through Node 4 are in their respective LANs, and they communicate with the ring implementation.

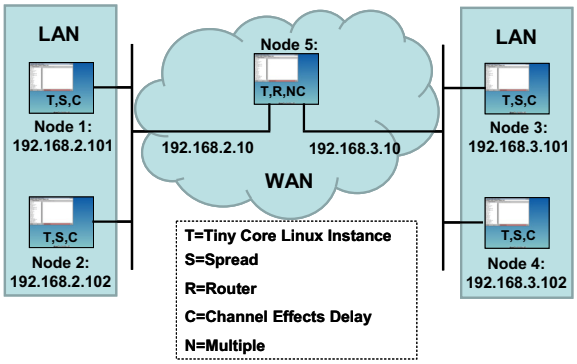

Figure 6. EmuNet with Channel Effects

6.2. Modeling the WARPnet Environment

To model the EmuNet as a WARPnet, we needed to get a baseline for what the max multicast QoS was for each testbed. Using the Multicast EmuNet and WARPnet QoS results from Table 3 and Table 5, we concluded that the max throughput for the EmuNet is 38.590 Mbps and the max throughput for the WARpnet is 36.958 Kbps. Given this data, we used the Linux programs tc and netem to throttle the EmuNet to match the WARPnet.

Figure 6 describes the basic setup for the experiment. At each Node 1 to Node 4, there is a channel effect delay C, and at Node 5 there is a channel effects delay $N*C$, where N is assumed to be a multiple of delay that a message would encounter across the WAN. We chose N to be 5 for our experiments. For the QoS tests with channel effects, we decided to use 1, 5, 10, and 100 ms delays.

In the following test, a Spread multicast message was sent to the multicast group including Node 1, Node 2, Node 3, and Node 4. Utilizing the channel effects programs discussed above, time delays were

added on each network adapter in the LANs. Also, a time delay was added to Node 5 that was five times larger than the delay specified for Nodes 1 through Node 4. Table 7 shows that we were able to throttle the EmuNet comparatively to the WARPnet's max QoS discussed in the control data section.

Table 7. Multicast Spread QoS: EmuNet

Delay, *C*, [ms]	Total Ave Throughput [Kbps]
1	3,336.74
5	1,037.25
25	276.06
125	47.07
250	24.14
375	15.75

Figure 7 depicts the Table 7 in a more revealing manner. From this graph, it is evident how increases in the delays on each Tiny Core affect the average throughput.

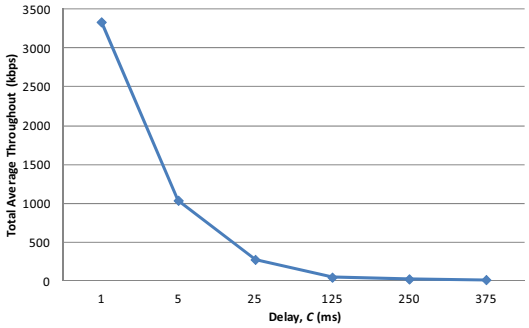

Figure 7. Spread with Channel Effects

7. Analysis and Conclusions

This paper shows that it is feasible to throttle down the EmuNet testbed to resemble the Spread Multicast QoS. Table 8 summarizes the throttled down EmuNet compared to the base WARPnet QoS. The WARPnet max QoS for the Spread protocol was 36.958 Kbps. When the EmuNet was throttled down with 125 ms and 250 ms delays, the QoS was 47.07 Kbps and 24.14 Kbps, respectively. These two data points are on the low and high sides of the WARPnet throughput. A throttled EmuNet between these two delays would give an even closer QoS match.

The EmuNet testbed was successful at achieving a throughout that was similar to the throughput of the WARPnet. From this conclusion, we know that the EmuNet is capable of simulating the WARPnet. We can now move forward and expand and combine our EmuNet to simulate a much larger multicast radio network.

Table 8. Summary of Spread QoS Throttling

Spread QoS Experiment	Delay [ms]	Throughput [Kbps]
WARPnet	0	36.958
EmuNet	1	3,336.74
EmuNet	5	1,037.25
EmuNet	125	276.06
EmuNet	125	47.07
EmuNet	250	24.14
EmuNet	375	15.75

8. Future Work

The authors' next goal is to continue to improve the realism of CR network. Three routes include:

- Scale the EmuNet environment to tens or hundreds of virtual nodes, and re-test the QoS.
- Create a mixed testbed that consists of both WARP boards and virtual nodes, and re-test the QoS.
- Port the Tiny Core Linux and Spread to run natively on the PowerPC hardcore processor on the WARP's FPGA, so that each wireless node operates as a stand-alone system.

The first of the two tracks given above would provide QoS information on the scalability of the Spread multicast message protocol. It also would provide insight on whether this protocol would be favorable for wireless radio networks. The second option would be to incorporate both the WARP boards and Tiny Core VMs to create a mixed wired and wireless network. This option could be scaled to include many nodes. In other words, the EmuNet and WARPnet could be combined into a single, consolidated Mixed Emulation-Wireless Network Testbed (MEWiNet), as shown in Figure 8. The MEWiNet would also receive emulated spectrum and whitespace data through the Dynamic Spectrum Emulator (DySE) system [3-5].

The third track would enable the authors to measure the QoS performance of both the WARPnet and the MEWiNet, by using complete, stand-alone radio nodes running on the WARP device. This would add greater realism to the measurements, would provide the first step toward full system deployment and field test and evaluation.

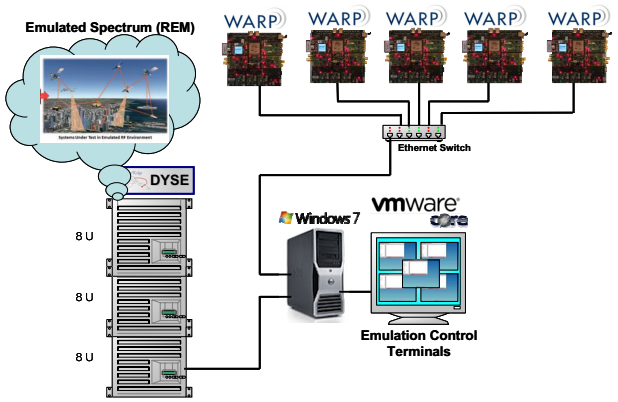

Figure 8. Mixed Emulation-Wireless Network Testbed (MEWiNet)

9. Acknowledgement

The views expressed in this document are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

The authors wish to thank AFRL/RYWE for their sponsorship of this research.

10. References

[1] R. Coram, *Boyd: The Fighter Pilot Who Changed the Art of War*. Boston: Back Bay Books, 2002.

[2] B. A. Fette, *Cognitive Radio Technology, 2nd Ed.*, Boston: Academic Press/Elsevier, 2009.

[3] R. K. McLean, B. N. Flatley, M. D. Silvius, and K. M. Hopkinson, "FPGA-Based RF Spectrum Merging and Adaptive Hopset Selection," *IEEE Aerospace Conference*, Big Sky, MT, March 2-9, 2013.

[4] R. K. McLean, M. D. Silvius, and K. M. Hopkinson, "Method for Evaluating k-Means Clustering for Increased Reliability in Cognitive Radio Networks," *IEEE Software Security and Reliability (SERE)*, Washington, DC, June 18-20, 2013.

[5] R. K. McLean, "An Architecture for Coexistence with Multiple Users in Frequency Hopping Cognitive Radio Networks," M.S. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 5, 2013.

[6] Y. Zhao, "Enabling Cognitive Radios through Radio Environment Maps," Ph.D. Dissertation, Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2007.

[7] Spread Concepts LLC, "The Spread Toolkit," March 2013, URL: http://www.spread.org.

[8] Y. Amir, C. Danilov, M. Miskin-Amir, J. Schultz, and J. Stanton, "The Spread Wide Area Group Communication System," *Technical Report CNDS-98-4*, The Johns Hopkins University, Baltimore, MD, URL: http://www.spread.org/SpreadResearch.html.

[9] Y. Amir, C. Danilov, and S. Stanton, "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area group Communication," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, New York, NY, June 25-28, 2000, pp. 327-336.

[10] J. R. Stanton, "A Users Guide to Spread Version 0.11," October 21, 2002, URL: http://www.spread.org/docs/guide/users_guide.pdf.

[11] X. Liming and J. Xiaohua, "QoS Multicast Routing and transmission scheduling in Multi-hop Cognitive Radio Networks," in *IEEE Global Communications Conference (GLOBECOM), Workshop on Pervasive Group Communications*, Miami, FL, December 6-10, 2010, pp. 1487-1491.

[12] H. Donglin, M. Shiwen, and J. H. Reed, "On Video Multicast in Cognitive Radio Networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, Rio de Janeiro, Brazil, April 19-25, 2009, pp. 2222-2230.

[13] H. Donglin, M. Shiwen, Y. T. Hou, and J. H. Reed, "Scalable Video Multicast in Cognitive Radio Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 28, Issue 3, April 2010, pp. 334-344.

[14] Rice University, "Wireless Open-Access Research Platform (WARP)," February 2013, URL: http://warp.rice.edu.

[15] VM Ware Workstation 8.0, February 2013, URL: http://www.vmware.com/products/workstation/overview.html.

[16] Tiny Core Linux Project, March 2013, URL: http://tinycorelinux.net.

[17] Internet Protocol Performance (Iperf) Tool 2.0, March 2013, URL: http://iperf.sourceforge.net.

[18] Rice University, "OFDM Reference Design - Wireless/Wired Bridge," March 2010, URL: http://warp.rice.edu/trac/wiki/OFDMReferenceDesign/Applications/Bridge.

[19] Xilinx, Downloads, 2013, Multiple Versions URL: http://www.xilinx.com/support/download/index.htm.