Integrating Multiple Radio-to-Router Interfaces to Open Source Dynamic Routers

Randy Charland, Leonid Veytser, Bow-Nan Cheng
Airborne Networks Group
MIT Lincoln Laboratory, Lexington, MA
{rcharland, veytser, bcheng}@ll.mit.edu

***Abstract*—Military radio systems are often constrained systems that require per link information to effectively institute QoS policies, route packets, manage topology, and tunnel application data. One of the first attempts at providing standard radio-to-router information through a common interface was point-to-point over Ethernet (PPPoE) RFC5578. Although PPPoE RFC5578 works effectively for directional radios that form point-to-point links, there are a number of limitations including added overhead, requirement of Ethernet as the medium between radio and router, and inefficient medium reuse in broadcast environments. As a result, many new protocols such as Dynamic Link Exchange Protocol (DLEP) and Radio-to-Router Control Protocol (R2CP) have emerged to address the limitations of RFC5578 and provide a standard method to share per link information with the network layer. In previous work, an open source routing solution using a modified Quagga router to support dynamic link metrics were developed to support an open source implementation of radio-aware routing with RFC5578. In this paper, we present significant changes to this system to support both DLEP and R2CP on the open source router (OSR). These changes facilitate open source support of all 3 radio-to-router protocols and enable performance comparisons.** [1]

I. Introduction

The current generation of military radio systems are limited-use systems that work well in a homogeneous radio environment, but require significant setup and configuration to interoperate with other radio systems. Each radio provides a subset of disparate link information in non-standard interfaces and often have built-in, home-grown or industry-based routers running potentially different routing protocols. In a heterogeneous radio system environment, wireless link characteristics change rapidly, often requiring direct link feedback from the radio to make informed routing decisions. In recent years, there has been a number of works [1], [2], [3], [4] in developing a common radio-to-router interface that standardizes a subset of per-link information to pass to the network layer for use in dynamic MANET routing. While these standards are being vetted through the Internet Engineering Task Force (IETF), many have yet to be implemented and tested in real-world environments.

Generally, radio-to-router interfaces (R2RI) are comprised of three major components: 1) The link information the radio can provide, 2) the transport mechanism to get this information to the router, and 3) a method for the router to make use of the information to make routing and possibly flow control decisions. Because many legacy military radio systems do not support many of the radio-to-router interfaces natively, a proxy system has been used to emulate the radio-side R2RI functionality by converting link metrics gleaned from various radio systems into a common R2RI format. Figure 1 shows the three radio-to-router interface elements along with a proxy configuration. As can be seen, R2RI functionality can be broken into a part that resides on the routing system and one part that resides on the radio system (or proxy system which interacts with the radio).

[1]This work is sponsored by the Department of the Air Force under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

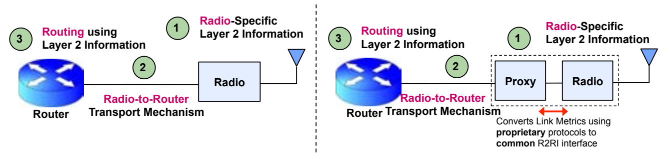
3
Routing using Layer 2 Information
1
Radio-Specific Layer 2 Information
2
Radio
Router
Radio-to-Router Transport Mechanism
3
Routing using Layer 2 Information
1
Radio-Specific Layer 2 Information
2
Proxy
Radio
Router
Radio-to-Router Transport Mechanism
Converts Link Metrics using proprietary protocols to common R2RI interface

Fig. 1. The three major components of any radio-to-router interface

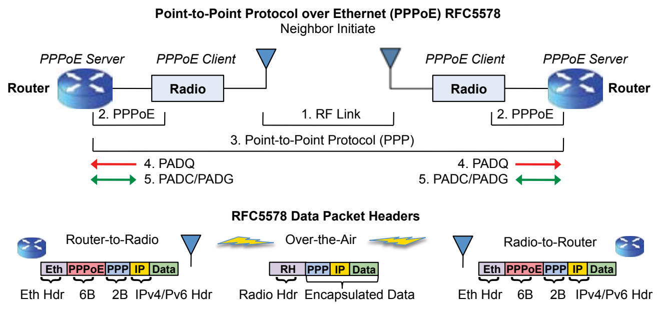

Fig. 2. Neighbor setup with RFC5578 and data packet headers

One of the first R2RI protocols proposed was Point-to-Point Protocol over Ethernet (PPPoE) with credit based flow control and link metrics extensions [5] (RFC4938). RFC4938 was obsoleted by RFC5578 [1] which added support for higher reported data rates and a few other small changes. For practical purposes, we utilize RFC4938 and RFC5578 interchangeably in this paper. Figure 2 illustrates the basic concept behind RFC5578. When an RF link is established, the radio, running a point-to-point protocol over ethernet (PPPoE) client initiates a PPPoE session with the router running a PPPoE server. Then, the router negotiates an end-to-end PPP connection with the router on the other side of the link. At periodic intervals, the radio probes the link and sends PPPoE Active Discovery Quality

(PADQ) packets to its local router containing link information such as per link latency, current and maximum data rate, relative link quality, resources, and neighbor up/down state. From this information, the routers dynamically calculate a link cost based on the quality of the link and the congestion, and weight the path choices accordingly. The radio periodically sends credits and credit grants to the server to throttle the rate of data send, implementing a flow control mechanism. As credits are used up, the packets are queued per link.

In previous work [6], [7], [8], [9], RFC4938/RFC5578 client and server/router-side code was developed to allow testing of the R2RI protocol in emulated and flight test environments. On the client/radio side, a proxy system was developed to convert link metrics from seven different legacy military radio systems into RFC4938-compliant messaging. On the server/router-side, a common virtual multipoint interface (CVMI) [10] was developed to aggregate link metrics and expose the information to a modified Quagga open source router (OSR). The open source router [11] received these per link metrics and dynamically calculated OSPF costs and recomputed shortest path calculations [12].

While RFC4938/5578 represented a good starting point to test R2RI applicability in DoD radio systems, its inefficient medium re-use with broadcast radio systems, and additional overhead of constructing PPP tunnels over the air, resulted in the development of two other R2RI technologies: Radio-Router Control Protocol (R2CP) [2] and Dynamic Link Exchange Protocol (DLEP) [3]. Although both protocols show promise and are currently in draft form in the IETF, evaluating each technology in emulated and real-world environments is necessary. Additionally, providing open source implementations allow for independent testing and increased extensibility.

To evaluate R2CP and DLEP, both client/radio-side code and server/router-side code must be developed. In this paper, we detail modifications made to open source implementations of R2CP and DLEP server/router code to interact with an open source Quagga router. Per link information is fed to Quagga and OSPF costs are dynamically calculated based on link metrics. The reported current data rate (CDR) is used to provide rate-based flow control on the radio-to-router interface. We rely on other work to describe implementation details of the R2RI client-side functionality and integration with existing network emulators. Key contributions include:

- An open source implementation of fully functional dynamic routing based on link metric capability with DLEP and R2CP support.
- An open source implementation of dynamic rate-based flow control based on radio link metric feedback.
- Functional evaluation of each of the key components

The rest of the paper is organized as follows: Section II overviews the key components of the dynamic open source router with R2RI extensions. Section III delves into each component, describing design decisions and implementation details. Section IV gives results from a functional evaluation of the setup. Finally, Section V concludes the paper with future work and lessons learned.

II. R2RI Server Overview

To provide R2CP and DLEP functionality on the router, several components had to be developed or modified. To understand the design decisions, it is important to first examine the previous architecture with the RFC4938 server. Figure 3 illustrates the previous R2RI server-side architecture and corresponding changes. Under the previous architecture, three major components functioned together to expose link metrics to the router and to perform dynamic routing. First, a PPPoE server was run on the same system to setup and manage RFC4938 PPPoE connections between the radio and router. Next, to support multicast and broadcast replication on PPP tunnels, a common virtual multipoint interface (CVMI) [10] was developed to inspect PPPoE link packets from the client and track per link metrics. These link metrics were exposed to the Linux userspace using Generic Netlink [13] messages. Lastly, these Generic Netlink messages containing per neighbor link metrics were received by a modified Quagga router [11], [14] that utilized metrics to calculate dynamic and time-varying per link OSPF costs [12].

To reuse as much of the infrastructure as possible, the same Generic Netlink interface was utilized to expose link metrics to the modified Quagga router. Instead of a PPPoE-Server and CVMI, DLEP and R2CP servers were used, which setup and maintain R2RI connections with the DLEP and R2CP clients. The DLEP/R2CP server exposed per link metrics using the Generic Netlink interface which was received by both the modified open source router and a basic QoS process (*qosd*), which utilized *ebtables* and *tc* filter commands to limit per link rate. The data path flowed to the Ethernet interface, via a common logical Ethernet bridge interface, instead of a CVMI interface as with RFC4938. In the following section, we detail each of the three major components of the R2CP/DLEP server-side setup.

III. Implementation Details

The radio-to-router interface, by its very nature, has components on both the radio and router. Previous work [6], [7], [9] laid the groundwork for supporting link metrics in an open source Quagga router implementation with RFC4938. Many of the same interfaces were re-used to feed DLEP and R2CP link metrics into the open source router.

A. Quagga OSPF Modifications

Quagga [11] is an open-source routing suite that provides implementations of various routing protocols including OSPF, RIP, BGP and IS-IS. It has been an important tool in designing and testing routing protocols as it is possible to change the source code and add modifications. Recently, the Quagga routing suite was modified by adding enhancements to support mobile ad hoc networks (MANETS) to the OSPFv3 routing protocol called OSPF-MDR [14]. To support R2RI protocols, link metric extensions were modified on Quagga to receive per link metrics through a Generic Netlink framework.

The modifications made to the Quagga routing suite to support R2RI protocols were designed with RFC4938 in mind but they also work with modified DLEP and R2CP servers.

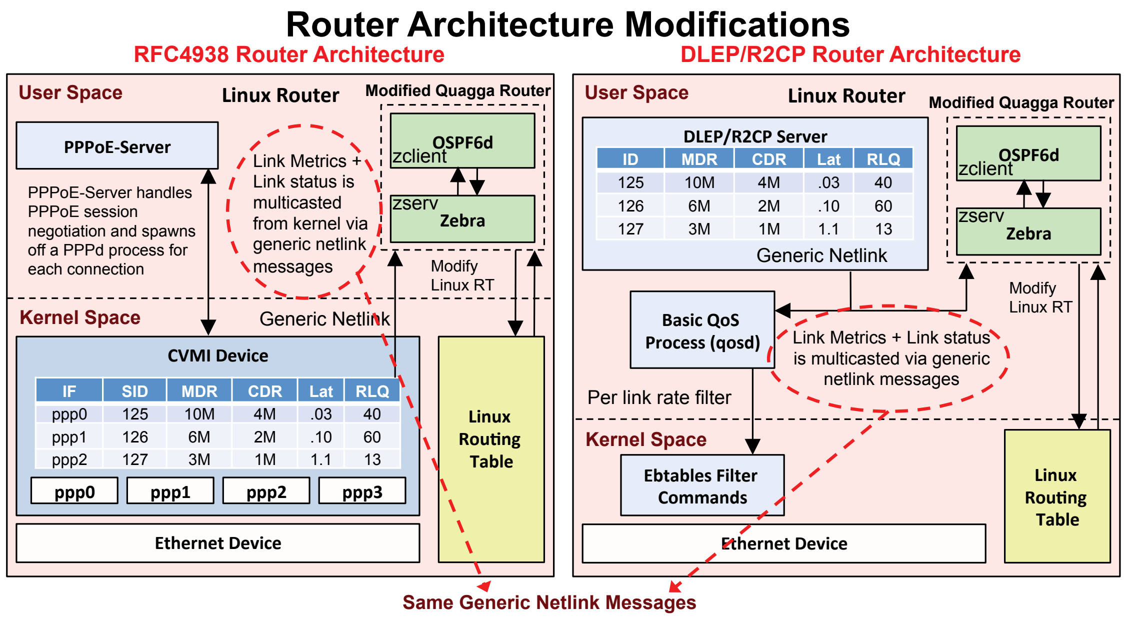

Fig. 3. R2RI Server-side Architecture and Modifications to the RFC4938/5578 architecture to support DLEP and R2CP. The same generic netlink interface to share link metrics with the modified Quagga router is used.

Quagga was modified to listen for link status and link metrics updates from the radios and then use this information in the OSPFv3 routing protocol to make better routing decisions. Figure 3 shows an example of the modified Quagga router interfacing with a DLEP [3], [15] session.

The interface between Quagga and router-to-radio protocol server instances is implemented using Generic Netlink protocol [16], [13]. The *zebra* instance of the Quagga routing suite subscribes to the custom Generic Netlink multicast group and listens for incoming multicast messages. There are three different commands that may be contained in those messages: a STATUS message that contains link status information (link up or down) of a particular neighbor, a PADQ message that contains link metrics information of a particular neighbor, and a PADQ REQUEST message that is a request that the Quagga router can send to find out link metrics of a specific neighbor. Each of these individual messages can contain various attributes, which include link status, neighbor MAC addresses, neighbor IP addresses, the various link metrics like CDR, MDR, latency, RLQ, and resources as well as protocol session and network interface ID's.

The *zebra* daemon listens for the Generic Netlink messages and upon receiving them passes the information along to the *ospf6d* daemon. When the *ospf6d* daemon receives a link up command for a neighbor that is currently down it immediately sends an expedited *hello* message to that neighbor and resets timers. The neighbor will immediately respond to that *hello* message and the process of creating an OSPF adjacency begins. After the adjacency is obtained, the *ospf6d* process asks the zebra daemon to send the PADQ REQUEST message on its behalf. The request message is sent via Generic Netlink multicast and the process responsible for this neighbor responds with the required link metrics. After receiving a link metrics update for a particular neighbor the *ospf6* daemon calculates the OSPF cost based on those metrics. Finally, when the ospf6d daemon receives a link down command, the adjacency and the route to that neighbor are immediately removed and the SPF tree is recalculated.

B. DLEP/R2CP Server

To provide DLEP and R2CP functionality on the server-side, open source implementations [17], [15] of the protocol were leveraged. Although development of ground-up DLEP/R2CP solutions was possible, we chose to leverage the open source implementation because: 1) the code was fairly stable and tested, 2) both DLEP and R2CP protocols are still in draft form in IETF with the potential for changes high (we didn't want to re-implement the changes for every draft release), and 3) the modularized implementation allows flexibility with other R2RI techniques. In order to integrate DLEP and R2CP servers with the Quagga router, while maintaining interoperability with prior RFC4938/5578 implementations, several minor modifications were made.

As described in the Quagga modifications section, the OSR had been previously modified to use dynamic link metrics reported by the radio. Originally sent from a kernel module (CVMI) using a Generic Netlink protocol, these link metrics were exposed by the user space DLEP and R2CP servers using the same Generic Netlink protocol interface. The event handlers of both DLEP and R2CP servers were modified to send out link status change information (link up and down events) and link metric information whenever the server received an update from the client. This information was received by Quagga and used in routing decisions. In addition to proactively sending updates, the servers were modified to listen to link metrics request messages and reply with requested information. These changes can be seen in Figure 3.

The current implementation has a few caveats worth mentioning. DLEP and R2CP are Layer 2 protocols and represent nodes by their MAC addresses (DLEP has an option of specifying IPv4 and/or IPv6 addresses as well, but it is not required). However, the Generic Netlink link metrics protocol requires either an IPv4 or IPv6 address to represent a node. To mitigate this, the implementation assumes that the IPv6 link local addresses have been generated from the MAC addresses using the standard EUI-64 method [18] and thus the IPv6 addresses required by the Netlink protocol are assumed using this method from the MAC addresses specified in DLEP/R2CP messages if IPv4/IPv6 addresses are not given.

Another caveat in the implementation is the data rate representation mismatch between DLEP and RFC4938 protocols for specifying CDR and MDR. RFC4938 specifies the data rate in Kbps in the PADQ messages but DLEP specifies the data rate in bps. Since Quagga router is oblivious to the router-to-radio protocol used and was originally designed with RFC4938 protocol in mind, it expects CDR and MDR data rates in Kbps. Therefore, the DLEP server was modified to convert the data rates it receives from the client from bps to Kbps before sending them via the Generic Netlink protocol.

Finally, the implementations of DLEP and R2CP server assume that a custom Generic Netlink group already exists and therefore requires some other process to create it. As a result, a small kernel module was developed to register the Generic Netlink group.

C. QoS Support

QoS support for the DLEP/R2CP protocols was implemented using Linux Traffic Control (*tc*) functionality along with Ethernet Bridge Tables (*ebtables*). QoS was implemented in an effort to prioritize and shape traffic in accordance to the bandwidth characteristics of a radio link, as specific by R2RI link metrics. Without explicit QoS control, Linux would effectively prioritize and shape traffic based on the underlying physical interface speed, most often a Fast Ethernet interface, in a prioritized First-In-First-Out (FIFO) nature. Much of the desired QoS behavior and scripts used for testing R2CP and DLEP was a result of previous work [6] with the Air Force Research Laboratory, when testing RFC5578.

The overall QoS behavior desired was to classify IPv4 traffic based on Differentiated Services Code Point (DSCP), prioritize the traffic based on the its DSCP, and then shape the prioritized traffic based on a percentage of the overall available bandwidth. From a priority perspective, packets marked with a DSCP of CS7 would have strict priority. Packets marked with a DSCP of CS6 through CS0 (and DSCP values in between) were prioritized and queued based on a Class Based Weighted Fair Queuing (CBWFQ) methodology. Fine-grained QoS was not implemented for IPv6 traffic as the primary traffic under test was IPv4-based and the only IPv6 traffic present during testing was OSPFv3 and ICMPv6. IPv6 link-local traffic, primarily used by OSPFv3 and ICMPv6, was manually placed in the same queue as CS7-marked IPv4 traffic (highest priority).

During normal QoS processing, IPv4 packets are classified using tc filters. Tc filters are packet-matching filters that select traffic based on packet characteristics, such as DSCP or other IPv4 header information, and then place the packet in a specific queue. However, for DLEP and R2CP scenarios, two stages of classification were needed due to the fact that each

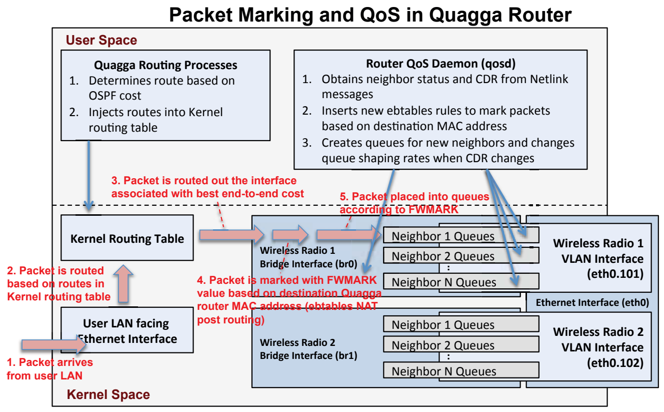
Packet Marking and QoS in Quagga Router
User Space
Quagga Routing Processes
1. Determines route based on OSPF cost
2. Injects routes into Kernel routing table
Router QoS Daemon (qosd)
1. Obtains neighbor status and CDR from Netlink messages
2. Inserts new ebtables rules to mark packets based on destination MAC address
3. Creates queues for new neighbors and changes queue shaping rates when CDR changes
3. Packet is routed out the interface associated with best end-to-end cost
5. Packet placed into queues according to FWMARK
Kernel Routing Table
Wireless Radio 1 Bridge Interface (br0)
Neighbor 1 Queues
Neighbor 2 Queues
Neighbor N Queues
Wireless Radio 1 VLAN Interface (eth0.101)
2. Packet is routed based on routes in Kernel routing table
4. Packet is marked with FWMARK value based on destination Quagga router MAC address (ebtables NAT post routing)
Ethernet Interface (eth0)
User LAN facing Ethernet Interface
Wireless Radio 2 Bridge Interface (br1)
Neighbor 1 Queues
Neighbor 2 Queues
Neighbor N Queues
Wireless Radio 2 VLAN Interface (eth0.102)
1. Packet arrives from user LAN
Kernel Space

Fig. 4. Packet marking and QoS in open source router

individual neighbor, located within a single layer-2 broadcast domain, is differentiated by destination MAC address instead of physical or logical interface (ppp or eth). This required the packet be first classified based on Layer 2 and then Layer 3 information. Layer 2 information consisted of the destination MAC address of the DLEP or R2CP neighbor router. Layer 3 information was based on IPv4 DSCP marking. The initial destination MAC address classification was implemented using a logical Ethernet bridge interface in the Quagga router along with Ethernet Bridge Table ($ebtables$) rules on that interface. For the Quagga router, the $ebtables$ POSTROUTING chain within the Network Address Translation (nat) table was used to first assign a Firewall Mark ($fwmark$) to a packet according to the destination router MAC address. This $fwmark$ was used to place the packet into a subset of additional queues on the outgoing Ethernet interface, which would then look at IPv4 DSCP markings and queue appropriately based on the link metrics to that neighbor. Figure 4 illustrates this process.

The queuing disciplines utilized under Linux tc were Hierarchical Token Bucket (HTB) and Stochastic Fair Queuing (SFQ). HTB queues and classes were structured in a way to implement the percentage-based, class-based queuing and SFQ was used to implement FQ within a given class. Differing priorities assigned to HTB classes were used to set servicing priority and allocate spare bandwidth to higher priority traffic classes.

The implementation of QoS was broken into to two parts:

- Shell scripts that would instantiate $ebtables$ rules and tc queues/classes/filters based on a neighbor ID and the available Current Data Rate (CDR) to a neighbor
- A process ($qosd$) that would monitor the status of neighbor links and call the appropriate scripts when neighbors were created or when link metrics changed.

On the OSR a process called $qosd$ monitored Generic Netlink messages being sent by the DLEP/R2CP server processes to Quagga. For DLEP and R2CP, the $qosd$ daemon would create $ebtable$ rules and tc queues/classes/filters when a new neighbor appeared and also change the CDR reported to the queues, for calculating bandwidth percentages, when a neighbor's link bandwidth changed. For multicast traffic a separate set of queues, of the same structure used for a unicast neighbor, was constructed to service multicast traffic.

IV. Functional Evaluation

In this section, we provide a functional evaluation of the system, taking results from a 5-node outdoor field test with actual 802.11 radios, and a 10-node emulated environment. The emulated network allow for repeatable tests on the system while the outdoor field tests help uncover real-world effects. The goal of this section is to show the system functioning in various test environments, not to provide a holistic performance review. As a result, only a sample of the results are given. For more test details, see [19]. It is also important to note that the specifics of the 802.11 implementation for this test are not detailed as they are not pertinent to the functional evaluation of the R2RI. The 802.11 radio used in these tests could be easily replaced by any broadcast, point-to-multipoint, or point-to-point radio.

A. Link Results

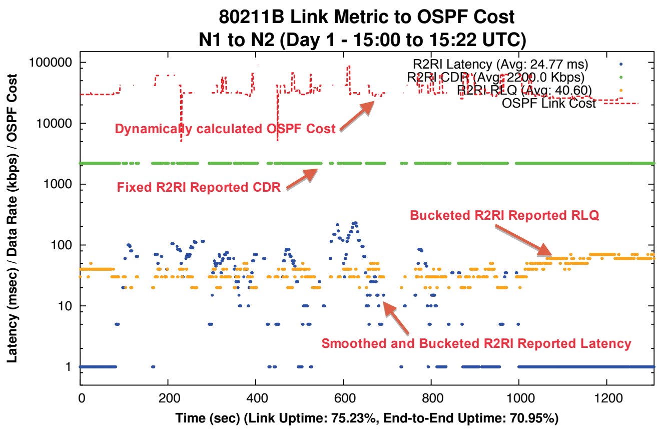

Fig. 5. Calculated OSPF cost for R2RI reported CDR, latency, and RLQ

Figure 5 shows 802.11B link data rates, latencies, RLQ, and OSPF costs seen in the field test, as reported by the R2RI protocol. In this particular example the reported R2RI CDR was fixed to 20% of the 802.11 operating data rate (11Mbps), which was 2.2 Mbps. Calculating an accurate per-neighbor data rate within an 802.11 ad-hoc network is complex and not the goal of this work. As such, operating data rates and scaling values were fixed to simplify CDR reporting, avoid 802.11 dynamic data rate auto negotiation, and avoid overrunning the shared medium. The figure shows varying reported latency, which shows jitter due to increased traffic load on the network as well as queue backlog when links were unavailable. The figure also shows an 802.11 RLQ metric, calculated by querying a customized MadWifi driver, obtaining a neighbor's RSSI, and normalizing the value to range of 0-100. Both latency and RLQ values were averaged and bucketed to address rapidly changing metrics and prevent link flapping. Lastly, the figure shows the calculated OSPF cost as a function of time based on the reported link metrics. Dynamic OSPF costs indicate successful passing of per link metrics to and from the DLEP and R2CP servers to the OSR.

While outdoor field tests illustrate how real radios operate, the results are often not repeatable exactly. For repeatable results, we turn to our 10-node emulation network. Table I

TABLE I
10-Node Avg. Measured Emulated Link Metrics

Emulated Radio	Avail	MLat	RLat	CDR	RLQ
802.11-1 (24Mbps)	81.9%	22.6ms	44.1ms	4.8Mbps	71.8
802.11-2 (48Mbps)	52.3%	22.5ms	48.2ms	9.6Mbps	68.2

shows the link availability (Avail), measured latency (MLat), reported smoothed and bucketed latency (RLat), scaled current data rate (CDR), and relative link quality (RLQ) averaged over all tests and all runs for both emulated 802.11 radios (802.11-1 and 802.11-2). As expected, the "802.11-1" radio with a 24 Mbps operating rate achieved higher availability than the "802.11-2" radio with a 48 Mbps operating rate because lower data rate leads to ability to decode packets at much higher distances. The operating rate for both radios were fixed to remove rate auto negotiation as a variable and for ease of evaluation. The CDR values were then scaled to 20% of the operating rate, as with the outdoor field tests. As mentioned earlier, successful receipt of link metrics indicates a functional server-side DLEP/R2CP implementation.

B. End-to-End Network Results

One of the basic statistics gathered for any kind of network system is end-to-end availability. End-to-end availability can be measured in many ways. In the following subsection, we present network availability measured through per second all-to-all link pings (for actual bi-directional data availability) as well as end-to-end route availability. Route availability, although an important metric, is often times misleading because even a stale route is seen as an available route. End-to-end link pings, however, are often affected by traffic load, congestion, and asymmetric path issues. By presenting both points of data, one can get a fairly good range of typical application availability.

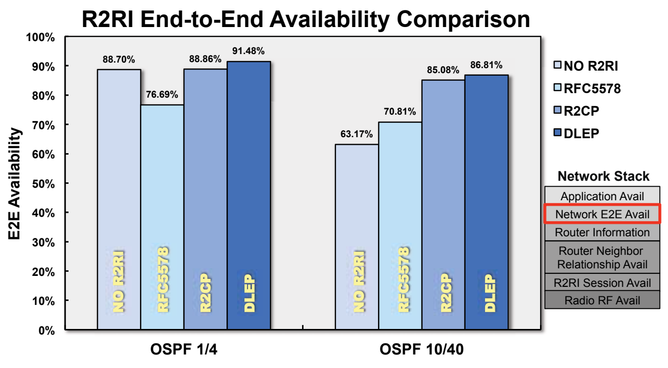

Fig. 6. R2RI protocols allow decreasing link polling and overhead while maintaining high availability

Figure 6 shows the average end-to-end ping availability for each of the R2RI protocols when utilizing two sets of OSPF hello/dead intervals. All results are averaged over 10 runs. As can be seen, when the OSPF hello interval is 1 second and the OSPF dead interval is 4 seconds (OSPF 1/4) OSPF is able to quickly discover new neighbors and recover from network outages fairly quickly, even with no R2RI protocol running, and there is high end-to-end ping availability. However, when hello and dead intervals are increased to 10 seconds and

40 seconds (OSPF 10/40) OSPF availability is more reliant on link up/down status messaging from R2RI protocols to discover neighbor adjacencies. As such, using R2CP and DLEP can achieve near-optimal availability with little OSPF polling. RFC5578 achieves the worst end-to-end availability due to the time needed for end-to-end PPP tunnel negotiation with new neighbors and/or recovery from lost PPP control packets during initial negotiation on a lossy network.

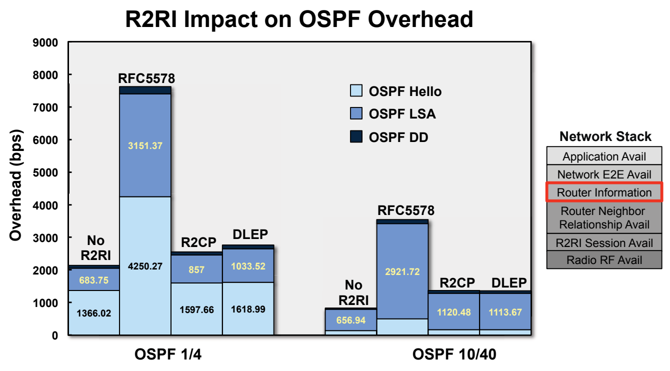

Fig. 7. DLEP and R2CP overhead is slightly higher than no R2RI, but RFC5578 is much higher due to multicast replication

Figure 7 compares the average OSPF overhead per node under each of the R2RI protocols with differing OSPF hello and dead intervals. As expected, overhead is much higher when the hello and dead intervals are set lower (OSPF 1/4) due to more OSPF hello messages and the increased chance of a neighbor needing to re-establish an adjacency due to lost hellos. Also, OSPF costs change over time with dynamic link metrics, when using R2RI protocols, resulting in more routing overhead in the form of LSAs. It is important to note that RFC5578 incurred significantly more overhead than DLEP or R2CP because of its multicast replication over all PPP tunnels. The more neighbors a node has, the more overhead is required for RFC5578. End-to-End network results and ability to pass data indicate functioning implementations of all R2RI server-side protocols.

Finally, though DLEP and R2CP are similar and perform almost equally having the option in DLEP to communicate IP address information for a neighbor, versus simply a MAC address, can be beneficial. If requirements dictate the use of non-EUI-64 IPv6 link-local addressing, and the R2RI protocol does not provide IP information, then alternative methods need to be used discover a neighbor's IP address information prior to OSPF adjacency formation. Such methods include gratuitous ARP or waiting the duration of an OSPF hello interval for a hello packet to be received. Requirements for additional end-to-end traffic to obtain IP addressing can result in worse performance on a lossy network, delaying the formation of OSPF adjacencies, much like during PPP negotiations for RFC5578.

V. Conclusion

In this paper, we detail modifications made to open source implementations of R2CP and DLEP server/router code to interact with an open source Quagga router. Per link information is fed to Quagga and OSPF costs are dynamically calculated based on link metrics. Reported current data rate (CDR) is used to provide rate-based flow control on the radio-to-router interface. Additionally, we provide brief test results from a 5-node field test and a 10-node emulated network to show functionality of the R2CP/DLEP server/router-side code. The server-side DLEP and R2CP implementations were leveraged in several tests involving emulated radios and live 802.11 radios [20], [19]. Future work includes developing a more robust dynamic QoS solution based on virtual interfaces.

References

[1] B. Berry, S. Ratliff, E. Paradise, T. Kaiser, and M. Adams, "PPP Over Ethernet (PPPoE) Extensions for Credit Flow and Link Metrics," IETF, RFC 5578, 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5578.txt

[2] D. Dubois, A. Kovummal, B. Petry, and B. Berry, "Radio-Router Control Protocol (R2CP)," IETF, Internet Draft (work in process) 00, 2011. [Online]. Available: http://tools.ietf.org/html/draft-dubois-r2cp-00

[3] S. Ratliff, B. Berry, G. Harrison, S. Jury, and D. Satterwhite, "Dynamic Link Exchange Protocol (DLEP)," IETF, Internet Draft (work in process) 02, 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-manet-dlep-02

[4] B.-N. Cheng, J. Wheeler, and L. Veytser, "Radio-to-Router Interface Technology and Its Applicability on the Tactical Edge," in *IEEE Communications Magazine*, October 2012.

[5] B. Berry and H. Holgate, "PPP Over Ethernet (PPPoE) Extensions for Credit Flow and Link Metrics," IETF, RFC 4938, 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4938.txt

[6] B.-N. Cheng, R. Charland, P. Christensen, A. Coyle, E. Kuczynski, S. McGarry, I. Pedan, L. Veytser, and J. Wheeler, "Characterizing Routing with Radio-to-Router Information in an Airborne Network," in *IEEE Military Communications Conference, MILCOM 2011*, November 2011.

[7] R. Charland, P. Christensen, J. Wheeler, and B.-N. Cheng, "A Testbed to Support Radio-to-Router Interrface Testing and Evaluation," in *IEEE Military Communications Conference, MILCOM 2011*, November 2011.

[8] B.-N. Cheng, R. Charland, P. Christensen, A. Coyle, L. Veytser, and J. Wheeler, "Characterizing Routing with CoTS Radio-to-Router Information in an Airborne Network," in *AIAA Aerodynamic Measurement Technology, Ground Testing, and Flight Testing Conference 2012*, June 2012.

[9] B.-N. Cheng, R. Charland, P. Christensen, L. Veytser, and J. Wheeler, "Evaluation of a Multi-hop Airborne IP Backbone with Heterogeneous Radio Technologies," in *ACM Mobihoc Airborne Networks and Communications Workshop 2012*, June 2012.

[10] L. Veytser and B.-N. Cheng, "An Implementation of a Common Virtual Multipoint Interface in Linux," in *IEEE Military Communications Conference, MILCOM 2011*, November 2011.

[11] "Quagga open source router source code." [Online]. Available: http://www.quagga.net

[12] "Mobile Ad Hoc Networks for Router-to-Radio Communications," Cisco Systems, Tech. Rep., 2007. [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_4t/ip_mobility/configuration/guide/ip_manet.html

[13] "Libnl Code." [Online]. Available: http://www.infradead.org/~tgr/libnl/

[14] "Boeing OSPF-Manet Quagga." [Online]. Available: http://downloads.pf.itd.nrl.navy.mil/ospf-manet/

[15] "DLEP open source code." [Online]. Available: http://dlep.sourceforge.net

[16] *Linux Generic Netlink Howto*. [Online]. Available: http://www.linuxfoundation.org/en/Net:Generic_Netlink_HOWTO

[17] "R2CP open source code." [Online]. Available: http://r2cp.sourceforge.net

[18] IEEE Standards Association, "Guidelines for 64-bit Global Identifier (EUI-64)Registration Authority," IEEE, Tech. Rep. [Online]. Available: http://standards.ieee.org/develop/regauth/tut/eui64.pdf

[19] B.-N. Cheng, R. Charland, P. Christensen, A. Coyle, I. Pedan, L. Veytser, and J. Wheeler, "Comparing Radio-to-Router Interface Implementations on Experimental CoTs and Open Source Routers," in *IEEE Military Communications Conference, MILCOM 2012*, October 2012.

[20] L. Veytser, R. Charland, and B.-N. Cheng, "Integrating Radio-to-Router Protocols into EMANE," in *IEEE Military Communications Conference, MILCOM 2012*, October 2012.