

# Customized Routing in Mesh Networks

Nadine Shillingford and Christian Poellabauer  
Department of Computer Science and Engineering  
University of Notre Dame  
Notre Dame, IN 46556  
{nshillin, cpoellab}@nd.edu

**Abstract**— Wireless mesh networks are increasingly used as multi-purpose networks, i.e., they serve multiple objectives and different applications simultaneously. As a consequence, a one-size-fits-all routing solution is difficult to achieve, particularly when the performance and QoS expectations of these applications differ. This work proposes CMR (Configurable Mesh Routing), a toolkit that supports the discovery of routes based on any combination of a number of supported QoS metrics. This enables network users to use customized routes that meet their unique needs. Our experiments on a 20-node mesh testbed illustrate how CMR can be used to (a) implement novel routing protocols on-the-fly and (b) emulate a variety of existing reactive routing protocols.

## I. INTRODUCTION

A wireless mesh network (WMN) is an ad-hoc network with two types of devices: mesh routers (forming the communication infrastructure) and mesh clients (e.g., mobile devices carried by network users). A rapidly increasing number of wireless mesh networks are used to provide broadband Internet access, wireless communications to rural or under-developed areas, ad-hoc communications among first responders, or backhaul communications for sensor and control networks. A trend in these networks is that they are increasingly being used for a large variety of objectives and applications, e.g., Oklahoma City's mesh network, which is the largest city owned and operated municipal WiFi mesh network covering an area of 555 square miles, provides access to more than 150 different software applications from the field. Typical wireless mesh networks base their communications on a single, one-size-fits-all, routing solution, e.g., protocols such as Dynamic Source Routing (DSR) [5] and Dynamic Destination Sequence Distance-Vector Routing (DSDV) [13] establish routes based on a single Quality-of-Service (QoS) metric, giving applications and users little choice or flexibility in route establishment. However, the QoS expectations of these applications may differ widely [1], necessitating more flexible solutions.

Toward this end, this work introduces the CMR (Configurable Mesh Routing) toolkit which provides an easy-to-use API for WMNs, allowing applications or users to implement their own reactive routing protocols based on any combination of a variety of pre-defined QoS metrics. Our current implementation supports five popular QoS metrics or constraints, but is easily extensible to support additional metrics. While CMR allows us to provide customized routing solutions on-the-fly, it is also a valuable tool for rapid prototyping, protocol evaluations, and teaching purposes.

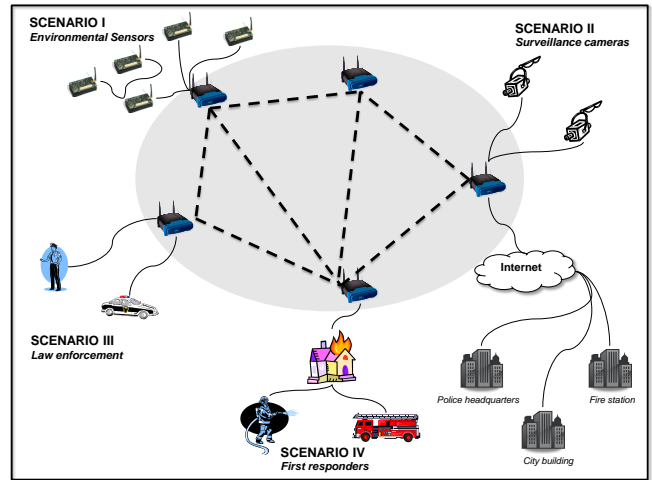


Fig. 1. Motivational scenario.

## II. MOTIVATIONAL SCENARIO

The mesh routers in a WMN serve as a backbone for numerous sensor networks or municipal wireless broadband access scenarios, i.e., the same mesh routers deployed throughout a city may service multiple data transmissions. Figure 1 shows a scenario with several applications relying on the same WMN:

- 1) environmental sensors collect information on weather and air pollution,
- 2) surveillance cameras monitor crime-ridden neighborhoods or malls,
- 3) law enforcement officers check license plates and transfer tickets directly to their headquarters, and
- 4) first responders coordinate their rescue activities via handhelds and other portable communication devices.

In each of these scenarios, the devices used serve as mesh clients connecting to the same mesh routers. While the same WMN supports multiple application scenarios, the differences in QoS route requirements for these applications become obvious. For example, these requirements may include constraints in latency (scenarios 2 and 4), reliability (3, 4), energy consumption (1-4), security (2, 3, 4), or bandwidth (2, 4). Using CMR, routes customized to each scenario can be established and operated simultaneously.

Metric	Protocol
Shortest path	DSDV, CGRS, WRP, DFR, IARP, MMRP, OLSR, TBRPF, AODV, DSR, TORA, ARA, Ariadne, AOMDV, BSR, CHAMP, DYMO, DNVr+, IERP, LUNAR, MOR, DQSR <sup>a</sup> , ZRP, GSR, FSR+, CBRP, LQSR, STAR
Link quality/strength	Babel, Guesswork, SSA, LQSR, QuaSAR, DQSR <sup>b</sup>
Reliability	Guesswork, BSR, LBR, ABR, MRP+
Bandwidth	LSQR <sup>b</sup> , ACOR+, AQOR, QuaSAR, DQSR <sup>b</sup> , FQMM, OLSMQ <sup>c</sup> , QRP using TDMA <sup>d</sup>
Latency	LSQR <sup>b</sup> , ACOR+, AQOR, QuaSAR, DQSR <sup>b</sup> , HSR
Energy/Battery power	QuaSAR, Guesswork, CC

<sup>a</sup>Distributed QoS Routing protocol (abbr.).

<sup>b</sup>A link-state QoS routing protocol (abbr.).

<sup>c</sup>The On-Demand Link-State Multi-path QoS Routing Protocol (abbr.).

<sup>d</sup>QoS Routing Protocol for Ad-Hoc Networks using TDMA (abbr.).

TABLE I  
ROUTING PROTOCOLS BY QoS METRICS

### III. RELATED WORK

Numerous routing protocols for ad-hoc and mesh networks [1] have been proposed over the last several years. Most ad-hoc network routing protocols consider one or two QoS metrics only and all applications and clients in such a network must use the same protocol and metrics. Once a network is deployed, these metrics can only be changed by modifying the routing layer on each network node. We have studied numerous routing protocols that have been proposed from 1994-2008 and Table I shows a list of popular routing protocols and the QoS metrics that they implement. The vast majority of protocols support a single metric while only 10% implement more than two QoS metrics. Three of the routing protocols investigated in our study [10, 12, 6] indicate that they can be extended to allow more QoS metrics (indicated by a ‘+’ sign in Table I), but proposals suggest the extensions as future work and remain to be completed. Table II gives a summary of the number of protocols which implement the six main metrics identified during our study.

QoS Metric	Number of Protocols
Shortest Path	29
Bandwidth	7
Latency	6
Reliability	5
Link Quality/Capacity	6
Energy	3

TABLE II  
ROUTING METRICS IN NUMBERS

Other interesting work in the area of routing protocols include frameworks such as the x-kernel [4]. However, according to [15], frameworks for ad-hoc routing are limited and require a significant amount of work from the protocol developer.

The Click router [7] is a component-based tool that can be used to implement flexible and configurable routers. Elements in Click are modules that perform tasks such as packet selection, modification, and scheduling. A Click configuration is a collection of push and pull elements in the form of a graph. Click is very extendable and it has been used to implement different routing protocols such as ClickDSR [3]. While CMR

has been developed as middleware tool for Unix systems, it could also be implemented by extending Click (which will be considered in our future work).

Work such as [2, 15] utilize generative programming to develop routing protocols. Unlike these approaches, CMR generates the routing protocols on-the-fly based on user QoS specification once the network has already been deployed.

Symbol	Values
symbol =	“>”   “<”
logical =	“&&”   “  ”   “!”
metric =	“HOP-COUNT”   “DELAY”   “BATTERY-LEVEL”   “LOAD”   “STABILITY”
quantity =	“ALL”   “ANY”   number

TABLE III  
CMR LANGUAGE BASIC SYMBOLS

In comparison to these previous efforts, an important design feature of CMR is its easy deployment and ease-of-use, e.g., it does not require dynamic code distribution or reconfiguration of a kernel and the QoS-metrics for route discovery are determined using a simple rule language.

### IV. THE CMR LANGUAGE

The operations of the CMR toolkit are primarily based on the CMR language. This is a syntax used by the source to specify the types of routes that should be selected. For the purpose of illustration we will use the Extended Backus-Naur Form metasyntax notation. The CMR language provides basic and advanced production symbols. A basic production symbol is either a terminal symbol or a system specific constant. These are shown in Table III.

The *metric* terminal symbol is of special significance to the CMR toolkit. This is a symbol representing a route metric. This may include constraints such as hop count, bandwidth, and reliability. The ability to include additional values to this

Symbol	Values
Composite =	("MIN"   "MAX") metric
MetricExpression =	(metric symbol (identifier   number))   (composite symbol (identifier   number)) [logical MetricExpression] [logical OverlapExpression]
OverlapExpression=	"OVERLAP" symbol (number   identifier)

TABLE IV  
CMR LANGUAGE ADVANCED PRODUCTION RULES

terminal symbol is the main contributor to the flexibility of CMR (these metrics are discussed in Section V-B).

Table IV lists the advanced production rules of CMR. These are a sequence of terminals assigned to a non-terminal symbol. Table V gives examples of these advanced production rules. The application may request a composite of a metric. Currently, the composites included in the CMR language are minimum (MIN) and maximum (MAX). Example 1 in Table V is used to select a route with the minimum hop count among a set of routes.

Construct	Example
1. Composite	MIN HOP-COUNT
2. MetricExpression	BATTERY-LEVEL > 50 && HOP-COUNT < 3
3. OverlapExpression	OVERLAP < 40

TABLE V  
CMR LANGUAGE EXAMPLES

Metrics can be compared with constants or variables. One or more of these comparisons comprise a *MetricExpression*. These are expressions used to determine whether a particular route meets a combination of metric criteria. Example 2 in Table V returns the route with average battery level greater than 50% and hop count less than 3.

The last expression in Table IV is the *OverlapExpression*. This type of expression is used to select routes with a certain level of disjointness. Mueller et al. [11] have studied the effect of link disjoint routes. They indicate that link disjoint routes are a good balance between node disjoint and non-disjoint routes. The protocol designed by [9] constructs maximally disjoint routes, that is, paths that have as few links in common as possible. The *OverlapExpression* allows the user or application to specify the amount of overlap allowed in the routes. Example 3 in Table V returns routes whose overlap is less than 40%.

## V. THE CMR ARCHITECTURE

The CMR architecture is based on five main modules - (1) the route discovery module (RDM), (2) the resource monitoring module (RMM), (3) the routing module (RM), (4) the CMR API module (API), and (5) the route cache service. The following subsections explain how the CMR modules interact.

### A. Route Discovery and Routing

Although CMR can be extended to accommodate both reactive and proactive approaches to routing, in this section we focus on a reactive route discovery scenario. Future work will focus on both reactive and proactive scenarios in CMR.

The RDM is responsible for discovering routes while the RM is responsible for routing data packets. When an application on the source needs to send data to a sink, it makes a call to the API.

Each CMR route cache entry consists of a destination IP address, an integer indicating the number of routes stored for this destination and a collection of routes. The source uses the *get\_route()* function to initiate a transmission request to CMR. This function contains two parameters - a QoS specification expressed in the CMR language and the IP address of the destination. This function allows the application to either retrieve routes from the route cache via the RDM or initiate a route discovery in the situation where a route to the sink is not stored in the route cache.

The route cache service is the module responsible for managing access to the route cache by the RDM and RM. If there is an entry containing a route to the sink in the route cache then data can be streamed directly to the sink. However, if no entry is found, then a message is sent to the RDM indicating that a route discovery process should be initiated. The RDM broadcasts a CMR request packet (CMR-RREQ) to the network layer.

The *CMR-RREQ* consists of all the QoS data including the QoS specification, the value of each metric as well as the minimum and maximum metric values for the route.

The Receiver Daemon (RD), a background process running on every CMR-enabled node in the network, receives CMR packets from the network layer and determines their type. Based on the type of the packet the RD directs the packet to the right module.

When the RDM receives a CMR-RREQ from the RD, the rule is parsed to determine which metrics have been requested for this particular CMR-RREQ. For example, if the application specified battery level and delay as its requirements, the rule parser in the intermediate node's RDM receives information about the node's metrics from the RMM (to be discussed in Section V-B).

Whenever the RDM on a node receives a CMR-RREQ, it determines whether it is the sink. The RDM parses the CMR rule stored in the CMR-RREQ packet header. This is different from the parsing done at the intermediate nodes. The RDM determines which metrics are specified within the rule and what values are required. The route is discarded if the total number of routes already stored in the route cache (for this CMR-RREQ) is equal to the number of routes requested or if the algorithm determines that the route does not meet the QoS specification.

The RDM reverses the selected route(s) and stores them in the header of a CMR-RREP packet. This packet is forwarded to the network layer where it is unicasted back to the source who initiated the route discovery.

Once the RDM on the source receives a CMR-RREP, it stores the routes in the route cache and passes the route options (including the QoS of each route) to the application. An application can then use the `send_data()` function call to CMR to send data to the sink using one or more of the route options. As a result of this call, the RM streams data using unicast to the sink. Similar to DSR, routes are stored within the data packet and each intermediate node determines which neighbor is the next hop toward the sink.

### B. Resource Monitoring and Route Maintenance

**Resource Monitoring in Linux.** The RMM is responsible for monitoring QoS metrics during the route discovery process. When an intermediate node receives a CMR-RREQ, the RMM on the node will determine the status of each QoS metric specified in the CMR-RREQ. The RMM determines the status of each QoS metric on-demand for the intermediate node. The following list describes how the RMM retrieves each of the five currently supported QoS metrics/constraints in a Linux environment.

- 1) Hopcount (HOP-COUNT): This is the number of hops that a CMR-RREQ is forwarded through before it arrives at the destination. This value is incremented every time an intermediate node forwards a packet. In the DSR RFC specification, this is equivalent to the `num_addresses` variable.
- 2) End-to-end delay (DELAY): A time stamp,  $t_1$  is added to the request packet when it leaves the source. When it arrives at its destination, the time is noted,  $t_2$ . The total latency of the transmission is  $t_2 - t_1$ . This particular method used by CMR requires time synchronization. Algorithms for time synchronization in mesh networks has been presented by [8, 16]. For experimental purposes we used the `ntpd` utility for time synchronization. Delay is expressed in time units.
- 3) Battery Power Level (BATTERY-LEVEL): This is the battery level of a node. To determine this value on Linux, the resource management module reads the `last full capacity` value in the file `/proc/acpi/battery/BAT0/info` and compares it to the `remaining capacity` value from `/proc/acpi/battery/BAT0/state`. The battery level is expressed as a percentage.
- 4) Load (LOAD): The load  $L_i$  of a node  $i$  is defined as the bandwidth utilization at each node divided by the manufacturer specified bandwidth capacity of  $i$ . Measurement of the route load can allow a user or application to avoid highly loaded/used nodes. The bandwidth utilization at each node can be retrieved from the `/proc/net/dev` file. This file contains the number of bytes received and transmitted on each network interface. The bandwidth utilization of node  $i$  is equivalent to the sum of the number of bytes transferred into  $i$  and the number of bytes transferred out of  $i$  during a one second interval. CMR includes a thread that runs in the background on each node. The thread performs this calculation and

stores the current bandwidth utilization in a simple text file. During the parsing at each intermediate node, this file is read and the value is retrieved. Bandwidth utilization is expressed in megabytes per second.

- 5) Stability (STABILITY): Routing protocols such as ABR [14] measure the temporal stability of nodes in the network using a beacon-based approach. In CMR, a node  $i$  sends out beacons at one second intervals containing the node's IP address,  $ip_i$  and the number of beacons it has transmitted  $b_i$  in a beacon interval,  $w$ . The receiving node stores a neighbor table which consists of  $ip_i$ ,  $b_i$ , and a stability value  $s_i$  indicating how many beacons have been received from neighbor  $i$  during the last  $w$  minutes. The STABILITY metric, therefore, is calculated using the formula  $s_i / b_i$  for each beacon interval,  $w$ . A higher stability ratio indicates a more stable route. The record for node  $i$  is reset to 0 if no beacons have been received for  $i$  within  $c * w$  minutes where  $c$  is a network-dependent constant. When CMR is deployed on a node a user may indicate whether beacons should be enabled. Beacons add additional traffic to the network so it is important that the user has the ability to control them.

**Extending the RMM.** Metrics can be added to the CMR implementation by first adding the metric to the `ROUTE` structure. Next, the user inserts code into the RMM to retrieve the metrics. The user also adds functionality to accommodate the new metrics in the sending, forwarding and receiving of requests in the RDM. The length of code added to the RMM will be metric-specific. However, changes to the RDM will be the same for every new metric.

**Route Maintenance.** The route maintenance mechanism of CMR is similar to that used in DSR. When node  $B$  receives a data packet from node  $A$ , it replies with a CMR-ACK. If node  $A$  does not receive the CMR-ACK from node  $B$  it retransmits the packet seven times until it receives a CMR-ACK or else node  $A$  sends a CMR-ERR message back to the source to indicate that a link breakage has occurred. The source will then either utilize an alternate route from its cache or initiate a new route discovery.

## VI. EXPERIMENTAL EVALUATION

Our experimental evaluation of the CMR toolkit consists of two different types of experiments. First, we conducted a case study to demonstrate the flexibility of CMR by implementing variants of two existing protocols and one new protocol. In our second set of experiments, we compared DSR with CMR-DSR and compared the overheads of both approaches. In both cases, our testbed consisted of Stargate devices which are small single board computers equipped with an Intel 400MHz XScale processor, 64MB SDRAM, and an AmbiCom wireless 802.11 card<sup>1</sup>. Each Stargate device uses the Linux operating system (version 2.4.19). We simulated a grid topology of

<sup>1</sup><http://www.xbow.com>

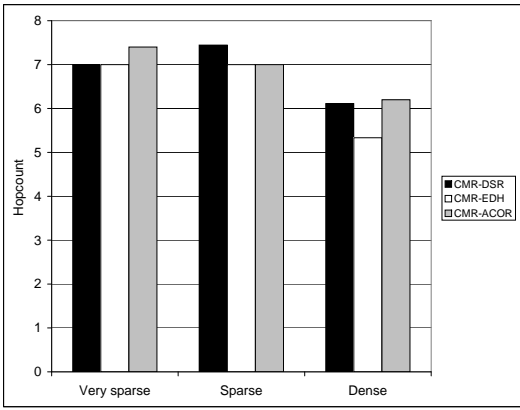


Fig. 2. Number of hops in route

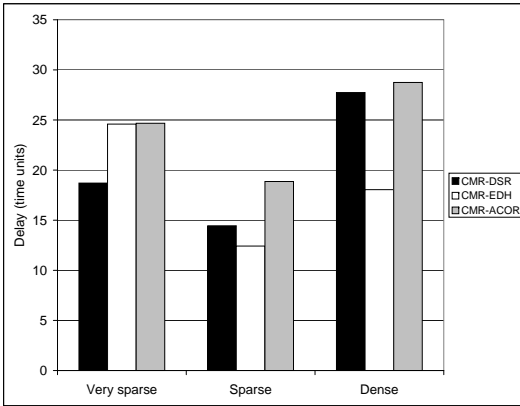


Fig. 3. End-to-end delay

Stargate devices by manipulating the iptables of each device for both experiments.

#### A. Case Studies

For demonstrative purposes, 20 Stargate devices were arranged in three different matrix topologies which we termed *very sparse* (1-3 neighbors), *sparse* (2-4 neighbors), and *dense* (3-5 neighbors). We compared the results of the experiments in terms of hopcount, delay, and load.

We implemented the Dynamic Source Routing protocol (DSR) [5] and Admission Control Enabled On-Demand Routing (ACOR) [6] using the CMR toolkit. In addition, we introduce a novel protocol called CMR-EDH. In all three cases, one route was returned that fits the QoS specifications.

- **CMR-DSR:** We implemented CMR-DSR using the QoS specification  $MIN\ HOP-COUNT$ .
- **CMR-ACOR:** The ACOR protocol proposed by [6] uses the metrics bandwidth and delay in route selection. To implement CMR-ACOR we use the following QoS specification  $MIN\ DELAY \ \&\& \ BANDWIDTH-USED < x$ .
- **CMR-EDH:** Our novel protocol, CMR-Energy-Delay-Hopcount or CMR-EDH, would be rather useful in the event that an application requires a path that has a particu-

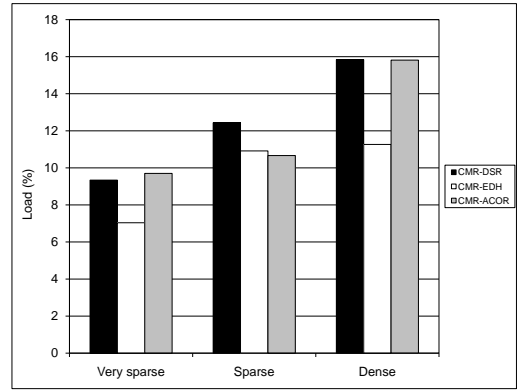


Fig. 4. Load

lar delay threshold in an energy constrained network. The QoS specification for this protocol is  $BATTERY-LEVEL > x \ \&\& \ DELAY < y \ \&\& \ HOP-COUNT < z$ .

Based on the observed hopcount, delay, and load of the routes selected during the CMR-DSR implementation we chose  $x = 2$  for CMR-ACOR and  $x = 50$ ,  $y = 30$ , and  $z = 10$  for the CMR-EDH experiments.

Overall, as indicated in Figure 2, the hopcount of the routes returned by CMR-DSR and CMR-ACOR protocols are similar with only slight variations of about 5% in the very sparse and sparse topologies. The CMR-EDH protocol appears to produce a lower hopcount than the other topologies, however, closer analysis of the data shows that the majority of the longer routes were rejected because of either low battery power or high delays in the routes. The multiple metrics in the CMR-EDH protocol resulted in a smaller number of routes added to the route cache.

The delay of the routes returned from our experiments are displayed in Figure 3. The results show a random trend in delay. The delay of the routes returned by the CMR-EDH protocol remains lower than the other protocols. The results for the dense topology show that the delay of the CMR-EDH protocol is as much as 10 time units lower than the other two protocols combined. CMR-EDH specifies a strict delay constraint therefore, any routes which do not have this requirement are rejected. These rejected routes may be accepted by other protocols. This explanation extends to the CMR-ACOR protocol as well. Although the CMR-ACOR protocol specifies a minimum delay requirement, it is not as strict as the CMR-EDH requirement.

Figure 4 shows that the load increases with the connectivity of the topology. This is because the individual nodes on the route to the destination have a higher load as the number of possible routes to the destination increases. The values for CMR-EDH increase with topology connectivity, however, it levels off in the dense topology. This is directly correlated to the drop in hop count from the sparse to dense topology. The CMR-EDH protocol is a more selective protocol, therefore, many of these routes were rejected based on other criteria

besides the load. The load component of the CMR-ACOR specification is very high, consequently, only 30% of the routes were rejected.

This case study shows that the CMR toolkit can be used to implement existing as well as novel protocols. It also shows the importance of fine tuning the QoS requirements. A constraint that is too high may result in a poorer quality of routes.

### B. Overhead Evaluation

For these tests, our network consisted of 16 Stargate devices where four devices had only two neighbors, four devices had three neighbors while the remaining eight devices had four neighbors.

We investigated the overhead introduced when the CMR toolkit is used to implement a routing protocol (CMR-DSR). The tests for CMR-DSR were run with all the CMR-specific functionality enabled while the tests for conventional DSR were run with all the CMR-specific functionality disabled. For the conventional DSR tests we simply returned the route stored in the first RREQ that arrives at the sink. This way we were able to compare the overheads of the CMR toolkit in terms of latency, CPU, and memory utilization.

When DSR is implemented using the CMR routing toolkit, the amount of time expended during the forwarding process on intermediate nodes is on average 3.6ms more than when DSR is implemented in the conventional method. This additional time can be attributed to the parsing of the rules included in the QoS specification.

In addition to measuring the latency overheads, we also observed the CPU utilization. We used the Unix `getrusage()` function for monitoring CPU usage by initiating a route discovery process every second for a total of 55 minutes. We observed a steady trend in CPU utilization with a slightly higher CPU utilization for the CMR implementation.

## VII. CONCLUSION AND FUTURE WORK

Future mesh networks will be increasingly required to support multiple applications. This paper introduces the CMR middleware toolkit which allows users/applications to dynamically implement existing and new protocols using any combination of supported QoS metrics. We propose the CMR language as well as a complete description of the CMR architecture. We implemented five metrics and provide instructions for further extensions. We showed that a minimal overhead of 3.6ms was observed during the forwarding process as well as a minimal increase in CPU utilization.

In our future work, we will enhance CMR to support end-to-end QoS management features, particularly focusing on autonomous monitoring and maintaining of routes and the effects newly established routes have on previously established routes. This is particularly important since CMR-based networks will operate routes using a variety of QoS parameters and constraints.

## ACKNOWLEDGMENT

This work was supported in part through NSF grant 0545899 and a DURIP grant by the Army Research Office.

## REFERENCES

- [1] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, March 2005.
- [2] M. Barbeau and F. Bordeleau. A protocol stack development tool using generative programming. In *In Proceedings of Generative Programming and Component Engineering (GPCE), 2002. Lecture Notes in Computer Science*, pages 248–7. Springer-Verlag, 2002.
- [3] S. Doshi, S. Bhandare, and T. Brown. An on-demand minimum energy routing protocol for a wireless ad hoc network. *Mobile Computing and Communications Review*, 6 (3), July 2002.
- [4] N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17:64–76, 1991.
- [5] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.
- [6] N. Kettaf, A. Abouaissa, T. Vuduong, and P. Lorenz. A cross layer admission control on-demand routing protocol for QoS applications. *International Journal of Computer Science and Network Security*, 6(9B), September 2006.
- [7] E. Kohler, R. Morris, B., J. Jannotti, and M. R. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.
- [8] T.-H. Lai and D. Zhou. Efficient and scalable ieee 802.11 ad-hoc-mode timing synchronization function. In *AINA '03: Proceedings of the 17th International Conference on Advanced Information Networking and Applications*, page 318, Washington, DC, USA, 2003. IEEE Computer Society.
- [9] S. J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. *IEEE International Conference on Communications*, 10:3201–3205, 2001.
- [10] Y. J. Lee and G. F. Riley. Dynamic nix-vector routing for mobile ad hoc networks. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 4, pages 1995–2001, March 2005.
- [11] S. Mueller, R. P. Tsang, and D. Ghosal. Multipath routing in mobile ad hoc networks: issues and challenges. *Lecture Notes in Computer Science*, 2965:209–234, April 2004.
- [12] G. Pei, M. Gerla, and T. Chen. Fisheye state routing in mobile ad hoc networks. In *Proceedings of ICDCS Workshop on Wireless Networks and Mobile Computing, April 2000, Taipei, Taiwan*, pages D71–D78, April 2000.
- [13] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 234–244, 1994.
- [14] C. Toh. Long-lived ad hoc routing based on the concept of associativity. Internet-draft, IETF MANET Working Group, March 1999. Expired.
- [15] P.E. Villanueva-Pena and T. Kunz. Gp-pro: The generative programming protocol generator for routing in mobile ad hoc networks. In *Proceedings of the 2nd IEEE Workshop on Wireless Mesh Networks (WiMesh 2006)*, pages 129–131, 2006.
- [16] D. Zhou, L. Huang, and T. H. Lai. On the scalability of ieee 802.11 ad-hoc-mode timing synchronization function. *Wirel. Netw.*, 14(4):479–499, 2008.