



Configurable Routing in Ad-Hoc Networks

Nadine Shillingford and Christian Poellabauer
Computer Science and Engineering
University of Notre Dame



Introduction

Problem: The actual use of a wireless ad-hoc network or its operational parameters may be unknown before deployment or they may change during the life time of a network.

Solution: Make customizability and configurability a key design choice for ad-hoc and mesh networks. With respect to routing, we need a toolkit that supports customized route discovery, i.e., each user or application in an ad-hoc network selects its own combination of Quality-of-Service parameters.

Our Implementation: CMR (Configurable Mesh Routing)

Motivation

Only 10% of 42 routing protocols studied implement more than 2 QoS metrics

Metric	Protocols
Shortest path	DSDV, CGRS, WRP, DFR, IARP, MMRPOLSR, TBRPF, AODV, DSR, TORA,ARA, Ariande,AOMDV, BSR, CHAMP,DYMO, DNVr, IERP, LUNAR, MOR, DQSR, ZRP, GSR, FSR, CBRP, LQSR, STAR
Link quality/strength	Babel, Guesswork, SSA, LQSR, QuaSAR, DQSR
Reliability	Guesswork, BSR, LBR, ABR
Bandwidth	LSQR, ACOR, AQOR, QuaSAR, DQSR, FQMM, OLSMQ, QRP using TDMA
Latency	LSQR, ACOR, AQOR, QuaSAR, DQSR, HSR
Energy/Battery Power	QuaSAR, Guesswork

CMR API

Two CMR External Functions

- `int send_rreq(char *destination, char *filename)`
initiates a route discovery to *destination* using the XML QoS specified in *filename.xml*
- `int send_data(char *destination, char *data)`
sends *data* to *destination*

Two CMR Internal Functions

- `int process_doc(RREQ *rreq, xmlNode *a_node)`
Iterates through the QoS xml tree starting from *a_node* and compares the requirements with the metrics stored in *rreq*
- `METRICS *retrieve_metric_values(xmlDoc *rule)`
Retrieves QoS metric values from forwarding nodes based on the QoS metric requirements stored in the xml tree *rule*

Architecture

Four CMR Modules

➤ Route Establishment

- responsible for initiating, forwarding and receiving route requests
- accepts or rejects routes based on QoS requirements

➤ Resource Management

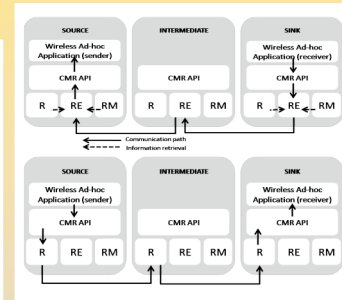
- manages different QoS resources in each node

➤ CMR API

- serves as bridge between user/application and the other four modules

➤ Routing

- responsible for the actual routing of data packets between nodes



The four CMR modules. The top figure shows the route establishment process while the bottom figure shows the routing process.

```
<rule id="1">
  <quantity>1</quantity>
  <parenthesis operator="AND">
    <requirement function="MIN">
      <name>delay</name>
      <value></value>
    </requirement>
    <requirement comparison-operator="<">
      <name>avg-bandwidth-used</name>
      <value>0.2</value>
    </requirement>
  </parenthesis>
</rule>
```

This XML QoS document specifies that the user requires the minimum delay route with an average bandwidth less than 20%.

Examples

```
<rule id="1">
  <quantity>1</quantity>
  <parenthesis operator="AND">
    <requirement function="MIN">
      <name>delay</name>
      <value></value>
    </requirement>
    <requirement comparison-operator="<">
      <name>avg-bandwidth-used</name>
      <value>2</value>
    </requirement>
  </parenthesis>
</rule>
```

CMR-ACOR – selects the route with the minimum delay and whose bandwidth is smaller than 2%.

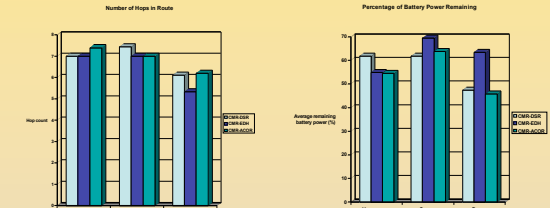
```
<rule id="2">
  <quantity>1</quantity>
  <parenthesis operator="AND">
    <requirement function="MIN">
      <name>hopcount</name>
      <value></value>
    </requirement>
  </parenthesis>
</rule>
```

CMR-DSR – Selects the route with the shortest hop count.

```
<rule id="1">
  <quantity>1</quantity>
  <parenthesis operator="AND">
    <requirement comparison-operator=">">
      <name>avg-battery-left</name>
      <value>50</value>
    </requirement>
    <requirement comparison-operator="<">
      <name>delay</name>
      <value>0.1</value>
    </requirement>
    <requirement comparison-operator="<">
      <name>hopcount</name>
      <value>10</value>
    </requirement>
  </parenthesis>
</rule>
```

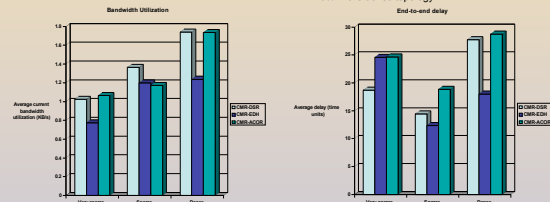
CMR-EDH – a novel protocol implemented in CMR which selects any route with an avg-battery-left of 50%, delay less than 0.1 seconds and a hop count of less than 10.

Experimental Results



Overall, the hop count of the routes returned by the three 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 hop count than the other topologies. 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 that are added to the route cache.

The CMR-DSR and CMR-ACOR protocols perform similar to each other in terms of remaining battery power. This is because these two protocols ignore energy requirements. The results of CMR-EDH are more interesting. Since the CMR-EDH protocol stresses the amount of battery power required by the routes, the CMR-EDH protocol has an increasing advantage over the other protocols in this metric with 6% in the sparse topology and 16% in the dense topology.



The average current bandwidth utilization increases with the connectivity of the topology. The bandwidth component of the CMR-ACOR rule is very high. Therefore, only 30% of the routes were rejected due to an average current bandwidth utilization greater than 2. These results show the importance of fine tuning the QoS requirements. A constraint that is too high may result in a poorer quality of routes.

The results show a random trend in delay. CMR-EDH and ACOR specify a minimum delay constraint. Therefore, any routes added to the route cache have this minimum delay requirement and routes which do not have this requirement are rejected. These rejected routes may be accepted by other protocols.

Future Work

- Developing resource management and admission control protocols to be implemented using the CMR toolkit. This includes traffic and flow specification in routing.
- Implementing cross-layer customization techniques (i.e., going beyond routing).
- Adding additional QoS metrics and using the CMR toolkit to experiment with existing and novel routing algorithms.

References

N. C. Hutchinson and L. L. Peterson. The x-kernel: An architecture for implementing network protocols. IEEE Transactions on Software Engineering, 17(1):64–76, January 1991.
 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.
 D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In IEEE OPENARCH, April 1998.