# Adapting to Route-demand and Mobility (ARM) in Ad hoc Network Routing*

Sungjoon Ahn and A. Udaya Shankar
(sjahn@cs.umd.edu, shankar@cs.umd.edu)
Computer Science Department
University of Maryland
College Park, MD 20742

February 14, 2001

**Abstract**

We present ARM (Adapting to Route-demand and Mobility), a control mechanism that allows any proactive routing protocol to dynamically adapt in a totally distributed manner to changes in node mobility and workload route-demands. Each node independently maintains a *mobility metric* indicating how fast its neighborhood is currently changing, and a *route-demand metric* indicating which destinations are currently involved in data forwarding. Control functions use these metrics to dynamically adjust the period and the content of routing updates. We apply ARM to the DSDV protocol, coming up with ARM-DSDV. For various mobility and workload scenarios, ARM-DSDV typically achieves the same data delivery as DSDV with update period optimized for the scenario, while saving up to 60% in routing cost. Lower cost gives data traffic more available bandwidth.

*This work is supported partially by DARPA contract "Information Dynamics and Agent Infrastructure" to the University of Maryland, College Park. It should not be interpreted as representing the opinions or views of DARPA or the U.S. Government.

# 1 Introduction

Ad hoc networks, also called MANETs (Mobile Ad hoc NETworks), are wireless data networks that do not require any communication infrastructure, unlike cellular networks and access point based wireless LANs. Ad hoc networks are suited for combat situations, search and rescue operations, and instant conferencing in infrastructure-absent geographic areas.

Ad hoc networks have characteristics that routing protocols for conventional networks need not deal with, among them dynamic topology, low bandwidth, short host battery life, and unreliable links. New routing protocols are needed for ad hoc networks and a number have been proposed, for example [1, 2, 3, 4, 5]. These routing protocols are usually classified as *proactive* or *reactive*. Proactive routing protocols [1, 2] periodically exchange routing updates to continuously maintain routes between all mobile host pairs, as in conventional wire routing protocols. Reactive protocols [3, 4, 5] send routing updates only when the data traffic demands routes and these updates are only for those routes.

However, in general no single routing protocol performs well over a wide range of mobility and route-demand patterns. In fact, few of the proposed routing algorithms adapt their behavior to mobility and route-demand patterns. Rather, most ad hoc routing protocols are optimized for certain operating conditions.

This paper presents ARM (Adapting to Route-demand and Mobility), a control mechanism that allows any proactive routing protocol to adapt in a totally distributed manner to changes in node mobility and changes in data traffic demand for routes. Each node independently maintains two metrics. One is *mobility metric* indicating how fast its neighborhood is currently changing, thereby reflecting the current rate of mobility. The other is *route-demand metric* indicating which destinations are currently involved in data forwarding, thereby reflecting the current demand for routes. These metrics are used by two control functions, called *update-period control function* and update-content control function, to dynamically adjust the period and the content of routing updates.

The ARM approach can be readily applied to any proactive routing protocol. In this paper, we apply it to DSDV (Destination Sequence Distance Vector) protocol [1], coming up with ARM-DSDV, and evaluate its performance relative to DSDV for several simple control rules using simulations. The simulator has a physical layer modeled by transmission range and bandwidth, a link layer based upon IEEE 802.11 (including CSMA/CA and RTS/CTS), and a workload layer of end-to-end connections. The performance metrics are data delivery ratio (fraction of data packets delivered to destinations) and routing cost (number of routing update octets transmitted). For various mobility and workload scenarios, ARM-DSDV typically achieves the same data delivery ratio as scenario-optimized DSDV while saving up to 60% in routing cost. (Scenario-optimized DSDV is DSDV with its update period optimized for the particular mobility and workload scenario.) Naturally, ARM-DSDV achieves higher data delivery ratio than non-optimized DSDV.

This paper is organized as follows. Section 2 addresses related work. Section 3 explains the ARM mechanism. Section 4 details the ARM-DSDV protocol. Sections 5, 6, and 7 describe respectively the performance evaluation model, the performance metrics, and the simulation results. Section 8 concludes.

## 2 Related Work

Proactive routing protocols [1, 2] periodically exchange routing updates to continuously maintain routes between all mobile host pairs, as in conventional wire routing protocols. Their advantage is that a data packet can be sent out immediately without any routing delay if a route exists. On the other hand, a proactive protocol wastes a large portion of available bandwidth when most of the routes it maintains are not used. Some optimizations are suggested to mitigate the routing overhead in [1]. One is *incremental dump* where each node advertises only the difference from the last update, reducing update message traffic. *Delayed update* defers broadcasting route update messages in hopes of better routes arriving shortly. Despite these optimizations, the routing update traffic of proactive routing is rather large. This is the main reason why proactive protocols perform inefficiently in certain network conditions [6], especially those characterized by skewed workload.

Reactive protocols [3, 4, 5] aim to eliminate the excessive overhead of proactive protocols by sending routing packets only when the data traffic demands routes. One disadvantage is the unavoidable initial delay in forwarding the first packet of a connection. Also, even though the goal is to reduce routing traffic, reactive protocols can suffer from high routing traffic overhead because they flood the network when discovering a new route [7]. The effect of flooding can be disastrous when the network is large and demand for new routes is high. Modifications can be introduced to alleviate the cost of flooding. For instance, geographic location of each node is exploited in LAR (Location Aided Routing) [8] to limit flooding area. However this assumes that all nodes are equipped with special devices like GPS. Another approach [9] makes use of prior routing histories to localize route request queries.

Hybrid protocols try to combine the strengths of proactive and reactive protocols. ZRP (Zone Routing Protocol) [7] is a hierarchical routing protocol that combines proactive and reactive protocols. A network is divided into zones inside which routing is done proactively. Interzone routing is performed reactively and only among zone leaders, which are elected nodes responsible for providing interzone routes to their zone members. By allowing proactive routing only within a zone, bandwidth is not wasted in advertising route entries of other zones. Flooding is limited by their route query control schemes. The zone radius is a configurable parameter. With zone radius of one, ZRP becomes purely reactive. With infinite zone radius, it becomes purely proactive. Through simulation the authors show that the optimal zone radius value depends on the call-to-mobility ratio (the ratio of number of calls or connections to node speeds). A high ratio (i.e., large number of connections or low mobility) favors large radius or more proactivity, while a low ratio favors small radius or more reactivity. ZRP assumes that the network condition (call-to-mobility ratio) is relatively static and known a priori for setting the optimal zone radius. Dynamically changing the zone radius involves reconfiguring zones and electing zone leaders, which can be pretty expensive and may require data forwarding to be halted during the transition.

In a survey paper of ad hoc routing protocols [10], the authors suggest switching between different protocols according to current network conditions. But no detailed work has been reported about intelligently switching between routing protocols based on dynamic network conditions. The cost of switching would be high in general because it has to occur globally throughout the network.

To summarize, few routing protocols adjust their operational parameters dynamically during their execution. ARM, proposed in this paper, allows any proactive routing protocol to adapt dynamically to changes in mobility and route-demand patterns. Furthermore it is completely decentralized. Adaptations do not require network-wide synchronization.

Each node adapts independently based on local observation and decision, spending a small amount of additional overhead.

# 3 ARM Mechanism

Recall that ARM has two controls. The update-period control maintains the mobility metric and dynamically adjusts the routing update period. The update-content control maintains the route-demand metric and dynamically adjusts the content of routing updates.

ARM can be applied to any routing protocol in which each node *periodically* sends routing updates. Specifically, in each period a node sends an routing update message constructed from its current routing information and builds new routing information based upon routing update messages received. At the end of the period, the new routing information becomes the current routing information, and the cycle repeats. Data packets are forwarded based on the current routing information.

Most proactive routing protocols behave this way. Also, these protocols broadcast routing update messages and hence can lose them. ARM is robust to such loss.

## 3.1 Update-period control

The intuition behind update-period control is that a node should update more frequently in high mobility, so as to accurately reflect the current network topology, and less frequently in low mobility, when frequent updates do not bring additional accuracy but consume bandwidth.

A node measures mobility by measuring the rate of change in its neighborhood, i. e., the set of nodes within radio range. The node maintains two neighbor tables: *current_neighbor_table*, based on updates received by the start of the current update period, and *new_neighbor_table*, based on updates received during the current update period. Both tables have entries of the type {*neighbor_id*, *t_expiration*}. Member *t_expiration* represents the time when the next update from the neighbor is expected to arrive. An entry is regarded as expired if *t_expiration* is less than current clock. This expiry time information is required because each nodes can have different update periods and update information from a node with a longer period should survive longer than that from a node with a shorter period.

To maintain these neighbor tables, routing update messages need to contain the sender's update period, in addition to the sender's id. Additional types of packets can be used to build the tables, depending on the particular underlying protocols, for example, sender information in hello messages and predecessor information in data packets.

Figure 1 describes the processing involved in the update-period control. At the start of a new period, an "instantaneous" mobility metric is obtained by dividing the number of nodes that became neighbors or stopped being neighbors by the previous update period. Smoothing the instantaneous metric over time interval *TW_SMOOTH* yields the mobility metric. The update-period control function maps the mobility metric to the new update period. The new neighbor table is merged into the current neighbor table and expired entries are deleted.

There are several points to note. In computing the mobility metric, an alternative to smoothing is to use weighted time averaging, as in TCP's RTT estimation [11]. Regarding update-period control function, it turns out that even simple update-period control functions, such as shown in Figure 2, can outperform non-adaptive update periods. The expiry times of entries in the neighbor tables is reasonably well-approximated by the sum of the

```
At start of an update period:
  // Mobility metric computation
   Remove expired entries from new_neighbor_table ;
   Let nbd_change be the number of nodes e such that
      e is expired in current_neighbor_table and not in new_neighbor_table
      or e is in new_neighbor_table and not in current_neighbor_table ;
   Let inst_nbd_rate_change be nbd_change divided by current update period ;
   Let mobility_metric be the inst_nbd_rate_change smoothed over TW_SMOOTH ;

  // New update period computation
   New update period is update-period control function value at mobility_metric ;

  // Neighbor tables update
   Remove expired entries from current_neighbor_table ;
   For each entry n in new_neighbor_table
      if match m is found in current_neighbor_table then
         assign n.t_expiration to m.t_expiration ;
      otherwise insert n into current_neighbor_table ;
   Empty new_neighbor_table ;

At reception of routing update message:
  Let expiration_time be sender's update period plus clock plus slack ;
  If sender's id has match in new_neighbor_table then
        update t_expiration in the table ;
     otherwise insert the sender's id and expiration time into the table ;
```

Figure 1: ARM update-period control

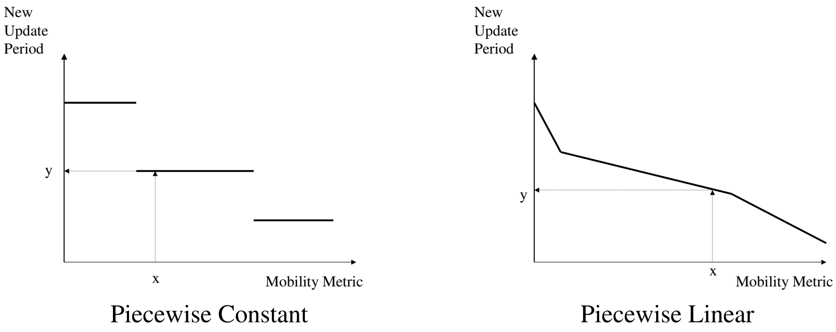


Figure 2: Simple update-period control functions

current time and the sender's update period; some slack can be added for a more conservative estimate. When ARM is instantiated in a routing protocol, the routing table entries are subject to the same expiry time constraints as neighbor table entries.

The update-period control computation is completely local to the node. The overhead consists of the extra field in routing update messages and the computation/storage of mobility metric, new update period, and neighbor tables.

### 3.2 Update-content control

The intuition behind update-content control is that a node does not have to send a piece of routing information in every routing update if that information is not being used by other nodes. In ARM, each node keeps track of which destinations it has forwarded packets to recently, specifically within a time window *TW_RECENT*. When constructing a routing update message, routing information for other destinations is included only if it satisfies a *update-content control function*.

```
At forwarding of a data packet:
   // Route-demand metric computation
   If the data packet's destination is in the route_demand_table then
      update the entry's t_last_forwarded to clock value ;
    otherwise create an entry in the route_demand_table with
      packet destination id and clock value ;

At construction of a routing update message:
  For every entry in the routing table
    // Update-content control function
    if there is a match m in route_demand_table with age less than TW_RECENT
        or if the entry passes the filter function ;
      then the entry is included in the message ;
```

Figure 3: ARM update-content control

To implement this, the node maintains a *route_demand_table* whose entries have the form $\{$*destination_id*, *t_last_forwarded*$\}$, indicating when the last forwarding occurred to the destination. The recent route demands recorded in the table represent *route-demand metric*. Figure 3 describes the update-content control operation.

The filter function can be implemented in a several ways, for example, in every $K$th routing message, randomly with some probability, etc. We point out that even if the update-content control decides not to include any routing information, the node still transmits a routing update (depending on update-period control) in order to advertise its own presence and help the update-period control of its neighbors.

The above description of update-content control assumes an underlying routing protocol which uses destination information, such as distance vector. However update-content control can be applied to other types of proactive protocols. For example, in a link state protocol the update-content control can decide whether or not to disseminate an outgoing link cost change based on how recently the link was used or whether the link is the current next hop to a "recently used" destination.

As in update-period control, update-content control is carried out locally and introduces minimal overhead, namely the computation/storage for *route_demand_table* and the filter function.

In most proactive protocols, every available entry (in routing table or next hop table) is included when update messages are constructed. But not sending all the information each time would not seriously affect the operation of the network. Thus a network in which only some nodes have ARM would still work.

# 4 DSDV and ARM-DSDV Protocol

Details of DSDV is presented first. How ARM is applied to DSDV producing ARM-DSDV is presented next.

## 4.1 DSDV protocol

DSDV uses improved version of Bellman-Ford algorithm. One of its main advantages over traditional distance vector protocol is the loop of freedom through the use of destination sequence numbers [1]. Original DSDV uses both periodic and triggered updates. We only focus on periodic updates in this paper.

```
CONSTANTS : UPDATE_PERIOD
VARIABLES : dest_seq_no, t_update
            current_routing_table, new_routing_table
              each routing table entry has members
                { dest_id, next_hop_id, num_hops, dest_seq_no }
```

Figure 4: Variables of DSDV protocol

```
MESSAGE FIELDS : sender_id, sender_dest_seq_no, routing_update_vector
                   each entry in the vector has members
                     { dest_id, num_hops, dest_seq_no }
```

Figure 5: Routing update message of DSDV protocol

In DSDV, nodes maintain variables as shown in Figure 4. Constant *UPDATE_PERIOD* denotes the globally fixed update period for all nodes. Variable *dest_seq_no* and *t_update* denotes destination sequence number and start of the next update period of the node, respectively. Routing entries to all possible destinations are maintained in *current_routing_table*, for data packet forwarding. *New_routing_table* stores routing entries based on the information in routing update messages received in the current period. Entries in routing tables have members denoting id of the destination, id of the next hop, number of hops to the destination, and sequence number originating from the destination, in the order they appear in the figure.

As shown in Figure 5, a routing update message contains id of the sender, current destination sequence number of the sender, and the routing update vector whose entries are copied from corresponding entries of the sender's *current_routing_table*. Table member *next_hop_id* is not included in vector entries because *sender_id* becomes *next_hop_id* for vector entries in the message when they are stored in routing tables.

```
EVENT : expiration of t_update
 dest_seq_no ++
 update current_routing_table using new_routing_table
 empty new_routing_table
 build routing update message from current_routing_table and send it
 t_update += UPDATE_PERIOD

EVENT : reception of a routing update message
 for x ranging over sender and vector entries
   if ( new_routing_table has no entry for x )
      // if x is sender, x.dest_id is sender_id of the message
   then
     insert x into new_routing_table
   else // let m be the matched entry in new_routing_table
   if ( (x.dest_seq_no > m.dest_seq_no) or
        ((x.dest_seq_no == m.dest_seq_no) and
         (x.num_hops + 1 < m.num_hops)        )
     // if x is sender, x.dest_seq_no is sender_dest_seq_no of the message
     // if x is sender, x.num_hops is zero
   then
       m.next_hop_id := sender_id of the message
       update other members of m using members of x
 endfor
```

Figure 6: Routing events of DSDV protocol

A node handles two routing events, expiration of *t_update* and reception of a routing update message as shown in Figure 6. On expiration of *t_update*, *dest_seq_no* of the node is incremented to favor the to-be-broadcast routing update message of the current period. Contents of *new_routing_table* is copied into *current_routing_table* for packet forwarding in the current period. A routing update message is constructed from entries in *current_routing_table*, and then broadcast to the neighbors. Finally, *t_update* is increased by *UPDATE_PERIOD* for the next update.

On reception of a routing update message, if entry to the sender does not exist in *new_routing_table*, the node creates an entry to the sender and stores *sender_dest_seq_no* along with it. Otherwise, only the destination sequence number is updated. Then the node processes the vector entries. A vector entry is inserted into *new_routing_table* if there is no entry with the same *dest_id*. Otherwise, entry with the same *dest_id* is updated in the table only when the vector entry is favored, i.e., it either has the larger destination sequence number or has smaller number of hops if sequence numbers are identical. As sequence numbers are monotonically increasing over time, this guarantees new routes are selected over old routes.

## 4.2 ARM-DSDV protocol

Figure 7 shows variables used in ARM-DSDV. Now *update_period* is varied to control update periods. Two neighbor tables are for update-period control and *route_demand_table* is for update-content control as described in section 3. Each entry in routing tables now has new member *t_expiration* to denote its expected expiration time.

```
VARIABLES : dest_seq_no, t_update
            update_period                              // new for ARM-DSDV
            current_neighbor_table, new_neighbor_table  // new for ARM-DSDV
              each entry  has members
                { neighbor_id, t_expiration }
            current_routing_table, new_routing_table
              each entry  has members
                { dest_id, next_hop_id, num_hops, dest_seq_no, t_expiration }
                  // t_expiration is new for ARM-DSDV
            route_demand_table                         // new for ARM-DSDV
              each entry  has members
                { dest_id, t_forwarded_last }
```

Figure 7: Variables in ARM-DSDV protocol

```
MESSAGE FIELDS : sender_id, sender_dest_seq_no
                 sender_update_period           // new for ARM-DSDV
                 routing_update_vector
                   each entry in the vector has members
                     { dest_id, num_hops, dest_seq_no }
```

Figure 8: Routing update message of ARM-DSDV protocol

As shown in Figure 8, the only additional field in ARM-DSDV's routing update messages is the sender's update period that is used to calculate *t_expiration* of table entries.

On expiration of *t_update*, additions to DSDV are computation of next update period and how *current_neighbor_table* is updated as shown in Figure 9. Both issues were addressed previously. *Current_routing_table* is updated using *new_routing_table* as follows. First, expired entries in both tables are thrown away. Second, for each entry $n$ in *new_routing_table*, insert $n$ into *current_neighbor_table* if no match is found. Otherwise, favored route of the two is selected and updated in *current_neighbor_table* if necessary. Entries in *current_neighbor_table* is used for forwarding in the current period regardless of their *t_expiration*. Routes updated by neighbors on shorter periods may be used beyond their *t_expiration*. An alternative

```
EVENT : expiration of t_update
 dest_seq_no ++
 update_period := value computed as in Figure 1 (section 3.1)
 update current_routing_table using new_routing_table
 update current_neighbor_table using new_neighbor_table as in Figure 1 (section 3.1)
 empty new_neighbor_table and new_routing_table
 build routing update message from current_routing_table and send it
 t_update += update_period

EVENT : reception of a routing update message
 tmp_t_expiration := clock + sender_update_period in the message + slack
 insert sender's id into new_neighbor_table as in Figure 1 (section 3.1)
 for x ranging over sender and vector entries
   if ( new_routing_table has no entry for x )
     // if x is sender, x.dest_id is sender_id of the message
     // entry with same id but expired t_expiration
     //  is thrown away and processed here
     insert x into new_routing_table
   else // let m be the matched entry in new_routing_table
   if ( (x.dest_seq_no > m.dest_seq_no) or
        ((x.dest_seq_no == m.dest_seq_no) and
         (x.num_hops + 1 < m.num_hops)        )
     // if x is sender, x.dest_seq_no is sender_dest_seq_no of the message
     // if x is sender, x.num_hops is zero
     then
       m.new next_hop_id := sender_id of the message
       update other members in m using members in x
 endfor
```

Figure 9: ARM-DSDV update-period control

would be stop using those entries for forwarding. But there is a possibility that those entries are being updated in *new_routing_table*. Another alternative would be to keep a single routing table and update entries in place whenever routing update messages are received. This prevents expired entries from being used for forwarding, but then the semantics of DSDV is somewhat compromised.

On reception of a routing update message, local variable *tmp_t_expiration* is calculated to be stored as *t_expiration* in neighbor and routing tables. Entries are updated in *new_neighbor_table* as explained in section 3.1. *New_routing_table* is updated like DSDV except that each expired entry in the table is thrown away and never considered as a match.

```
EVENT : forwarding of a data packet
 if ( dest_id of data packet exists in route_demand_table )
 then
  update t_forwarded_last to clock value
 else
  insert the dest_id with t_forwarded_last

EVENT : expiration of t_update
 // when building routing update messages
 for e ranging over current_routing_table entries
   if ( ( e has match m in route_demand_table and less than
          TW_RECENT passed since m.t_last_forwarded         )
        or ( filter function allows copy )                    )
   then
     copy e into the routing update message
 endfor
```

Figure 10: ARM-DSDV update-content control

Figure 10 illustrates update-content control of ARM-DSDV. For ARM-DSDV, destinations are recorded in *route_demand_table* to be compared with routing table entries when constructing routing update messages. Other than this, no additional processing specific to DSDV are required for update-content control.

## 5 Performance Evaluation Model

To compare the performance of DSDV and ARM-DSDV, we have a simulator (written in CSIM [12]) with the layered structure as shown in Figure 11. DSDV and ARM-DSDV share the common lower layers as well as the upper workload layer. The figure also summarizes parameters of each layer.

| |
|---|
| **End-to-end workload** :<br>*CONNECTION_START_TIME*, *CONNECTION_DURATION*<br>*DPKT_LENGTH*, *DPKT_RATE* |
| **Routing layer** :<br>DSDV : *UPDATE_PERIOD*<br>ARM-DSDV : *TW_SMOOTH* (for update-period control)<br>*TW_RECENT* (for update-content control) |
| **Link layer** :<br>*RTS_LEN*, *CTS_LEN*, *ACK_LEN*<br>*PKT_HEADER*, *SLOT_TIME*, *QUEUE_LEN* |
| **Physical layer** :<br>*TX_RANGE*, *BANDWIDTH*, *TA_TIME* |
| **Mobility** |

Figure 11: Layer model and parameters

### 5.1 Mobility

The mobility of a node is modeled as a series of pauses and motions. A pause is defined by a time duration during which the node does not move. A motion is defined by direction, speed, and time duration. A motion can be followed by a pause or another motion. By juxtaposing two motions, variable speed can be modeled.

### 5.2 Physical layer

The physical layer is characterized by three parameters: *TX_RANGE*, *BANDWIDTH*, *TA_TIME*. Two nodes are able to transmit packets to each other only when they are in transmission range, specified by parameter *TX_RANGE*. The transmission time is determined by dividing the packet length by *BANDWIDTH*. Parameter *TA_TIME* specifies the time length required to turn the antenna from receiving mode to sending mode and vice versa.

Transmission delay is treated as zero because of the relatively short transmission distances. For a successful transmission of a packet from one node to another, the two nodes must stay within transmission range during the entire packet transmission time, and the receiver's antenna must be in receiving mode and should not receive (part or whole of) another packet during the same period of time.

We do not see any substantial reason to sacrifice simplicity and fast simulation by modeling physical layer details like multi-path fading, attenuation of signal strength, and noise. Also the model does not depend on any particular spread spectrum implementation (FHSS, DSSS, etc).

### 5.3 Link layer

The link layer model replicates IEEE 802.11 standard [13] MAC layer operations as described below.

The MAC layer uses CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) to share the channel among all nodes. In CSMA/CA, a node starts a transmission only if the channel is free and this is determined by both physical and virtual carrier sensing. Physical carrier sensing is done by analyzing signal strength at the air interface. Virtual carrier sensing is done by looking up the NAV (Network Allocation Vector), a data structure indicating if the channel is currently free of ongoing RTS/CTS handshakes. If the node finds the channel busy, the node performs binary back-off, waiting for its back-off counter to reach zero. The counter is initialized to a random integer that doubles on average at every consecutive failed transmission attempt. The counter decrements when the channel appears free for a specified period of time called *slot time* (typically set to 20 $\mu$ seconds).

The RTS/CTS handshake is as follows. A node with a data packet to send first broadcasts an RTS frame, advertising its intention to send a data frame. The intended destination station responds with CTS, advertising that it is ready to receive. The sender transmits the data packet. On valid reception, the destination sends back an ACK. On receiving or overhearing a RTS, CTS or data frame, a node sets its NAV busy for the time duration inferred from data packet length in the frame header. Since RTS/CTS frames are much shorter than data packets, recovery from a collision is much cheaper. RTS/CTS handshakes are not mandatory. They are not used for broadcast packets.

In the model, fragmentation and reassembly are avoided by having packets be smaller than fragmentation threshold. Every node has a send queue that holds both routing and data packets. The maximum size of the queue is given by *QUEUE_LEN* in number of packets. Routing packets have precedence over data packets. FIFO is used as the queuing discipline within data packets and LIFO is used within routing packets. When a routing packet arrives from the upper layer to a full send queue, the data packet most recently enqueued is dropped; if the queue has only routing packets, the oldest routing packet gets dropped.

*PKT_HEADER* octets are added to upper layer packets in order to accommodate link layer protocol header information. Parameter *SLOT_TIME* specifies the unit in which the back-off counter decrements.

Unlike the physical layer, it is important to have a detailed model of the link layer. Bandwidth consumption by link layer control frames and retransmissions have significant impact on instantaneous residual bandwidth for upper layers.

### 5.4 Routing layer

For both DSDV and ARM-DSDV, all the operations described previously are modeled into the simulator. We do not model the incremental dump and delayed update features of the original DSDV.

### 5.5 End-to-end workload

The end-to-end workload is modeled in terms of connections and packets. A connection is defined by source node, destination node, start time (*CONNECTION_START_TIME*), duration (*CONNECTION_DURATION*), data packet length (*DPKT_LENGTH*), and data packet rate (*DPKT_RATE*). The packet interarrival time is constant.

# 6 Performance Metrics

A simulation scenario is characterized by mobility pattern, end-to-end workload, average node speed, simulation duration, and routing protocols—ARM-DSDV with specified control functions and DSDV with specific update period. For a simulation scenario, we have the following performance metrics:

- **Delivery ratio**: number of data packets that arrived at their destinations divided by the number of data packets sent out from source nodes.
- **Routing cost**: total number of octets in all the routing update messages sent.
- **Relative cost**: routing cost of ARM-DSDV divided by routing cost of DSDV.

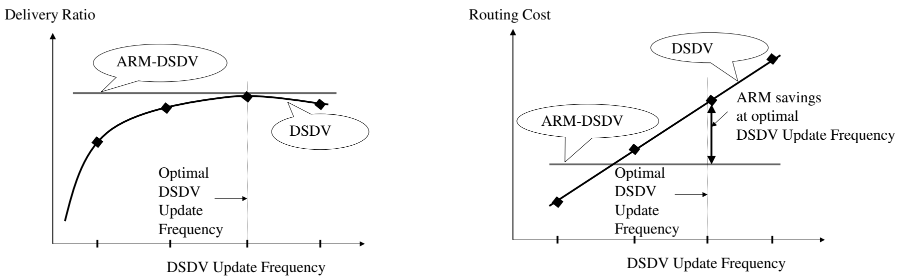


Figure 12: Generic result

For a mobility pattern, workload, node speed, and simulation time, we expect ARM-DSDV and DSDV performance to be as illustrated in Figure 12. The solid lines connects the results for DSDV over different update frequencies. The shaded line indicates the result for ARM-DSDV. The delivery ratio of DSDV increases rapidly as the update frequency approaches the optimum. After that point, it either plateaus or decreases due to the contention introduced by excessively frequent updates. We expect the delivery ratio of ARM-DSDV to be similar to what DSDV achieves at optimal update frequency. The cost is compared between the cost at an update frequency of DSDV and the cost of ARM-DSDV. The routing cost of DSDV grows with update frequency. We expect the routing cost of ARM-DSDV to be much lower. Even if the routing cost is similar, ARM-DSDV still has an advantage over DSDV in that it does not require to manually configure optimal update frequency.

# 7 Simulation Results

We present simulation results for two mobility patterns. Figure 13 lists common parameter values for lower layers that we use for all our simulations.

| |
|---|
| **Physical layer** : |
| $TX\_RANGE$ = 100 m, $BANDWIDTH$ = 2 Mbps |
| $TA\_TIME$ = 10 $\mu$ sec |
| **Link layer** : |
| $RTS\_LEN$ = 40 octets, $CTS\_LEN$ = 40 octets |
| $ACK\_LEN$ = 34 octets, $PKT\_HEADER$ = 58 octets |
| $SLOT\_TIME$ = 20 $\mu$ sec, $QUEUE\_LEN$ = 100 packets |

Figure 13: Common parameter values

## 7.1 Mobility pattern 1

Mobility pattern 1 models wireless nodes on vehicles crossing each other at a highway interchange as shown in Figure 14. There are four groups of vehicles in a 2km $\times$ 2km geographical area. Each group has two rows of five vehicles. The rows are separated by 50m. Groups moving in opposite direction on adjacent lanes are separated by 20m. All groups have same group speed but individual vehicle speed varies within $\pm 20\%$ of the group speed. Adjacent nodes in each group start with 50m separation but the separation varies due to varying individual speeds. We have four different group speeds of 5 m/sec, 8 m/sec, 9 m/sec, and 10 m/sec (or 18 km/hr, 28.8km/hr, 32.4km/hr, and 36km/hr, respectively).

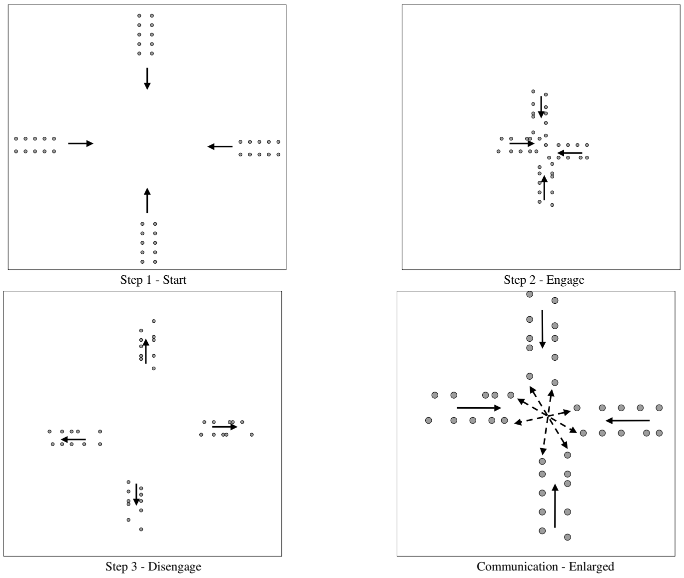


Figure 14: Mobility pattern 1

As the figure shows, each group starts from the fringe and moves towards the middle. Connections are opened at about the time all nodes obtain paths to each other. Connections are closed before the vehicles pass by each other completely and all paths break down. The

**Routing layer** :

DSDV :

$UPDATE\_PERIOD$ = 2, 1, 0.5, 0.2, 0.1, 0.05, 0.02 sec

ARM-DSDV :

$TW\_SMOOTH$ = 1 sec, $TW\_RECENT$ = 3 sec

$$\text{update-period control function} = \begin{cases} 0.50 & \text{if mobility metric is } 0 \\ 0.15 & \text{" " " is in } (0,1] \\ 0.12 & \text{" " " is in } (1,10] \\ 0.10 & \text{" " " is in } (10,150] \\ 0.05 & \text{" " " is in } (150,\infty) \end{cases}$$

filter function = skips one in every two advertisement opportunities

**End-to-end workload** :

$CONNECTION\_START\_TIME$ = when nodes start to have paths to each other

$CONNECTION\_DURATION$ = 5 sec

$DPKT\_LENGTH$ = 100 octets, $DPKT\_RATE$ = 1 packets/sec

**Values of (node speed, simulation time, number of runs)** :

( speed = 5 m/s, 220 sec, 10 runs )

( speed = 8 m/s, 170 sec, 10 runs )

( speed = 9 m/s, 150 sec, 10 runs )

( speed = 10 m/s, 130 sec, 10 runs )

Figure 15: Scenario parameters of mobility pattern 1

last picture of Figure 14 shows which vehicles become connected. Each row has exactly one node that transmits packets, resulting in a total of eight connections in the network.

Figure 15 gives scenario parameters. The simulation time depends on the speed of vehicles; the slower the speed, the longer the simulation time in order for vehicles to complete their trips. The update-period control function used in this paper is obtained empirically. It is subject of future research to develop a systematic way of constructing such functions.

Figure 16 presents delivery ratios achieved by DSDV and ARM-DSDV. Curves for DSDV clearly show that it performs well only around optimal update frequency. ARM-DSDV achieves good delivery ratios, specifically 99.5%, 99.3%, 96.2%, and 97.8% for increasing speeds. Best delivery ratios of DSDV are 100.0% at 10 updates/sec, 99.5% at 20 updates/sec, 97.9% at 20 updates/sec, and 99.5% at 20 updates/sec. Thus ARM-DSDV has practically the same delivery ratios. Apparently, ARM-DSDV finds appropriate values of update frequency.

Figure 17 illustrates the routing cost of both protocols. As expected, DSDV results in linear increase in the routing cost regardless of vehicles speed. One might wonder why the routing cost decreases as vehicles travel faster. As the vehicles travel the same distance, more speed means less simulation time, and hence less occurrences of updates. In addition, more speed gives more distance between vehicles resulting in sparser topology because of variable individual speed. Thus each routing update message carries less number of vector entries. These two factors offset the more frequent updates in high mobility.

The routing cost of ARM-DSDV is much less than that of DSDV at all speeds. Relative cost at optimal DSDV update frequency are 57.3%, 31.1%, 29.9%, and 30.0% for the different speeds. However, all the speeds do not show big differences between delivery ratios at the optimal update frequency and the next optimal (but less frequent) update frequency for DSDV. For these update frequencies, DSDV delivery ratios are 98.3%, 98.8%, 94.4% and 99.0% compared to 100.0%, 99.5%, 97.9%, and 99.5% obtained at optimal frequency. The new relative cost is 113.0%, 59.9%, 57.5% and 57.7%, respectively. Still, ARM-DSDV has

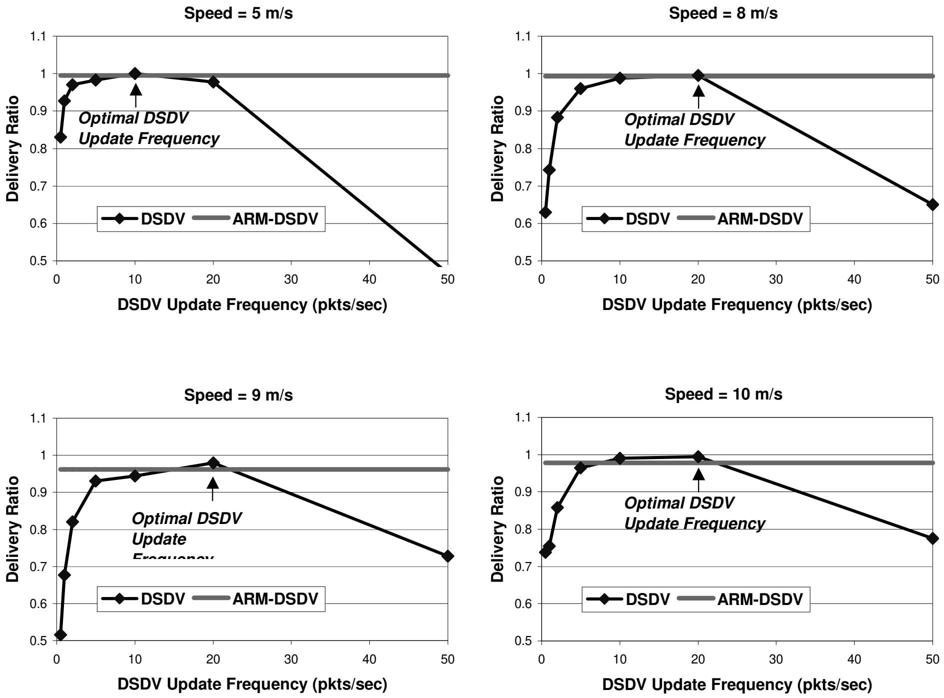


Figure 16: Delivery ratio of mobility pattern 1

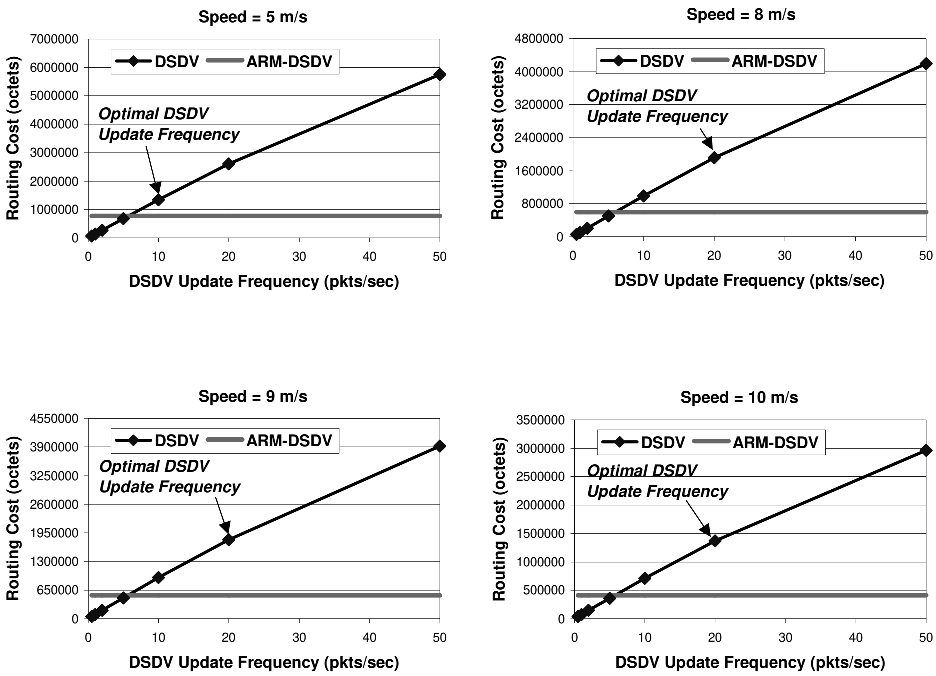


Figure 17: Routing cost of mobility pattern 1

far less routing cost in three of the four speeds. It spends about 10% more than DSDV when the speed is 5 m/sec.

## 7.2 Mobility pattern 2

Mobility pattern 2 models a search and rescue operation. Nodes are located relatively close in the beginning. After halting for individually different time periods (less than 5 seconds), all nodes repeat 5 seconds of moving and 5 seconds of pausing until the end of simulation. Speed is identical for all nodes. There are four different node speeds— 2 m/s, 5 m/s, 10 m/s, and 15 m/s (or 7.2 km/hr, 18 km/hr, 36 km/hr, and 54 km/hr, respectively). Figure 18 shows how nodes move over the time when the node speed is 10 m/s; at time 100 seconds, the nodes are spread over 1km $\times$ 1km. There are 40 nodes and each has a single connection to a randomly selected peer.

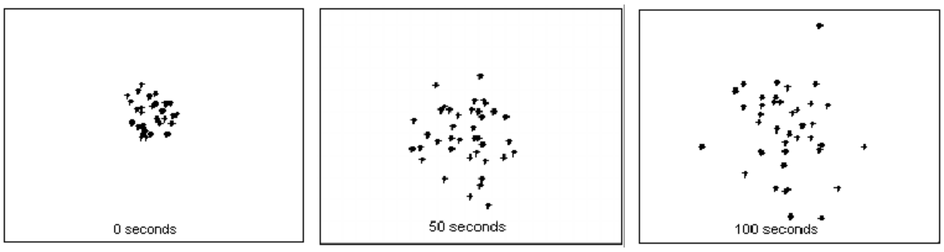


Figure 18: Mobility pattern 2

Figure 19 shows the scenario parameters.

**Routing layer** :
<u>DSDV</u> :
*UPDATE_PERIOD* = 2, 1, 0.5, 0.2, 0.1, 0.05 sec
<u>ARM-DSDV</u> :
*TW_SMOOTH* = 1 sec, *TW_RECENT* = 3 sec
update-period control function and filter function = same as in Figure 15
**End-to-end workload** :
*CONNECTION_START_TIME* = uniformly distributed along simulation time
*CONNECTION_DURATION* = 5 sec
*DPKT_LENGTH* = 100 octets, *DPKT_RATE* = 1 packets/sec
**Values of (node speed, simulation time, number of runs)** :
( speed = 2 m/s, 100 sec, 10 runs )
( speed = 5 m/s, 100 sec, 10 runs )
( speed = 10 m/s, 100 sec, 10 runs )
( speed = 15 m/s, 100 sec, 10 runs )

Figure 19: Scenario parameters of mobility pattern 2

Figure 20 shows delivery ratios of ARM-DSDV and DSDV. ARM-DSDV achieves 99.4%, 98.1%, 94.3%, and 77.7% while the best delivery ratios of DSDV are 99.9% at 5 update/sec, 99.4% at 10 updates/sec, 96.0% at 10 updates/sec, and 78.1% at 10 updates/sec. The ratios are very similar between ARM-DSDV and DSDV. One can also observe that, as the group speed increases, the overall delivery ratio is getting low regardless of the protocol. This is

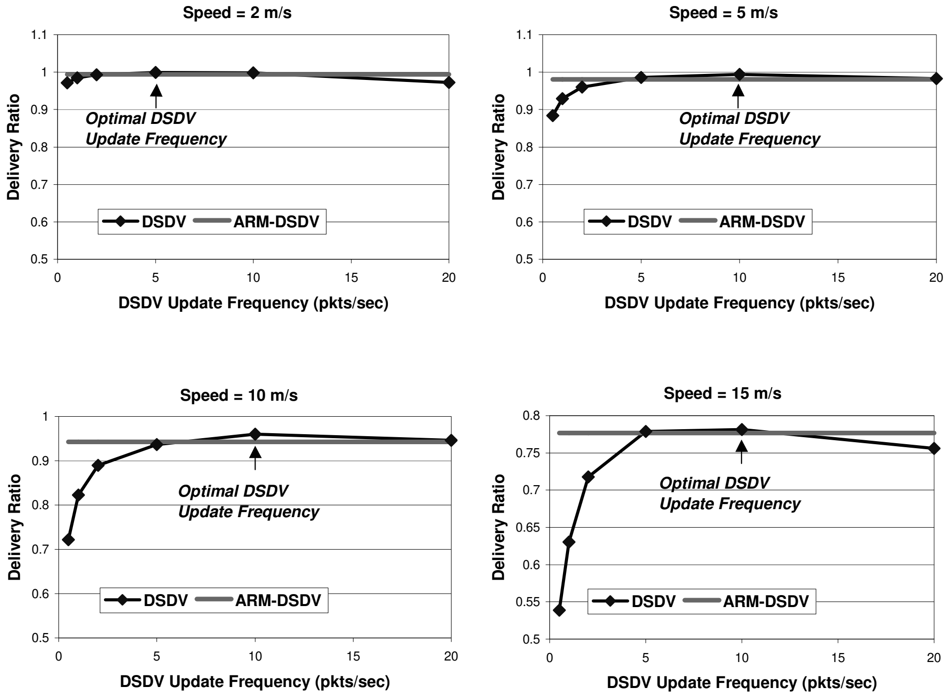


Figure 20: Delivery ratio of mobility pattern 2

because more dynamics makes it harder for a routing protocol to accurately maintain route information.

The routing cost is shown in Figure 21. The relative cost at optimal DSDV update frequency is 129.3%, 76.1%, 70.0%, and 68.1%, respectively. Except the case of speed 2 m/sec, ARM-DSDV saves from 23.9% to 31.9%. Compared to mobility pattern 1, savings are smaller. One of the reasons is that mobility pattern 1 has fewer connections of eight compared to forty connections in mobility pattern 2. Update-content control has more opportunities to cut down on vector entries in mobility pattern 1.

# 8 Conclusion

The ARM control mechanism presented here allows a proactive routing protocol to dynamically adjust the period and content of its routing updates in order to adapt to the mobility and route-demand pattern. Furthermore, ARM is completely decentralized, allowing each node to adapt independently, and its overhead is low.

We applied ARM to the DSDV protocol, coming up with ARM-DSDV. We showed that for various mobility and workload scenarios, ARM-DSDV typically achieves the same data delivery ratio as DSDV with update period optimized for the mobility and workload scenario, while saving up to 60% in routing cost. Lower cost gives data traffic more available bandwidth, a valuable resource in ad hoc networks. Naturally, ARM-DSDV achieves higher data delivery ratio than non-optimized DSDV.

Changes in neighborhood appear to be a good approximation to the actual extent of mobility. Keeping track of forwarded packets appears to be a useful yet inexpensive way of assessing the route-demand patterns.

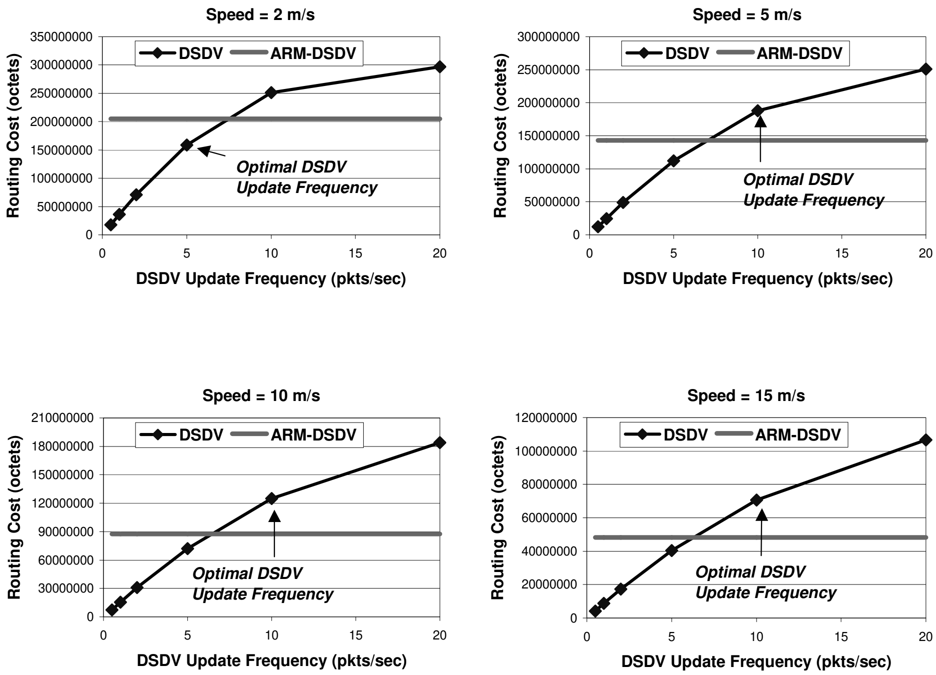


Figure 21: Routing cost of mobility pattern 2

Regarding future work, clearly ARM requires much more analysis. Alternative ways to obtain mobility and route-demand metrics need to be investigated and compared with current metrics. Sophisticated control functions may outperform the simple ones used in the paper.

Another area of future work is to make reactive protocols adapt to mobility pattern. This appears to be conceptually harder than ARMing proactive protocols. A proactive protocol can be made adaptive by slowing down its proactivity, reducing routing information being exchanged among nodes. But for a reactive protocol, one would need to add new mechanisms of information gathering.

# References

[1] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance vector routing for mobile computers. In *Proceedings of ACM SIGCOMM*, August 1994.

[2] S. Murphy and J .J .Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Networks and Applications Journal*, October 1996.

[3] C. Perkins and E. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, February 1999.

[4] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, Kluwer Academic, 1996.

[5] V. Park and M. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE INFOCOM*, April 1997.

[6] D.A. Maltz, J. Broch, and D. Johnson. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of ACM MOBICOM*, October 1998.

[7] Z. Haas and M. Pearlman. Performance of query control schemes for the zone routing protocol. In *Proceedings of ACM SIGCOMM*, August 1998.

[8] Y. Ko and N. Vaidya. Location-aided routing (LAR) in mobile ad hoc networks. In *Proceedings of ACM MOBICOM*, October 1998.

[9] R. Castaneda and S. Das. Query localization techniques for on-demand routing protocols in ad hoc networks. In *Proceedings of ACM MOBICOM*, August 1999.

[10] E. Royer and C-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications, 6(2):46–55*, 1999.

[11] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, August 1998.

[12] H. Schwetman. CSIM User's Guide. *Microelectronics and Computer Technology Corporation*, 1992.

[13] B. Crow, I. Widjaja, J. Kim, and P. Sakai. IEEE 802.11 wireless local area networks. *IEEE Communications Magazine*, September 1997.