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

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. Simulations for various mobility and workload scenarios show that ARM-DSDV typically achieves better data delivery, while keeping the routing cost at reasonable levels, when compared to DSDV with update period optimized for the scenario.

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 changes in mobility and route-demand patterns. For example, most proactive protocols use a fixed update frequency even if the mobility patterns are changing.

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 bet-

1092-1658/01 $17.00 © 2001 IEEE

ter data delivery ratio than scenario-optimized DSDV while keeping the routing cost at reasonable levels. 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*, *mobility_metric*}. 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 different 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. Member *mobility_metric* indicates the mobility metric of the neighbor, and is used in computing the update period as described below. To maintain these neighbor tables, routing update messages need to contain the sender's update period and the sender's mobility metric in addition to the sender's id.

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. An "aggregate" mobility metric is obtained by averaging the mobility metric of this node and the mobility metrics of the neighbors (available in new neighbor table). The update-period control function maps the neighborhood mobility metric to the new update period. The new neighbor table is merged into the current neighbor table and expired entries are deleted.

Data packets received or overheard by the node are also used to build the neighbor tables. Because data packets do not carry the sender's update period and sender's mobility metric, which are required for neighbor tables, the following is done: the sender's update period is set to the minimum update period of the update-period control function, and the sender's mobility metric is set to a negative value (to indicate that it should not be used in computating the aggregate mobility metric).

There are several points to note. In computing the mobility metric, an alternative to smoothing is to use weighted

```
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
 Let aggregate_mobility_metric be the average of mobility
  metrics of this node and neighbors (from neighbor table)
 New update period is update-period control function value
  at aggregate_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;
    assign n.mobility_metric to m.mobility_metric;
  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 and mobility_metric in the table;
 otherwise insert the sender's id, expiration time,
  and mobility metric into the table;

At reception/overhearing of data packet:
 Let expiration_time be minimum update period
  plus clock plus slack;
 Let mobility_metric be a negative value;
 If sender's id has match in new_neighbor_table then
  update t_expiration and mobility_metric in the table;
 otherwise insert the sender's id, expiration time,
  and mobility metric into the table;
```

Figure 1. ARM update-period control

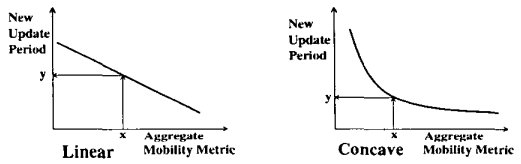

Figure 2. Simple update-period control functions

time averaging, as in TCP's RTT estimation [11]. The expiry times of entries in the neighbor tables is reasonably well-approximated by the sum of the 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. Regarding update-period control function, it turns out that even simple update-period control functions, such as shown in Figure 2, outperform non-adaptive update periods.

In obtaining the update-period control function, we rely on simple and intuitive procedures instead of a thorough analysis. According to simulations we have done with different control functions, the range of the new update periods is more important than the shape of the function. The range of aggregate mobility metric is obtained empirically and the maximum value between 20 and 50 is reasonable.

The maximum value for the new update period is roughly determined by considering the maximum end-to-end delay

```
At forwarding of a data packet:
 // Route-demand metric computation
 If the data packet's destination is in
  the current_routing_table then
   update the entry's t_last_forwarded to clock value;
  otherwise drop the packet;

At construction of a routing update message:
 For every entry in the current_routing_table
  // Update-content control function
  if the last forwarding has 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

within which routing information is desired to be propagated from one end of the network to another. For example, 10 hops of network diameter and 5 seconds of desired maximum end-to-end delay yields 0.5 seconds of maximum new update period. The minimum value for the new update period is roughly determined by considering the allowed bandwidth share of routing update messages within a neighborhood together with average number of neighbors, average update message size, bandwidth, and link access delay. For example, 10 neighbors, 150 bytes of update message, 2 Mbps of bandwidth, 0 seconds of link access delay, and 5% routing bandwidth share yields 0.13 seconds of the minimum update period.

The update-period control computation is completely local to the node. The overhead consists of the extra fields 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 a destination is included only if it satisfies a *update-content control function*.

To implement this, the node maintains an additional field, *t_forwarded_last*, for each entry in the current routing table to indicate when the last forwarding of a packet occurred to that destination. The recent route demands recorded in this additional field represent the *route-demand metric*. Figure 3 describes the update-content control operation.

A *filter function* slows down the rate at which routing information of an unpopular destination is advertised. It can be implemented in several ways, for example, including the routing information only in every Kth routing message, or only 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.

Determining the update-content control function involves choosing a proper value for *TW_RECENT* and the filter function. *TW_RECENT* should be in the order of interarrival times of connections and much larger than packet interarrival times. A few seconds should be reasonable.

If the filter function is aggressive, savings in routing traffic would be high. However it would result in longer delay when opening a connection to a new destination. Thus skipping once in every few updates should be reasonable. For instance, skipping every other update would double the delay in the worst case, but reduce the number of routing entries whose destinations are not currently being used by 50%.

The above description of update-content control assumes an underlying routing protocol which uses destination-based 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 the extra routing table entry field, *t_forwarded_last*, 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

Due to space limitation, the following descriptions of DSDV and ARM-DSDV are abbreviated. For details, see [12].

4.1 DSDV protocol

DSDV is a distance vector routing protocol that achieves loop freedom through the use of destination sequence numbers [1]. The original DSDV uses both periodic and triggered updates, but we consider only periodic updates in this paper.

Each node maintains a constant *UPDATE_PERIOD* denoting the globally fixed update period for all nodes. Variables *dest_seq_no* and *t_update* denote respectively the destination sequence number and the start of the next update period for the node. Routing entries to all possible destinations are maintained in the current routing table, for data

packet forwarding. The new routing table stores routing entries based on the information in routing update messages received in the current period. Each routing entry has members indicating id of the destination, id of the next hop, number of hops to the destination, and destination sequence number of the destination.

A routing update message contains id of the sender, destination sequence number of the sender, and routing update vector indicating the sender's current routing table except for id of the next hop.

A node handles two routing events: expiration of *t_update*, and reception of a routing update message. 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 the new routing table is copied into the current routing table. A routing update message is constructed from entries in the 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 an entry for the sender does not exist in the new routing table, the node creates an entry for the sender and stores the destination sequence number of the sender along with it. Otherwise, the destination sequence number of the existing entry is updated. Then the node processes the vector entries. A vector entry is inserted into the new routing table if there is no entry with the same destination id. Otherwise, the entry with the same destination 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. Because sequence numbers are monotonically increasing over time, this guarantees new routes are selected over old routes.

4.2 ARM-DSDV protocol

ARM-DSDV maintains more state information than DSDV. *UPDATE_PERIOD* is now a variable. The current and new neighbor tables are added. The current and new routing tables have additional fields; specifically, all entries now have a field indicating expiry time and entries in the current routing table have a field indicating the time at which a packet was last forwarded (for update-content control).

ARM-DSDV's routing update messages have two additional fields, sender's update period and sender's mobility metric.

ARM-DSDV's processing of routing events does the following, in addition to what is done in DSDV. On expiration of *t_update*, next update period is computed and current neighbor table is updated. Current routing table is updated using new routing table by throwing away expired entries, retaining non-expired entries, and selecting favored routes. Entries in current neighbor table are used for forwarding in the current period regardless of their expiry times.

On reception of a routing update message, the expiry time is calculated and stored in neighbor and routing tables. New routing table is updated like in DSDV except that expired entries in the table are thrown away and never considered as matches.

On reception or overhearing a data packet, new neighbor table is updated using the packet's predecessor field.

For the update-content control of ARM-DSDV, destination information is recorded on packet forwarding and compared with routing table entries when constructing routing update messages.

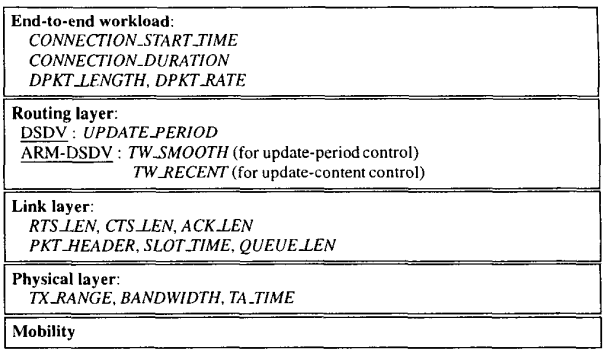

Figure 4. Layer model and parameters

5 Performance Evaluation Model

To compare the performance of DSDV and ARM-DSDV, we have a simulator (written in CSIM [13]) with the layered structure shown in Figure 4. DSDV and ARM-DSDV share the common lower layers as well as the upper workload layer. The figure also summarizes parameters of each layer. The link layer and physical layer sections can be skipped by readers familiar with IEEE 802.11.

5.1 Mobility

The mobility of a node is modeled by 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 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 interval 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 [14] 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 μ 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 performance**: delivery ratio of ARM-DSDV divided by delivery ratio of DSDV.
- **Relative cost**: routing cost of ARM-DSDV divided by routing cost of DSDV.

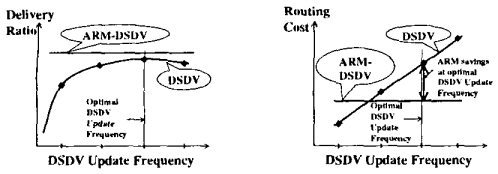

Figure 5. Generic result

Physical layer:
TX_RANGE = 100 m, *BANDWIDTH* = 2 Mbps
TA_TIME = 10 μ sec
Link layer:
RTS_LEN = 40 octets, *CTS_LEN* = 40 octets
ACK_LEN = 34 octets, *PKT_HEADER* = 58 octets
SLOT_TIME = 20 μ sec, *QUEUE_LEN* = 100 packets
Routing layer:
DSDV :
UPDATE_PERIOD = 2, 1, 0.5, 0.2, 0.1, 0.05 sec
ARM-DSDV :
TW_SMOOTH = 5 sec, *TW_RECENT* = 5 sec
update-period control function =

$$\begin{cases} -0.02 \times \text{ag\_mob\_metric} + 0.5 & \text{if ag\_mob\_metric is in (0,20]} \\ 0.1 & \text{otherwise} \end{cases}$$

filter function = skips one in every two advertisement opportunities

Figure 6. Common parameter values

For a mobility pattern, workload, node speed, and simulation time, we run DSDV several times spanning different update periods and ARM-DSDV once. Over a simulation run, ARM-DSDV keeps adapting its period and contents of routing updates. We expect ARM-DSDV and DSDV performance to be as illustrated in Figure 5. 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 6 lists common parameter values that we use for all our simulations. The update-period control function of ARM-DSDV is linear and the maximum and the minimum update periods are 0.5 seconds and 0.1 seconds, respectively. *Ag_mob_metric* in the figure denotes the aggregate mobility metric.

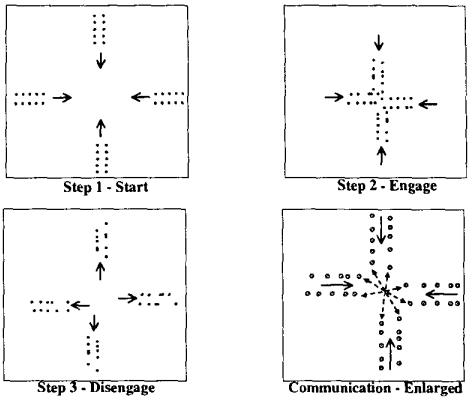

Figure 7. Mobility pattern 1

End-to-end workload:
CONNECTION_START_TIME =
1st time: when simulation begins
2nd time: when nodes start to have paths to each other
CONNECTION_DURATION = 5 sec
DPKT_LENGTH = 100 octets
DPKT_RATE = (10,3) and (15,5) packets/sec
number of connections = 40
Values of (node speed, simulation time, number of runs):
(speed = 15 m/sec, 70 sec, 10 runs)
(speed = 20 m/sec, 50 sec, 10 runs)

Figure 8. Mobility pattern 1 scenario parameters

7.1 Mobility pattern 1

Mobility pattern 1 models wireless nodes on vehicles crossing each other at a highway interchange as shown in Figure 7.

There are four groups of vehicles in a 1.5km $\times$ 1.5km geographical area. Each group has two rows of five vehicles. The rows are separated by 40m. Groups moving in opposite direction on adjacent lanes are separated by 20m. All groups have same group speed but individual vehicle speed varies within $\pm 5\%$ of the group speed. Adjacent nodes in each group start with 40m separation but the separation varies due to varying individual speeds. We have different group speeds of 15 m/sec and 20 m/sec (or 54 km/hr and 72 km/hr, respectively).

As the figure shows, each group starts from the fringe and moves towards the middle. The connections between the nodes are organized into two phases. The first phase of connections start when the simulation begins, and it consists of intra-group connections only. The second phase of connections start when all nodes obtain paths to each other. It consists of inter-group connections only, and these connections are closed before the vehicles pass by each other completely and the paths break down. The last picture of Figure 7 shows which vehicles become connected for the second phase.

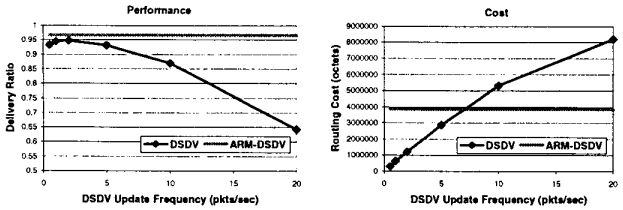

Figure 9. Performance/cost of mobility pattern 1

We have a low load condition where packet rate is 10 packets/sec for the first phase and 3 packets/sec for the second. The high load condition has packet rates of 15 packets/sec and 5 packets/sec, respectively.

Figure 8 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. For each node speed, all the possible combinations of network connections and packet rates are simulated.

We present graphs in Figure 9 for 40 total connections and node speed of 15 m/sec as an example. Due to the space limitation, results from other scenarios are presented in Table 1. The left graph of Figure 9 presents delivery ratio achieved by DSDV and ARM-DSDV. Curve for DSDV clearly shows that it performs well only around optimal update frequency. ARM-DSDV achieves better delivery ratio at 96.6%. Best delivery ratio of DSDV is 94.7% at 2 updates/sec. Apparently, ARM-DSDV finds appropriate values of update frequency. For DSDV protocol, the next optimal (but less frequent) update frequency often yields similar delivery ratio to that of the optimal update frequency. The suboptimal frequency is 1 update/sec, which gives 94.5% of delivery ratio.

The right graph of Figure 9 illustrates the routing cost of both protocols. As expected, DSDV results in linear increase in the routing. One can observe that ARM-DSDV spends more in routing traffic compared to DSDV at optimal or suboptimal update frequencies. However, the routing traffic is kept at reasonable levels to leave bandwidth for the data traffic, indicated by ARM-DSDV's better delivery ratio.

Table 1 presents relative metrics for all the simulation scenarios of mobility pattern 1. The second numbers in the update frequency column are DSDV update frequencies one step below the optimal update frequencies. The relative performance and cost at these frequencies are presented as the second numbers in the relative metrics columns.

When compared at the optimal DSDV update frequencies, ARM-DSDV achieves better delivery ratios. This indicates that ARM-DSDV operates with proper value of update frequencies. The routing costs of ARM-DSDV are reasonable for most of the scenarios.

Packet rate (pkts/s)	Number of connections	Speed (m/s)	Optimal and suboptimal dsdv update frequency (msgs/s)		Relative performance (%)		Relative cost (%)	
10 then 3	40	15	5	2	100.1	102.0	133.9	316.6
		20	5	2	100.6	105.5	153.2	361.2
15 then 5	40	15	2	1	102.0	103.8	318.6	604.6
		20	5	2	101.3	105.3	150.5	357.1

Table 1. Relative metrics of mobility pattern 1

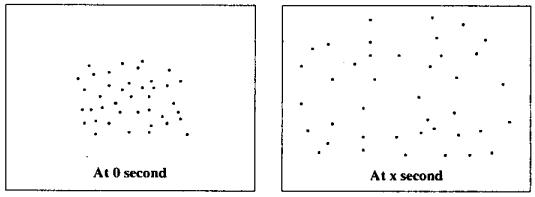

Figure 10. Mobility pattern 2

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. Initially all nodes move with 2 m/sec (7.2 km/hr) and then, after 200 seconds, they move at the increased speed of 20 m/sec (72 km/hr) until the end of simulation. We have another case of mobility where node speeds change from 0 m/sec to 30 m/sec (108 km/hr).

Figure 10 shows how nodes move over the time; at time 100 seconds, the nodes are spread over 0.5km × 1km. There are forty nodes. Each node has identical number of connections to randomly selected peers.

Figure 11 shows the scenario parameters. For each node speed, all the possible combinations of network connections and packet rates are simulated also.

Figure 12 shows delivery ratios of ARM-DSDV and DSDV for 80 connections with 5 packets/sec packet rate. The node speeds are 0 m/sec then 30 m/sec. ARM-DSDV achieves 85.6% while the best delivery ratio of DSDV is 79.2% at 10 update/sec.

End-to-end workload:
CONNECTION_START_TIME = uniformly distributed along simulation time
CONNECTION_DURATION = 5 sec
DPKT_LENGTH = 100 octets
DPKT_RATE = 1, 5 packets/sec
number of connections = 40, 80
Values of (node speed, simulation time, number of runs):
(speed = 2 m/sec then 20 m/sec, 400 sec, 10 runs)
(speed = 0 m/sec then 30 m/sec, 400 sec, 10 runs)

Figure 11. Mobility pattern 2 scenario parameters

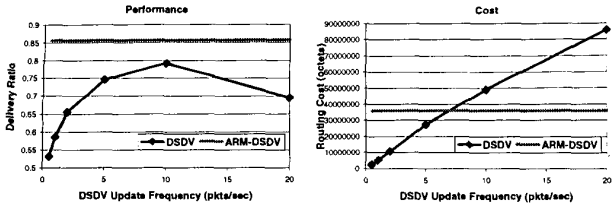

Figure 12. Performance/cost of mobility pattern 2

Packet rate (pkts/s)	Number of connections	Speed (m/s)	Optimal and suboptimal dsdv update frequency (msgs/s)		Relative performance (%)		Relative cost (%)	
1	40	2 then 20	10	5	102.5	108.4	69.2	132.4
		0 then 30	10	5	105.3	110.8	70.0	130.0
	80	2 then 20	10	5	101.0	104.2	69.2	132.4
		0 then 30	10	5	108.0	114.8	73.1	132.6
5	40	2 then 20	10	5	100.5	103.6	69.3	132.4
		0 then 30	10	5	106.3	111.8	73.8	134.2
	80	2 then 15	10	5	100.1	101.7	70.7	134.0
		0 then 30	10	5	109.7	115.6	74.6	134.3

Table 2. Relative metrics of mobility pattern 2

The routing cost is shown in the right graph of Figure 12. The relative cost at optimal DSDV update frequency is 73.1%. ARM-DSDV achieves better delivery ratio while spending much less in routing traffic.

Table 2 presents relative metrics for all the simulation scenarios of mobility pattern 2. Compared to mobility pattern 1, the savings in routing cost are much greater. Relative delivery ratios are better for ARM-DSDV in all the cases, indicating that ARM-DSDV adapts well to various mobility and route-demand patterns.

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 better delivery ratio than DSDV with update period optimized for the mobility and workload scenario, while spending reasonable amount in routing cost. 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.

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 S. 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] S. Ahn and A. U. Shankar. Adapting to route-demand and mobility (ARM) in ad hoc network routing. Technical Report CS-TR-4214, Computer Science, University of Maryland, August 2001.

[13] H. Schwetman. CSIM user's guide. *Microelectronics and Computer Technology Corporation*, 1992.

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