Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol

Elizabeth M. Royer
Dept. of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA 93106
eroyer@alpha.ece.ucsb.edu

Charles E. Perkins
Networking and Security Center
Sun Laboratories
Menlo Park, CA 94025
cperkins@eng.sun.com

Abstract

An *ad-hoc* network is the cooperative engagement of a collection of (typically wireless) mobile nodes without the required intervention of any centralized access point or existing infrastructure. To provide optimal communication ability, a routing protocol for such a dynamic self-starting network must be capable of unicast, broadcast, and multicast. In this paper we extend Ad-hoc On-Demand Distance Vector Routing (AODV), an algorithm for the operation of such ad-hoc networks, to offer novel multicast capabilities which follow naturally from the way AODV establishes unicast routes. AODV builds multicast trees as needed (i.e., *on-demand*) to connect multicast group members. Control of the multicast tree is distributed so that there is no single point of failure. AODV provides loop-free routes for both unicast and multicast, even while repairing broken links. We include an evaluation methodology and simulation results to validate the correct and efficient operation of the AODV algorithm.

1 Introduction

The idea of ad-hoc networks of mobile nodes dates back to the days of the DARPA packet radio network [11]. In more recent years, interest in these networks has grown along with improvements in laptop computers. These improvements include greater power, longer battery life, and decreased weight. Because so many laptop computers are now in use, and because these computers are easily portable due to their compact and lightweight design, the ability to communicate from one such computer to another, and from one such computer to a fixed network, is desired.

To facilitate such communication, many routing protocols have been developed [5, 10, 13, 14, 16]. While each of these protocols is able to provide unicast capability to network nodes, none offers multicast communication ability. Although multicast is not necessary to establish communication between nodes, it is frequently a desired feature for a network. A few protocols have been created to provide the multicast communication which these other protocols lack. The Lightweight Adaptive Multicast (LAM) protocol [9] is an example of one of these protocols. LAM is tightly coupled with the Temporally-Ordered Routing Algorithm (TORA) [14] as it depends on TORA's route finding ability and cannot operate independently. An advantage of LAM is that, since it is tightly coupled with TORA, it can take advantage of TORA's route finding ability and thereby reduce the amount of control overhead generated. However, LAM has the disadvantage that it relies on a *core* node, thus has a central point of failure. Other protocols specified in internet drafts [3, 8, 22] are also able to provide multicast communication, but they too depend on an underlying routing protocol for correct operation. Additionally, the routing protocol described in [3] can suffer from transient routing loops.

Unlike other protocols, the Ad-hoc On-Demand Distance Vector Routing (AODV) [17, 18] protocol is capable of unicast, broadcast, *and* multicast communication. Unicast and multicast routes are discovered on-demand and use a broadcast route discovery mechanism. Broadcast data delivery is provided by AODV by using the Source IP Address and Identification fields of the IP header as a unique identifier of the packet. The destination address of broadcast data packets is set to the well-known broadcast address 255.255.255.255. The redundant processing and propagation of a data packet multiple times by a single node is prevented because each node records the Source IP Address and Identification fields of the IP header of the packet. All additional copies of a data packet are discarded after the original reception.

There are numerous advantages to combining unicast and multicast communication ability in the same protocol. A protocol which offers both forms of communication can be streamlined so that route information obtained when searching for a multicast route can also increase unicast routing knowledge, and vice versa. For instance, if a node returns a route for a multicast group to some source node, that source node, in addition to learning how to reach the multicast group, will also have learned of a route to the node returning that information. AODV can take advantage of this to enhance general routing knowledge. In a mobile environment, any reduction in control overhead is a significant advantage for a routing protocol. Additionally, combining both types of communication into a single protocol simplifies coding. Lastly, we expect that continued improvements to the basic algorithm (e.g., for Quality of Service (QoS) applications, for client/server discovery, or for utilizing asymmetric routing paths) will benefit both unicast and multicast data transmission. AODV currently utilizes only symmetric links between neighboring nodes, but otherwise does not depend specifically on particular aspects of the physical medium across which packets are disseminated.

The remainder of this paper is organized as follows. In Section 2, the basic data structures required for operation of the AODV algorithm are presented. Section 3 describes the route request/route reply query cycle used for unicast route discovery. Section 4 describes, in detail, the multicast algorithm. Simulation results are presented in Section 5. Section 6 describes our plans for future work, and finally Section 7 concludes the paper.

2 Routing Tables

Each node running AODV maintains two routing tables. The first of these is the *Route Table*. The route table is used for recording the next hop for routes to other nodes in the network. The fields of the route table are as follows:

- Destination IP Address
- Destination Sequence Number
- Hop Count to Destination
- Next Hop
- Lifetime

New entries are placed in the route table following the reception of route requests (RREQs) and route replies (RREPs). When a node receives one of the listed message types, and it does not already have a route entry for the source of the message, it places an entry in the table listing the indicated information. Associated with each entry is a *lifetime*, indicating the length of time the route entry is valid. Routes are deleted from the table if they are not been updated or used within the indicated lifetime.

The second routing table that a node maintains is the *Multicast Route Table*. This table contains entries for multicast groups of which the node is a router (i.e., a member of the multicast tree). Each entry in the multicast route table contains the following information:

- Multicast Group IP Address
- Multicast Group Leader IP Address
- Multicast Group Sequence Number
- Hop Count to Multicast Group Leader
- Next Hops
- Lifetime

New entries are placed in this table after the node becomes a router for a multicast group. Associated with each Next Hop entry is an *Enabled* flag. This flag is used to indicate whether the link has been officially added on to the multicast tree. The Enabled flag of a next hop entry is set only after the activation of a route by the reception of a Multicast Activation (MACT) message, as described in Section 4.5.1. For multicast route entries, there may be more than one next hop entry.

A third table, called the *Request Table*, is a small table that contains only two fields:

- Multicast Group IP Address
- Requesting Node IP Address

Each node in the network that supports multicast routing maintains this table, regardless of whether it is a member of the multicast group. This feature is used solely for optimization and does not affect the correct operation of the protocol. When a node receives a RREQ to join a multicast group, it checks its request table for an entry for that group. If no entry for the group exists in the table, the node records the IP address of the group, together with the IP address of the node requesting a route to the group. Because the first node to request membership in a group typically becomes the multicast group leader, the entries in the table represent the group leaders. If a node later wishes to join a multicast group, it can check its request table to determine who the group leader is. If it has a route to that node, it can unicast its RREQ instead of broadcasting it.

3 Route Discovery

Route discovery with AODV is purely on-demand and follows a route request/route reply discovery cycle. When a node needs a route to a destination, it broadcasts a RREQ. Any node with a current route to that destination (including the destination itself) can unicast a RREP back to the source node. Route information is maintained by each node in its route table. Information gleaned through RREQ and RREP messages is kept with other routing information in the route table. AODV uses sequence numbers to eliminate stale routes. Routes with old sequence numbers are aged out of the system.

AODV's primary objectives are as follows:

- To provide unicast, broadcast, and multicast capability to all nodes in the ad-hoc network
- To minimize the broadcast of control packets.
- To disseminate information about link breakages to those neighboring nodes that utilize that link.

The following section briefly describes route discovery in AODV. For further details, please see [18].

3.1 Reverse Route Establishment

Route discovery with AODV is on-demand and occurs when a node requires a route to a destination for which it does not already have a recorded route. Such a node initiates route discovery by broadcasting a RREQ packet [5]. The fields of the RREQ are as follows:

$$< J\_flag,\ R\_flag,\ Broadcast\_ID,\ Source\_Addr,\ Source\_Seq\#,\ Dest\_Addr,\ Dest\_Seq\#,\ Hop\_Cnt >$$

The *J_flag* and *R_flag* (*join* and *repair* flags, respectively) fields are used only for multicast group RREQs (described in Section 4.1).

Each node in the network is responsible for maintaining two separate counters: a sequence number and a broadcast ID. The sequence number ensures the freshness of routes to the node. The broadcast ID, together with the source node's IP address, uniquely identifies each RREQ. The sequence number is increased when the

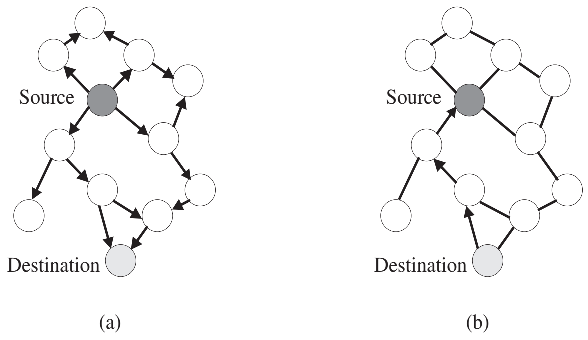


Figure 1: The RREQ / RREP Message Cycle.

node acquires new neighbor information, and the broadcast ID is incremented for each RREQ the node initiates. The node requesting the route places its IP address, current sequence number, and broadcast ID in the *Source_Addr*, *Source_Seq#*, and *Broadcast_ID* fields, respectively. The IP address of the destination and the last known sequence number for that destination are placed in the *Dest_Addr* and *Dest_Seq#* fields.

A node receiving a RREQ first updates its route table to record the sequence number and next hop information for the source node. This *reverse route* entry may later be used to relay a RREP back to the source. The node then checks this table to see whether it has a route to the requested destination. In order to respond to a RREQ, a node must either be the destination itself, or must have an unexpired route to the destination with a sequence number at least as great as that indicated in the *Dest_Seq#* field of the RREQ. A node having such a route is said to have a 'fresh enough' route to the destination. If this is the case, the node generates a RREP as described in Section 3.2 below. Otherwise, it rebroadcasts the packet to its neighbors. Figure 1(a) illustrates the broadcasting of RREQs.

A node may receive the same RREQ multiple times. When a node receives a RREQ, it records the source address and broadcast ID of the packet. If it later receives a RREQ with this same information, it does not process the packet but instead discards it.

3.2 Forward Path Setup

As stated above, a node can respond to a RREQ if it is the destination itself, or if it has a fresh enough route to the destination. When a node fulfills these requirements, it sends a RREP back to the source node. The RREP contains the following information:

$$< R\_flag, U\_flag, Dest\_Addr, Dest\_Seq\#, Hop\_Cnt, Lifetime >$$

The *Dest_Addr* field is set to the destination address specified in the RREQ, and the *Dest_Seq#* is set to the responding node's record of the destination's sequence number. The *Hop_Cnt* field is set to the distance of the responding node from the destination, or zero if the destination itself sends the RREP. The *R_flag* and *U_flag* (*repair* and *update* flags) fields are used only for multicast routes, as described in Section 4.3.

The responding node unicasts the RREP back along the next hop towards the source node. The node receiving the RREP increments the *Hop_Cnt* field by one and then updates its entry for the destination node in its route table, thereby establishing the forward path to the destination. It then unicasts the RREP to its recorded next hop to the source node. This continues until the RREP reaches the source node. Figure 1(b) is an example of the destination node responding by sending a RREP back to the source. Nodes that are not along the path determined by the RREP delete the reverse pointers after `active_route_timeout` (3000 msec).

Once the source node receives the RREP, it can use the route to send data packets to the destination. In the event that it receives a RREP in the future with a greater destination sequence number or a smaller hopcount, the source node updates its route table information for the destination and instead uses the new route.

It is likely that an intermediate node will receive more than one RREP for a given source/destination pair. In this case, the node checks the *Dest_Seq#* and *Hop_Cnt* fields against its recorded information. If the destination sequence number is greater than the node's recorded value, or if the sequence number is the same but the *Hop_Cnt* is smaller, the node updates its information for the destination and forwards the RREP to the source. Otherwise, if the information contained in the RREP is not as good as that which the node already has in its route table and has sent to the source, it will discard the RREP and not forward it.

3.3 Local Connectivity Management

Nodes learn of their neighbors through packet transmissions. When a node sends a packet, its neighbors hear the transmission and update their local connectivity information to ensure that it includes this neighbor. In the event that a node has not transmitted anything within the last `hello_interval` msec, it broadcasts to its neighbors a Hello message. This informs its neighbors that it is still within their transmission range. A Hello message is a special unsolicited RREP which contains a node's IP address and current sequence number. The Hello message is prevented from being rebroadcast outside the neighborhood of the node because it contains a time to live (TTL) value of 1. Neighbors that receive this packet update their local connectivity information to include the node. The failure to receive any transmissions from a neighbor in the time defined by the periodic transmission of `allowed_hello_loss` Hello messages is an indication that the local connectivity has changed, and the route information for this neighbor should be updated (see also Section 4.6.1).

4 The Multicast Algorithm

The multicast algorithm uses the same RREQ/RREP messages as previously described. Only one new message, the Multicast Activation (MACT), is needed. As nodes join the multicast group, a multicast tree composed of group members and nodes connecting the group members is created. Multicast group membership follows the model of the Mbone in that it is dynamic; nodes are able to join and leave at any time [6]. A multicast group leader maintains the multicast group sequence number. Multicast group members must also agree to be routers in the multicast tree.

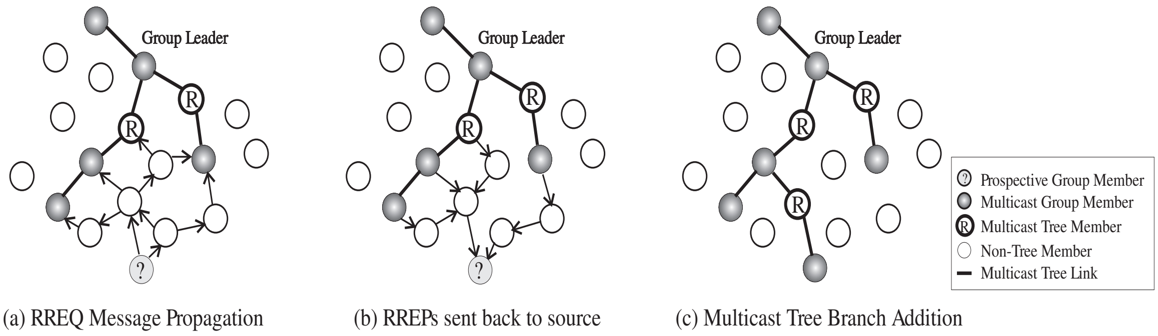


Figure 2: Multicast Join Operation.

4.1 Route Request Message Generation

A node sends a Route Request (RREQ) message when it wishes to join a multicast group, or when it has data to send to a multicast group and it does not have a route to that group. The *Dest_Addr* of the RREQ is set to the IP address of the desired multicast group, and the destination sequence number is set to the last known sequence number for that group. If the node wishes to join the multicast group, it sets the *J_flag* of the RREQ; otherwise, it leaves the flag unset. The RREQ may be either broadcast or unicast depending on the information available at the source node. If the source node has a record of another node (the multicast group leader) previously requesting a route to that multicast group, and if the source node has a valid route to that node, it includes an extension field containing the IP address of the group leader and *unicasts* the RREQ along the known path to the group leader. Otherwise, if the source does not know who the group leader is or if it does not have a route to the group leader, it broadcasts the request. Figure 2(a) illustrates the propagation of a broadcast RREQ.

Only a member of the desired multicast tree (i.e., a router for the group) may respond to a *join* RREQ. If the RREQ is not a *join* request, any node with a fresh enough route to the multicast group may respond. If a node receives a *join* RREQ for a multicast group of which it is not a member, or if it receives a RREQ and it does not have a route to that group, it rebroadcasts the RREQ to its neighbors.

If the source node does not receive a RREP before timing out, it broadcasts another RREQ with *Broadcast_ID* increased by one. If it does not receive a RREP to this RREQ, it continues broadcasting route requests up to `rreq_retries` total rebroadcasts. After this number of attempts, it can be assumed that either the multicast group is unreachable, or there are no other members of that multicast group in its connected portion of the network. In this case, the node becomes the multicast group leader, and initializes the group sequence number (i.e., sets equal to 1). If the original RREQ is unicast to the group leader and a RREP is not received, all further RREQs are broadcast, because it is possible that either the group leader is unreachable or that the node specified in the unicast RREQ is *no longer* the group leader. The *Dest_Addr* of each broadcast RREQ is set to the IP address of the multicast group, and the extension containing the IP address of the group leader is not included.

Nodes receiving a *join* RREQ check their request table for an entry for the requested multicast group. If there is no entry for the multicast group, the node enters the multicast group address, together with the IP address of the requesting node, in its request table. If there is no previous entry for the group, the requesting node may become the group leader. A node wishing to join a multicast group consults its request table to determine the group leader.

4.2 Reverse Route Establishment

As the RREQ is broadcast across the network, nodes set up pointers to establish the reverse route. Propagation of non-*join* RREQs for multicast groups is similar to that described in Section 3.1. A *join* RREQ, however, requires a few processing differences. A node receiving a *join* RREQ maintains a corresponding route entry in its multicast route table, in addition to its (unicast) route table. The *Enabled* flag for this entry is set to FALSE, and only later is set to TRUE if the route is selected to be added to the multicast tree (see Section 4.5.1). A node can only respond to a *join* RREQ if it is a member of the multicast tree. The generation of route replies is described below.

4.3 Route Reply Message Generation

If a node receives a *join* RREQ for a multicast group, it may reply if it is a router for the multicast group's tree and its recorded sequence number for the multicast group is at least as great as that contained in the RREQ. Additionally, the group leader can always reply to a *join* RREQ for its multicast group. The responding node updates its route and multicast route tables by placing the requesting node's next hop information in the tables, and then generates a RREP. The node then unicasts the RREP back to the node indicated by the *Source_Addr* field of the received RREQ. Figure 2(b) illustrates the path of the RREPs to the source node.

The RREP contains the last known sequence number for the multicast group and the IP address of the multicast group leader. In addition, it includes a special extension field called *Mgroup_Hop*. This field is initialized to zero and incremented each time the packet is forwarded. When the RREP is received by the source node, the *Mgroup_Hop* field indicates the distance (in hops) of the source node from the nearest member of the multicast tree. The IP address of the group leader is also placed in an extension field, called *Group_Leader_Addr*.

As nodes along the path to the source node receive the RREP, they add both a route table and a multicast route table entry for the node from which they received the RREP, thereby creating the forward path. They increment the *Hop_Cnt* and *Mgroup_Hop* fields of the RREP and then continue forwarding the RREP back towards the source node.

In the event that a node receives a unicast RREQ with its own IP address in the Multicast Group Leader extension, and if the node is in fact not the group leader, it simply ignores the request and does not propagate the RREQ any further. The source node will timeout and broadcast a new RREQ without the multicast group leader extension. This event should never happen; however to protect against the possibility that a node has out-dated group leader information in its request table, a mechanism is included to handle a RREQ with such invalid information.

4.4 Group Hello Messages

The first member of the multicast group becomes the leader for that group. This node remains the group leader until it decides to leave the group, or until two partitions of the multicast tree merge (see Section 4.6.2). The multicast group leader is responsible for maintaining the multicast group sequence number and for disseminating this number to the multicast group. Periodically (every `group_hello_interval` seconds), the group leader broadcasts a Group Hello message. The Group Hello message is an *unsolicited* RREP with a TTL greater than the diameter of the network, so that it is propagated across the entire network. The Group Hello contains extensions which indicate the multicast group IP addresses and corresponding sequence numbers of all multicast groups for which the node is the group leader. The sequence number for the group is incremented for each Group Hello broadcast by the group leader. The *Hop_Cnt* of the Group Hello is initialized to zero and is incremented by each node that receives it, thereby indicating the distance in hops from the group leader.

Nodes use the Group Hello information to update their request table. When a node receives the Group Hello, it checks its request table for an entry for the advertised multicast group. If the table does not contain an entry for that group, the node enters the group and group leader IP addresses. Nodes that are members of the multicast tree use the Group Hello to update their current distance from the group leader. The Group Hello is also used for merging partitioned multicast trees, as is described in Section 4.6.2.

4.5 Multicast Tree Maintenance

Because the network consists of mobile nodes, links on the multicast tree are likely to break. Link breakages must be repaired in a timely manner to maximize multicast group connectivity. Multicast tree maintenance can be divided into three main categories: selecting and activating the link to be added to the tree when a new node joins the group, pruning the tree when a node decides to leave the group, and repairing a broken link. Repair involves re-establishing branches when a link fails and reconnecting the tree after a network partition.

At any interior node in a multicast tree, the route entry for the multicast group has multiple next hops. When a data packet addressed to the multicast group is received by a multicast tree member, the Source IP Address and Identification fields of the data packet's IP header are recorded. The packet is then multicast by the node to its next hops. If the node is a group member, the packet is processed. A node on the multicast tree may receive the same data packet multiple times if it receives a data packet, retransmits the packet to its next hops, and then receives that same data packet when its next hops retransmit the packet to their next hops. The node will detect this redundancy by checking the Source IP Address and Identification fields of the IP header, and it will then discard the packet.

4.5.1 Multicast Route Activation

When a source node broadcasts a RREQ for a multicast group, it often receives more than one reply. Because each of the RREPs sets up a potential addition to the multicast tree, one and only one of the RREPs must be selected as the next hop. In this way, only one branch is added to the tree, and loops are thereby avoided. This is accomplished as follows. The source node waits `rte_discovery_timeout` milliseconds after sending the RREQ before selecting a route. `rte_discovery_timeout` is a configurable parameter which may be set according to the size of the network. During this time period, the node keeps the received route with the greatest sequence number and the shortest number of hops to the nearest member of the multicast tree; it disregards other routes. At the end of this period, it enables the selected next hop in its multicast route table, and then *unicasts* a Multicast Activation (MACT) message to this selected next hop. Each MACT message contains the following fields:

$$< P\_flag, GL\_flag, Source\_Addr, Source\_Seq\#, Dest\_Addr >$$

The *Dest_Addr* is set to the IP address of the multicast group. The *P_flag* and *GL_flag* fields, which are used for pruning and choosing a new group leader, respectively, are explained in Sections 4.5.2 and 4.6.1.

The next hop, on receiving the MACT message, likewise enables the entry for the source node in its multicast route table. If this node is a member of the multicast tree, it does not propagate the MACT any further. However, if this node is not a member of the multicast tree, it will have received one or more RREPs from its neighbors. It keeps the best next hop for its route to the multicast group, unicasts a MACT to that next hop, and enables the corresponding entry in its multicast route table. This process continues until the node that originated the RREP (because it was already a member of the tree) is reached. Nodes that had generated or forwarded RREPs delete the entry for the requesting node if they do not receive a MACT activating their route after `mtree_build` milliseconds. Figure 2(c) illustrates a multicast tree created in the described manner.

The MACT message ensures that the multicast tree does not have multiple paths to any tree node (and, thus, is in fact a tree). Nodes only forward data packets along activated routes in their multicast route table. This prevents the possibility of data packets being delivered to a source node by multiple next hops before a MACT message is received.

4.5.2 Pruning

During normal network operation, a multicast group member may decide to terminate its membership in the multicast group. If the node is not a leaf node of the tree, it may revoke its member status but it must continue to serve as a router for the tree. Otherwise, if the node is a leaf node, it may prune itself from the tree by using the MACT message. In this case, the *P_flag* (*prune*) of the MACT is set, and the *Dest_Addr* is set to the IP address of the multicast group. A leaf node necessarily has only one next hop for the multicast group, so it unicasts the MACT message to that next hop. After sending the message, the node removes all information for the multicast group from its multicast route table. The next hop, on receiving the MACT, notes the *P_flag*, and consequently deletes the entry for the sender node from its multicast route table. If this node is itself not a member of the multicast group, and if the pruning of the other node has made it a leaf node, it can similarly prune itself from the tree by the method described. Tree branch pruning terminates when either a multicast group member or a non-leaf node is reached.

4.6 Repairing Broken Links

Multicast group tree links may break due to node mobility or route expiration timers. Unlike in the unicast scenario, however, a link breakage necessarily triggers route reconstruction because of the necessity of keeping the multicast group members connected during the lifetime of the group. The re-establishment of tree links after breakages and network partitions is described below.

4.6.1 Link Breakages

Nodes promiscuously record the reception of any neighbor's transmission. A link breakage is detected if no packets are received from the neighbor in the time

$$\texttt{hello\_interval} \times (1 + \texttt{allowed\_hello\_loss}).$$

If a neighbor transmits other packets during that time, the neighbor is no longer obligated to transmit any Hello packets because the other packets serve the purpose of signaling its presence. The neighbor is also expected to forward any data packets received to their next hop(s) within `retransmit_time` msec. Failing to receive any transmissions from a neighbor will cause the expiration of the route timer associated with that route.

When a link breakage is detected, the node *downstream* of the break (i.e., the node that is further from the multicast group leader) is responsible for repairing the broken link. This distinction is made because, if both nodes tried to repair the link, it is possible they would establish different paths and thus form a loop. The downstream node initiates the repair by broadcasting a RREQ with *Dest_Addr* set to the IP address of the multicast group leader and with the *J_flag* set. The *Dest_Seq#* is set to the last known sequence number of the multicast group, and the Multicast Group Hop Count (*Mgroup_Hop*) extension is set to the distance of the node from the multicast group leader. The only nodes which may reply to a RREQ with the *Mgroup_Hop* extension are nodes that are at least as close to the group leader as indicated by this field, or the group leader itself. This prevents nodes on the same side of the break as the initiating node from responding, thereby ensuring a new route to the group leader is found.

Because the node with which the initiating node lost contact is likely to still be nearby, the initial TTL value of the RREQ is set to a small value. In this way, the effects of the link breakage can be localized. If no RREP is received within `rte_discovery_timeout` milliseconds, all successive RREQs (up to `rreq_retries` additional attempts) are broadcast across the network. Any node that is a part of the multicast tree and that has a fresh enough multicast group sequence number and a hopcount to the multicast group leader smaller than that indicated by the *Mgroup_Hop* field can respond to the RREQ by unicasting a RREP. Forward path set up and subsequent route deletions occur as described in Sections 4.3 and 4.5.1.

If no RREP is received at the source node after `rreq_retries` attempts, it can be assumed that the network has become partitioned and the tree cannot (at this time) be reconnected. In this scenario, the partition of the tree that is downstream of the break is left without a group leader. A new group leader must be chosen. This occurs in one of two ways. If the node that initiated the route rebuilding is a multicast group member, it becomes the new multicast group leader. On the other hand, if it was not a group member and has only one next hop for the tree, it prunes itself from the tree by sending its next hop a MACT message with the *P_flag* set. On receiving the MACT, the node notes that the message came from its link to the group leader. This indicates that a network partition has occurred and that the next hop has pruned itself from the tree. If this node is a multicast group member, it becomes the new group leader. Otherwise, it also prunes itself from the tree, and this process will continue until a multicast group member is reached.

In the event that the node that initiated the rebuilding is not a group member and has more than one next hop, it cannot prune itself from the tree because doing so would leave the tree partitioned. Instead, it selects the first of its next hops and unicasts a MACT with the *GL_flag* (*group leader*) set. This flag indicates that the next group member to receive the MACT should become the new group leader. Hence, if the next hop receiving this message is a group member, it becomes the group leader. Otherwise, if it is not a group member, it similarly selects one of its next hops and unicasts a MACT with the *GL_flag* set. This process continues until a multicast group member is reached.

After becoming the new multicast group leader, the node broadcasts a Group Hello across its connected part of the network (partition). This message has the *U_flag* (*update*) set, indicating that it is the new group leader and all nodes should update their multicast route table and request table information accordingly.

After a multicast tree link breakage is discovered, if the node *upstream* of the break is a not a group member, and if the link breakage causes this node to become a leaf node, it sets a timer and waits for the tree branch to be reestablished through it. If it does not receive a MACT from a downstream node within `route_expiration` msec, either another node was chosen as the next hop on the tree, or the network has become partitioned and the link

Parameter Name	Meaning	Value
`allowed_hello_loss`	# of Allowed Hello Losses	2
`group_hello_interval`	Frequency of Group Hello Broadcasts	5 sec
`hello_interval`	Frequency of Hello or Other Broadcasts	1000 msec
`max_retrans`	Maximum # of Retransmissions	10
`mtree_build`	Time to Wait to Receive a MACT	2000 msec
`retransmit_time`	Time to Wait for Data Packet Retransmissions	1000 msec
`rev_route_life`	Time to Keep Reverse Route Entries	3000 msec
`route_expiration`	Lifetime of Route Table Entry	3000 msec
`rreq_retries`	Max # of RREQ Retransmissions	2
`rte_discovery_timeout`	Max Time to Wait for a RREP	1000 msec

Table 1: Simulated Parameter Values.

could not be reestablished. In either case, it prunes itself from the tree in the manner described in Section 4.5.2.

4.6.2 Reconnecting Partitioned Trees

After the multicast tree becomes disconnected due to a network partition, there are two group leaders. If the partitions reconnect, a node eventually receives a Group Hello for the multicast group that contains group leader information that differs from the information it already has. If this node is a member of the multicast group, and if it is a member of the partition whose group leader has the lower IP address, it can initiate the reconnection of the multicast tree. The node must already be a member of the group in order to minimize the number of tree branches of the group, and its group leader must have the lower IP address so that only one of the group leaders attempt to rebuild the tree, thereby avoiding loops.

If a node meets the above criteria, it unicasts a RREQ with the *R_flag* (*repair*) set to its group leader. The *R_flag* indicates that the RREQ needs special handling. The group leader, after receiving such a RREQ, grants the node permission to rebuild the tree by unicasting a RREP back to the node. It notes that it has given this node rebuilding permission and must not grant any other node such permission unless the current rebuild fails. Again, this is to prevent multiple nodes from attempting repairs (which would likely cause the formation of loops).

After receiving a RREP granting it rebuilding permission, the node unicasts a RREQ to the other group leader, using the node from which it received the Group Hello as the next hop. This RREQ contains the current value of the partition's multicast group sequence number. When it receives the RREQ, the other group leader notes the set *R_flag*, takes the larger of its record of the group's sequence number and the received sequence number for the group, and increments this value by one. It then unicasts a RREP back to the source node. This group leader becomes the leader of the reconnected tree. As the RREP travels back to the source, it grafts a branch on to the tree. Having noted the *R_flag*, the next time the group leader sends a Group Hello, it sets the *U_flag*. All members formerly contained in the other partition (including the partition's group leader) note the new group leader information, and the merging of the two trees is then complete.

5 Simulations and Results

We have simulated AODV using an event-driven, packet-level simulator called PARSEC [2], which was developed at UCLA as the successor to Maisie [1]. The PARSEC language is suited to the simulation of dynamic topologies and routing algorithms. The main objective of the simulations is to show that AODV accurately builds a multicast tree on-demand, and that this tree can be used to efficiently route data packets between multicast group members.

5.1 Simulation Environment

Our simulations were run using a network composed of 50 nodes. Nodes are initially placed randomly within a fixed-size $L \times L$ area. During the simulation, nodes are free to move anywhere within this area. Each node has a predefined speed between zero and one meter per second. It then travels towards a random spot within the $L \times L$ area. The node moves until it reaches that spot, then chooses a rest period from a uniform distribution between 60 and 300 seconds. After the rest period, the node travels towards another randomly selected spot. This process repeats throughout the simulation, causing continuous changes in the topology of the underlying network.

The communication radius R_{max} of the nodes is a major contributor to the interconnection pattern of the ad-hoc network. In our simulations, the communication radius is held constant at 10m. Two nodes can communicate directly, and are thus considered each other's neighbors, if they are less than R_{max} distance apart. If they are farther apart than R_{max}, they cannot hear each others transmissions.

The channel model used in the simulation is CSMA. Before beginning a transmission, carrier sensing is performed by a node to determine whether any of its neighbors are transmitting. If a node detects an ongoing transmission by a neighbor, it calculates an exponential back-off based on the number of times it has attempted the retransmission and waits this amount of time before sensing the channel again. A node attempts to transmit a packet `max_retrans` times before dropping the packet.

Nodes in the simulation may suffer from the *hidden terminal* problem [21]. If node A transmits to node B, and node C, unable to hear node A's transmission, simultaneously transmits to node B, the packets are assumed

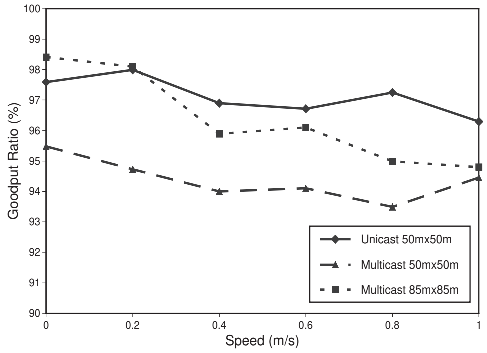


Figure 3: Goodput Ratio as a Function of Speed.

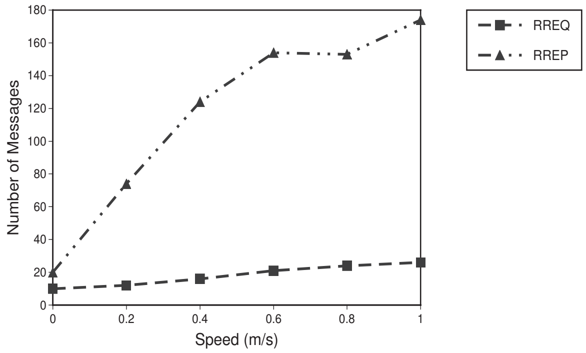


Figure 4: Control Overhead for Unicast Simulations.

to collide at node B and both packets are dropped.

Data sessions begin at randomly selected times throughout the simulation. Data packets are 64 bytes in length and the number of data packets transmitted per session is a geometric distribution with average 3,000. The data rate is 1.0 Mbit/sec. The simulations were run for 300 seconds, and new sessions are generated throughout the simulation. New data sessions are generated according to a geometric distribution with average of 25 minutes. This amounted to eight generated sessions per unicast simulation. In addition, once a node is a member of the multicast group, it generates new sessions for that multicast group according to a geometric distribution with average of 12 minutes. This produced approximately fifteen data sessions per multicast simulation. Because sessions are generated throughout the simulation, we keep track of and account for any data packets in transit at the end of the simulation.

As stated earlier, multicast group membership is dynamic. Non-group members are also able to create sessions and send data packets to members of the multicast group.

Table 1 gives the values of the essential parameters for the AODV simulation. The parameter values were chosen because they minimize network congestion while allowing the algorithm to operate as quickly and as accurately as possible.

5.2 Results and Discussion

To examine the accuracy of AODV's multicast operation, we ran simulations of both unicast and multicast communication. In [18], we present various simulations of AODV which demonstrate that AODV's unicast operation is both accurate and efficient. The unicast and multicast simulations discussed here use the parameter values given in Section 5.1. We examine the results produced by the simulations to show that AODV's multicast performance is comparable to its unicast performance. Particularly, we examine the goodput ratio and the amount of control overhead produced by the simulations. We define the goodput ratio as the number of data packets received compared to the number of data packets sent.

In the first simulations, a room size of 50m×50m is used. This size room, with 50 nodes and a transmission radius of 10m, allows the vast majority of nodes to be able to reach all other nodes, in one or more hops, throughout the simulation. This enables us to verify that AODV builds a multicast tree between group members and then maintains that tree throughout the lifetime of the group. With this room size there are few, if any, partitions of the multicast tree. We performed simulations of both unicast and multicast communication in the 50m×50m room.

In the second set of multicast simulations described, the room size is increased to 85m×85m while the transmission radius is held at 10m. With a room this large there are many small network partitions which are isolated from each other. Many of these network partitions contain multicast group members. As the simulation progresses and nodes move about the room, we are able to verify that group members recognize when they come into contact with another partition and that consequently the multicast trees merge and one group leader is selected. A unicast simulation of the 85m×85m network is not included because, with such a sparsely connected network, it is a frequent occurrence that a route to a desired destination does not exist.

Each class of simulations was run for six different speeds of node movement. The speeds ranged from 0 m/s to 1 m/s. For each movement speed, ten simulation runs were completed, where each run had a different initial network configuration. The results of these simulations were averaged together to produce the resulting graphs.

In the multicast simulations, there is one multicast group which nodes may choose to join. No unicast sessions are created; all data traffic is multicast. As nodes decide to join the multicast group, they broadcast RREQs in the manner described in Section 4.1. Hence, at the beginning of the simulation, there are no multicast group members. The number of multicast group members then increases and decreases as nodes decide to join and leave the group. At any given time in the simulation, there are as many as 10 nodes which are members of the multicast group. A node may send data packets to the multicast group regardless of whether it is a member of the multicast group. If a node is not a member of the group, it finds a route to the multicast group and then transmits its data packets along that route. In the unicast simulation, however, all generated sessions are point-to-point. Any node can potentially be selected as a sender or a receiver for a given session.

Because nodes are frequently moving and routes between nodes break, the goodput ratio is not likely to be 100%. AODV does not retransmit data packets that are

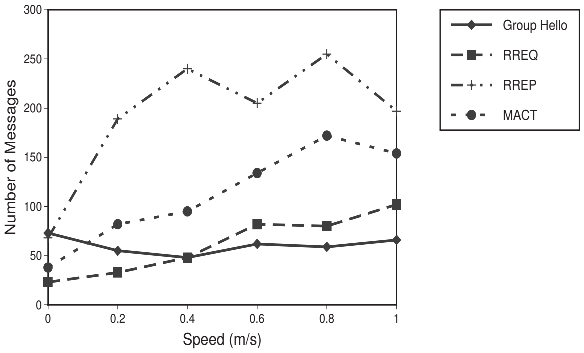


Figure 5: Control Overhead for 50m×50m Multicast Simulations.

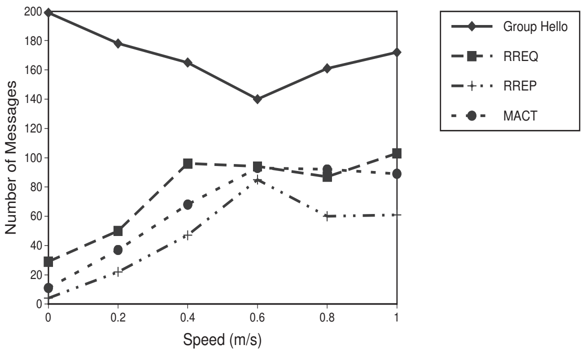


Figure 6: Control Overhead for 85m×85m Multicast Simulations.

lost due to node movement, and hence does not guarantee packet delivery. However, it does find good routes for IP's best-effort delivery, and the goodput ratio is high.

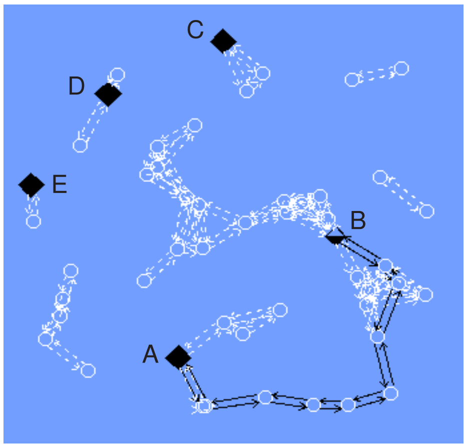
Figure 3 indicates the achieved goodput ratio for each of the simulation scenarios at different speeds. The 50m×50m multicast simulations show a slightly decreased goodput ratio compared to the results of the similar unicast simulations. This results from the fact that, while each data packet in the unicast simulations must only be received by one node, each data packet in the multicast simulations must be received by multiple nodes. That is, every member of the multicast group in the connected portion of the network must receive the data packets. This results in an increased likelihood of collisions. Although nodes buffer packets while they rebuild routes, packets that are sent during reconstruction of tree branches have the possibility of being lost if the nodes on each end of the break are not a part of the reconnected branch. This is due to the fact that there are no retransmissions of data packets. Hence there is a greater likelihood of packet loss in the multicast simulations since there are many more routes which must be maintained.

The multicast simulations of the 85m×85m network demonstrates AODV's operation under continual network partitions and merges. Because the connectivity of the network is so low, most multicast group members are singleton members of their partition, and hence they are group leaders. However, whenever two network partitions, each having one or more multicast group members, merge, the multicast trees must also merge and an overall group leader must be chosen. Similarly, whenever a portion of the network with two or more group members partitions, where each of the network components then has one or more group members, the component without the group leader must choose a new group leader. In the simulations of speed 0 m/s, the goodput ratio is high because there were typically between only one and three multicast group members in a single partition, and so the data packets did not need to be delivered to a large number of group members. As the speed of movement of the nodes increases, however, the goodput ratio decreases. With such a small communication radius in a large room, group members often no sooner discover each other than they are out of transmission radius of each other. This is especially true in the .8 m/s and 1.0 m/s simulations. Nodes reconnect the tree and start sending data packets, and then the tree quickly becomes partitioned again. The data packets in transit must be dropped.

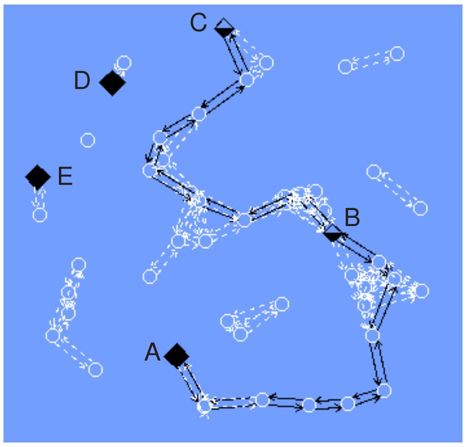
Figures 4, 5, and 6 represent the number of control packets that are generated during the simulations. The unicast simulations work as expected, with the number of RREQs and subsequent RREPs increasing as the rate of movement and link breakages increases. The 50m×50m multicast simulations work similarly, with the number of RREQs, RREPs and MACTs increasing with the speed of movement. There are more RREQs produced in the multicast simulations than in the unicast simulations due to the greater number of routes which must be maintained.

Because there are many small clusters of nodes in the 85m×85m multicast simulations, each of which may contain multicast group members, there are many different group leaders for the multicast group. Consequently, there are many more Group Hellos generated in these simulations than in the comparable 50m×50m multicast simulations, since the multicast group in the 50m×50m network was generally not partitioned. However, because network connectivity in these networks is low and there are numerous isolated network components, the overall impact of the increased number of Group Hellos is small. Like the other network scenarios, the 85m×85m multicast simulation shows an increase in the number of RREQs and RREPs as the speed of movement increases. However, after .6 m/s, these simulations show a slight decrease in the number of RREPs and MACTs generated. With the faster movement speeds and the small transmission radius relative to the room size, network partitions often only momentarily came into contact, and thus do not have enough time to reconnect two partitions of the same multicast tree. Since the trees do not have time to reconnect, one of the group leaders does not need to relinquish its group leader status. Hence the number of Group Hello messages produced begins to increase again for faster movement speeds.

Packet loss in the simulations is the result of either a collision, or a node transmitting a packet to a node that has been its next hop along the path, but this next hop has already moved out of transmission range from the sending node and hence does not receive the packet. AODV is able to find a route to the multicast group each time it is needed, and it is able to successfully maintain the links of the multicast tree for the lifetime of the group. The lifetime of the multicast group begins when



(a) The Partitioned Network before the Merge



(b) The Network after the Merge

Figure 7: Network Snapshot Before and After Multicast Tree Merge.

the first node requests to join the group and continues until the end of the simulation. If AODV were to be run over a MAC-sublayer protocol such as IEEE 802.11, data packets would rarely be dropped. However, AODV does not require such a protocol, because even without an underlying MAC-sublayer protocol, its performance is good.

One other result from the simulations to examine is the route acquisition latency. The route acquisition latency is the time between when a node discovers it needs a route to some destination, and the time that it acquires that route and can begin using it. Because a node wishing to join the multicast tree must always wait `route_discovery_timeout` before selecting its next hop and unicasting a MACT, that timeout will be a lower bound on the latency for acquiring a multicast route. For more details on the route acquisition latency for unicast routes, please see [18].

As an illustrated example of the merging operation, Figure 7 represents a snapshot of the nodes in the simulation immediately prior to and following a merge of two partitions of the multicast tree. In the figure, the solid diamonds are the group leaders, the partially filled smaller diamonds are multicast group members, and the unfilled circles are nodes in the network that are not group members. The dark solid lines represent links on the multicast tree, while the light dashed lines illustrate that the two nodes the lines connect are capable of communication. In Figure 7(a), nodes A, C, D, and E are group leaders for their partitions of the network. Node B is a member of node A's group. Figure 7(b) shows the partition containing node A and C after the merge. C has joined the tree and given up its group leader status, and A has remained the leader of the group. Notice that no new branches between nodes A and B had to be added to the tree.

6 Future Work

There are many areas of investigation that are relevant to AODV. To begin, we plan to continue our simulations of AODV, including the utilization of different channel models to determine how the protocol will function in a variety of environments. Though we do not feel that differing channel models will have a significant impact on AODV's relative performance and the results obtained from the simulations, we plan to complete simulations with differing channel models to verify these claims.

One of AODV's biggest sources of protocol overhead arises from the system-wide broadcasts that are used to disseminate RREQs. There are other protocols (notably CEDAR [20]) that establish a distinguished set of *cores* that are given the responsibility of managing the dissemination of such control messages. We believe that AODV could benefit from the integration of such mechanisms into its route discovery process.

On another front, it has been shown [4] that buffering can be used to enable *smooth handoff*, for instance in the context of Mobile IP [19]. This same idea of smooth handoff and buffering can be adapted to the context of AODV. When a link is broken in a routing path, subsequent re-establishment could be accompanied by delivery of some number of buffered packets.

Clearly, security is a major concern. Key distribution, authentication, and encryption in the ad-hoc networking context remain largely unsolved problems. We would like to specify an authentication procedure to avoid the disruption of valid routes by malicious nodes.

Reliable delivery of packets is another major concern. The current state of AODV does not provide for guaranteed delivery of data packets. However, AODV could be enhanced to provide this service. AODV's basic multicast algorithm elegantly lends itself to improvements already done for multicast in networks of stationary nodes, such as those described for Scalable Reliable Multicast (SRM) in [7].

There have been numerous proposals for scaling ad-hoc network protocols to greater node populations. One such proposal is gathering sub-populations into clusters and restricting the dispersal of route table information based on whether a desired destination is in a local or in a remote cluster. To the extent that such techniques are beneficial, we believe that they can equally well be adapted for use with AODV. As the tradeoffs for control traffic between cluster maintenance and route discovery become better understood, we will endeavor to incorporate clustering techniques into AODV. The gains in scalability will probably be even more favorable for multicast operations than for unicast, since our multicast algorithm places more reliance on network-wide broadcasts.

We have recently defined *Quality of Service* (QoS) extensions for AODV to enable route establishment between nodes that have certain well-defined traffic flow requirements. We would like to perform further simulations to verify our intuition that AODV will retain its high degree of efficiency and accuracy, even when the requirements for establishing valid routes are broadened to include QoS constraints.

Mobile IP [15] has been standardized within the IETF to enable seamless roaming for mobile nodes. However, Mobile IP assumes that a mobile node has been assigned a *home address* and that there is a *home agent* that can receive packets destined for the mobile node. Since there may not be any such home network in an ad-hoc network, it is not easy to see how Mobile IP can be applied. However, if just one of the ad-hoc network nodes has connectivity to the global Internet, it becomes possible for every mobile node in the ad-hoc network to achieve connectivity to the global Internet. Furthermore, any such mobile node can send a Mobile IP Registration Request to its home agent to describe its current *care-of address*, as described in [12]. We would like to augment AODV, and its recently proposed *transit networking* extension, to implement this type of Mobile IP connectivity and make it available to all AODV nodes. This would also allow AODV nodes to subscribe to Internet-based multicast groups.

7 Conclusion

We have presented a routing protocol for ad-hoc mobile networks that is capable of unicast, broadcast, *and* multicast communication. AODV has an advantage over other ad-hoc network routing protocols because it provides all three types of communication without being dependent on or requiring the use of any additional routing protocols. The main features of AODV are as follows:

- Unicast, Broadcast, and Multicast communication is provided.
- Routes are established on-demand with small delay.
- Multicast trees connecting group members are maintained for the lifetime of the multicast group.
- Link breakages in active routes are efficiently repaired or reestablished.
- All routes are loop-free through the use of destination sequence numbers.
- Inactive routes are quickly aged out because they are likely to go stale.

Through simulation, we have shown that AODV is able to obtain a high goodput ratio for both unicast and multicast communication. Additionally, it is able to offer this communication with a minimum of control packet overhead. AODV is an excellent choice for establishing communication within an ad-hoc network. It is suitable for a variety of applications, including conferencing, emergency search-and-rescue operations, and community-based networking. We look forward to continuing to enhance AODV by reducing the need for system-wide broadcasts, incorporating security and reliable delivery mechanisms, exploring QoS extensions, and implementing Mobile IP connectivity.

References

[1] R. Bagrodia and W. Liao. Maisie: A Language for Design of Efficient Discrete Event Simulation. *IEEE Transactions on Software Engineering*, April 1994.

[2] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. PARSEC: A Parallel Simulation Environment for Complex Systems. *IEEE Computer*, 31(10):77–85, October 1998.

[3] E. Bommaiah, A. McAuley, R. Talpade, and M.-K. Liu. AMRoute: Adhoc Multicast Routing Protocol. *IETF Internet Draft, draft-talpade-manet-amroute-00.txt*, August 1998. (Work in Progress).

[4] R. Caceres and V. N. Padmanabhan. Fast and Scalable Handoffs for Wireless Internetworks. *Proceedings of the 2^{nd} ACM International Conference on Mobile Computing and Networking*, pages 56–66, November 1996.

[5] M. S. Corson and A. Ephremides. A Distributed Routing Algorithm for Mobile Wireless Networks. *ACM/Baltzer Wireless Networks Journal*, 1(1):61–81, February 1995.

[6] H. Eriksson. MBONE: The Multicast Backbone. *Communications of the ACM*, 37(8):54–60, August 1994.

[7] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[8] M. Gerla, G. Pei, S.-J. Lee, and C.-C. Chiang. On-Demand Multicast Routing Protocol (ODMRP) for Ad-Hoc Networks. *IETF Internet Draft, draft-ietf-manet-odmrp-00.txt*, November 1998. (Work in Progress).

[9] L. Ji and M. S. Corson. A Lightweight Adaptive Multicast Algorithm. *Proceedings of IEEE GLOBECOM*, pages 1036–1042, Sydney, Australia, December 1998.

[10] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. Tomasz Imielinski and Hank Korth, editors, *Mobile Computing*, pages 153–181. Kluwer Academic Publishers, 1996.

[11] J. Jubin and J. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, 1987.

[12] H. Lei and C. E. Perkins. Ad Hoc Networking with Mobile IP. *Proceedings of the 2^{nd} European Personal Mobile Communications Conference*, pages 197–202, October 1997.

[13] S. Murthy and J. J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *Mobile Networks and Applications*, 1(2):183–197, October 1996.

[14] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. *Proceedings of IEEE Conference on Computer Communications*, pages 1405–1413, Kobe, Japan, April 1997.

[15] C. E. Perkins. IP Mobility Support. *RFC 2002*, October 1996.

[16] C. E. Perkins and P. Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. *SIGCOMM '94: Computer Communications Review*, 24(4):234–244, October 1994.

[17] C. E. Perkins and E. M. Royer. Ad Hoc On Demand Distance Vector (AODV) Routing. *IETF Internet Draft, draft-ietf-manet-aodv-02.txt*, November 1998. (Work in Progress).

[18] C. E. Perkins and E. M. Royer. Ad-hoc On-Demand Distance Vector Routing. *Proceedings of the* 2^{nd} *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, New Orleans, LA, February 1999.

[19] C. E. Perkins and K.-Y. Wang. Optimized Smooth Handoffs in Mobile IP. *Proceedings of the IEEE Symposium on Computers and Communications*, Red Sea, Egypt, July 1999.

[20] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm. *Proceedings of IEEE INFOCOM*, pages 202–209, New York, NY, March 1999.

[21] A. S. Tanenbaum. *Computer Networks, Third Edition*, chapter 4, pages 263–264. Prentice Hall, Englewood Cliffs, 1996.

[22] C. W. Wu, Y. C. Tay, and C.-K. Toh. Ad hoc Multicast Routing Protocol Utilizing Increasing Id-numbers (AMRIS) Functional Specification. *IETF Internet Draft, draft-ietf-manet-amris-spec-00.txt*, November 1998. (Work in Progress).