Differential Destination Multicast–A MANET Multicast Routing Protocol for Small Groups

Lusheng Ji and M. Scott Corson
Institute for System Research
University of Maryland
College Park, MD 20742
e-mail: lji, corson@isr.umd.edu

Abstract—**In this paper we propose a multicast routing protocol for mobile ad hoc networks (MANETs). The protocol—termed Differential Destination Multicast (DDM)—differs from common approaches proposed for MANET multicast routing in two ways. Firstly, instead of distributing membership control throughout the network, DDM concentrates this authority at the data sources (i.e. senders) thereby giving sources knowledge of group membership. Secondly, differentially-encoded, variable-length destination headers are inserted in data packets which are used in combination with unicast routing tables to forward multicast packets towards multicast receivers. Instead of requiring that multicast forwarding state to be stored in all participating nodes, this approach also provides the option of stateless multicasting. Each node independently has the choice of caching forwarding state or having its upstream neighbor to insert this state into self-routed data packets, or some combination thereof. The protocol is best suited for use with small multicast groups operating in dynamic networks of any size.**

Keywords—**MANET, multicast routing, small group multicast**

I. Introduction

There are generally two mainstream approaches used for multicast routing in fixed networks: Group Shared Tree (GST) and Source-Specific Tree (SST). Both construct data forwarding paths interconnecting all group members, where a member is understood to be a multicast *receiver*. Data is firstly forwarded to the tree then along the tree paths to reach all group members. Several protocols include data sources as part of the forwarding tree as well. The GST approach builds one tree for the whole group regardless of where the data sources are located. The SST approach organizes multicast forwarding by data source. A "session" is associate with each source and is identified by the combination of group ID and source ID. For each session there is one distribution tree formed from the union of all shortest paths between the source and the group members. This form of multicasting usually results in lower end-to-end delay since the data is forwarded using the shortest source-receiver paths, but typically consumes a greater amount of network bandwidth than shared-tree approaches.

A characteristic shared among these traditional multicast protocols is that the multicast computation is *distributed* in the network. Not only are the forwarding states constructed and maintained by the network, group membership control (or lack thereof) is also distributed over the network. While this approach improves scalability with respect to group size, it has certain drawbacks. Firstly, distributed membership management may make aspects of security more difficult due to the lack of admission control. Without natural support for end-to-end signalling between sources and receivers, such approach needs to rely on external mechanisms for security management. At the same time, billing management becomes more complicated as the information property owner, typically the source, has no control and knowledge over how and to whom its property (data) is distributed. Secondly, distributed per group forwarding state maintenance may result in large router resource usage. The number of possible multicast groups formed among n members grows combinatorially. While there is no efficient way to aggregate multicast routing table entries, the linearly growing multicast routing table (with respect to the number of active multicast groups) may quickly become too much of a storage burden for routers. The issue is worsen by the number of routers involved in forwarding since all routers along the multicast forwarding paths need to participate in multicast routing state maintenance.

To address this issue, recently there is an attempt of shifting towards stateless multicast routing for small groups. [1] and [2] lift multicasting out of routing layer so that multicasting no longer requires router support. Multicast data is encapsulated in unicast envelop and transmitted between end receivers. Therefore no multicast routing state is installed on routers. [3] and [4] propose connectionless, small group multicast as a "stateless" approach to multicast routing. In these approaches, variable-length destination lists are placed in packet headers that are self-routed towards the destinations using the underlying unicast forwarding tables.

We consider the problem of multicast routing in Mobile Ad hoc Networks (MANETs). In MANETs all mobile nodes are equipped with wireless communication interfaces and can move at will. Here we assume their communications occur using omni-directional antennas over broadcast media. The combination of node mobility and a wireless environment can result in MANET topologies subject to rapid and unpredictable changes. Because of these dynamics, and the fact that communication is carried over a bandwidth-constrained broadcast media, the multicast routing problem in MANETs differs from that in fixed networks.

During recent years several multicast protocols have also been designed specifically for MANETs (e.g. CAMP [5], ODMRP [6], MAODV [7] and LAM [8]). These protocols all follow the traditional multicast approaches, i.e. distributed group membership management and distributed multicast routing state maintenance. In addition to the security and resource use issues mentioned before, these approaches, especially when applied for use with small and sparsely distributed (potentially numerous) groups, may become even less efficient and more expensive to function in MANETs due to bandwidth constraints,

network topology dynamics, and high channel access cost.

II. Protocol Description

A. Overview of Proposed Approach

Aiming at the issues briefed in the previous section, we propose the Differential Destination Multicast (DDM) protocol. This approach is motivated, in part, by the approach to unicast routing of Dynamic Source Routing (DSR) [9] and derived, in part, from the work of [3] and [4].

In DDM, the sources control multicast group membership to ease certain aspects of security administration. More importantly, and a departure from other proposed MANET multicast protocols, DDM encodes the destinations (i.e. the multicast group members to whom the data needs to be delivered) in each data packet header in a fashion different from [3] and [4]. This "in-band" information can be used to establish soft-state routing entries if desired. Using such an approach has two advantages for MANETs. Firstly, there is no control overhead expended when the group is idle; a characteristic shared with DSR. Since many multicast applications do not have continuous traffic flows, it can be expensive in terms of network control overhead to maintain multicast forwarding state in routers during idle periods. In-band control avoids this problem because if there is no data traffic, there is no need for any control information either. Secondly, it is not necessary for the nodes along the data forwarding paths to maintain multicast forwarding state if it chooses to run under stateless mode. When one intermediate node receives a DDM data packet, it only needs to look at the DDM header to decide how to forward the packet; another similarity to DSR. Assuming that routers can handle this processing cost, this stateless mode can be very reactive and efficient. This stateless approach also avoids loading the network with pure signaling traffic; a third trait shared with DSR. In so doing, the hope is that the unicast algorithm can converge much *faster*, with DDM then making immediate use of new unicast routing knowledge.

While the fixed networks and MANETs can both benefit from stateless, explicit multicasting because of its savings in storage complexity, it is even more desirable to follow this approach in MANETs due to the special characteristics of the MANETs. Firstly, in wired Internet, network topology change rarely occurs. Therefore if group membership is stable, after a multicast tree is constructed, the maintenance effort is small and tree links seldom need repair. On the other hand, MANET topology is subject to constant and sometimes dramatic changes. Multicast forwarding topology built in MANETs needs constant repair even rebuild. Maintaining multicast routing state is a much more expensive operation in MANETs than in wired networks. This is even worsen by the tight bandwidth constrains of MANETs. Secondly, the cost of medium access is high in wireless broadcast networks due to the MAC mechanism and broadcast transmission's blocking effect to neighboring nodes. Although packing routing information together with data traffic will enlarge data packet size, it reduces the total number of channel accesses because it reduces the number of pure control packets generated by the protocol. Therefore such approach can be more efficient overall in many scenarios. Thirdly in broadcast networks, when a node needs to send to more than one neighbor, it only needs to broadcast the packet once. So this approach has better bandwidth consumption and channel access properties in broadcast networks than in point-to-point networks. Lastly, the performance bottleneck for Internet routers is typically at the forwarding processing. The relatively complex processing of the destination encoded headers prevents fast path forwarding at Internet routers. On the other hand presently in MANETs, the effect of per-packet processing on forwarding rate is less significant relative to bandwidth, medium access, and energy constraints. Also since mobile computing devices are typically less resourceful comparing to their stationary and wired counterparts due to weight and physical dimensional limits, saving storage resource use becomes more important in MANETs.

In scenarios where the stateless approach is not favorable, DDM may also operate in a "soft-state" mode. It is here that DDM markedly departs from the work of [3] and [4]. In this mode, as data packets with in-band control information are routed through the network, each node along the forwarding path remembers the destinations to which it forwarded the last time and how the data was forwarded (i.e. which next hop was used for each destination). By caching this information, the protocol no longer needs to list all the destinations in *every* data packet header. When changes occur in the underlying unicast routing or destination list, an upstream node only needs to inform its downstream neighbors (i.e. its next hops) regarding the *differences* in destination forwarding since the last packet; hence, the name "Differential Destination" Multicast. Reporting only these differences significantly reduces DDM header sizes. Ideally, in a stable network where the topology and membership remain unchanged, only the first data packet needs to contain destination addresses and all subsequent packets would contain no destination information. In practice, the state kept at each node along the forwarding paths is "soft". Each time data forwarding occurs, this state is refreshed. Stale state eventually times-out and is removed. This mode is better suited for applications which generates small data packets at relatively high rate.

DDM is not a general purpose multicast protocol in the conventional sense. The header-encoded destination mechanism does not scale well with group size. The stateless mode, however, does scale well with the number of multicast groups, as no per-group state is required in any routers. In MANETs, if the number of multicast groups is small enough to permit state storage, then the soft-state version using differentially-encoded header processing can be used to reduce average packet size and save bandwidth. This approach, however, has essentially little applicability to fixed networks as the complexity of the differentially-encoded header processing would significantly slow down forwarding rates in high-speed networks.

B. Membership Management

In contrast with traditional multicast algorithms, a multicast data source plays an important role in the DDM protocol. The protocol proceeds independently for each source sending to a multicast group (a session), and the remaining description applies to a single session. The source acts as an admission controller for the information itself is sending. When a node (the

"joiner") is interested in a particular multicast session, it needs to join the session by unicasting a JOIN message to the source for that session. This JOIN message only needs to include the ID of the group to which the node wants to join. Other useful fields, namely the "joiner" ID and the source ID, are already included in the unicast packet as the source and destination fields of the IP header. Then it is up to the source to decide if the JOIN can be accepted. The admission policies are beyond the scope of this paper.

Upon receiving a JOIN message, if the joiner passes the session's admission requirements, the source adds the joiner into its member list (ML) and the joiner becomes a receiver of the session (also referred as a destination of multicasting in following context). The source then acknowledges the JOIN message by unicasting an ACK message to the joiner.

Back at the joiner, after the JOIN messages are sent, the joiner waits for one JOIN_WAITING_PERIOD. If it has not received any ACK till the end of this period, the joiner needs to resend the JOIN message. When sending the JOIN messages, the waiting period is exponentially backed off for every consequent JOIN sent. The sending of JOIN messages is stopped when either an ACK message is received (a success) or MAX_JOIN_RETRY JOINs have been sent and the waiting period for the last JOIN message has passed (a failure). Reception of a DDM data packet intended for this joiner prior to reception of an ACK may or may not indicate a successful join, depending on the security mechanism (if any) in use and whether or not an ACK is required as part of this mechanism. Security-related mechanisms are beyond the scope of this paper. This paper essentially describes an unsecured session. When join process is successfully completed on ACK reception, any activate join waiting timer is canceled and no further JOIN messages are generated.

The ML kept at the source node needs to be refreshed from time to time to maintain up-to-date membership information. Due to the dynamic nature of wireless networks, node reachability may change unpredictably. When operating in hostile environment such as battle field, nodes also may be destroyed before any graceful exit procedure can be executed. Thus the source needs to be able to purge stale members. In DDM membership refreshing is source-initiated. Once every MEMBERSHIP_REFRESH_PERIOD data packets, the source sets a POLL flag in the next outgoing data packet. Upon receiving such data packet, a multicast session member needs to unicast a JOIN message again to the source to express its continued interest. If after MAX_REFRESH_TIMEOUT such polling data packets have been sent and there is still no JOIN message received from a particular member, the source assumes that this member has left the multicast session. This member is then removed from the ML and excluded from future forwarding computations. JOIN "implosion" at the sender is not expected to be a problem due to small expected group sizes. If necessary, random delay jitter may be added to the transmission times of JOINs sent in response to POLL packets to reduce congestive effects at the source.

This "polled" membership refreshment is used as a secondary mechanism to detect an absent member. An explicit LEAVE message is defined as the preferred way for a session member to leave the source's ML. It is a unicast message sent from the leaving member to the session source. When received by the source, this LEAVE message terminates the member's membership. The member is removed from the ML and is excluded from future forwarding computations. To increase robustness, instead of sending just one message, MAX_LEAVE_RETRY LEAVE messages are unicast to the source when a node leaves the session. After these transmissions, a member node removes all information associated with the multicast session. The source may also dismiss a receiver by removing it from the ML at any time if membership control policy suggests so.

C. Forwarding Computation

The key notion of DDM is forwarding computation based on destinations (i.e. multicast receiver addresses) encoded headers, may or may not be differentially-encoded depending on in which mode a node operates. In this section, we describe the operation under soft-state mode. The operation for stateless mode is more straight forward.

C.1 Packet Formats

DDM employs two types of packets: control packets and data packets, where it is understood that data packets may also contain control information. There are four types of control packets: JOIN, ACK, LEAVE, and RSYNC. The first three are used only by the membership control part of the algorithm. Each of these control packets contains one field for the packet type and one for the group address. Since they are unicast packets, the source and destination of the packets are already included in the packet's IP header. They are used in admission processing as well. The fourth packet type, RSYNC, is used by a node to request its upstream neighbor to re-synchronize the stored destination lists on both nodes. The detailed used of RSYNC message will be explained later in section II-C.4 and II-C.5. This is a unicast message with TTL of 1. It contains a three-state flag in addition to the message type field.

Multicast data packets contain a payload and a DDM header. Each DDM header is composed of a *summary section*, and at least one *DDM block*. The summary section contains fields such as *membership refresh request flag*, *DDM header aggregation length*, and the sender's address, etc. Each *DDM block* is constructed for a particular downstream neighbor. Each *DDM block* contains the *intended receiver*, the *DDM block type*, *DDM block sequence number* and some other fields depending on the type. There are three types of the *DDM blocks*: Empty (E) blocks, Refresh (R) blocks and Difference (D) blocks. A D block can be either an incremental block (D_i), or decremental block (D_d). Oftentimes only one *DDM block* is needed to describe the change in destination list for a downstream neighbor. However, when both D_d and D_i blocks are needed to describe a difference, both blocks are constructed and put into the header in a way that is consistent to all nodes (i.e. always put the D_d block immediately ahead of the D_i block).

The E block does not need any other field. It is followed directly by the payload. The R block has a destination list L. There is an incremental list (L_i) in each D_i block to tell the recipient that it also needs to forward data to the destinations in this list in addition to whom it is already forwarding. The decremental list (L_d) in a D_d block informs the recipient that it no longer needs to forward to the destinations in this list. In both R

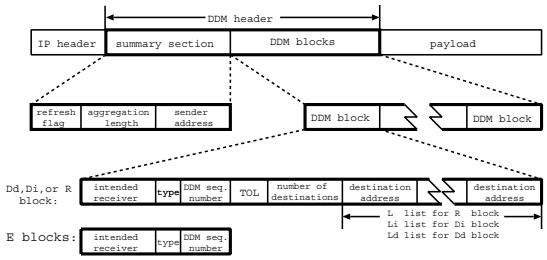

Fig. 1. DDM Header Format Example

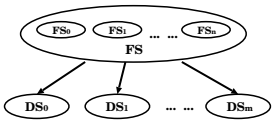

Fig. 2. Forwarding Computation Data Structure

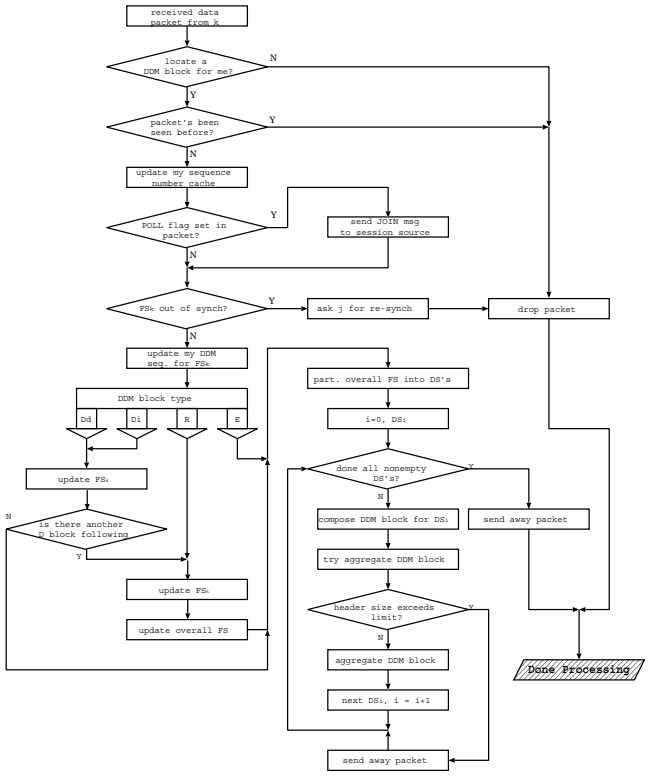

Fig. 3. Forwarding Computation

and D blocks, there is a *Time Of Life* (TOL) value. Both D and R blocks may be referred to as "informative" blocks in following text since they contain forwarding list update information.

When used in broadcast media networks, *DDM blocks* for different downstream neighbors may be aggregated together to reduce the number of transmissions. When this feature is in use, all aggregated DDM headers share the same *summary section* and their *DDM blocks* are concatenated together. The *DDM header aggregation length* of the *summary section* indicates the number of *DDM blocks* being so aggregated. Each *DDM block* is tagged by the intended receiver for the block so that when this packet is received, the receiver can locate the correct *DDM block* and the address list for itself. Figure 1 offers an example of an aggregated DDM data packet header. More detailed specification for packet format as well as other parameters can be found in [10].

C.2 Data Structures

The data structures involved in the forwarding computation is shown in Figure 2. They are used by a node to remember to whom and how it forwards multicast data. Such data structure is only needed for nodes operating in "soft-state" mode. At each node, there is one Forwarding Set (FS) for each active multicast session. It records to which destinations this node needs to forward multicast data. At the source node, the FS is the same as the ML. At other nodes, this FS is actually the union of several smaller sets. Each small FS_k set records the destinations included in data packets received from upstream neighbor k. After receiving a data packet from a neighbor k, the receiving node will update the corresponding subset FS_k and then the unioned overall set FS where

$$FS = \cup_k FS_k, \quad \text{for all upstream neighbors } k.$$

Associated with each set FS_k, there is one sequence number (SEQ_k^{FS}). This is used to record the last *DDM block sequence number* seen in data packet received from upstream neighbor k. The reason for this sequence number is to detect the loss of DDM data packets containing forwarding set updates. For each multicast session, there is a cache for storing recently seen data sequence numbers. Such cache is used to prevent data packet looping and duplication.

The overall FS set contains all the destinations to which this node needs to forward. However, these destinations may be reached via different paths (i.e. next hops). Therefore, the FS needs to be partitioned into subsets according to the next hops. The destinations in the FS who use the same downstream neighbor (next hop) will be put into the same subset. These subsets resulted from partitioning the FS are called Direction Sets (a DS_l exists for each downstream neighbor l). For each DS_l there is again a sequence number SEQ_l^{DS}. It is initialized to 0 when the DS_l is created for the first time. Each DS_l also contains a "forced refreshing" flag. This flag has three states: "no forcing", "forcing once" and "forcing always". The use of this flag will be explained in sections II-C.4 and II-C.5.

For a pair of tree neighbors u and d, the DS_d set on the upstream node u should contain the same list of destinations as the FS_u set on the downstream neighbor d.

C.3 Processing

Since MANET nodes commonly use broadcast, here we assume that DDM header aggregation is in use. Most of the processing consists of set operations. The computation complexity on each participating node is thus bounded by $O(n \log n)$, with n being the number of members in the multicast session. The forwarding computation can be found in the flow chart in Figure 3.

When a node receives a DDM data packet from an upstream neighbor k, it first tries to locate the *DDM block* intended for itself by looking at the *intended receiver* field of each enclosed

DDM block. If no such block can be found, the data packet is dropped and the forwarding processing stops. After finding the *DDM block*, the receiving node compares the sequence number of the data packet (the identification and fragmentation offest fields of IP header can be used for this purpose) with the contents of local sequence number cache for this multicast session. If this packet has been seen before, it is discarded and no further processing is needed.

Then the receiving node accesses the FS_k set. If no such set is found and the received *DDM block* is a R block, a new FS_k is created along with its associated data objects. The newly created set is initialized to empty. The SEQ_k^{FS} number is initialized to be 1 lower than the *DDM block sequence number* in the received *DDM block*. However, if the block is E or D-type, this means the FS_k set is out of synchronization with the Direction Set on the sender k. In this case, the receiving node sends a RSYNC packet back to the upstream neighbor k.

Before a node proceeds further, it compares its SEQ_k^{FS} value with the *DDM block sequence number* in the received *DDM block*. The reason for this step is to detect DDM update losses. Because of the differential approach, it is very important to quickly detect the loss of any packet which contains destination list updates. When a node sends out a *DDM block*, it stamps the block with a *DDM block sequence number*. This sequence number is incremented by one every time the node sends out an informative *DDM block*. The sending of E blocks does not increment the sequence number since there is no change in the destination list. Still, an E block carries the current sequence number so a receiving node may detect previous losses.

If the received *DDM block*'s *DDM block sequence number* is at least 2 greater than the SEQ_k^{FS} and the block is an E or D type, the receiving node knows that at least one packet was lost and that packet(s) contains destination list updates intended for its viewing. So the receiving node sends a RSYNC packet back to the upstream neighbor, and the data packet is dropped. This check is unnecessary for R blocks since they always contain the entire destination list.

After verifying the *DDM block sequence number*, the SEQ_k^{FS} value is updated to the sequence number in the received *DDM block*. If the received *DDM block* is an E block, there is no change since the last data forwarding. Therefore the states left by last forwarding computation can be reused: the receiving node only needs to forward one copy for the data packet to each downstream neighbor l whose DS_l is not empty.

If the block is a R block, the node replaces its FS_k by the list L in the block.

$$FS_k = L$$

When a node receives a D_i or D_d block, it updates the corresponding FS_k according to the D block: removing the addresses in the L_d from its FS_k and adding the addresses in the L_i into its FS_k set. Also the receiving node needs to look one block beyond in the DDM header to see if the next block is also a D block for itself. If so, the L_i or L_d list in that D block needs to be processed together.

$$FS_k = (FS_k - L_d) \cup L_i$$

After updating the FS_k, the receiving node updates the union FS. Then it partitions the new overall FS set into DS'_l sets according to the "next hop" l used by the unicast routes towards the destinations in the FS. All destinations in the same DS'_l share a common "next hop" l.

By comparing the contents of the new DS'_l set and the existing DS_l set (result of the forwarding of the last data packet) for the same next hop l , the node assembles new *DDM blocks* for next hop l for the outgoing DDM header. If there is one DS'_l but there is no matching DS_l, a R-type *DDM block* is constructed. The sequence number for this *DDM block* is set to 0. Alternatively, if there is one DS_l set but there is no DS'_l, no *DDM block* is constructed for l as l is no longer used as downstream next hop.

For the case that there are both DS'_l and DS_l, the processing checks if the DS_l's "forced refreshing" flag is set to either "forcing once" or "forcing always". If so, a R block is constructed which contains all destination addresses in the DS'_l. Otherwise the contents of them need to be compared. If the DS'_l set is the same as the DS_l set, an E block is used since there is no change. Otherwise, R or D block(s) is constructed. By default D block(s) is tried first. All destination addresses that are in the DS'_l (new Direction Set) but not in the DS_l (old Direction Set) form the *incremental list* L_i. All addresses that are in the DS_l but no longer in the DS'_l form the *decremental list* L_d. Addresses that are common in both sets appear in neither list. If the total length of the L_i and L_d lists is not shorter than the destination list in DS'_l, a R block is used instead to shorten the overall list length. For D or R-type *DDM blocks*, the SEQ_l^{DS} is incremented by one while it remains the same for E blocks. The final SEQ_l^{DS} is used as the block sequence number in the constructed *DDM block*.

$$R \;\; header : L = DS'_l$$

or

$$D \;\; header : \begin{cases} L_i = DS'_l - DS_l \\ L_d = DS_l - DS'_l \end{cases}$$

After the *DDM block* is ready it is packed into the header of the data packet. The "forced refreshing" flag is reset to "no forcing" if its current value is "forcing once". The contents of the DS_l set are replaced by the DS'_l set and kept in memory for the next forwarding computation. DS'_l is self is discarded after forwarding.

If DDM header aggregation is not in use, the data packet can be forwarded to the downstream neighbor l right away. Otherwise, more *DDM blocks* will be packed into the same data packet header until it becomes full. The above *DDM block* construction is repeated until all DS' sets are processed. New *DDM blocks* are inserted into the packet header if the header size does not exceed the limit. Otherwise, the filled data packet is sent out and a new data packet with the same payload is allocated. New *DDM blocks* are then inserted into the header of this new data packet.

C.4 Forwarding Set Synchronization

When operating in soft-state mode, usually only the *differences* of the destination lists are included in data packet headers. Therefore it is very important to keep the *Direction Set* on

the upstream side and the *Forwarding Set* on the downstream side synchronized. The DDM algorithm uses sequence number to maintain synchronization. For a pair of neighbors, the corresponding sequence number SEQ_u^{FS} on a downstream node d should match with the SEQ_d^{DS} on its upstream neighbor u.

This sequence number is carried inside of each *DDM block* and its advertisement is *data-driven*. Only nodes which are *not* exchanging data may remain out of synchronization for extended periods. If a receiver d detects missing sequence numbers, it knows that some data messages containing informative *DDM blocks* intended for itself have been lost and its FS_u set is out of synchronization with the DS_d set on the upstream neighbor u. It needs to notify the upstream neighbor u using a RSYNC (request to synchronization) message. When used for resynchronization purpose, this message contains a flag telling the upstream neighbor to set the "forced refreshing" flag for the DS_d set to "forcing once". The message is unicast to the upstream neighbor.

Node u, upon receiving a RSYNC message, locates the DS_d set used for the sender of this RSYNC message (d) and sets the "forced refreshing" flag to what the message says, in this case "forcing once". When u is forwarding the next data packet, it will see the "forced refreshing" flag at the DS_d set and the *DDM block* construction algorithm described previously will produce a R block. Since R block contains the full list, upon reception the downstream neighbor d will have its FS_u set synchronized with the contents of the DS_d set on node u again.

C.5 Timers and Dual Modes of DDM

DDM is a dual-mode algorithm where each participating node can operate in either "stateless" mode or "soft-state" mode. Timers, the "forced refreshing" flag, and the TOL field in each *DDM block* are the mode control parameters.

There are timers associated with each set (both FS_k sets and DS_l sets). When any of the timers expires, the corresponding set is removed. Every time there is a packet received from neighbor k, the timer associated with the FS_k is reset to expire in TOL seconds as specified in the received *DDM block*.

Every time a DS_l set is used for forwarding, the forwarding node picks an expiration period for the timer associated with this DS_l. Then in the outgoing *DDM block*, its TOL field is set to:

$$TOL = DS_l \;\; life\ timer\ value * R \quad R > 1$$

This action helps ensuring that, over a given link, the timer for the *Direction Set* on the upstream side expires *before* the timer for the *Forwarding Set* on the downstream side. Otherwise if the DS set lives *longer* than the downstream's FS set, the upstream neighbor may use an E or a D block and the downstream node will have to start the resynchronization process. Although the timer setting scheme does not completely avoid this from happening (packet loss may still cause the same problem), it should sufficiently reduce the probability of the occurrence of such undesired event. Also, by making the timer value for the DS sets a local decision, nodes are given the opportunity to adapt the timer values to their local environment. For example, if a node moves rapidly relative to its surrounding nodes, it is reasonable to choose a smaller timer value.

The preceding applies to "soft-state" operation. If "stateless" forwarding is desired at a node, it may need to inform its upstream neighbor about its decision. If the upstream neighbor is also operating in "stateless" mode, there will be no need for the notification since all *DDM blocks* coming from it will be R-type. If the upstream neighbor is in "soft-state" mode, indicated by the reception of an E or D block from it, the "stateless" node needs to send back a RSYNC message with the "forced refreshing" flag set to "forcing always". After receiving such RSYNC message the upstream neighbor will only send R blocks from the next data packet onwards since the DS set corresponding to this "stateless" downstream neighbor has a "forcing always" flag. This flag will not be reset after each outgoing *DDM block* is constructed. In a "stateless" node, all FS and DS set timers are set to zero so that none of the sets are kept. Later on, if a "stateless" node ever wants to switch back to soft-state again, it only needs to send another RSYNC message to its upstream neighbor to cause the latter to reset its "forced refreshing" flag to "forcing once".

Using a combination of RSYNC message, TOL, and "forced refreshing" flag, neighboring nodes can operate in different modes but still work together.

D. Route Correctness Discussion

DDM is loop-free as long as the underlying unicast routing is loop-free. Since the DDM routing (FS to DS's partition) calculation is carried out for every data packet, only the most recent unicast routing information is used. Transient loops are still possible during unicast routing convergence period, but will disappear as the unicast protocol recovers. During this period, some data packets may have been errantly sent using erroneous route information. In this case, these data packets will either be dropped when the IP TTL reaches zero, or will exit the loop and head towards the destination when some forwarding node finally corrects the loop.

Broken routes is also handled by unicast routing. It is not necessary for DDM to learn about link status. The old DS and the old FS sets on two ends of a broken link are left alone to be deleted on time-out since no data packet is forwarded using them. If there is alternative route available, or as soon as the unicast routing finds one, DDM will use the next hop of the new route when forwarding the next multicast data packet. When there is new link to be added, DDM does not react either. In fact, DDM is not aware of the change if this new link is not used for reaching DDM destinations. Only after the unicast routing protocol designates this link as a next hop for some DDM destination will DDM utilize the new link. DDM will then set up a new DS for this new link. By relieving DDM from multicast forward path monitoring and maintenance, the only pure control traffic DDM generates during data forwarding is the RSYNC packet.

Data packets may be lost during transmission. If the lost packet is an E packet, it is simply a data loss. If the packet is a D or R packet, the loss may de-synchronize the FS sets on the receiving ends from the matching DS sets on the sending ends. However, since all packets contain *DDM block sequence number*, the receiving end of the lossy link will discover the loss of an informative *DDM block* when receiving the next data

packet. A RSYNC is then sent back asking for a R block to re-synchronize the FS sets and the DS sets.

III. Performance Evaluation

A. Simulation Environment

The simulations are implemented using the NS-2 network simulation package [11]. The simulation environment models a MANET of 50 mobile nodes. At the beginning of the simulation, nodes are randomly placed in an 1000m by 1000m area. Node movement uses the random way-point model of [12] with no pausing. When the simulation starts, each node randomly picks a destination and moves towards it with a random constant speed. After a node reaches its destination, it selects a new destination and starts to move towards it at a newly selected speed. Each simulation is executed for 900 seconds. There is no network partition during the course of simulation.

The network stack of each mobile node consists of a link layer, an ARP module, an interface priority queue, a MAC layer, and a network interface. IEEE 802.11 is used as the MAC protocol. The radio transmission range is approximately 250 meters. Each network interface transmits data at a rate of 2 Mbits/sec.

In all simulation runs, CBR traffic flows are injected into the network from source nodes. The size of data payload is 512 bytes. Data packets are generated at each source at a rate of 4 packets per second. Group members and sources are randomly selected among all 50 nodes. Source nodes may or may not be group members themselves. To reduce side effects, membership control features are turned off. All group members join the multicast group at the beginning of the simulation and remain members till the end of simulation.

B. Protocols

The performance of the DDM protocol is studied in two flavors. The basic version of DDM runs above an omniscient unicast routing algorithm which always knows the full topology of the network and computes shortest path to destination. The shortest path computation is purely based on network topology. This decouples DDM from the characteristics of any given unicast algorithm, and allows us to focus on its own behavior under admittedly ideal circumstances. The performance obtained in this case is probably an upper bound on that achievable with DDM. Then DDM is also run above the MANET unicast routing protocol AODV (Ad Hoc On-Demand Distance Vector) [13]. This is done to assess the effectiveness of DDM over AODV, and to see the side effects of using DDM over an on-demand protocol.

The AODV routing agent is already provided in the NS-2 package. When working with an on-demand unicast routing protocol such as the AODV, DDM needs to be modified slightly to work with the "reactivity" of the on-demand protocol. When DDM needs a route towards one particular destination and the unicast routing does not have a route, DDM encapsulates the data packet in a unicast packet for that destination and passes it to the unicast routing algorithm for forwarding. Since there is now a "demand" for a route, the unicast algorithm will start to build a route and eventually this data packet is delivered to the waiting group member. Route information for the same destination is also ready for future DDM queries.

Other than DDM, we also implemented simple flooding and the On-Demand Multicast Routing Protocol (ODMRP) [6] for comparison. The ODMRP multicast routing agent implements the specification of [6] without mobility prediction. In simulation, ODMRP protocol parameters are set to the values used in [14], which is produced by the authors of ODMRP. In addition, the Forwarding Group timeout value is set to 4 times of the route refresh interval, as suggested by [6].

C. Parameters

We alter the simulations by varying three parameters: the maximum node speed, the number of members in a group, and the number of data sources for the group. We only simulate one multicast group. The first variable represents different levels of node mobility, and thus the relative amounts of network topology dynamics. Each node's speed is set to a value evenly distributed between 0 and the maximum node speed. The second parameter controls multicast group size. It translates into two factors, membership density and overall network traffic level, which both affect the simulation at the same time. The last parameter controls traffic load. When the number of sources varies, the total amount of data traffic circulating within the network also changes.

The simulation runs under different combinations of parameter values. The standard case is for a multicast group with 2 sources and 10 members. All nodes are moving at a random speed between 0 and 2 *m/s*. When we are studying the effect of one particular parameter, it is set to different values while the others are fixed at the standard case. The standard case has relatively low traffic load and low mobility. This way we can focus on the performance changes resulted from the change of the parameter of interest. For the effect of node mobility on protocol performance, the simulated range of maximum node speed is between 1 m/s and 20 m/s. For group sizes, simulations are run for groups with from 2 to 50 members. For the impact of number of sources, we use from 1 to 10 sources. For each simulation data point, results from multiple runs are averaged. These runs are for the combinations of 3 different random node placement/movement patterns and 3 different group member distributions.

D. Simulation Results

D.1 Data Packet Delivery Ratio

The data packet delivery ratio is defined as the ratio of the number of data packets actually received by group members to the number of data packets that should have been be received. Since the membership is static during the entire simulation (and since there are no partitions), the number of data packets that should be received equals the product of the number of members and total number of data packets generated by all sources.

From Figure 4(a) we can see that the throughput of ODMRP (which does not require any unicast routing support) is impressive across the whole mobility spectrum. So too is the delivery ratio of flooding. They are doing well because they use redundant transmissions for data delivery. ODMRP uses mesh for-

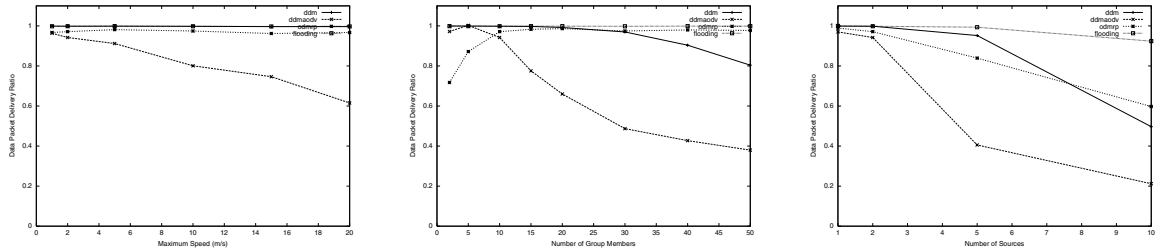

Fig. 4. Data Packet Delivery Ratio vs. (a) Max. Node Speed, (b) Group Size, (c) Number of Sources

ward topology instead of traditional tree topology. Data packets are flooded within the mesh called "forwarding group" (FG). A simplified view of which is that the FG is constructed as the union of all Source Specific Trees for the multicast group. Given a group with 10 members and 2 sources, the FG can contain enough nodes to provide the redundancy needed for high delivery level. The basic DDM also offers good throughput at all mobility levels. Since it runs over near perfect routing information, this result is not surprising. In addition, since the network traffic level is low, there is little data packet collision. However, the throughput of DDM over AODV decreases as the node mobility level increases. This is mainly due to inaccurate route information AODV provides. When nodes move faster, the topology changes faster. It becomes harder for unicast routing protocol to maintain up to date route information.

When the group size changes, different protocols react differently. Flooding does well for all group sizes. ODMRP offers high data delivery ratio when the multicast group is more populated. When the group is small, the FG constructed by the protocol contains a small amount of nodes and thus is relatively fragile. The FG is easier to be broken by node movements. When the group gets larger and larger, the FG size increases and thus becomes more tolerant to node movements, – if one path is broken, the FG may very well contain other backup paths. Both DDM variations behave the opposite from ODMRP: they provide high data delivery ratio when the group is small but the performance degrades when the group gets larger. The basic DDM's data delivery ratio drops for large groups mainly due to high membership density. When membership density is high, data packet collision is more likely to occur since more nodes need to participant in data forwarding. At the same time, DDM block aggregation is more likely to be used since the number of downstream neighbors is increased. DDM block aggregated data packets are sent in multicast envelops. 802.11 sends them without the RTS/CTS handshake and thus they become more vulnerable to collisions. In addition such congestion related packet loss happens more frequently near sources. A data packet dropped near source has more negative impact on delivery ratio than a packet dropped father away. All these factors reduce the DDM's delivery ratio for larger multicast group. The DDM over AODV exhibits a faster performance drop due to the additional fact that when there are more members, the AODV simply needs to maintain more routes. The high data traffic level resulted from high membership density also has negative impact on AODV route accuracy since many AODV control packets are broadcast packet. They are as vulnerable to collisions as the data packets.

Adding the number of sources increases network traffic level even faster. Very soon, the whole network is badly congested and all protocols suffer. Flooding is the least sensitive to network load as it's forwarding topology is the most redundant. And it does not need any control messages to maintain this forwarding topology. ODMRP is not as good as flooding due to two reasons. Firstly its forwarding topology is smaller than flooding's. Secondly ODMRP needs to exchange control message to construct and maintain its forwarding topology. These control messages can become victims of collisions too when the network is congested. When traffic level is very high, such control message loss costs the drop of ODMRP delivery ratio. The basic DDM does not need control message other than RSYNC. However, as a tree forwarding protocol, DDM does not provide redundant forwarding path. Thus data packet losses have more serious impact on delivery ratio than packet losses in ODMRP. For DDM over AODV, it is the worst since the consequences of data packet losses is compounded with what resulted from the overly stressed unicast routing.

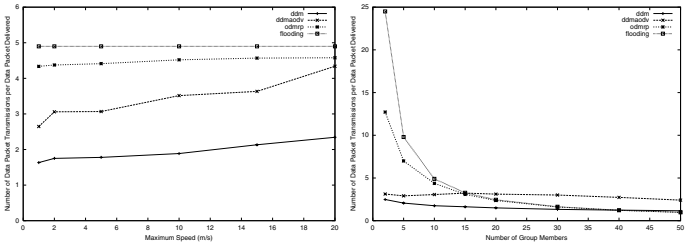

Fig. 5. Number of Data Packet Transmissions per Data Packet Delivered vs. (a) Maximum Node Speed, (b) Group Size

D.2 Data Forwarding Efficiency

We measure data forwarding efficiency using the "number of data packets transmitted per data packet delivered". The greater this number is, the less efficient a protocol is. This partially measures the bandwidth efficiency of a protocol. Since data payloads tend to be large in size compared to protocol control information, the number of data packet transmissions is usually the most important measure for bandwidth consumption of a multicasting protocol in terms of bytes.

The number of data packets transmitted includes the transmissions of all packets containing a user payload. For DDM, this includes all data packets. For ODMRP, this includes both data packets and JOIN_QUERY packets since those contain user data as well.

Figure 5 shows the average number of data transmissions required to deliver each data packet. For all simulated node mo-

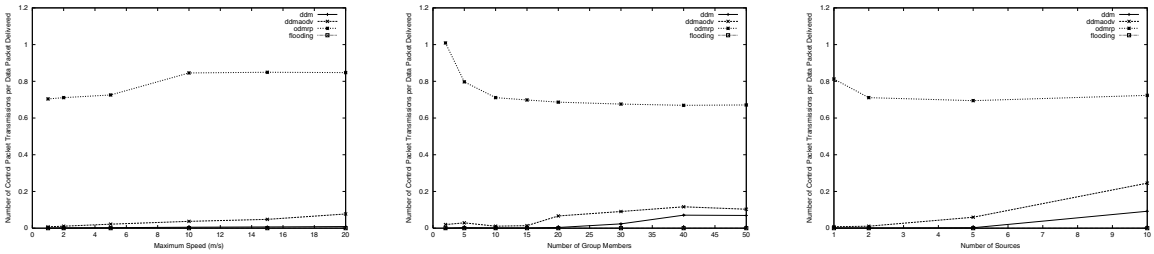

Fig. 6. Number of Control Packet Transmissions per Data Packet Delivered vs. (a) Maximum Node Speed, (b) Group Size, (c) Number of Sources

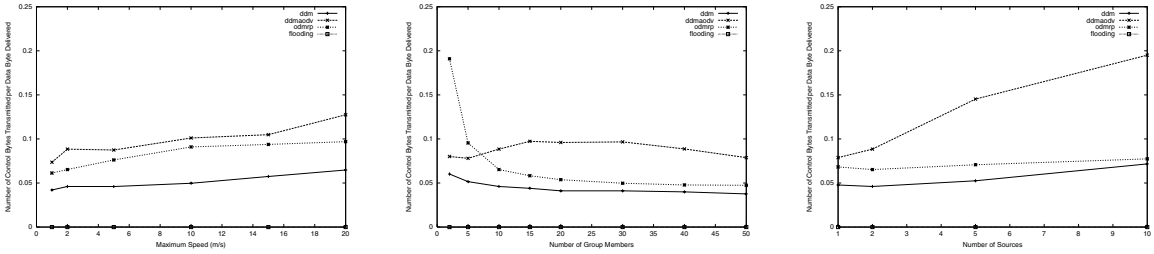

Fig. 7. Number of Control Bytes Transmitted per Data Byte Delivered vs. (a) Maximum Node Speed, (b) Group Size, (c) Number of Sources

bility levels, ODMRP requires more data transmissions to deliver data packets to receivers. One reason for ODMRP's high data transmission number is its JOIN_QUERY message. This message, with data payload piggy-backed, is flooded throughout the whole network periodically. Another reason is that ODMRP uses mesh forwarding. The mesh contains all source-to-member paths and the size of mesh is fairly large compared to the size of multicast group. Partially, the large size of the FG's is also the result of the time for a node to stay as an FG node. After an FG refreshing, a node of the previous FG may not be refreshed if it is no longer on the paths between sources and members. However, it may remain in the FG and keeps forwarding data if its FG flag has not expired. This adds forwarding path redundancy and thus improve data delivery ratio. However, it increases the size of FG and reduces data forwarding efficiency. Using tree forwarding, DDM, either running above ideal routing or AODV, uses small number of transmissions to deliver each data packet. When the node mobility increases, both protocols forward data packets less efficiently mainly due to the drop of packet delivery ratio.

When the group size increase, data transmission efficiency of both flooding and ODMRP increases due to the fact that more and more members are included in their forwarding topologies, whose sizes do not increase much because they are already quite large. DDM is relatively insensitive to group size changes. When group size increases, the overall forwarding efficiency tends to increase slightly because the paths between sources and receives are more likely to overlap. However, the increase is affected by the drop of number of data packets delivered as group size increases.

D.3 Multicast Routing Protocol Overhead

We observe multicast routing protocol overhead from two perspectives: pure control packet overhead and control byte overhead. Due to the high cost of channel access in wireless broadcast environment, the former is more significant for MANET protocols than it is for point-to-point networks. Flooding does not exchange any control information so both of its overheads remain 0.

• Control Packet Overhead: Control packet overhead is defined as the "number of control packets transmitted per data packet delivered". It shows relatively how much extra wireless channel accesses is required for the protocol to exchange control information. Here we only count pure protocol control packets that do not carry user payload. For DDM, this only counts RSYNC packets. For ODMRP, the count includes only JOIN_REPLY and ACK packets. The plots are shown in Figure 6.
Both variations of DDM rarely use pure control packets as they mostly rely on control information enclosed in data packets. Thus the number of protocol control channel accesses is kept low. The control packet overhead of both DDM's increase when data packet loss increases, which occur when (a) node mobility increases , (b) group size increases, and (c) number of sources increases. There are two reasons for this increase. Firstly, the loss of data packets which contain destination list update will drive the downstream nodes to send out RSYNC messages to their upstream neighbors. Secondly, the lowered data delivery ratio decreases the denominator of the overhead and thus increases the overhead.
ODMRP's control packet overhead is higher than the rest. This is due to the mechanism it uses to construct the FG. When a group member receives a JOIN_QUERY, it replies with a JOIN_REPLY. This JOIN_REPLY will trigger all nodes between the originator of the JOIN_REPLY and sources to send more JOIN_REPLY towards the sources. After each member receives JOIN_QUERY, it needs to reply with JOIN_REPLY and such chain reaction is triggered again.
ODMRP's control packet overhead increases slightly as node mobility level increases. ODMRP does not employ adaptive mechanisms to dynamically adjust FG refreshing period so the JOIN_QUERY's are flooded out at the same rate for all node mobility levels. The small increase is due to the following rea-

son. When nodes move faster, the reverse routes to sources learned when receiving JOIN_QUERY become less reliable as the next hops become more likely to move away. If the triggered JOIN_REPLY is targeting at a next hop which is no longer there, the JOIN_REPLY will be lost. In this case, after ODMRP retransmits JOIN_REPLY for several times without success, the node needs to find another route to source on the spot. The is done by broadcasting another packet specifying the route it is looking for. All neighbor nodes receiving such packet need to generate their own JOIN_REPLY's. This causes the increase of control packet overhead when topology changes faster. ODMRP's packet overhead decreases as group size increases and number of sources increases. Although the number of JOIN_REPLY messages generated also increases as the group size gets bigger and there are more sources, the denominator, the number of data packets delivered to members, increases even faster.

- Control Byte Overhead: Control byte overhead is defined as the "number of control bytes transmitted per data byte delivered". The number of control bytes includes both the whole length of protocol control packets and embedded protocol control information in data packets. For DDM, the whole packet size of RSYNC packets and the DDM header bytes embedded in each data packet are both counted. For ODMRP, the control bytes only include those of the control packets. For JOIN_QUERY messages that contain data payload, only bytes used by ODMRP are counted. As we can see from Figure 7, in this criteria, DDM is not particular advantageous compared to ODMRP.

ODMRP does not piggyback control information in regular data packets. Control information is enclosed in JOIN_QUERY packets. However since these packets are flooded through the network so this portion of control bytes remains at the same level for different parameters. Other control bytes are from pure control packets. Therefore we see that roughly the change of byte overhead follows the change of packet overhead for ODMRP.

For DDM, things are different. Majority of the control bytes are from DDM headers enclosed in data packets. Thus we see less correlation between packet overhead and byte overhead. When the node mobility level increases, both DDM's have higher control byte overhead due to more frequent route changes. Thus DDM needs to use more D and R blocks to inform the downstream neighbors about the route changes. These blocks enlarge the DDM header size. When the group size increases, the byte overheads of both DDM's tend to decrease. Because paths between sources and receivers overlap each other more, one data transmission can effectively forward data for more members. Also since routes are relatively slow changing due to low mobility, most of the data packets only carry E blocks. However, this trend is offset by the drop of data delivery ratio. When number of sources increases, both DDM's experience byte overhead increase. This is due to data packet loss, the use of more RSYNC, and larger DDM header size for resynchronization.

IV. Conclusion and Future Work

We have proposed a multicast routing protocol intended for use with small multicast groups in ad hoc networks of any size. The protocol is source-initiated and controlled, and uses in-band, destination-encoded data packet headers to control a distributed routing computation. The result is a flexible, efficient and robust protocol suitable for small multicast groups. The protocol offers the option of choosing between "stateless" and "soft-state" modes.

The simulation results show that DDM is very efficient both in terms of data forwarding and control channel access. Especially for small groups, such characteristic stands out among all simulated protocols. When the network traffic level is low, the throughput of the protocol is affected by performance of the unicast routing protocol DDM is operating upon. When the unicast protocol provides good route information, DDM delivery high throughput across all situations. DDM performs particularly well for small groups. It is nearly ideal for multicasting to small groups in networks that already have unicast routing support. On the other hand, ODMRP is better suited for another end of MANET multicasting, multicasting to large and dense groups.

The simulation also shows that DDM is more sensitive to network traffic level compared to protocols using redundant forwarding path. This is because data packet loss has more serious consequence on protocol performance. When data packet loss occurs, the DDM control information embedded in data packets is also lost. With no redundant data forwarding, packet loss also reduces packet delivery ratio.

In future studies, different modes and options of the DDM protocol will be explored. More comparison studies will also be included in future plan. Improvements to the protocol, especially those targeting at reducing data packet vulnerability and the impact of data packet loss will be attempted.

References

[1] I. Stoica, T. Ng, and H. Zhang, "Reunite: A recursive unicast approach to multicast," *In Proceedings of IEEE INFOCOM'00*, March 2000.

[2] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," *In Proceedings of the International Conference on Measurements and Modeling of Computer Systems*, 2000.

[3] Rick Boivie, "A new multicast scheme for small groups," *IBM Research Report RC21512(97046)*, June 1999.

[4] D. Ooms and W. Livens, "Connectionless multicast," *Internet Draft draft-ooms-cl-multicast-01.txt, work in progress*, October 1999.

[5] J. J. Garcia-Luna-Aceves and E. Madruga, "A multicast routing protocol for ad-hoc networks," *Proceedings of IEEE INFOCOM'99*, pp. 784–792, March 1999.

[6] S. Lee, W. Su, and M. Gerla, "On-demand multicast routing protocol (odmrp)," *Internet Draft draft-ietf-manet-odmrp-02.txt, work in progress*, June 1999.

[7] E. Royer and C. Perkins, "Multicast using ad hoc on-demand distance vector routing," *Proceedings of ACM/IEEE MOBICOM '99*, 1999.

[8] L. Ji and M. Corson, "Light-weight adaptive multicast," *In Proceedings of IEEE GLOBECOM '98*, November 1998.

[9] D. Johnson and D. Maltz, "Dynamic source routing in ad hoc wireless networks," *Mobile Computing, edited by Tomasz Imielinski and Hank Korth*, pp. 153–181, 1996.

[10] L. Ji and M. Corson, "Differential destination multicast (ddm) specification," *Internet Draft draft-ietf-manet-ddm-00.txt, work in progress*, July 2000.

[11] UC Berkeley and LBL and USC/ISI and Xerox PARC, http://www-mash.cs.berkeley.edu/ns, *ns notes and documentation*, October 1999.

[12] J. Broch, D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," *Proceedings of ACM/IEEE MOBICOM '98*, pp. 85–97, October 1998.

[13] C. Perkins, E. Royer, and S. Das, "Ad hoc on-demand distance vector (aodv) routing," *Internet Draft draft-ietf-manet-aodv-04.txt, work in progress*, October 1999.

[14] S. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A performance comparison study of ad hoc wireless multicast protocols," *INFOCOMM '2000*, March 2000.