Reliable Ad Hoc Group Communication using Local Neighborhoods

Lawrence Klos, Golden G. Richard III (lklos@uno.edu, golden@cs.uno.edu)
CS Dept., University of New Orleans, New Orleans, LA 70148

***Abstract* - In this paper an enhanced reliability protocol added to the ODMRP multicast ad hoc protocol is described. This protocol increases the overall data packet delivery ratio by adding packet storage and retransmission operations coordinated by the multicast source. Storage responsibilities are assigned based on localized 'neighborhoods' of nodes with minimal spanning hopcount, within the group. Simulation results are presented that reflect the protocol overhead of both ODMRP and the reliability component, broken down by operational phase.**

***Index Terms* – Mobile Networking, Ad Hoc Wireless, Multicast, Reliable, ODMRP.**

I. INTRODUCTION

Ad hoc networks consist of mobile wireless nodes communicating with no fixed infrastructure support. Communication occurs on a node to node basis, with the links established between nodes forming the overall ad hoc network. Applications for ad hoc networks range from an informal collection of conference participants communicating in a ballroom, to soldiers organizing in a battlefield or rescuers coordinating efforts on a remote mountaintop. Group, or multicast, communication is a natural extension for ad hoc networks. It is necessary for many applications. Where group communication is required, reliability is often an important issue: in situations where lives are at stake, missed messages can have very negative consequences.

II. RELIABLE AD HOC GROUP COMMUNICATION

Reliable group communication research has a relatively long history in wired networks. It is becoming a core enabler for organizations providing distributed services where specific communication reliability guarantees between nodes at multiple sites are required. Each node in such systems processes all generated messages, none of which can be dropped without serious consequences.

To date, it is not feasible to implement reliable communication protocols in Ad Hoc networks. It is not even possible to guarantee delivery of data packets. If a node stays out of range of others, it will never receive data packets. Most existing ad hoc multicast protocols try to deliver the greatest number of data packets to other nodes on a best effort basis.

Golden G. Richard III is an Associate Professor with the University of New Orleans, New Orleans, LA 70148 (email: golden@cs.uno.edu).
Lawrence Klos is a PhD student with the University of New Orleans, New Orleans, LA 70148 (email: lklos@uno.edu).

This paper is an initial study of factors involved in eventually reaching 100% reliability. In general, three factors are responsible for missed data in ad hoc networks. The first is dropped packets due to network link contention from control overhead, the second is dropped packets due to contention from data forwarding overhead, and the third is dropped packets due to non existent links. The first two factors are more common in dense networks, while the third factor occurs increasingly as network node density lessens. The reliability of most basic ad hoc multicast protocols are affected by these three factors, since the protocols rely on initial best-effort data delivery. For these protocols, some mechanism must be added if the goal is eventual 100% reliability.

A mechanism of storing and resending data packets is one potential solution to missing data packets. Whether packets are undelivered due to network contention or missing links, future data packet resends can allow for improved chances of eventual data delivery. This mechanism will be successful if the dropped packet count due to the new overhead generated from the resend mechanisms is more than outweighed by an overall increase in data delivery. The goal then, is to develop mechanisms that increase network contention due to the new control overhead and data packet resend overhead as minimally as possible, while increasing endpoint delivery of missing data packets as much as possible.

In this paper, the strategy taken was to implement data packet storage and retransmit techniques coordinated by the multicast source. ODMRP was taken as the underlying multicast group communication protocol, since it is well documented and simulations[7] have shown it to have a comparatively high data delivery ratio. The storage and retransmit mechanisms distribute the responsibility for message storage and retransmission to all group members, with the source of each data stream being responsible for the partitioning of nodes into neighborhoods and assignment of storage responsibilities. With group members partitioned into neighborhoods and each neighborhood storing a 'sliding window' of the full dataset, multiple datasets can be stored across the network as the group member count increases, and NACK requests for data packets are increasingly likely to be answered locally, reducing both network overhead and data latency as the network grows.

The choice of the source node as coordinator for these responsibilities is not an intuitive one. It is common knowledge that ad hoc multicast protocols are distributed algorithms, and any centralized component is to be avoided. However, several factors suggested that this R-ODMRP centralized mechanism could be successful:

- An implicit centralized component, the data source, is unavoidable for any multicast ad hoc communication protocol. If the central source of data dies or is partitioned, there will be no data to reliably deliver.
- ODMRP, while categorized as non-centralized, still relies on the near-centralized two step process of join query/reply for datapath establishment. In the extreme case (with receivers existing only at the outer edge of a network), this mechanism duplicates R-ODMRP's mechanism of reliable join query/reply, with the final step of *nodes adjacent to the source sending a packet back to the source* removed.

Section 3 discusses related work, section 4 presents a brief overview of ODMRP, section 5 describes R-ODMRP, and section 6 describes the neighborhood creation algorithm in detail. Section 7 presents a performance evaluation of R-ODMRP, giving a detailed description of the packet overhead for each phase of the protocol. Finally, section 8 covers future work, and section 9 concludes the paper.

III. Related Work

Multicast reliability in ad hoc networks is a relatively new area of research, with results appearing in the recent years. Mechanisms from protocols can be broken down into three categories: Congestion Control mechanisms such as RALM, Forward Error Correction mechanisms such as Packet Erasure Recovery, and Data Packet Retransmission mechanisms such as RMA, Anonymous Gossip and Hyperflooding.

RALM[10] is a reliability protocol that achieves a higher data delivery ratio by enforcing error and congestion control similar to TCP. Reliable data delivery is guaranteed to one group member at a time, in round-robin fashon. This is accomplished by requiring the source to select a neighbor to transmit the data to. The neighbor will reply with either a positive ACK, showing the data was successfully received, or a negative NACK, requesting retransmission of missing data. This feedback from the neighbor is also used to adjust the sources window size. This window increases linearly, but if losses begin to occur it is halved. In this approach decreased throughput is traded off for increased reliability. Another downside is that even with congestion control, dropped packets are resent from the source rather than locally, creating greater network overhead from both the receiver NACK and the source retransmit.

Packet Erasure Recovery [9] uses error correction codes to provide reliability. Data packets are encoded and split into pieces, and these pieces are then transmitted to receivers. If a certain number of packet pieces arrive at a receiver, the data packet is correctly reassembled and processed. The downside of this approach is that forward error correction work best when loss rates are predictable. In ad hoc networks, worst case behavior for a node to drop packets is unpredictable: a node may go out of range of the network and stay out of range.

RMA [5] is a reliable ad hoc multicast protocol that assumes sources know the Ids of all receivers in the network, and ensures they receive all data packets by recording positive ACKS from all receivers. Retransmissions from the source provide reliability. Stable links are given preference by using the routing metric of the expected lifetime of a link. A downside of this approach is that all receivers sending ACKs back to the source makes scalability of the protocol questionable, due to ACK implosions.

Anonymous Gossip[3] allows for randomly selected pairs of group nodes exchanging information on received and lost packets. A node missing data packets will periodically send a gossip request, containing data on lost packets, sequence number of next expected packet and source and group address, to a random neighbor node. If the neighbor node is a group member, it unicasts a reply back to the initiator, otherwise it randomly selects a neighbor to forward the message to. The multicast protocol in use must provide nodes with hopcounts to their nearest neighbors, to accomplish this. An unavoidable downside of this approach is that the increased network traffic created by the protocol negatively impacts the data loss the protocol attempts to correct. Also, given that request / replies occur between random nodes, this is a best effort technique.

Hyper flooding [8] is an adaptive technique using flooding as the base protocol, with modifications to prevent loops. Nodes record neighbors by listening to and sending hello messages. Received data packets are stored, and rebroadcasts of these data packets occur when a packet is received from a node that is not on the neighbor list, or when receiving a hello message from a new node. In these cases, all packets in the data packet cache are retransmitted. The downside of this approach is a far greater amount of network overhead, and the large amount of storage required at each node.

IV. ODMRP overview

ODMRP[6] is a mesh-based on-demand ad hoc protocol for group communication. It performs scoped flooding of data packets to all group members by establishing a 'forwarding group' of network nodes between a source and group members. Route refreshes update the broken links arising from node mobility or resource changes. Route setup and refresh each consist of two phases: Request and Reply.

A. Route setup - Request phase

When a source has multicast data to send but no knowledge of receivers, it builds a "Join Query" packet, adds its IP address, and broadcasts it. Each node receiving the Join Query will store the source IP address and packet ID, add the IP addresses of the upstream node and originating source to its routing table, add its own IP address into the last hop IP address field, and rebroadcast it downstream. The Join Query floods the network, reaching all receivers.

B. Route setup - Reply Phase

A group member, upon receiving a Join Query, completes the processing described above for the Join Query, then initiates a "Join Reply" packet once the multicast route is selected. The receiver node adds all source and next hop IP addresses for the group from its routing table, adds its own IP address into the previous hop field, and broadcasts the Join Reply packet upstream. Each neighbor node receiving this packet checks the set of next hop IP addresses. If a next hop IP address matches the neighbor node's own, the node is on the forwarding path between source and receiver, and is part

of the forwarding group. The node sets its Forwarding Group flag, looks into its own routing table entries for the group ID and builds a Join Reply packet to broadcast upstream.

C. Forwarding Data and Maintenance

When a node receives a multicast data packet, it checks for duplicates, then checks its Forwarding Group flag. If the flag is set, the node is a forwarding group member, and will rebroadcast the packet to its neighbors.

Periodically, the source will refresh routes with another Join Query. All forwarding group members will then be reset according to the new network topology. Group membership is preserved in a soft state. Once a source has no data to multicast, it stops sending periodic Join Query packets. All forwarding nodes will then eventually timeout and revert to non-forwarding status for that source. If a receiver wants to leave the group it stops sending Join Reply packets.

D. Unicast Functionality

Using the same Join Query/Join Reply protocol with a unicast IP address as the destination, a unicast sender can discover a route to a unicast receiver. Since duplicate Join Query packets are dropped (based on source IP address and data packet sequence number), the route created by unicast operation is a single path.

E. ODMRP Data Structures

Following are the standard data structures of ODMRP.

- *Message Cache*: When a node receives a Join Request or data, it stores the source ID, sequence number and group address of the packet in this cache to detect duplicates. This cache is timed out in Round Robin fashion.
- *Member Table*: This table holds the multicast address and source node address combination that the current node is a receiver or forwarder for. An expiration time variable is in the table in order to expire stale entries.
- *Forwarding Group Table*: This table holds the multicast addresses and expiration times for all multicast groups for which the current node is a forwarding group member.
- *Routing Table*: This table holds the multicast and source addresses for all multicast senders the current node is a receiver for, along with the next hop (upstream) address on the path to the source. This next hop address is used as the destination for Join Reply packets from the current node.

V. RELIABLE ODMRP

The technique outlined here allows each source a means to work with the two parameters of reliability and overhead cost, moving the reliability ratio up or down dynamically, over a single multicast session, if desired.

A. Overview

In R-ODMRP the responsibility for data storage and retransmit is assigned to all receivers of the multicast group, with the source of each data stream coordinating responsibilities. All group members are divided up by the source into sets of local neighborhoods. The source sets the number of nodes per neighborhood, with the option of determining the node's storage overhead. With each neighborhood member storing a portion of the data packets, each local neighborhood stores a distributed "sliding window" of all transmitted data packets. Nodes Nacking missing packets will be answered by neighbors unicasting replies.

B. Packet Storage

In R-ODMRP, When a source initially sends out a Join Query, it becomes a Reliable Join Query (RJQuery) packet. The RJQuery packet has a timeout value attached. Once the RJQuery packet is sent, each node receiving it (whether a receiver node or not) will decrement this timer value by a preconfigured "two hop time" before sending the RJQuery downstream. After the RJQuery timer expires at each node, each receiver node will send a Reliable Join Reply (RJReply) back upstream. If a node with an expiring timer is not a receiver, it will send an RJReply only if it receives other RJReplies from downstream.

Each RJReply contains a 2D table, known as the Network Datapath table. When a node (receiver node or not) receives RJReplies from downstream nodes, it stores their Network Datapath table as a block in its own table sorted relative to other received blocks with the topmost block having the longest datapath. On timer expiration, just before the table is sent upstream in an RJReply, each table entry is shifted such that entry (x, y) becomes entry (x, y + 1), emptying the leftmost column, column 0. The node stores an entry for itself in entry (0,0) containing its id, branch count (the number of RJReplies received from downstream), and receiver status, and then forwards the table upstream in its own RJReply.

The end result of the RJQuery/RJReply phase is that the source obtains a full positional listing of all receivers and forwarding group members in the network. RJQuery/Reply operations occur periodically, but at a lower frequency than the standard Join Query/Reply operation.

The source will then set a number for the "nodes per neighborhood" count, and, with the Network Datapath table as input, partition all receiver nodes into local neighborhoods using its "Source Neighborhood" Algorithm. The source then assigns data packet storage responsibilities such that the set of nodes within any given neighborhood will store the full set of data packets in sliding window fashion.

On the next multicast data packet after a Reliable Join Query, the source piggybacks a table defining the range of packet sequence numbers each receiver in each neighborhood is responsible for storing. Each receiver then begins storing its share of data packets. This recovery scheme does not depend on which node stores the packets, only that they are stored somewhere in each neighborhood.

As nodes leave the group, their storage responsibilities are reassigned on new RJQuery/Reply rounds. However, as more and more nodes join over time, more neighborhoods are created and duplicate storage responsibilities will be assigned. The individual neighborhoods storing the duplicate packets will become smaller and smaller, relative to the overall network. Additionally, the source can reassign neighborhood size and data packet storage responsibilities on any RJQuery/RJReply round, dynamically adjusting reliability versus overhead over the course of a multicast.

C. Packet Retransmission

The second responsibility, data packet retransmission, will be initiated by a receiver node noticing a gap in data packets. It will broadcast a Resend Request packet to its local neighborhood, with a local time-to-live scope, listing all packets needed by sequence number. The requestor will give its ID for unicast replies. Upon receiving the packet, neighbor nodes will check their storage for the requested sequence numbers and unicast found data packets back along a single path. If the requesting node receives an incomplete reply or no reply at all, it will retain all gap sequence numbers, sending them out in its next Resend Request.

D. R-ODMRP Structures

These structures are in packets sent to other nodes:

- *Network Datapath Table*: This table holds the current node's network positional information (node id, branch count and receiver status), accumulated from nodes on all downstream datapaths. This table is inserted into an RJReply packet, just before sending. The bandwidth requirements for a single node entry in the 2D Network Datapath Table in this implementation is 2 ½ bytes, based on a 15 bit node id, a 4 bit branch count (holding a maximum of 15 branches from a node), and a 1 bit boolean receiver status. A single 64k data packet will hold data describing approximately 26,200 nodes. As yet, it is unclear what the maximum feasible size of ad hoc networks will be, but one line of thinking holds them to be smaller than this, such as an informal gathering of conference attendees, or a lone group of rescuers.
- *Data Packet Gap List*: This list contains sequence numbers of all data packets that have not been received by the local node. The bandwidth requirements used in this implementation were 2 bytes per packet id number.
- *Storage Responsibility Table:* This table holds data packet storage responsibilities for all receiver nodes in the network. It is multicast out by the source to all receivers after the source neighborhood algorithm completes. The bandwidth requirements for a single receiver node entry in the 2D Storage Responsibility Table in this implementation is (2 + 1/n) bytes, with n being the 'nodes per neighborhood' count. For a 'nodes per neighborhood' count of 3, a Storage Responsibility Table handling the node count of 26,200 described above will fit into a 64k data packet for the worst case, where every network node is a receiver.

These structures are needed for a node's internal processing:

- *ResendRequestReply Cache*: This table holds data packets a node is responsible for storing. Additionally, it holds snooped data packets carried in resend replies forwarded by a node. To identify each data packet, the group address, id of the source node, address of the request originator and previous hop forwarder, originators sequence number of the request, and the id of the replier are all stored. Entries are aged out in Round Robin fashion. All replies are stored so that if any nearby receiver node sends a request for the packet in the future, it can be answered locally. The sending of resend requests for different nodes are staggered by a random time, in order to make this likely to happen. If a local group of nodes all miss a data packet but get the next one in sequence, one node will send out a request for a data packet while others wait. By the time others begin to initiate a request for the same packet, they will likely find it stored in their cache already and stop the send process.
- *Data Packet Sequencer*: This list holds recently received data packet sequence numbers along with their received time, for a given source. When a sequence number is received causing a gap, and over two seconds has elapsed since its reception, the missing number is listed as a gap in the received sequence. The data packet sequencer list is added to from the tail, and the head is periodically trimmed. Trimming occurs either periodically when received data packets are all in sequence, or when gaps have been identified and loaded to the gaps list.

VI. R-ODMRP Neighborhood Creation

A. Overview of Neighborhood Building

As the group of receivers grows in size, neighborhood partitions and node data storage responsibilities are dynamically reallocated by the source, allowing partitioned neighborhoods to be composed of a diminishing percentage of network receiver nodes that are more closely grouped. As the number of receiver nodes and neighborhoods grow in an ad hoc network, Resend Requests and replies will travel fewer hops, reducing overall network traffic. Scalability is built in to the data storage and retransmit process.

B. R-ODMRP Neighborhood Building Parameters:

A set of parameters govern R-ODMRP Neighborhood Building operations. Some variables are simply inputs to the Neighborhood Building algorithm, while others are configured by the source. In initial simulations they are set to a fixed amount, but will be varied to study various network conditions in future work. They are described below:

- *Size of a node's data packet storage buffer (NodSiz):*It is assumed that all nodes in the network will be homogenous, and all will have the same fixed amount of storage space to devote to reliable communication.
- *Amount of network data produced per sec (AmtSec):* Fixed based on a single source generating a set number of fixed size packets per second.
- *Number of seconds of data to store (NbrSec):* Set to a value greater than the average time a node going out of range stays disconnected from the network.
- *Total storage capacity for a neighborhood(TotStg):* Set by the formula: TotStg = AmtSec * NbrSec
- *Number of nodes per neighborhood (NbrNod):* Set by the formula: NbrNod = TotStg / NodSiz

Following are R-ODMRP timing parameters:

- *Source timeout after RJQuery, before processing RJReplies (SrcTimOut):* Set based on the maximum simulation time for the first RJQuery to travel to the farthest receiver, and the RJReply to return to the source.
- *Time for a packet to travel one hop and back (TwoHopTim):*Used to set timers for Resend Requests,

and to determine the amount to subtract from the SrcTimOut remainder at each downstream node.

C. Neighborhood Building Algorithm:

The algorithm takes as input the Network Datapath Table and uses the NbrNod variable to partition the network into neighborhoods. Then it builds a table assigning each neighborhood node packet storage responsibilities, and a maximum hop count between nodes for each neighborhood. It inserts this table into the next JQuery packet before broadcasting it. The algorithm works as follows:

1. First, a pass is done through the Network Datapath Table, identifying the number of receiver nodes in each row (data path), summing to find the total number of receivers. The total number of receivers divided by NbrNod will give the number of neighborhoods the receivers will be partitioned into, as well as a remainder. The source keeps track of both the NbrNbrhds variable and the RmdrNbrhds variable.
2. Next, construction of the Storage Responsibility Table begins. Starting with the receiver at the far right end of the top row in the array, processing moves left, the hopcount is tracked, and receivers are added until a neighborhood is completed or a branch node is reached. Once a full neighborhood of receivers is identified, the source loads a row in its Storage Responsibility Table and records the maximum spanning hop count.
3. Upon reaching a branch node, if a full neighborhood is not yet built, the algorithm loads the branch node position on a stack, and steps down to the end of the next row. It continues to build the current neighborhood from the furthest node from the source forward, tracking hop count. Similarly, if a branch node is reached in this row, processing steps down another row, but never moving outside a block. Once processing again reaches the node originally stepped down to on a given row, the stack node is popped and processing continues with it.
4. Once the first block is complete, the algorithm moves on to the next. The algorithm continues on in this manner until all receiver nodes within each block are either partitioned into a neighborhood or the count of remaining receiver nodes within each block is less than NbrNod.
5. Remaining receivers in all blocks are closest to the source. They are partitioned in the following way:
   - Unpartitioned nodes are sorted from bottom to top, with associated hopcount to source retained.
   - Selection of nodes for a new neighborhood begins at the bottom, and works sequentially to the top.
   - Hopcount per neighborhood is the sum of the two greatest numbers from either the max node hopcounts to the source from nodes in different blocks, or the max hopcount between two nodes in one block.

   This algorithm will result in a table of partitioned neighborhoods, each with a spanning hopcount. Storage responsibilities are assigned by assigning a data packet sequence number range to each column in the table. This table is then put into the next JQuery packet and broadcast out to all receivers. Each receiver, upon receiving this packet, will learn its storage responsibilities and begin storing packets in a circular buffer.

D. Example of Neighborhood Building

Figure 1 shows a diagram of an example ad hoc network. The bold outlined node is the source, dotted outlined nodes are forwarding nodes and solid outlined nodes are receivers. Figure 2 shows example node network datapath tables, sent from the listed nodes to those upstream. Eventually the source will receive four RJReply Network Datapath Tables, sort them by block, and build a Network Datapath Table representing the composition of the overall ad hoc network.

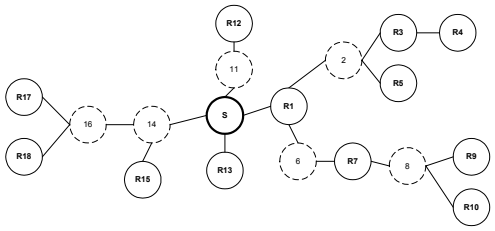

Figure 1: Example Ad Hoc Network.

Each entry in this Network Datapath Table is a structure with three elements: NodeID: the node's individual id, Branch_Count: the number of downstream branches (table

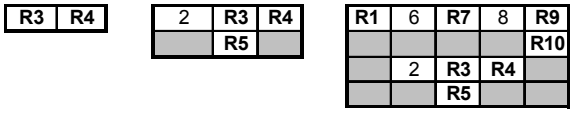

Figure 2: Example Network Datapath Tables (R3, 2 and R1)

rows) extending from the node, and Receiver_Status: a Boolean indicating receiver/forwarder status. The Network Datapath Table constructed by the source is shown in Figure 3. This table has four blocks, built from four RJReplies.

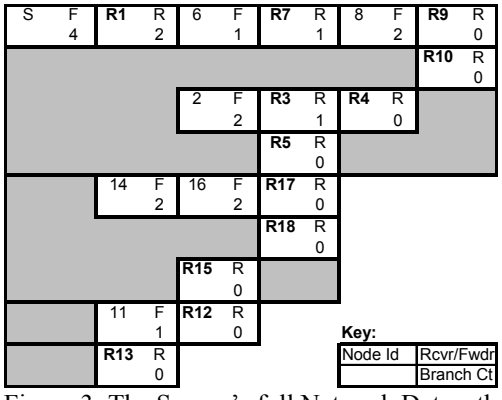

Figure 3: The Source's full Network Datapath table.

The source then begins the task of partitioning this table into neighborhoods. If the node count per neighborhood (NbrNod) is three, for example, the partitioning would happen in the following manner:

- R9 is selected for the first neighborhood. Node 8, a branch node, is placed on the stack, and R10 is added. Node 8 is popped, and R7 completes the neighborhood. The neighborhood's max hop count is set to 2.
- R1 is reached and added to neighborhood 2. R1 is seen as a branch, placed on the stack, and nodes R4 and R3 are

added to neighborhood 2. The max hop count is set to 3, and R5 is a block remainder.

- Next, R17 is selected as the first node of neighborhood 3. 16 is placed on the stack and R18 is added to the neighborhood. Then node 14 is placed on the stack and R15 is added to the neighborhood, which is set to have a max hop count of 3.
- Now the algorithm shifts to phase two. The resorted array of remainder nodes is shown in Figure 4. R1 has already been partitioned into a neighborhood, so a flag is set for the entry to indicate this.

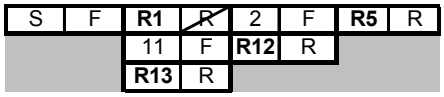

Figure 4: Network Datapath Table Remainders

- Starting with shortest hopcount to the source first, this array is traversed bottom up. First, the algorithm selects R13 for neighborhood 4. Next, R12 is selected for the neighborhood, and finally R5 is selected. The max hopcount is 5.
- The algorithm completes with the Storage Responsibility Table shown in Figure 5. For every 100 data packets sent from the source, nodes in the first column will store packets 1-33, nodes in the second will store packets 34-66 and nodes in the third will store packets 67-100.

Pkts 1-33	Pkts 34-66	Pkts 67-100	Nbrhd Hopct
R8	R10	R7	2
R1	R4	R3	3
R17	R18	R15	3
R13	R12	R5	5

Figure 5: Node Packet Storage Responsibility Table.

VII. Protocol Performance Evaluation

R-ODMRP was implemented in the ns-2 network simulator [4], developed by the University of California, Berkeley, and the VINT project, with Carnegie Mellon's Monarch Project mobile and wireless ns-2 extensions[11] incorporated. The ns-2 simulator is commonly used in networking research. [2] provides a full description of the software layers and the IEEE 802.11 MAC protocol used in these simulations. The USC/ISI ns-2 implementation of ODMRP[12] was also used.

A. Simulation Details

The ODMRP and R-ODMRP simulations all executed with identical randomly generated baselines of network traffic and node movement files to more accurately compare performance. This baseline consisted of five node movement scenarios and six traffic pattern scenarios. All scenarios established fifty mobile nodes with a single node as multicast source within a 1000m x 1000m area. The radio propagation range for each node was 250 meters, and the channel capacity was 2 Mbits/sec. Each simulation executed for 600 seconds of simulated time. Once all nodes joined the group the multicast source began transmission of 512 byte packets with a constant bit rate of 3 packets per second. The traffic pattern scenarios had 25, 30, 35, 40, 45 and 49 receiver nodes respectively.

30 simulation runs were executed each for ODMRP and R-ODMRP. A total of 60 simulations were performed. This baseline was chosen because simulations [7] have shown that ODMRP performs best in conditions of relatively good network connectivity and low network traffic load and speed, and any protocol with the goal of increasing its reliability would have to outperform standard ODMRP under these conditions. The reliability technique proposed in this paper likely has its greatest advantages in sparse networks with frequent longer partitions, however.

For ODMRP and R-ODMRP, parameters were set to 3 seconds for the Join Query flood interval and 9 seconds for the forwarding state timout, the values used by ODMRP's creators in their simulation studies. R-ODMRP sets a flag in every fourth Join Query packet, turning it into a Reliable Join Query packet. The node count per neighborhood for R-ODMRP was set at 3, and all nodes were preset to store a maximum of 500 data packets, in Round Robin fashion.

B. Initial Simulation Experiments

Beginning experiments lead to some modifications to the basic protocol of R-ODMRP that produced better end results. Originally, the time-to-live hopcount for a resend request packet was set to the maximum distance between nodes within a given neighborhood, but this produced relatively poor results. Data packets that would have been correctly delivered under ODMRP were dropped due to network traffic contention with the Resend Requests, causing the R-ODMRP portion of the protocol to work that much harder to try to fill the gaps, leading to further network contention. In the end, for these simulations of high network connectivity, a TTL of 1 gave best results for Resend Request packets. A consequence of this was that data packets that were undelivered to a group of receiver nodes tended to "bubble" across nodes over many cycles, increasing latency for those packets.

C. Simulation Results

Initial results for *Total Data Packets vs. Delivered Data Packets* (Packet Delivery Ratio) were encouraging. Table 1 shows that when ODMRP ran alone Packet Delivery Ratio varied between 92.8% and 93.8% for the thirty simulations, given the same number of network nodes and an increasing percentage of receivers.

Scenario	ODMRP	R-ODMRP	
		(ODMRP part)	(both parts)
50n25r	92.8%	92.0%	97.1%
50n30r	93.2%	92.2%	97.1%
50n35r	93.4%	92.1%	97.5%
50n40r	93.7%	92.5%	97.7%
50n45r	93.8%	92.4%	97.6%
50n49r	93.7%	92.2%	97.5%

Table 1: Packet Delivery Ratio

Scenario	ODMRP	R-ODMRP	
		(ODMRP part)	(both parts)
50n25r	2.24	2.17	2.74
50n30r	2.04	2.00	2.56
50n35r	1.82	1.79	2.36
50n40r	1.70	1.68	2.26
50n45r	1.61	1.56	2.16
50n49r	1.53	1.50	2.13

Table 2: Control Overhead

When R-ODMRP ran, the ODMRP portion operated between 1% and 1 ½% worse than its standalone counterpart, due to the added network contention, but the reliability portion

increased Packet Delivery Ratio by approximately 4% overall, to between 97.1% and 97.7%.

Other metrics showed the tradeoff for this increased reliability, however. The *Ratio of Data and Control Packets vs. Delivered Data Packet* (Control Overhead), shown in Table 2 reflects a consistent and unavoidable increase for R-ODMRP. The differential in this metric represents greater channel contention, working against the basic goal of reliable data delivery. Though an increase must exist, since R-ODMRP uses control packets, the increase shown for R-ODMRP scales similarly to that of ODMRP, rising a similar percentage as the number of receivers in the 50 node network declines.

Data Packets Forwarded vs. Data Packets Delivered (Forwarding Overhead), shown in Table 3, also shows an unavoidable increase for R-ODMRP. The differential in this metric also represents greater channel contention, working against the basic goal of reliable data delivery. An increase here must exist, given the store and retransmit mechanism, but the differential between ODMRP and R-ODMRP increases with an increase in receiver count. The mechanism used for Resend Request/Reply will be modified to increase scalability of this portion of R-ODMRP.

Scenario	ODMRP	R-ODMRP	
		(ODMRP part)	(both parts)
50n25r	19.8	19.0	21.8
50n30r	21.6	20.9	24.5
50n35r	22.0	21.3	26.6
50n40r	23.2	22.8	29.3
50n45r	24.5	23.5	31.6
50n49r	25.0	24.0	33.8

Table 3: Data Forwarding Overhead

The data delivery latency of the two protocols shows the greatest differential, however. While the average latency of ODMRP, and the ODMRP portion of R-ODMRP averaged about 10ms across all receiver counts, the extra packets delivered by the Resend Request/Reply portion tended to have a latency of seconds, due to several factors. One is the fact that two seconds elapse after a gap is noticed and a Resend Reply packet is sent. Another is that a random delay before sending was added to allow snooping of other node's Replies before sending a request. A third is the mechanism used to trigger requests, which causes data to "bubble" across nodes.

The competing metrics involved in enhancing reliability for ODMRP have been clarified as a result of this work. Four central factors balance against each other: Packet Delivery Ratio ("Reliability"), Ratio of Data and Control Packets per Delivered Data Packet ("Control Overhead"), Forwarding Efficiency ("Forwarding Overhead") and Data Packet Delivery Latency to all Receivers ("Latency"). Comparing the basic ad hoc multicast protocol of ODMRP to R-ODMRP, overall latency tends to be lower, reliability is based on the basic protocol's best-effort delivery technique, and network traffic overhead is lower. When the store and retransmit reliability components are added to ODMRP, reliability increases, overall latency increases and network traffic overhead increases, due to the control and forwarding mechanisms. A successful reliability component will, under various network conditions, always increase reliability (by a varying amount, depending on the scenario and the strength of the reliability component), increase overhead by an 'acceptable' amount (acceptable meaning low enough so that the extra overhead causes minimal additional network contention resulting in minimal additional dropped data packets), and increase data packet latency minimally as possible. Of the three competing factors, the two overhead metrics are more tightly linked to increased reliability, and latency is the least linked metric.

In most multicast ad hoc protocols, reliable packet delivery falls off sharply as network node density becomes more sparse, with fewer links between nodes. It is expected that the sparser the network, the more successful a store and retransmit reliability component such as R-ODMRP will be in achieving its goals. In sparse networks increased network traffic overhead required by the reliability component will have a lesser negative effect, since contention is less of an issue. It is expected that latency will be affected to a greater degree, since packets that would have been undelivered will be delivered much later, when a link is finally obtained, but latency will be due to the unavailability of a link rather than the mechanisms of the reliability component.

D. Protocol Results by Phase

Statistics were gathered for the normalized packet counts for each phase of the ODMRP portion and the reliability portion of R-ODMRP. Figure 6 reflects the normalized packet counts

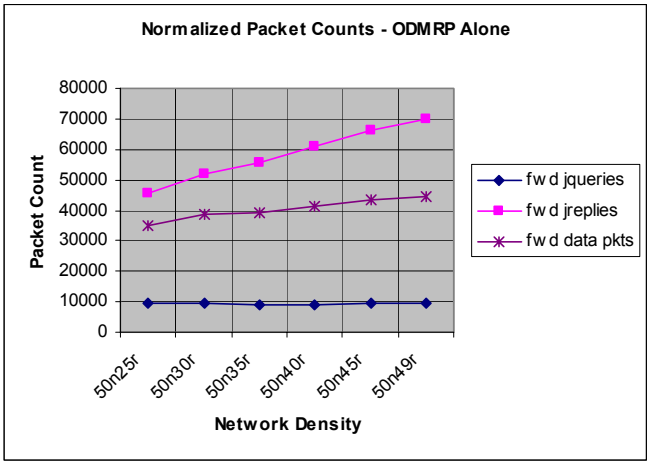

Figure 6: Normalized Packet counts for ODMRP.

for all phases of ODMRP. Here it can be seen that the number of forwarded JQuery packets holds flat across the 6 scenarios, while the forwarded data packet count rises gradually. This makes sense, because as more receivers are added, data packets will at times be forwarded to further endpoints, given the same network. The JReply packet count shows a sharper increase, however. This portion of ODMRP would be the first to investigate in order to raise ODMRP's overall efficiency.

Figure 7 shows the corresponding normalized packet counts for the phases of R-ODMRP added in over the baseline series of runs. Here it can be seen that the number of RJQueries

holds flat. This is expected, since the ODMRP protocol is reused for this component. The count of RJReplies rises very gradually, almost holding flat, as the number of senders is increased. This count reflects the new timeout mechanism for gathering all downstream RJReplies before initiating one upstream. This metric shows that ODMRP's network contention due to JReply traffic can be reduced by adopting the R-ODMRP mechanism. This would increase ODMRP's scalability and efficiency by reducing control overhead network traffic. The R-ODMRP counts for Resend Requests and Resend Replies rise at a similar steep pace relative to the other protocol components, however. The Resend Request/Reply mechanism would be the first to look at in terms of increasing the efficiency of the overall R-ODMRP protocol. A technique to unicast out a Resend Request should help reduce this packet count. This will have the secondary effect of reducing the Resend Reply count.

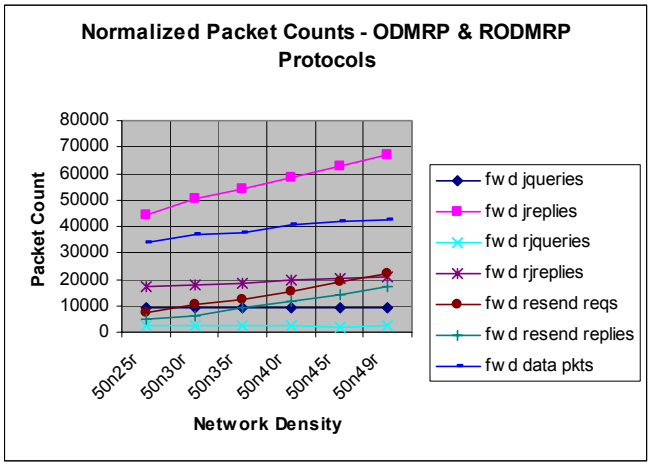

Figure 7: Normalized Packet counts for Both.

VIII. Future Work

Near term goals include economizing the Resend Request /Reply mechanism for dense networks. Examination of transmission and reception simulation detail data will provide information that will help close the gap to full packet delivery.

Other areas for future work would be to examine other network scenario baselines. For example, sparse networks would seem to be an area where R-ODMRP could operate to greatest advantage. A new mechanism for Resend Requests must be developed for this scenario, and merged with the existing mechanism to work across cases.

IX. Conclusions

This paper described R-ODMRP, a reliability protocol added to ODMRP. R-ODMRP consists of operations to store and retransmit sequenced data packets between receiver nodes, with overall coordination by the source. R-ODMRP has been implemented in ns-2 and run against a baseline of a dense network with increasing receiver count, ideal conditions for the base ODMRP protocol. Results show that R-ODMRP does outperform ODMRP under these conditions in terms of reliability, at an acceptable cost of an increase in routing efficiency and forwarding efficiency. The data delivery latency metric is expected to improve in future work, with fine tuning on the Resend Request /Reply protocol phases.

References

[1] K. Birman, "Building Secure and Reliable Network Applications", Manning Publishing Company, Greenwich, CT, and Prentice Hall, 1997.

[2] 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 the Fourth Annual ACM/IEEE International Conference on Mobile Computing And Networking, Dallas, TX, October, 1998.

[3] R. Chandra, V. Ramasubramanian, K. Birman, "Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks", Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, Arizona, April, 2001.

[4] K. Fall, K. Varadhan, editors, "The ns Manual", The VINT Project, UC Berkeley, LBL, USC/ISI, and XEROX PARC, April, 2002. Available at http://www-isi.edu/nsnam/ns/.

[5] T. Gopalsamy, M. Singhal, D. Panda, P. Sadayappen, "A Reliable Multicast Algorithm for Mobile Ad Hoc Networks", Proc. of the Dist. Cmptg Systems Workshops, pp. 563-570, Vienna, Austria, July 2002.

[6] S.J. Lee, W. Su, M. Gerla, Internet Draft, "On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks", draft-ietf-manet-odmrp-02.txt, January, 2000.

[7] S.J. Lee, W. Su, J. Hsu, M. Gerla, R. Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols", Proceedings of IEEE INFOCOM 2000, Tel Aviv, Israel, March, 2000.

[8] K. Obraczka, G. Tsudik, K. Viswanath, "Pushing the Limits of Multicast in Ad Hoc Networks", Proceedings of the 21st International Conference on Distributed Computing Systems, Phoenix, Arizona, April, 2001.

[9] L. Shu, D. Poppe, "Assuring Message Delivery in Mobile Ad Hoc Networks with Packet Erasure Recovery", Proceedings of the Distributed Computing Systems Workshops, pp 14-19, Vienna, Austria, July, 2002.

[10] K. Tang, K. Obraczka, S.J. Lee, M. Gerla, "A Reliable Congestion-Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks", The 5th Int'l Symposium on Wireless Personal Multimedia Communications, pp 252-256, Hololulu, USA, October, 2002.

[11] "The CMU Monarch Projects wireless and mobility extensions to ns", The CMU Monarch Project, August, 1999. Available at http://www.monarch.cs.cmu.edu/.

[12] The USC/ISI ns-2 version of ODMRP, previously available on Univ. of California website. Supported by NSF's NGE Program and the IMAHN Project. Copyright 1991-1997, Regents of the University of California.