Extensions for Multicast in Mobile Ad-hoc Networks (XMMAN): The Reduction of Data Overhead in Wireless Multicast Trees

Michael Edward Christman

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science in
Electrical Engineering

Dr. Scott F. Midkiff, Chair
Dr. Nathanial J. Davis, IV
Dr. Luiz A. DaSilva
Dr. Jahng S. Park

July 31, 2002
Blacksburg, Virginia

Keywords: Multicast, MANET, Wireless Mobile Networks

Extensions for Multicast in Mobile Ad-hoc Networks (XMMAN): The Reduction of Data Overhead in Wireless Multicast Trees

Michael Edward Christman

Dr. Scott F. Midkiff, Chair
Electrical Engineering

(ABSTRACT)

Mobile A*d hoc* Network (MANET) routing protocols are designed to provide connectivity between wireless mobile nodes that do not have access to high-speed backbone networks. While many unicast MANET protocols have been explored, research involving multicast protocols has been limited. Existing multicast algorithms attempt to reduce routing overhead, but few, if any, attempt to reduce data overhead.

The broadcast nature of wireless communication creates a unique environment in which overlaps in coverage are common. When designed properly, a multicast algorithm can take advantage of these overlaps and reduce data overhead. Unlike a unicast route, in which there is one path between a sender and receiver, a multicast tree can have multiple branches between the sender and its multiple receivers. Some of these paths can be combined to reduce redundant data rebroadcasts.

The extensions presented in this thesis are a combination of existing and original routing techniques that were designed to reduce data rebroadcasts by aggregating multicast data flows. One such optimization takes advantage of the multipoint relay (MPR) nodes used by the Optimized Link State Routing (OLSR) unicast protocol. These nodes are used in unicast routing to reduce network broadcast, but can be used to help create efficient multicast data flows. Additionally, by listening to routing messages meant for other nodes, a host can learn a bit about its network and may be able to make routing changes that improve the multicast tree.

This protocol was implemented as a software router in Linux. It should be emphasized that this is a real implementation and not a simulation. Experiments showed that the number of data packets in the network could be reduced by as much as 19 percent. These improvements were accomplished while using only a small amount of routing overhead.

Acknowledgements

I would like to thank all of those who made this thesis a success. First, I would like to express my appreciation to the Office of Naval Research (ONR), without whom, the Navy Collaborative Integrated Information Technology Initiative (NAVCITTI) project and, subsequently, this thesis would have never been possible.

A big "Thank You" goes to the members of the thesis committee, Dr. DaSilva, Dr. Davis, Dr. Park, and most of all, Dr. Midkiff. Without Dr. Midkiff's guidance and patience this research would never have been completed. Dr. Park's assistance with the construction of the test bed was also greatly appreciated.

Special thanks are extended to all the members of the lab, especially John Wells and Tao Lin, whose technical and moral support was crucial to the development and testing of the multicast router. Their seemingly infinite patience with the author's incessant nagging was greatly appreciated.

Finally, I would like to thank my friends and family, both here in Blacksburg and around the world, for their moral support during this most challenging of experiences. Without them I would have lost my sanity long before I even proposed the thesis topic!

Table of Contents

Table of Figures

Table of Tables

Chapter 1. Introduction

Traditional network routing techniques fall short when asked to provide mobile hosts with a reliable connection in a wireless environment. Wireless links allow for a high degree of mobility, but are problematic in that they support low data rates and have a limited range that can lead to frequent link failures. These two obstacles necessitate a new approach to routing protocols. An emerging class of networks, known as Mobile *Ad Hoc* Networks (MANET) [1], promises to provide connectivity among hosts in a highly volatile environment, while minimizing routing overhead.

As its name implies, a MANET is a network in which wireless mobile nodes operate independently of a backbone infrastructure. While hosts in traditional networks often rely on a designated router to forward data, every host in a MANET is required to route and forward data. Among other things, these networks rely on innovative network layer routing techniques to improve the reliability of the network.

This thesis discusses and attempts to improve multicast routing in MANETs. Whereas unicast routes provide connectivity between two end hosts and broadcast routes connect all hosts in a network, multicast routes provide a path between specific groups of nodes. Multicast routes allow for one-to-many, many-to-one, or many-to-many communication. Examples of multicast applications are video conferencing, chat rooms, electronic mail groups and online gaming.

Research into MANET routing protocols is relatively new. Although many fledgling unicast routing algorithms have been proposed, one has yet to be standardized. Most of the research on MANETs has focused on the problem of unicast routing, although a few multicast algorithms have been developed. Like unicast MANET routing protocols, multicast protocols attempt to maintain efficient and accurate routes while keeping routing overhead to a minimum. Efficient unicast routes provide the shortest path between hosts in terms of the number of hops. By ensuring that data traverses the shortest path available, unnecessary data rebroadcasts can be eliminated, and bandwidth can be conserved. In a multicast tree, the shortest path between a source and its receivers may not always be the most efficient route. An efficient multicast tree can reduce the total number of data rebroadcasts by aggregating data flows. This efficient route may not necessarily provide the shortest hop path between the source and receiver.

The thesis builds on existing multicast protocols and proposes a combination of preexisting and new optimizations that conserve data overhead. These extensions to the routing protocol attempt to reduce control overhead by taking advantage of existing routing mechanisms.

The following chapters describe the operation of existing multicast protocols, the new protocol, the implementation of the multicast router, and the results of the experiments involving this implementation. A brief description of existing unicast and multicast MANET algorithms immediately follows this introduction as Chapter 2. Chapter 3 presents in detail the problem that this thesis attempts to solve. Chapter 4 offers a solution to this problem. Chapters 5 and 6 describe the implementation of the multicast router and the results of the experiments, respectively. Unfortunately, at the time that the experiments were run no known MANET implementations with which to compare the proposed multicast protocol were available. Wired multicast algorithms would not work in a wireless environment without extensive modifications and could not be included in the experiments. Because of this, the experiments measured the relative effectiveness of specific proposed optimizations.

The optimizations presented in this thesis focus on the creation and maintenance of an efficient multicast tree. This is an approach that is foreign to most multicast protocols. It is the author's intention to not only present methods by which to build efficient multicast trees, but to also convince the networking community to take a new approach when developing multicast algorithms for wireless networks.

Chapter 2. Background and Prior Work

2.1. Introduction

Today's wired networks depend on high-speed backbone networks to quickly transfer large amounts of data to remote locations. While these backbone networks provide for the fast and reliable transmission of data over large distances, they confine the network to a fixed physical location. This thesis discusses new techniques that allow mobile wireless networks to act independently of backbone networks. Specifically, the development and implementation of a multicast algorithm for wireless mobile networks are presented.

2.1.1. Providing Mobile Wireless Services

Wireless networks do not share the robust and high-speed links enjoyed by their wired counterparts. Wireless connections have a small data carrying capacity, a relatively high error rate, and are unreliable when compared to traditional wired connections. Mobile *ad hoc* networks (MANET) [1] may be an adequate solution to the wireless networking problem. MANETs operate independently of a fixed backbone network, conserve bandwidth, and react quickly to changes in network topology. Other mobile or wireless protocols, such as the Mobile Internet Protocol (Mobile IP or MIP) [2] or typical deployments of IEEE 802.11 [3], depend on intermediate networks for connectivity between mobile nodes. Often, traditional wired networks are used to connect access points or home and foreign agents. Mobile IP networks require a backbone network in order to provide connectivity between mobiles nodes. This suggests that Mobile IP is not a solution for truly *ad hoc* networks. IEEE 802.11 specifications, including Wi-Fi or 802.11b specifications, define the PHY and MAC layer protocols. They do not directly influence network and other higher layer protocols and, therefore, do not provide routing between nodes that are not directly connected at the MAC layer. Nonetheless, IEEE 802.11 technology is becoming an increasingly popular way of providing link-level access to wireless hosts. Most incarnations require that nodes communicate directly with a static base station, which then uses a traditional network to provide connectivity between hosts. It is clear that neither the Mobile IP nor the current

Wi-Fi models provide the independence and autonomy that is inherent in a MANET wireless network.

MANET routing algorithms are developed under the assumption that nodes operate in a wireless and mobile network. This presupposes that links have low bandwidth, a high error rate, fail with a relatively high probability and do not have access to a high-speed backbone network. The network must have the ability to react quickly to link failures, and at the same time, it must reduce the amount of unnecessary routing overhead. Unfortunately, these are two diametrically opposed objectives. To preclude the need for a backbone network each node in a MANET acts like a router. Without a backbone network individual host-routers must have the ability to maintain routes and forward data to downstream nodes.

At last count, close to a dozen different MANET routing protocols have been proposed (none of which have been standardized) [4]. These algorithms attempt to reduce overhead and/or increase robustness in different ways. Several unicast (host-to-host) algorithms have been tested in code and made available. Despite the plethora of unicast algorithms, few MANET multicast routing protocols have been developed. Further, at the start of this project no MANET multicast implementations were known to exist. Because of the scarcity of work done on this subject, there is promise for progress in this field.

2.1.2. Uses for MANETs and Multicast

As was the case with the first networking protocols, research with MANETs has largely been motivated and funded by those with military applications in mind. One such project involves the integration of heterogeneous communication systems among an international naval task force. Much of the United States Navy's communication occurs via satellite. A United States warship is unique in this respect, as most other navies do not have persistent access to high bandwidth satellite systems. When the United States Navy wishes to work and communicate with a foreign vessel they are often loath to share access to highly sensitive satellite systems with a foreign military. A MANET may be a useful mechanism for communicating between ships without the benefit of satellite systems.

Fred Templin suggested on the IETF MANET mailing list that a MANET could be used to control a squadron of unmanned aerial vehicles (UAV) [5]. At present, UAVs are operated from remote locations by a team of operators. It is envisioned that a squadron of UAVs could operate autonomously and communicate by forming an airborne mobile *ad hoc* network. Preliminary field tests have been conducted using remote controlled helicopters equipped with a wireless network card. In general, MANETs are ideal for establishing communication in remote military and emergency situations, where a backbone infrastructure is not available.

Any military commander will agree that accurate information and the ability to communicate is imperative to the success of a military campaign. In his definitive work, *On War*, the Prussian general Carl von Clausewitz described a phenomenon that he called the "fog of war" [7]. He noted that even the most meticulous planning does not prepare an army for combat. In the heat of battle unpredictable events occur, communication breaks down, and chaos dictates the course of the battle. Multicast applications are by no means a solution to the "fog of war," but they can help to soften its effect by providing real-time communication among commanders in remote locations. Multicast applications include chat rooms, video conferencing and live video streaming. One can imagine a group of naval commanders planning an operation using a video conferencing application without having to leave their ships.

Non-military uses for MANETs are showing up in local coffee shops and apartment complexes. Neighborhood area networks (NAN) [6] are slowly becoming a popular way to share Internet access in high-density urban settings. In a NAN, a friendly Internet service subscriber shares their high-speed network connection, e.g., Digital Subscriber Link (DSL) or cable modem connection, by setting up a wireless gateway. Neighbors with wireless network cards can access the network with this gateway directly, or can find a route through other users with access to the gateway. Because of limits on range and power, one can imagine that an effective NAN would be dependent on a dense network, such as an apartment complex or shopping center. This network may not be mobile in the traditional sense, but changes in topology may occur as users power their home computers on and off. If a user does have a mobile unit, such as a laptop, even a

small change in location may cause a link failure due to the signal attenuation caused by walls or other obstructions found in buildings.

2.2. MANET Unicast Routing Protocols

MANET routing protocols can be classified as being proactive (or table driven) or reactive (or on-demand) or a combination of the two [8]. There are tradeoffs in routing overhead and route convergence time between reactive and proactive MANET protocols. Reactive protocols are generally more efficient than proactive algorithms in terms of routing overhead. However, low routing overhead often leads to longer link downtime and stale routes. To illustrate these issues, examples of a reactive and a proactive algorithm are presented. The Dynamic Source Routing (DSR) [9] algorithm is one of the earlier MANET protocols and is a reactive protocol. The Optimized Link State Routing (OLSR) [10] algorithm is an example of a proactive routing protocol.

Like the popular wired routing algorithm, Open Shortest Path First (OSPF) [11], OLSR is a link state algorithm. This means that routers broadcast their one-hop topology to the entire network, from which each node can create a map of the network. From this, a node can compute the shortest path route to every node in the network. OLSR is particularly important in this research. An implementation of OLSR provides the underlying routing mechanism for the development and testing of the multicast protocol presented in this thesis. It should be noted that OLSR was not originally chosen because of its superior properties, but because a simple implementation was readily available.

2.2.1. The DSR Protocol

DSR provides a simple example of how a reactive, or on-demand, protocol works. Many of the MANET multicast algorithms incorporate on-demand mechanisms when building a multicast source tree. A detailed description of the DSR protocol provides the reader with insight into the workings of on-demand MANET routing protocols.

Routing updates occur only when a route is needed. Imagine the situation where node A wants to send a data packet to node N, but does not have a valid route to the destination. In this case, node A begins a route discovery process by sending a ROUTE_REQUEST packet to every node in the network. Nodes that receive the packet

forward it to their neighbors unless they are the destination or they have a valid route to the destination. In both cases the node, say node N, drops the ROUTE_REQUEST packet and sends a ROUTE_REPLY packet back to the source. If the router can assume that the network is using a MAC protocol that requires bi-directional links, such as IEEE 802.11, and node N has a route to node A, the router then sends the ROUTE_REPLY packet via unicast to A. In the case that N does not have a route to node A the node can use the data in the ROUTE_REQUEST header to find a route to the source. As the ROUTE_REQUEST packet traverses the network, it adds the address of each intermediate node to its header. This behavior is associated with the technique of source routing. Node N can then simply reverse this sequence of addresses and add them to the header of its ROUTE_REPLY message. This sequence is illustrated in Figures 2.1 and 2.2.

If node N cannot assume that the links are bidirectional then the node must discover its own route to A through the use of a new ROUTE_REQUEST message. In this case the ROUTE_REPLY must be piggy backed on the new ROUTE_REQUEST message.

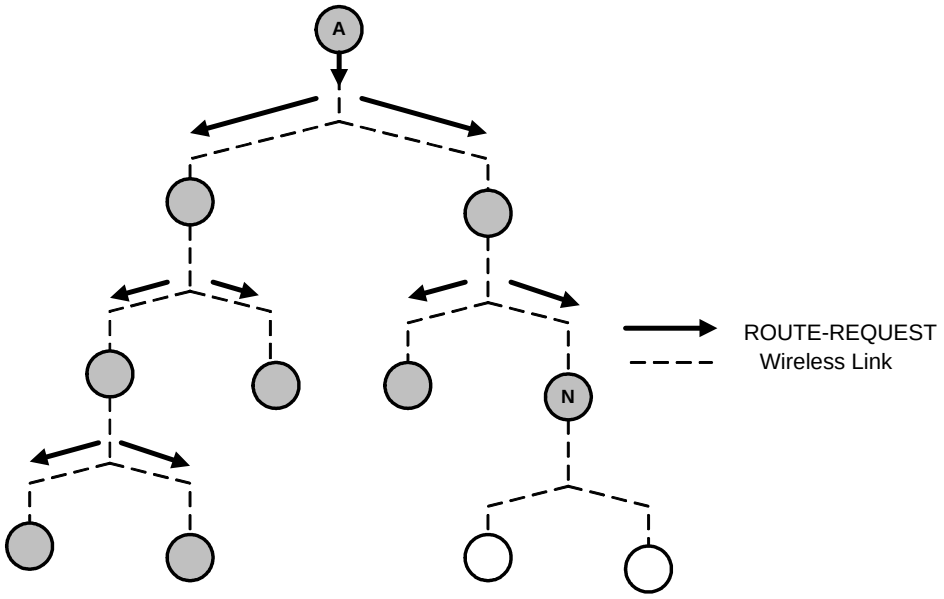


Figure 2.1. DSR Route Request flood where node A requests a route to node N.

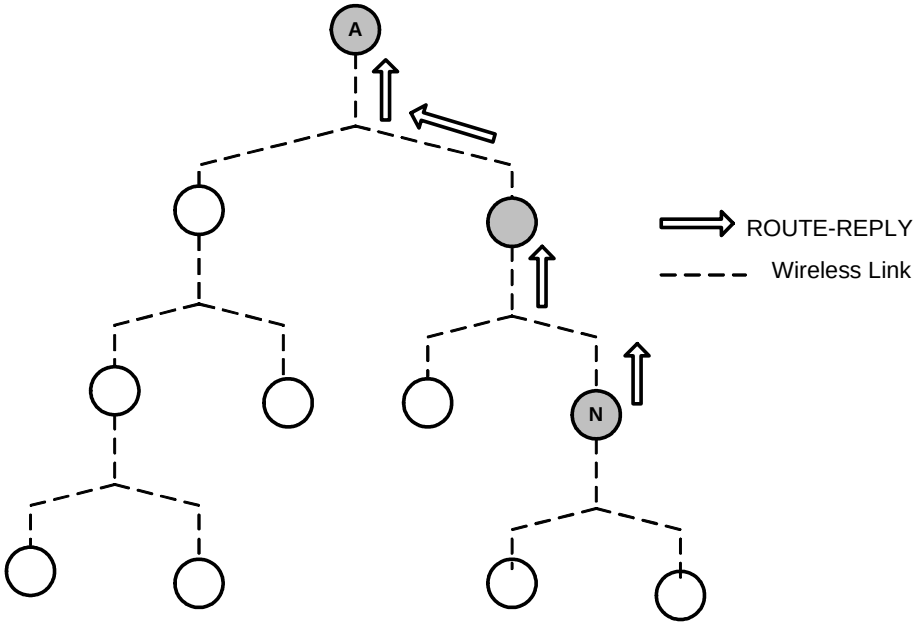


Figure 2.2. DSR Route Reply.

Through source routing, intermediate nodes can use the information in data packet headers to create their own routes. A DSR node that "overhears" a data packet can look into its header and use the source routing information to maintain its own routes. The DSR draft states that "the source route used in a data packet, the accumulated route record in a Route Request, or the route being returned in a ROUTE_REPLY MAY all be cached by any node" [9]. Exploiting this overheard information helps to reduce control overhead, but it should be noted that encouraging nodes to listen to packets may prove to be a security risk or, more likely, it may be necessary to remove the ability to "snoop" packets to ensure security.

Whenever possible, DSR reduces the number of acknowledgment packets by exploiting features of certain MAC layer wireless protocols. The DSR Internet draft states that the "confirmation of receipt [of a packet] in many cases may be provided at no cost to DSR, either as an existing standard part of the MAC protocol in use (such as the link-level acknowledgement frame defined by IEEE 802.11), or by a 'passive acknowledgement'" [9]. This passive acknowledgment occurs when a node overhears a downstream node forward its packet. Take for example the situation show in Figure 2.3. Here, node A is transmitting to node E. In this situation, node A can verify that B receives a packet sent by A if it overhears node B forward the packet to node C. If

neither MAC layer nor passive acknowledgment is available, network level acknowledgements are needed.

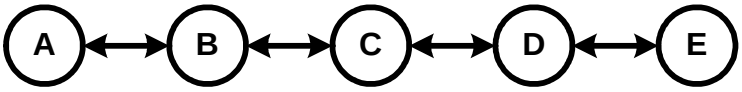


Figure 2.3. Network assuming 802.11 bi-directional links.

If a node is unable to acknowledge the receipt of a packet (e.g., due to a link failure), a ROUTE_ERROR packet is returned to the sender. Assume that the link between nodes D and E fails. In this case, node D cannot verify that node E has received the packet and sends a ROUTE_ERROR to A, as shown in Figure 2.4. If node A has another route to node E, it should send the data packet that way. If it does not have another route, it should perform a new route discovery, which requires node A to flood the network with a ROUTE_REQUEST packet.

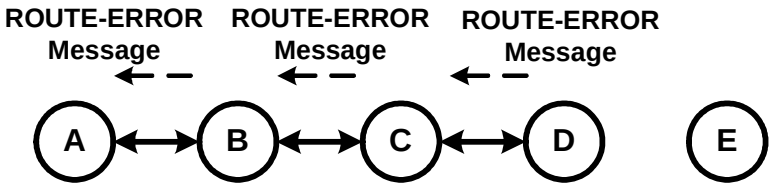


Figure 2.4. Route maintenance example.

If node D has another route to node E it can replace the original route in the packet with its own route and send the packet. A ROUTE_ERROR message with the new route to node E should be sent to node A. The DSR draft refers to this technique as "packet salvaging" [9].

Another way that DSR attempts to minimize overhead is by reducing the number of route reply "storms." A route reply storm occurs when many nodes send a ROUTE_REPLY in response to a ROUTE_REQUEST. As one might expect, some of the routes returned may be better than others in terms of the number of hops, delay, etc. To prevent redundant or inferior routes, DSR requires that nodes wait a random time *t* before replying to a ROUTE_REQUEST. The random time *t* is defined as:

$$t = H \times (h - 1 + r)$$

Here, h is the number of hops of the returned route, H is a small constant delay greater than at least twice the maximum wireless link delay, and r is a random number between 0 and 1. This ensures that all nodes with a shorter route to the source respond before this node, and that all nodes with a larger hop count respond after this node. While the node is waiting to send the ROUTE_REPLY, it listens for data packets from the source to the destination node. If it overhears a data packet with a source route length less than or equal to h, the node can assume that the source has already received an equally good, or better route. In this case, the node cancels its ROUTE_REPLY.

It is clear that by creating routes only when explicitly needed, reactive protocols such as DSR reduce the routing overhead in the network. On the other hand, reactive protocols generally show slow rates of convergence. Reactive routing protocols work well when mobility is low, but as the number of link failures increase performance decreases drastically. Rather than actively checking the status of the network, reactive protocols wait until a link failure is detected before updating the routing table. This can cause long periods of down time while the source searches for a new route.

It has been shown that the Transmission Control Protocol (TCP) behaves poorly in mobile *ad hoc* networks [12]. In a wired network packet loss is often the result of congestion, while in a wireless network dropped packets are more likely caused by bit errors or stale routes. Most TCP implementations assume that congestion causes packet loss and TCP shrinks its sending window size to reduce the amount of traffic on the network. A long route convergence time, as might be expected in some environments with a reactive routing protocol, can lead to unnecessary reductions in TCP's window size. Short route convergence times, such as those found in proactive protocols, can avoid this problem. If the network is able to recover from a routing failure quickly, a destination node may be able to respond with a TCP acknowledgment before TCP causes the sender to time out and reduce its sending window size. If it were possible to fix routing failures before nodes needed to actually send data this issue could be avoided altogether.

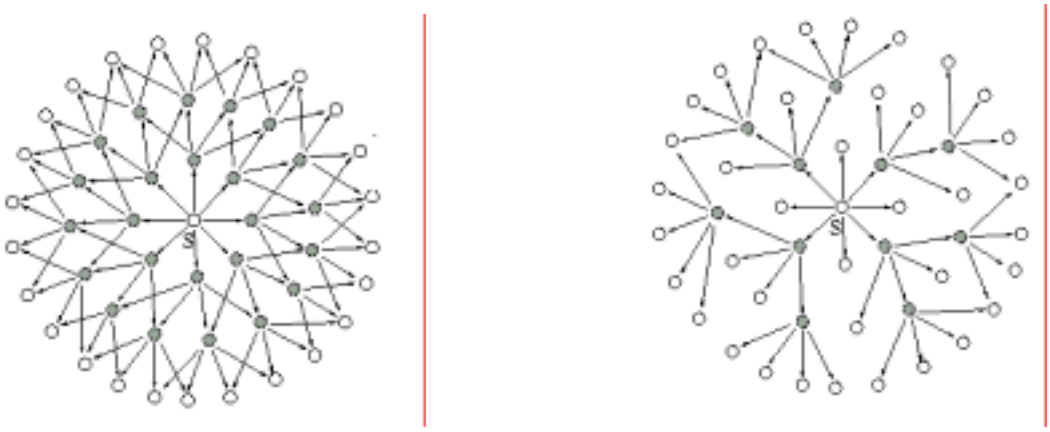
2.2.2. The OLSR Algorithm

The approach of fixing routing failures before the route is actually used is the philosophy that drives proactive MANET routing protocols. At the expense of additional overhead, proactive routing algorithms try to fix routing errors before they become a problem. These algorithms continuously probe the network, which allows nodes to find new routes and ensures that stale ones are removed from the routing table. In short, proactive routing protocols attempt to create a full and up-to-date routing table, regardless of the need for the route.

Proactive MANET routers maintain their current state through the use of periodic beacons, commonly referred to as "HELLO" messages. Nodes broadcast these packets to inform one-hop neighbors of their status and to provide state information to their neighbors. Like OSPF, OLSR is a link state algorithm. Link state algorithms require that routers tell the entire network about their immediate neighbors. Using this information, routers are able to create a map of the full network topology. From this, routers perform Dijkstra's algorithm to find the shortest path tree [13].

While the frequent flooding of control messages helps routers to discover routes, it can consume a significant amount of network bandwidth. This is especially true in a mobile network, where broadcasts need to occur frequently in order to paint an accurate picture of the network. OLSR addresses this issue by reducing the number of nodes that forward data during a network-wide broadcast.

The OLSR protocol uses multi-point relay (MPR) nodes to reduce the number of unnecessary forwarding of network broadcasts. During a network flood only those routers selected as MPR nodes actually forward data. Figure 2.5 illustrates how a network using MPR nodes reduces unnecessary rebroadcasts. These figures were taken from [14].



(a) (b)

Figure 2.5. Broadcast (a) without MPR nodes and (b) with only MPR nodes forwarding data. Shaded nodes are MPR nodes [14].

To inform neighbors as to its status, an OLSR node periodically sends a HELLO message to its one-hop neighbors. Each HELLO message contains a list of the source's one-hop neighbors. Once a node receives a HELLO message from all of its neighbors it has knowledge of every node within a two-hop radius. From its list of one-hop neighbors a node selects the set of MPR nodes such that it can reach every two-hop node through at least one of the MPR nodes. Rebroadcasts are then optimized within a two-hop radius.

OLSR nodes periodically flood the network with topology control (TC) messages that contains a list of the source node's MPR selector set. A MPR selector set is the list of addresses that have chosen the node as a MPR. From the information in TC messages, nodes in the network can construct an accurate picture of the network topology and then use the shortest path algorithm to determine their routing tables.

2.3. Multicast Protocols for Wired Networks.

It is not unusual for a wired multicast algorithm to depend on a specific underlying unicast routing mechanism. For example, Multicast OSPF (MOSPF) [15] is built on top of OSPF. By integrating multicast functionality with a specific routing protocol, a multicast router can exploit the advantages gained by the specific unicast algorithm. Like unicast algorithms, multicast protocols can sometimes be categorized as being either a link state or distance vector algorithm. Examples of link state and distance

vector multicast implementations are Multicast OSPF (MOSPF) [15] and the Distance Vector Multicast Routing Protocol (DVMRP) [16], respectively. A third type of multicast algorithm does not depend on a specific routing protocol. This algorithm is aptly named Protocol Independent Multicast (PIM). Two incarnations of PIM exist, sparse mode (PIM-SM) [17] and dense-mode (PIM-DM) [18]. Discussion in this thesis is limited to PIM-SM since this is the protocol that is most commonly used. It is important to discuss multicast algorithms for wired networks for two reasons. The first reason is to provide the reader with a better understanding of the foundations upon which MANET multicast algorithms have been built. The second reason is that in order to make accurate comparisons between MANET and traditional algorithms, one must understand the operation of the wired protocols. A good overview of these algorithms is given by Peterson and Davie [19], which was the basis for the following descriptions.

2.3.1. Link State Multicast (MOSPF)

Multicast functionality is relatively simple to integrate into an existing link state algorithm. Each router has enough information to create a complete picture of the entire network. From this point it can use Dijkstra's algorithm to construct the shortest path tree. As mentioned before, a link state algorithm floods the network to disseminate routing information. When additional information is included in these messages to indicate which networks wish to receive multicast traffic, it is simple for routers to compute the multicast tree. Routers need only communicate with hosts on their network to determine from which multicast groups they wish to receive traffic.

2.3.2. Distance Vector Multicast (DVMRP)

Integrating multicast in a distance vector routing protocol is bit harder to do because routers do not have complete knowledge of the network. Recall that routers only know the next hop and cost to a destination. DVMRP uses a method called Reverse-Path Multicast (RPM) to build the multicast tree. A DVMRP node that sends multicast traffic before the multicast tree has been created broadcasts this data to every node in the network. Only after data has been sent are branches pruned from the broadcast tree to reveal the multicast tree. Pruning begins when a leaf node (a node without any

downstream links) determines that it does not wish to receive multicast traffic. The router then tells its upstream node that it does not wish to receive data. The upstream router then stops forwarding multicast traffic to that node. If the upstream router does not wish to receive multicast data, it informs its upstream node, which in turn stops routing to its downstream node. This sequence iterates towards the source until it reaches a router that wishes to receive multicast traffic or it reaches the source.

Putarsti states some of the advantages of DVMRP [16].

a) Pure source specific multicast distribution trees provide a simple model to deploy and troubleshoot.
b) Join latency is minimized since new sources automatically send data to all receivers.
c) DVMRP builds a broadcast tree per source network with the routing updates, so it does not have to flood links that are not part of the broadcast tree for a source.
d) DVMRP uses its own topology discovery mechanism allowing both faster adaptation to change and a more stable steady state.

Small join latency, quick adaptation to network changes and eliminating unnecessary network floods are important when building a system for a mobile wireless network. However, Putarsti also notes that DVMRP is unsuitable for low-data rate networks. He states, appropriately, that the initial network flood consumes a large amount of bandwidth. Clearly, this inefficiency precludes DVMRP from being a good candidate for a wireless mobile *ad hoc* network.

2.3.3. Protocol Independent Multicast (PIM)

As the name implies, PIM algorithms do not rely on an underlying routing algorithm and were developed in response to the scaling problems found in networks. Routers use PIM protocol messages to explicitly join or leave multicast groups. These messages are sent by multicast receivers to rendezvous point (RP) routers indicating whether a node wishes to join or leave a group. Each group is assigned an RP and this node acts as an intermediary between the source and the group. As receiver nodes send join messages to the RP, intermediate nodes mark in their routing tables which link to forward multicast data for the specific group. In this way, a source-independent multicast

tree is built with the RP as the root. Multicast senders then tunnel multicast traffic to the RP, which in turn, forwards the data down the multicast tree. Clearly this method is inefficient as a shorter route may exist between a direct Source ↔ Destination pair than through a Source ↔ RP ↔ Destination route.

The network topology in Figure 2.6 shows an example of this inefficiency. In this case the multicast source S is sending data to the destination nodes D1 and D2. The original multicast tree is indicated by the solid lines and shows that S must send data to the RP via node N2. The RP then forwards the traffic to N1 and the destination nodes. However, a more direct link exists between S and N1, as indicated by the dashed line. An efficient multicast tree would have S send data directly to N1. Fortunately, PIM-SM allows for such optimizations when a specific source sends a sufficient amount of multicast traffic.

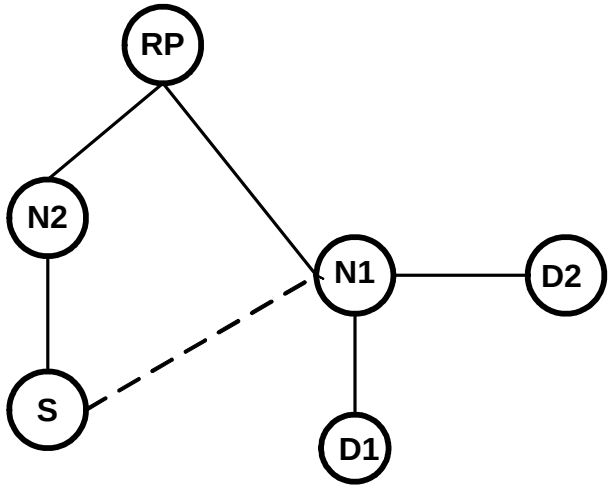


Figure 2.6. Example PIM-SM multicast tree.

Not only does the use of the RP node create inefficient multicast routes, but it also reduces the stability of the network in the case of a link or node failure. The failure of a central node upon which many multicast trees depend would cause more severe disruption than if the responsibility were distributed. This is especially important in a MANET, where link and node failures can occur fairly regularly.

2.4. MANET Multicast Algorithms

Existing MANET multicast routing algorithms use a hybrid of link state and distance vector algorithms to create a source tree. As mentioned before, multicast is a relatively new topic in the field of MANETs, and only a few algorithms have been proposed. Examples of some MANET multicast algorithms are Multicast AODV (MAODV) [20], On-Demand Multicast Routing Protocol (ODMRP) [21], Multicast Zone Routing (MZR) [22], Multicast OLSR (MOLSR) [23], Ad hoc Multicast Routing (AMRoute) [24], Ad hoc Multicast Routing Protocol utilizing Increasing id-numbers (AMRIS) [25], Core-Assisted Mesh Protocol (CAMP)[26], Adaptive Demand Driven Multicast Routing (ADMR) [27], Lightweight Adaptive Multicast Algorithm (LAM) [28], Differential Destination Multicast (DDM) [29], and Multicast Core Extraction Distributed Ad-Hoc Routing (MCEDAR) [30]. Among the most mature are MAODV, ODMRP, MOLSR and MZR. Lee, Su, Hsu, Gerla and Bagrodia [31] make comparisons between some of these protocols and provide a brief overview of the operation of the algorithms. The following subsections provide a description of MZR, MOLSR, ODMRP, AODV and ADMR.

2.4.1. Multicast Zone Routing (MZR)

The MZR protocol [22] is based on, but is not dependent upon, a specific routing mechanism. It takes into consideration the hierarchical structure used by the Zone Routing Protocol (ZRP) [31]. A ZRP network is partitioned into zones. Each node computes its own zone, which is determined to be the set of nodes that lie within a certain zone radius of the node. Devarapalli [22] describes zone routing as “a hybrid approach between the proactive and reactive routing protocols, where routing is proactive inside the zones (i.e., a unicast route is proactively maintained between every pair of nodes in a zone) and reactive between the zones (i.e., a route between two nodes in different zones is created when needed).” He goes on to exude the benefits of zone routing in his statement that “this hybrid approach provides a balance between the efficient packet delivery of proactive routing and the low maintenance cost of reactive routing.”

Routes within a zone are proactively maintained using a distance vector algorithm. To create a zone, a MZR node A broadcasts an ADVERTISEMENT message

with a time-to-live (TTL) equal to ZONE-RADIUS, or the radius of the zone. A node B within the zone radius decrements the TTL and forwards the message if appropriate. Node B makes an entry in its routing table for node A, with the last hop of the ADVERTISEMENT message as the next hop towards the destination, i.e., the source of the ADVERTISEMENT message. The distance is set to the hop count of the packet. Nodes that are ZONE-RADIUS hops away from node A become border nodes, and play a special role in zone routing and multicast zone routing. Border nodes serve as a gateway between node A's zone and the rest of the network.

In the spirit of zone routing, MZR begins its search for a multicast tree within the zone before extending the search outward. MZR is a source-specific algorithm, meaning that a multicast tree is created for each source-group pair. When a source wants to start sending multicast traffic it initiates the construction of a multicast tree. The source node sends a TREE-CREATE packet to each node in its zone. Nodes that receive this message and want to receive multicast data respond with a TREE-CREATE-ACK packet. A TREE-CREATE-ACK packet is sent back to the source of the multicast session and, as the packet travels up the tree, intermediate nodes mark in their routing tables the last hop of the TREE-CREATE-ACK as a downstream node. In this way a reverse multicast path is created. When a source finishes the creation of a multicast tree within its immediate zone it unicasts a TREE-PROPAGATE to its border node. This indicates to a border node that it should create a multicast tree within its own zone. Tree creation continues in the new zone in the same way as described above. Upon receipt of a TREE-CREATE-ACK, the border node unicasts a TREE-CREATE-ACK to the multicast source. This creates a link between the border node and the source. This sequence continues until every node in the network receives a TREE-CREATE message. Devarapalli suggests that a flag could be set in a TREE-CREATE message that would tell border nodes to initiate a multicast tree search in their zones. This would eliminate the need for a TREE-PROPAGATE message.

Routes in MZR are updated through the use of TREE-REFRESH packets. These packets are periodically sent by the source to its multicast receivers indicating that the source still has data to send. If a node on the multicast tree fails to receive a TREE-REFRESH message after a certain time it deletes its multicast entry. Devarapalli states

that TREE-REFRESH packets should be piggybacked on multicast data whenever possible.

Zone routing performs well when a link failure occurs. If a downstream node detects a link failure and it is still interested in the multicast session, it initiates branch reconstruction by sending a JOIN packet to all the nodes within its zone. If a node within the zone has a route to the multicast source it responds with a JOIN-ACK. A new route between the lost node and the source of the JOIN-ACK is created in a way similar to the initial multicast route creation method. If a search within the zone fails to produce a route, the lost node sends a JOIN-PROPAGATE to its border nodes, which in turn, look for a route within their zones. If they find a route, they respond with a JOIN-ACK to the lost node. If not, they continue the search with a JOIN-PROPAGATE to their border nodes. Essentially, if a route is not found within the lost node's zone, the search for a route is propagated throughout the entire network. However, if a route is found within the confines of the lost node's zone, the search is limited to those nodes and bandwidth is conserved.

Tree pruning is a relatively simple process. A node N that wishes to leave a multicast group sends a PRUNE message to its upstream nodes. If node A is an upstream node of node N and node N is node A's only one-hop downstream node, node A will then stop forwarding multicast traffic. If node A does not want to receive multicast data itself and it does not have any other downstream nodes it sends a PRUNE message to its upstream node. This continues until the PRUNE reaches a node that wishes to receive multicast traffic or it reaches the source node. Nodes that wish to join an existing multicast session can perform a JOIN in the same way that a lost node does.

One advantage of the MZR protocol is that it creates a source specific, on-demand multicast tree with a minimal amount of routing overhead. Multicast zone routing attempts to reduce the amount of overhead incurred in route maintenance by preventing routing updates from spreading unnecessarily throughout the network. It would seem that the tradeoff in complexity and routing overhead incurred by the zone routing mechanism does not necessarily offset the advantages presented by MZR. It seems as if zone routing is advantageous when attempting to route unicast packets, but not necessarily when creating multicast source trees.

Clearly, the hierarchical approach of MZR does not conserve bandwidth during the initial TREE-CREATE flood. In fact, MZR can introduce extra latency when a TREE-CREATE flood occurs and TREE-PROPAGATE messages are used. MZR requires that multicast tree creation beyond the source's immediate zone occur only after the intra-zone multicast tree has been created. Tree creation latency could certainly be reduced if foreign zones did not have to wait for the source zone to complete its tree before creating their own multicast trees.

It seems as if the advantages gained through the use of zones may be accomplished through simpler means. The MZR algorithm does have the advantage of limiting multicast re-joins within the zone. But if a node must look outside its zone for a new route the entire network is flooded. Only when a new route lies within ZONE-RADIUS hops from the lost node is bandwidth conserved. This situation may be common in the case of link failures, but not in the case that a node wishes to join an existing multicast tree for the first time. Instead of using MZR, a node in need of a new multicast route could simply send an initial JOIN message with a small TTL, i.e., on the order of a zone radius. If after a certain time this JOIN does not produce a valid route, i.e., it does not receive a JOIN-ACK message, it could resend the JOIN with a larger TTL.

2.4.2. Multicast Optimized Link State Routing (MOLSR)

As in MZR, the multicast extension of OLSR [23] creates a source specific multicast tree. Unlike MZR, MOLSR is dependent on OLSR as an underlying unicast routing algorithm. Multicast-capable routers in an OLSR network periodically advertise their ability to route and build multicast routes with a MC_CLAIM message. This message carries no information and is only used to declare the router's capabilities to the network. MC_CLAIM messages are sent to every node in the network every MC_CLAIM_PERIOD seconds. Because the information in a MC_CLAIM message does not change over time, a relatively long MC_CLAIM_PERIOD should be used. Using this and the information provided by the TC messages, MOLSR nodes can calculate shortest path routes to every potential multicast source. This is done in the

same manner seen in OLSR, except that now the routes consist entirely of multicast - capable OLSR routers.

Multicast routes are built in a backward manner that is similar to the method used in MZR. A source that wants to send multicast traffic advertises its intentions by broadcasting a SOURCE_CLAIM message to every node in the network. Before responding to the SOURCE_CLAIM, a multicast receiver first checks its multicast routing table. If an entry does not already exist, the node creates one and sets the timer to the SOURCE_HOLD_TIME and the list of child nodes to null. The node then sets the parent node to the next hop towards the multicast source, as determined from its multicast routing table, and sends a CONFIRM_PARENT to the parent node. If an entry does exist the node simply updates the timer, and does not send a CONFIRM_PARENT message to its parent. In either case, if the node is an MPR node for the last hop it forwards the SOURCE_CLAIM.

When a node receives a CONFIRM_PARENT message it checks its multicast routing table for an entry. If the entry does not exist, it creates one and sets the sons and parents to null. If the last hop of the CONFIRM_PARENT packet does not exist in the sons list it adds it and updates the son timer to SON_HOLD_TIME. If the son does exist it simply updates the SON_HOLD_TIME. The node then sets the parent address to the next hop in the multicast routing table to reach the source, as determined by the multicast routing table.

A multicast source periodically sends a SOURCE_CLAIM message to every node in the network. The reason for this is twofold. First, it informs multicast receivers that the source is still sending data and that all of the nodes in the multicast tree should update their multicast timers. Secondly, it allows unattached hosts to join the multicast group. If a node detects that the next hop entry towards the multicast source has changed, a node must inform the new entry that it is now a multicast parent. To do this the node must send a CONFIRM_PARENT message to the node. If the old parent is reachable the node may send a LEAVE message to the old parent to disable the route. A node periodically sends out CONFIRM_PARENT messages to inform its parents that it still wishes to receive multicast traffic.

A MOLSR node that wants to leave a multicast group and that has no sons sends a LEAVE message to its parent. The parent removes this node from the son list and, if the list becomes empty, it sends a LEAVE message to its parent. This continues until a node that wants to receive data or a node with at least two sons is reached. As mentioned above, a node may send a LEAVE message if a change in network topology causes a change in the multicast tree. Jacquet, *et al*. [23] also discuss the case when a node that does not have multicast capabilities sends multicast data. The research described in this thesis assumes that all nodes have multicast capability and does not consider a situation when this is not true. Nonetheless, this is an interesting topic, and it should be noted that Jacquet, *et al.* [23] present a proposal for a new Wireless Internet Group Management Protocol (WIGMP).

2.4.3. On-Demand Multicast Routing Protocol (ODMRP)

ODMRP [21] is an on-demand protocol, meaning that nodes maintain state information only when a multicast session is active. In this way, ODMPR is able to reduce overhead when information about the multicast tree is not needed. A network of ODMRP nodes form a mesh of hosts that forward multicast data. This group of nodes is called a Forwarding Group (FG) and the protocol that uses a FG is called the Forwarding Group Multicast Protocol (FGMP). There are two versions of FGMP. In one, the creation and maintenance of the FG is rooted at the source. The second version is rooted at the receiver. These protocols are referred to as FGMP-SA and FGMP-RA, respectively, and differ only in which node is the root of the session. Multicast sessions more often than not have more receivers than sources, so the source-based algorithm is typically be used. ODMRP is a specific case of FGMP and can be rooted at the source or at the receiver.

ODMRP is a proactive algorithm and a sender (or receiver) that wants to send to (or join) a multicast session periodically floods the network with a JOIN-DATA packet. This message advertises the multicast session in which the node is interested in joining and contains:

- Multicast Group ID
- Sender (or Receiver) ID

- Sequence Number
- TTL
- Last Hop ID
- Hop Count

Nodes that receive a non-duplicate version of this packet store the upstream node ID and rebroadcast the packet. Nodes that receive this packet and wish to join, or are already a part of the group respond by broadcasting a JOIN-TABLE message to its neighbors. Each JOIN-TABLE message contains a list of multicast senders (or receivers) and their next-hop pairs. If a node receives a JOIN-TABLE and finds its own ID in the list of next-hop nodes, it then knows that it is the next hop towards the source (or receiver) and that it should send a new JOIN-TABLE message to its neighbors with a new list of [source(or receiver), next hop] pairs. This list is compiled from the information provided from the received JOIN-DATA messages. As this procedure iterates to the source of the JOIN-DATA packet, the multicast tree is built and multicast routes are created in a backward manner.

Nodes update the tree by periodically broadcasting JOIN-DATA packets. This allows nodes to leave a multicast tree without explicitly notifying the group. A node that no longer wishes to receive multicast traffic can simply refuse to reply to a JOIN-DATA message and allow its upstream nodes to time out. ODMRP is unique in that it does not require the presence of any underlying unicast protocol. One-way unicast routes can be formed through the use of JOIN-DATA and JOIN-TABLE messages in the same way that the multicast tree is created. Figure 2.7 shows an example of a how Forwarding Group might look in an ODMRP network. It also shows the forwarding tables of a network some of the nodes. The figure was taken from [21].

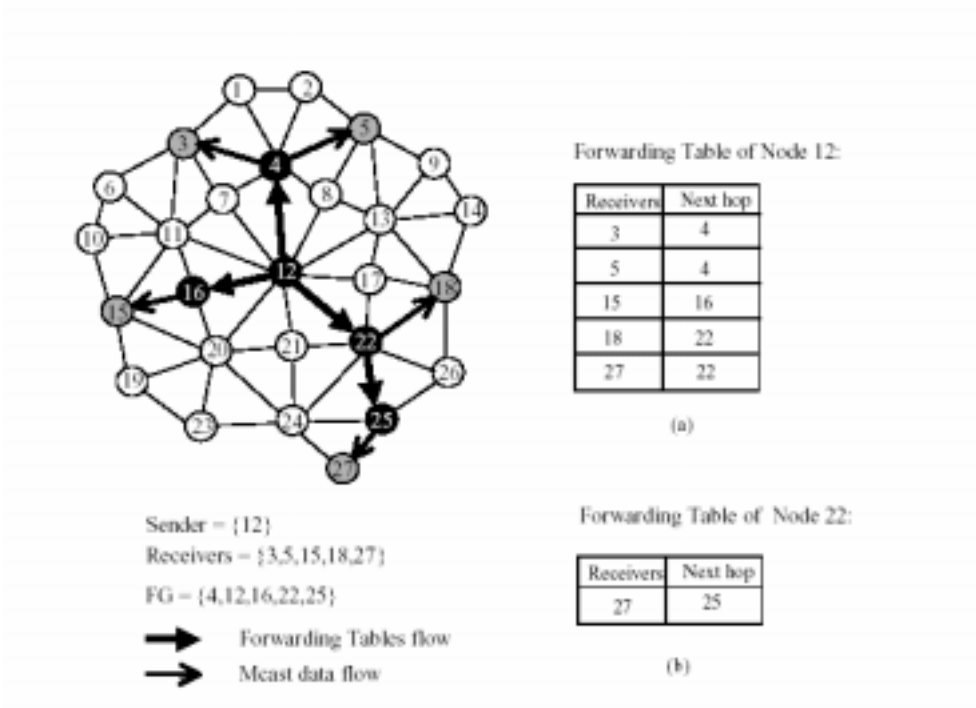


Figure 2.7. Example of forwarding tables for ODMPR (from [21]).

2.4.4. Multicast AODV (MAODV)

The MAODV protocol [20] is similar to ODMRP in that they both build multicast routes through backward learning. That is, a sender initiates a multicast session by flooding an advertisement to the entire network and routes are built as the response to the advertisement makes its way back up the tree towards the source. However, unlike ODMRP, MAODV is not proactive. Route maintenance in MAODV is performed locally, and as such, routing overhead is reduced. One advantage of MAODV is that it is a simple extension of the unicast protocol AODV [32]. Like MOLSR, MAODV effectively reduces overhead by incorporating both unicast and multicast functionality.

When a node wishes to join or send data to a multicast group and does not have a valid route to the group, it floods the network with a Route Request (RREQ) message. If a node wishes to join a group, i.e., it wants to receive data but does not want to send data, it sends a RREQ with the *J_flag* set, otherwise the flag is left unset. MAODV uses the concept of a group leader and if the node is aware of the group leader it unicasts, rather than broadcasts, the RREQ to the appropriate node. A group leader is usually the first node in the group and serves as the root of the tree and provides additional functionality.

The use of the group leader in MOADV is similar to the Rendezvous Point (RP) used in the PIM multicast algorithms and may cause some inefficiencies in the multicast tree.

A node that receives a *join* RREQ and is a member of the multicast group or a node that receives a RREQ and has a valid route to the multicast group responds with a Route Reply (RREP) message. Otherwise, the node rebroadcasts the RREQ. If the node that sent the original RREQ does not receive a reply before a timer expires it rebroadcasts the RREQ (or *join* RREQ). If a node does not receive a RREP after RREQ_RETRIES attempts it assumes that there are no members within its immediate network. The node then stops sending RREQ messages and becomes a multicast group leader. If the node unicasts a RREQ to the group leader and the sender does not receive a RREP before a time out, the sender assumes that either the destination is no longer the group leader or that the group leader is no longer reachable. In either case, the node proceeds to broadcast all subsequent RREQ messages. As RREP messages make their way back towards the source of the RREQ, reverse routes are created and the multicast tree is formed. Figure 2.8 illustrates an example of a MAODV RREQ and RREP sequence. This figure was taken from [20].

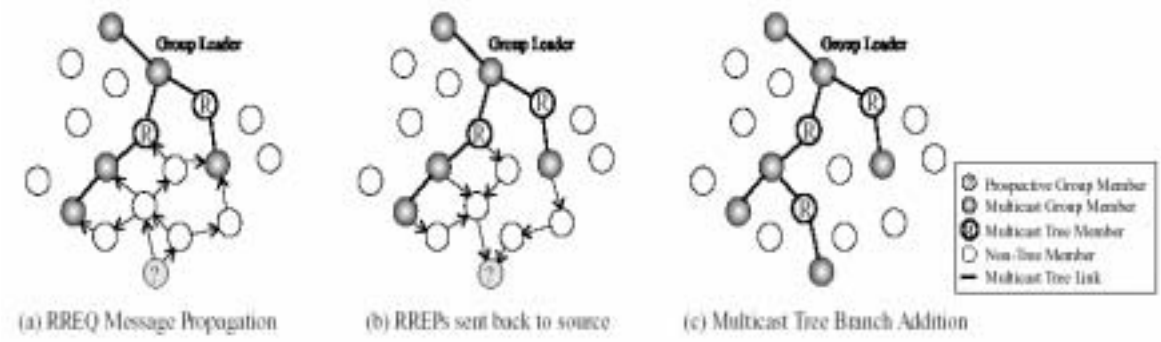


Figure 2.8. Example of a multicast join operation [20].

Because a node may receive a number of RREP messages in response to the RREQ message that it sent, it waits a certain time before choosing an upstream node. After this timer has expired the node chooses the best route in terms of number of hops to the source and unicasts a Multicast Activation (MACT) packet to the appropriate node. If the recipient of the MACT is not already a member of the multicast group it checks for

the best route (from the list of RREP messages) and responds with a MACT to its upstream node. This continues until the originator of the RREP receives the MACT.

The multicast group leader is responsible for maintaining the multicast group sequence number and for disseminating this number to the members of the multicast group. A group leader does this through the use of periodic Group Hello messages. This is an unsolicited RREP message that is sent to the entire network. This message informs nodes that are not a part of the multicast tree which node is the group leader and allows members of the group to update their multicast group sequence number. The multicast group sequence number is used to ensure that nodes only react to current routing updates. The Group Hello can also be used to choose a new multicast group leader if two previously partitioned networks merge.

A downstream node detects a link failure when it fails to receive a packet from its upstream neighbor before a timer expires. Downstream nodes react to link breaks by sending a RREQ message with the *J-flag* set. Along with the multicast session ID, the number of hops to the multicast group leader is included in the message. Because a node with a path to the multicast group leader is likely to be nearby, the node sends the RREQ with a TTL of one. If within a certain period of time the node does not receive a RREP it increments the TTL and resends the RREQ. If, after RREQ_RETRIES, the node has not received a valid RREP, the node assumes that the network has been partitioned and that the multicast group leader is unreachable. In this case the nodes in the recently partitioned network, i.e., the node and its downstream children, renegotiate a new multicast group leader. Only those nodes that have a valid route to the multicast group and have a hop count less than or equal to the multicast group leader may reply to the RREQ. The hop count field prevents nodes downstream of the link failure from responding and providing, now faulty, routing information.

2.4.5. Adaptive Demand Driven Multicast Routing (ADMR)

The ADMR protocol [27] and the multicast version of AODV are similar in that they are both source-based, on-demand, multicast protocols. Like ODMRP, the ADMR protocol works independently of any underlying unicast algorithm. One feature of ADMR is that nodes are capable of creating or joining a source-specific multicast group.

A session that uses the traditional multicast model [33] is only dependent upon the multicast group address. A new source-specific model [34] allows nodes to join, leave, and send to a specific [source, group] session.

A node that is using ADMR and wants to begin sending multicast traffic floods the first multicast data packet to the entire network. This is done by adding an ADMR header with the *network flood* flag set. If the *network flood* flag is set, the message is sent to every node in the network. Conversely, if the *tree flood* flag is set, the packet is sent to every node in the multicast tree. The sending node then buffers any subsequent data packet until it receives a valid response from a potential multicast receiver. A valid response comes in the form of a RECEIVER JOIN packet from a node that wishes to join the multicast group. A multicast receiver that is not yet part of the tree and has not received an advertisement from this source sends a RECEIVER JOIN up the reverse path that the original advertisement took. As the RECEIVER JOIN makes its way from the receiver to the source, intermediate routers mark in their routing tables the last hop of the RECEIVER JOIN. During the original network flood nodes mark the last hop of the advertisements as their upstream node. From these two pieces of information nodes are able to create multicast routes.

Multicast senders periodically flood a data packet to the entire network so that receivers who, for some reason, have not received a multicast advertisement may join the group. A node that lies between the receiver that initiated the RECEIVER JOIN and the multicast source may receive multiple copies of the RECEIVER JOIN from several multicast receivers. Because there is no guarantee that the RECEIVER JOIN will actually reach the multicast source, intermediate nodes forward up to three RECEIVER JOIN messages for a given multicast [source, group] pair.

During the delivery of multicast data the source includes in the ADMR header an *inter-packet time* interval that notifies downstream nodes of how often they should expect data packets. If the application layer at the source stops sending data packets, the source sends a KEEP-ALIVE message to the multicast tree. The interval at which these messages are sent is multiplied my some multiple until a maximum interval is reached, at which point, the node assumes the application is done sending data. If a downstream node does not receive data or a KEEP-ALIVE message within a multiple of the keep-

alive time it assumes that the source has finished sending data and the node will leave the multicast tree.

Nodes may proactively join a group by sending a MULTICAST SOLICITATION message to the entire network. When a multicast source receives a MULTICAST SOLICITATION it replies in one of two ways. If the source has received several MULTICAST SOLICITATION messages within a short period it may choose to advance the time of the next network flood session advertisement. The source may also respond by sending an ADMR KEEP-ALIVE message down the reverse path that the MULTICAST SOLICIATION took to reach the source. After the multicast receiver receives the reply from the source it responds with a RECEIVER JOIN message, thus completing the three-way handshake and activating the multicast routes.

A node detects a link failure when it fails to receive a multicast data packet or a KEEP-ALIVE before the node's *disconnection timer* expires. The value of the timer is based on the inter-arrival time of data and the number of hops that the node is from the source. The further that the node is from the source, the longer its *disconnection timer*. This prevents downstream nodes from attempting to fix a broken link before an upstream node can complete its repairs. Once a node detects a disconnection it sends a REPAIR NOTIFICATION message to its downstream nodes. This informs the downstream nodes that a link repair is underway and that they should not respond to a link repair request. After sending its REPAIR NOTIFICATION packet, the node waits a *repair delay* period of time before proceeding with its repair. If during this time the node receives a REPAIR NOTIFICATION from its upstream node it will not proceed with the link repair. The receipt of this message notifies the node that the actual problem is further upstream and that it should not attempt to repair the link. If, after the delay, the node has not received a REPAIR NOTIFICATION from an upstream node, it floods the network with a RECONNECT message. A node that has received a RECONNECT packet and has not received a REPAIR NOTIFICATION assumes that it is not downstream of the link failure, i.e., its route to the source is still valid and it then unicasts the RECONNECT message to the source. If the source is still interested in sending multicast traffic it sends a RECONNECT REPLY down the reverse path of the RECONNECT towards the

initiator. Along the way multicast forwarding is enabled in those nodes that were not previously forwarding data.

Rather than explicitly notifying the group that it wants to leave a multicast group, nodes use passive acknowledgements to determine if they should prune themselves from a group. After an intermediate node forwards data it listens for one of its downstream nodes to resend the data packet. If the node fails to hear this passive acknowledgement before a timer expires, the node assumes that no downstream nodes are interested in the session and, if it is not interested in the session itself, it prunes itself from the tree and stop forwarding data. This method requires that the last multicast receiver on the branch rebroadcast the ADMR header of the data packets that it receives.

2.5. Summary

While only a handful of MANET multicast algorithms were discussed here, perhaps a dozen or so have been proposed. With varying degrees of success, all MANET multicast algorithms attempt to increase the reactivity of the protocol and to reduce control overhead. However, few are concerned with reducing unnecessary data rebroadcasts. Reducing even a small percentage of the unnecessary data rebroadcasts can greatly reduce the overall bandwidth consumption. In the following chapters, the development and prototype implementation of a new multicast routing protocol is discussed. This new protocol attempts to build on the foundations laid by earlier pioneers in the field of MANETs. This new protocol borrows some of the good ideas present in previous algorithms and introduces a few mechanisms for reducing data rebroadcasts. Like ADMR and ODMRP, the new protocol conserves bandwidth by creating the tree only when absolutely necessary and is rooted at the source. Unlike previous protocols, the new multicast system attempts to reduce the number of unnecessary data rebroadcasts.

Chapter 3. Problem and Methodology

3.1. Introduction

Engineers have previously developed MANET multicast algorithms with two objectives in mind, to create an accurate picture of the network topology and to reduce control overhead. Existing algorithms have approached this problem in a number of ways, several of which were described in the preceding chapter. A proposed third objective is to create and maintain a multicast tree that efficiently delivers multicast packets, i.e., to reduce data overhead. The simplest solution to accurate multicast delivery is to have every node rebroadcast every data packet that it receivers. This method assures the delivery of data, but is extremely inefficient in its use of bandwidth.

The efficient use of bandwidth is imperative in wireless networks where data rates are usually severely limited as compared to their wired counterparts. Not only do wireless networks have to contend with low data rates, but the broadcast nature of wireless communication can introduce additional congestion and data collisions into the system. Consider the simple network shown in Figure 3.1.

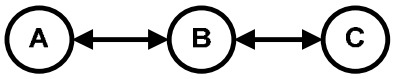


Figure 3.1. A simple wireless network.

In this wireless network node A is sending data to node C via node B. In such a network any data retransmitted by node B would be overheard by node A, and may interfere with transmissions that node A wishes to perform. A wired network would not have this problem, as node B would have two interfaces, one for communications with node A and one for communication with node C.

It should be clear that the conservation of bandwidth is imperative to the success of any wireless network. While previous MANET multicast protocols focused on the reductions of control overhead, the multicast protocol investigated in this research attempts to reduce the amount of bandwidth used by the network both in terms of control overhead and data rebroadcasts. It can usually be assumed that data transmission consumes more bandwidth than control overhead. Even a small decrease in data retransmissions should substantially improve network performance. Unlike previously

proposed MANET multicast algorithms, this new protocol will focus on reducing the amount of unnecessary data rebroadcast though the creation of a more efficient multicast tree.

3.2. Approach for the New Protocol

The DVMRP [16] multicast protocol is widely used in wired networks, but it is not appropriate for use in a MANET for several reasons. First, it is extremely bandwidth inefficient. Routers using DVMRP flood the network with multicast data before the tree has been completely constructed, wasting bandwidth. Secondly, DVMRP does not allow for route optimization once the original tree has been created. The algorithm assumes that the network topology changes infrequently and that routing changes or optimizations are rarely needed. In an *ad hoc* network the most efficient route between the source and its receivers may change quickly. Link state algorithms can react more quickly to changes in the network, but in doing so they add extra overhead to the system. It can usually be assumed that multicast routes are not needed or used as often as unicast routes. For this reason a bandwidth conserving reactive protocol should be used for this multicast protocol.

The proposed multicast protocol creates a source-specific multicast tree and maintains routes reactively. Like DVMRP [16], ODMRP [21], ADMR [27], and MAODV [20], the proposed algorithm creates multicast trees on-demand. Tree creation is triggered by a source when it decides that it wants to start sending data. Instead of flooding the network with data and then pruning the tree as nodes decide whether or not they wish to be a part of the session, this MANET multicast protocol advertises a multicast session by flooding the network with a small control packet. This message informs receiving nodes that a multicast session is about to begin. Nodes that receive this advertisement and who wish to join the session notify the source of their intentions through an appropriate response. Multicast routes are formed in a backward learning manner that is similar to the method used in ADMR. Unlike ADMR, route maintenance is performed locally. In ADMR a node that loses its connection with its upstream node must query the source for a new route. The new multicast algorithm allows intermediate nodes to respond to route requests.

While the multicast routing protocol does not depend on a specific unicast protocol in order to operate, it does rely on an underlying routing daemon to provide an accurate list of one hop nodes. It is suggested that OLSR [10] be used to provide these routes. This and other MANET multicast algorithms depend on network broadcast to build and maintain the multicast tree. The OLSR algorithm has been shown to reduce network floods through the use of MPR nodes [14]. The proposed multicast algorithm can use these nodes to reduce control overhead. Furthermore, the use of MPR nodes can help to aggregate multicast data flows and reduce unnecessary data rebroadcasts. The example network in Figure 3.2 shows a multicast tree that does not use MPR nodes to build a multicast tree. In this illustration grey nodes are multicast receivers, black nodes forward data, white nodes are not a part of the tree, and node 12 is the multicast source. In this multicast tree nodes 7, 8, 16, 18, 21 and 24 rebroadcast data, for a total of six data rebroadcasts.

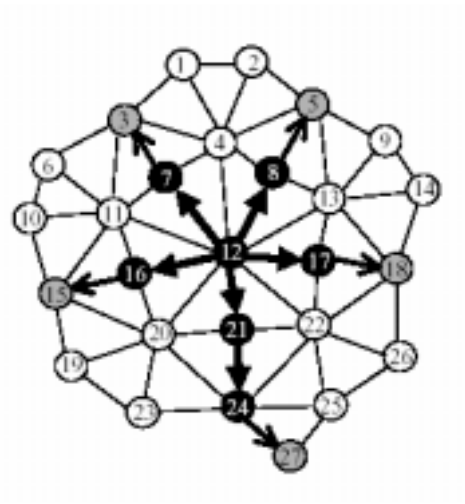


Figure 3.2. A potential multicast tree, from [21].

However, more efficient multicast routes are possible if MPR nodes are used in the creation of the multicast tree. Figure 3.3 shows a network where, through the use of MPR nodes, a more efficient multicast tree has been built. In this case only four nodes, 4, 16, 22, and 25, rebroadcast data.

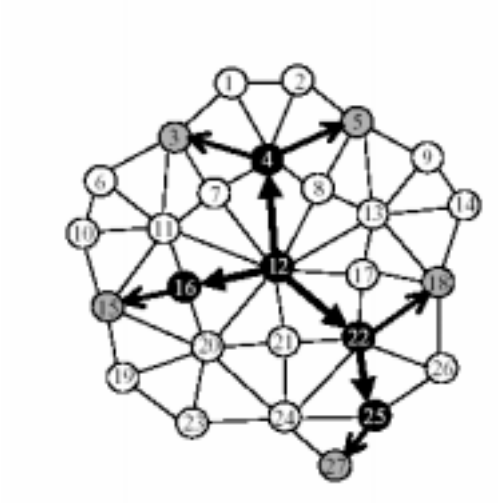
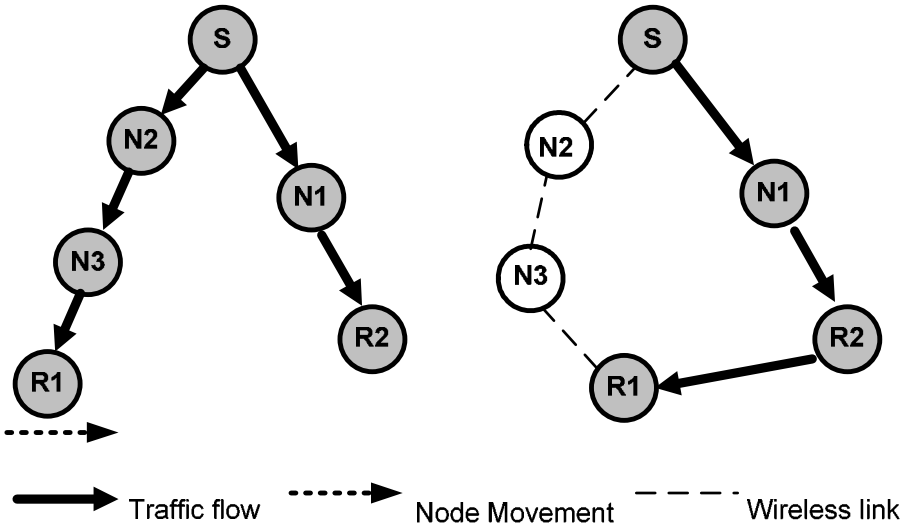


Figure 3.3. A multicast built tree using MPR nodes, from [21].

It should be noted that it is possible that the multicast tree in Figure 3.2 could be formed without the use of MPR nodes. However, the use of OLSR and MPR nodes guarantees the creation of the tree in Figure 3.3.

This protocol can make further reductions in the number of data rebroadcasts by allowing nodes to improve the efficiency of the multicast after the multicast tree has been created. A multicast tree branch may include intermediate nodes that needlessly retransmit data. By changing the branch of the multicast tree on which it resides, a node may be able to reduce the number of non-receiving nodes in the network. The following example illustrates such a situation.

The example network shown in Figure 3.4 is made up of a multicast source S, two multicast receivers R1 and R2, and three nodes N1, N2 and N3 that do not want to receive multicast data but are willing to forward traffic. The two-branch tree in Figure 3.4(a) shows the initial tree topology. In this example the motion of R1 is such that it eventually moves close enough to R2 so that the two nodes can communicate directly. In this case the network can be improved by eliminating N2 and N3 from the tree. This will reduce the number of unnecessary rebroadcasts in the network. This improvement occurs when R1 chooses to become a child of R2, rather than one of N3. This situation is shown in Figure 3.4(b).



(a) **(b)**

Figure 3.4. Network with source S, receivers R1 and R2, and intermediate nodes N1, N2 and N3. Shaded nodes are a part of the multicast tree. Arrows show the path of the traffic. A two-branch tree exists in (a). If R1 and R2 move sufficiently close together, a new, more efficient tree (b) is possible.

3.3. Methodology

The research described in this thesis was performed in three main steps. The first step involved the formulation and definition of the multicast algorithm. The original groundwork of the algorithm can be found in work presented by earlier designers. A synopsis of these protocols is presented in Chapter 2. This new algorithm attempts to improve upon past research by combining pre-existing and novel optimizations. The focus of the new multicast protocol is on the reduction of needless data retransmissions through improvements in the efficiency of the multicast tree. A detailed description of the protocol is given in Chapter 4.

Following the formulation of the multicast protocol, a prototype multicast router was implemented. This router was developed in Linux and is described in detail in Chapter 5. It should be emphasized that is a prototype implementation, not a simulation. It will allow the host to route “real” multicast data packets. The coding is done in the C programming language under the Linux 2.4 kernel [36].

While this router is intended to be deployed on wireless hosts, performance evaluation of the routing algorithm was done on a wired test bed. The use of a wired test bed allows the user to control network topologies more easily than could be done with a wireless interface. Network topologies can be controlled through the use of a dynamic switch [37]. The dynamic switch is essentially a modified Linux machine with multiple interface cards that allows the user to specify which interfaces are connected. Connections between interfaces on the dynamic switch are transparent. This means that all packets that arrive on an interface are forwarded to each of the interfaces to which it is connected. The use of the dynamic switch allows the user to maintain a consistency between different experiments that may not be possible when dealing with unpredictable wireless links. The dynamic switch also allows the user to simulate wireless links by controlling data rates and incurring random packet loss.

3.4. Summary

Bandwidth is a precious resource in a wireless network. The new multicast protocol attempts to manage this commodity by eliminating some of the inefficiencies inherent to most multicast trees. Not only do wireless networks have to contend with low data rates, but the broadcast nature of wireless transmissions can cause additional congestions and data collisions. These problems can be curtailed by reducing the amount of control and data packets in the network. Development of the multicast protocol was performed with this in mind. The proposed multicast algorithm, described in detail in Chapter 4, attempts to reduce the overall overhead in the network through the reduction of superfluous data retransmissions. Performance will be discussed in Chapter 6.

Chapter 4. Protocol

4.1. Introduction

This chapter describes in detail the operation of the new multicast protocol. This algorithm attempts to create a more efficient multicast tree to reduce the number of needless data rebroadcasts. It does this by informing nodes as to the status of the small portion of the network in which they are interested. This is accomplished by taking advantage of pre-existing notification mechanisms which introduces only a modest increase in control overhead in the network. This protocol uses many of the mechanisms seen in earlier multicast algorithms. The innovations presented here lay in a few small, but effective, optimizations. For this reason, the protocol will be referred to as XMMAN (eXtensions for Multicast in Mobile Ad-hoc Networks)[1].

The XMMAN protocol creates an on-demand, source-specific multicast tree. Each multicast tree is rooted at the source and can be identified by a unique [source, group] pair. While the operation of this algorithm does not depend on a specific unicast protocol, an underlying routing algorithm must provide an accurate one-hop topology. To provide this, it is suggested that a proactive routing algorithm, such as OSLR [14], be used in conjunction with XMMAN. This algorithm conserves overhead by creating a multicast tree in an efficient manner only when the tree is needed. The multicast algorithm can also utilize the MPR nodes as assigned by OLSR.

4.2. Definitions

This section uses RFC 2119 [38] to define the use of capitalized words such as MUST, SHOULD, MAY, etc. A list of technical definitions used in the protocol description is given below.

Node – Any device operating in the MANET. This thesis often uses the terms “node,” “router,” and “host” interchangeably.

[1] This acronym is also a play on the author’s last name. Where “Christmas” is often abbreviated as “X-mas”, “Christman” can be abbreviated as “X-man”.

One-Hop Nodes – The set of nodes with which a router has direct peer-to-peer contact at the link layer.

Upstream Node – The next hop node towards the source of the multicast tree.

Downstream Nodes – The set of one-hop nodes in the multicast tree away from the source to which a node forwards data packets.

Parent Nodes – The set of nodes that lie between directly between a router and the multicast source.

Child Nodes – The set of nodes that lie between a router and the leaf nodes, away from the multicast source.

Multicast Route – Route from a multicast source to a group of multicast receivers where all nodes have agreed to forward multicast traffic for a given session.

Multicast Session – Identified by a specific [source, group] pair.

Optimized Link State Routing (OLSR) – Unicast MANET routing algorithm that MAY be used as the underlying routing algorithm. An OLSR router uses a set of MPR nodes to reduce unnecessary rebroadcasts of data. [14]

Multipoint Relay (MPR) Node – "A node which is selected by its one-hop neighbor, node X, to 're-transmit' all the broadcast messages that it receives from X, provided that the same message is not already received, and the time to live field of the message is greater than zero." [14]

4.3. Routing Tables

4.3.1. Multicast Routing Tables

Each node has a multicast routing table in which it stores information relevant to the multicast tree. Entries are maintained for each multicast session of which the node is a member. A timer is kept for each of these entries. This timer is updated whenever it receives a refresh message from the source. If this timer is allowed to expire, the route is deleted. Each entry SHOULD have, at a minimum:

- Session ID [source, group]
- Pointer to upstream node
- List of downstream nodes and their timers
- Timer
- Hop count
- Routing flags

While this is the minimum set of fields that must be present in the multicast routing table, the actual implementation includes several extra variables.

4.3.2. Duplicate Table

Each node MUST cache control packets in a duplicate table to detect the receipt of out of order or duplicate control packets. The node MAY use a least recently used (LRU) scheme to limit the size of the table.

4.3.3. Neighbor Table

Each node MUST keep a list of its one-hop neighbors and the interface that is used to reach them. This information SHOULD be maintained by observing the unicast routing tables.

4.3.4. MPR Table

If a node is using MPR nodes it MUST keep a list of the nodes that have selected it as an MPR node. This is also called the MPR selector set.

4.4. Protocol

As a part of a multicast tree, a node can have a number of relationships with the other members of the session. Given a particular multicast session, a node can be a source, a receiver, an intermediate node, a source and a receiver, or a receiver and an intermediate node. Note that a host cannot be a source and a forwarding router for a specific [source, group] tree.

The protocol is similar to AODV [32] and AMDR [27] and uses many of the naming conventions and packet formats used in the Multicast Zone Routing (MZR) [22] protocol. Multicast session addressing information is distributed separately and is not within the scope of this document. It is assumed that nodes use standard existing mechanisms to determine the group IP addresses of multicast sessions of interest. RFC 2119 [39] and RFC 2974 [40] describe some ways in which this can be done. Generally, these protocols use a central server that hosts can query for multicast addresses.

4.4.1. Multicast Tree Creation

A node that wants to begin sending multicast data initiates the creation of a multicast tree. Nodes can determine if they are multicast senders in several ways. A node MAY listen for data packets from the application layer. If a node hears a data packet for which it is not already a source, it can buffer the subsequent data until it has created a valid multicast tree. A node MAY also force the multicast application to notify the router that it wishes to begin sending data. The actual mechanism used to initiate tree creation is implementation dependent.

A source node starts tree creation by flooding a TREE-CREATE message to every host in the network. This message contains the source and group IP addresses of the session, a sequence number, the time to live (TTL), and the hop count of the packet. A host that receives a non-duplicate version of this message decrements the TTL and rebroadcasts the message. Nodes that receive a TREE-CREATE message with a TTL of zero do not forward the message. This limits each branch of the multicast tree to a maximum of TTL hops from the source. Before forwarding the TREE-CREATE message, a node MUST increment the hop count field and record the value in its

multicast routing table. If MPR nodes are being used, a node only rebroadcasts a TREE-CREATE message that is receives from a member of its MPR selector set.

A node that receives a new TREE-CREATE packet inserts an entry for the [source, group] pair with the upstream node pointer pointing to the last hop of the packet and sets the timer to the REFRSH_HOLD_TIME. The list of downstream nodes is set to NULL. A node that wishes to join this multicast session responds to the TREE-CREATE with a unicast TREE-CREATE-ACK message to the upstream node, i.e. the last hop of the TREE-CREATE, with the multicast source and group IP addresses in the body of the message.

A node that receives a valid TREE-CREATE-ACK packet updates the timer of the appropriate multicast entry and adds the address of the last hop of the message to its set of downstream nodes. If this node is not the source and does not have a valid route to the source, it forwards the TREE-CREATE-ACK packet to its upstream node. The node sets the downstream node timer to DOWN_HOLD_TIME. This process continues until the TREE-CREATE-ACK reaches the multicast source or another node that has a valid multicast route to the source. Nodes that receive TREE-CREATE packets but never receive a TREE-CREATE-ACK message assume that they do not lie between a source and a receiver and will simply time out their multicast entries.

By taking advantage of the MPR nodes as provided by OLSR, unnecessary rebroadcasts of TREE-CREATE messages can be reduced. Not only will TREE-CREATE rebroadcasts be reduced, but a tree that reduces unnecessary data rebroadcasts will be created. If OLSR is used as the underlying unicast routing protocol, the multicast daemon MAY communicate with the OLSR daemon to exchange a list of MPR nodes. A slight modification of the OLSR router implementation is necessary to facilitate communication between the two processes.

4.4.2. Route Maintenance

Frequent changes in network topology can cause links failures that disrupt communication between nodes. Because of the highly dynamic nature of the network link failures must be detected and fixed quickly. The following section describes how a

source disseminates state information throughout the tree and how nodes respond to link failures.

4.4.2.1. Tree Refresh Messages

A multicast source periodically disseminates state information through the periodic use of TREE-REFRESH messages. These packets are sent by the multicast source to every node in the multicast tree and are identified by a unique [source, group] pair and a sequence number. Member nodes that receive a valid TREE-REFRESH message update their timers and forward the message to their downstream nodes. A source MUST send a TREE-REFRESH message every REFRESH_INTERVAL seconds. Usually the REFRESH_HOLD_TIME is some integer multiple of the REFRESH_INTERVAL (e.g. 3). Once a multicast source stops sending data it MUST stop sending TREE-REFRESH packets. Nodes that fail to receive a TREE-REFRESH before the timer expires will delete their routes. This ensures that multicast tree maintenance only occurs when the multicast session is active. As the TREE-REFRESH message moves down the tree intermediate nodes MUST decrement the TTL and increment the hop count field in the message. If the hop count advertised by the TREE-CREATE is different from the hop count in their routing table the node will update the value. This value may change if a change in network topology occurs. Nodes MUST NOT forward TREE-REFRESH messages with a TTL that is less than or equal to zero. TREE-REFRESH packets SHOULD be piggybacked on data packets. Routers SHOULD set the REFRESH_HOLD_TIME long enough to allow for link repair in the case that the failure to receive a TREE-REFRESH message is due to a link failure.

TREE-REFRESH messages can also be used to disseminate information that allows nodes to perform route optimization. Adding fields that describe the status of the last hop and the number of multicast receivers on the branch may allow nodes to make localized route optimizations. This feature is described in Section 4.4.6.

4.4.2.2. Reaction to Link Failures

4.4.2.2.1. General Operation

Downstream nodes are responsible for repairing link failures. A node detects a link failure by looking at its neighbor list. If a router finds that a direct connection between it and its upstream node no longer exists, the host will initiate branch rediscovery. Branch rediscovery proceeds in much the same way as does multicast route discovery, except in reverse. A node looking to rejoin the multicast tree floods the network with a JOIN packet. Each JOIN packet contains the [source, group] pair of the session of interest and the address of the node initiating the branch rediscovery. As the JOIN message traverses the network, intermediate nodes mark an entry in their multicast routing tables, set their timers to HOLD_REFRESH_TIME, and store the address of the last hop of the JOIN. Nodes that receive a new JOIN message and do not have a valid route to the source decrement the TTL and forward the message if the TTL is greater than zero. If the node has a valid route to the source it adds the last hop of the JOIN to its downstream list and unicasts a JOIN-ACK message to the last hop of the JOIN. Nodes forward the JOIN-ACK message until it reaches the lost node. The initiator of the JOIN may receive many JOIN-ACK messages, each advertising a different path to the source. To enable one of these branches, the initiator MUST send a JOIN-OPT message back up the tree. The lost node unicasts this JOIN-OPT message to its new upstream node. If a node receives a JOIN-OPT message and multicast routing has not been enabled, the node enables routing and forwards the JOIN-OPT message to its upstream node. The JOIN-OPT message travels towards the multicast source, enabling forwarding, until it reaches the node that sent the original JOIN-ACK.

When performing branch rediscovery a lost node MUST not respond to a JOIN-ACK from a downstream node, i.e. a node that is part of the partitioned network. This problem can be solved by requiring that nodes respond to a JOIN only if their hop count to the source is smaller than or equal to the hop count advertised by the lost node. If a node has a larger hop count than the lost node it is possible that it is downstream of the link failure and, therefore, its route is no longer valid. For this reason a JOIN message MUST include in the message the minimum hop count that a node must have in order to

respond with a JOIN-ACK. However, this prevents the lost node from joining a node with a valid route to the source but with a larger hop count.

4.4.2.2.2. The JOIN-NOTIFY Message

During branch rediscovery nodes MAY notify their children that they are performing branch rediscovery. To do this, a node would send a JOIN-NOTIFY message to its downstream nodes. This message is sent just before a host sends a JOIN message and accomplishes two goals. First, it informs nodes that an upstream node is attempting to rejoin the multicast tree and that they MUST NOT respond to a branch request for the multicast session or forward a JOIN from the originator of the JOIN-NOTIFY. Secondly, it updates the timers of the downstream nodes. Failing to do this may cause nodes to time out before the link can be repaired. JOIN-NOTIFY messages have the same format as JOIN messages.

A JOIN-NOTIFY message is sent with a TTL set to JOIN-NOTIFY-TTL. The value of JOIN-NOTIFY-TTL SHOULD be large enough that it can reach every node in the network (e.g. 16). The node floods the message to each of its downstream children in the same way that the TREE-REFRESH traverses the network. Upon the receipt of a JOIN-NOTIFY message a host checks to see if the message came from its upstream node. If so, it sets a timer equal to JOIN_WAIT seconds, resets its REFRESH_HOLD_TIME timer, and forwards the message if it has any downstream nodes. For JOIN_NOTIFY_HOLD_TIME time the node will not respond to or forward a JOIN messages originating from the initiator of the JOIN-NOTIFY. The use of the JOIN-NOTIFY message can reduce the total overhead in the network and can help to eliminate routing loops.

4.4.2.2.3. Exponential Increase JOIN

A node initiating branch rediscovery MAY send a JOIN message with a small FIRST_JOIN_TTL value (e.g. 2 or 3) in the hope that a valid route exists locally. If a timeout occurs before a valid JOIN-ACK is received the node doubles the TTL of the message and resends the packet. The node continues to double the TTL and resend the packet until it receives a valid JOIN-ACK or reaches a maximum TTL value of

JOIN_TTL. If the implementer prefers, this method of exponentially increasing of the TTL value can be skipped and the JOIN can be sent with a large value of JOIN_TTL.

When used properly this method allows a network to converse bandwidth. However, if a second JOIN is necessary those nodes within FIRST_JOIN_TTL hops of the lost node will receive multiple requests (which they cannot answer!) for a path to the multicast source. Increasing the value of FIRST_JOIN_TTL increases the probability that a route will be discovered in the first re-join attempt. However, as this value increases, so does the penalty paid for not finding a route. A second JOIN with a large FIRST_JOIN_TTL will be resent to a large number of nodes.

4.4.3. Tree Pruning

Nodes MAY choose to leave a multicast group at any time. A router that is no longer interested in multicast traffic MUST check for any downstream nodes before removing itself from the multicast tree. A node that finds at least one downstream node stops sending data packets to its upper layer applications, but does not stop forwarding that data to downstream nodes. While the application layer multicast program may choose to ignore multicast packets, the router still forwards traffic to its downstream nodes. If a node finds that it does not have any downstream nodes, i.e. the router is a leaf node, it deletes the entry from its multicast routing table and sends a PRUNE message to its one-hop neighbors. Each PRUNE message contains the address of the multicast group that it wishes to leave. This protocol follows the traditional multicast model as described in [34] and does not allow a node to join or leave a source-specific multicast session, as described in [35]. A slight modification of the protocol could provide for this capability. A node that receives a PRUNE message checks to see if the last hop of the message is a member of its downstream list. If the address is in the downstream list the node removes the address. If this deletion empties the list and the node is not interested in the session itself, it stops data forwarding and prunes itself from the group. If the deletion empties the downstream list, but the node is still interested in the group, the router stops forwarding data.

4.4.4. Maintaining Downstream Nodes

Link failures in a wireless network may prevent nodes from explicitly leaving a multicast group. For this reason a node MUST keep track of which of its downstream nodes are also one-hop neighbors. Each downstream entry keeps a timer that is updated periodically if the node finds the address in its list of one-hop neighbors. If the address cannot be found before the timer expires, the entry is removed. If the downstream list is empty and the node does not want to receive multicast data ,the node MUST send a PRUNE message to its upstream neighbors. The timer SHOULD be long enough to allow a JOIN message to reach the node if a link fails and the downstream node is still interested in the session, i.e. the downstream timer SHOULD be longer than the upstream timer.

4.4.5. Joining an Existing Multicast Session

If a node wishes to join a group and has not heard a TREE-CREATE in FIND_TREE_TIME seconds, it SHOULD flood a JOIN message to the network. These JOIN message are sent with the source address set to 255.255.255.255. A node that receives this message responds in the same way that it would for any other JOIN except for that its sends a JOIN-ACK for every source of this advertised group. A node will continue to send JOIN messages every JOIN_WAIT seconds until it receives a valid response.

4.4.6. Route Optimization

4.4.6.1. Route Optimization Using TREE-REFRESH Messages

A member of the multicast tree MAY perform a route optimization if it manages to overhear a TREE-REFRESH message broadcast by a source other than its upstream node. If a node overhears such a message it can check the information stored in the message to determine whether, if by joining the new branch, inefficiencies in the network can be reduced. If a node can determine that it is an only child, it MAY leave its upstream node and join the new node. This move would allow its original parent to prune itself from the multicast tree if it is non-receiver, or allow the node to stop forwarding multicast data if it is a multicast receiver.

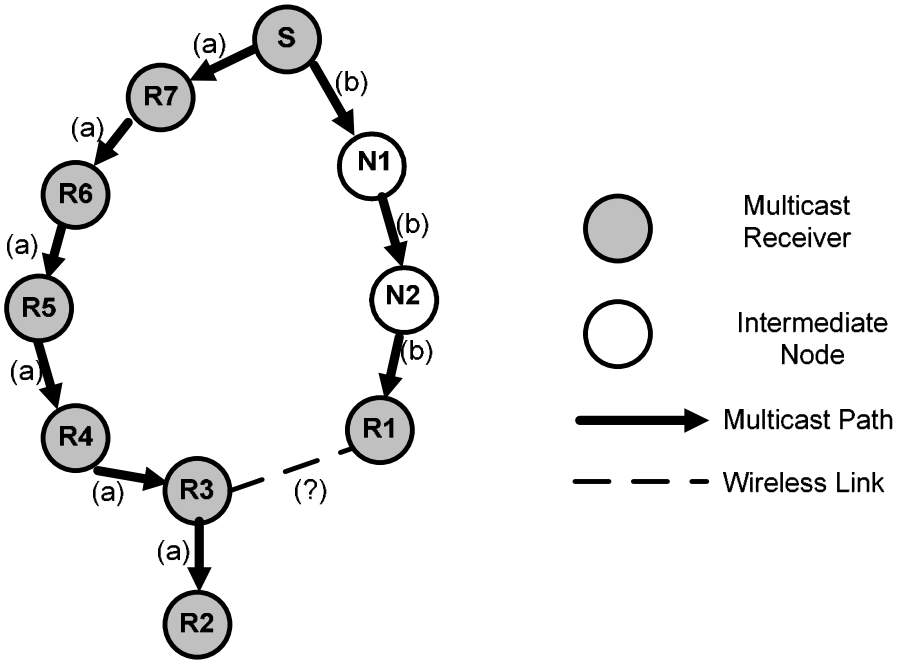


Figure 4.1. Which branch is better for the network? Path (a) requires fewer unnecessary rebroadcasts, but path (b) has fewer hops to the source.

Figure 4.1 shows a multicast tree with two branches (a) and (b). If node R1 overhears a TREE-REFRESH message from R3 it has the option of staying on branch (b) or moving to branch (a). If node R1 were to move to branch (a), it would be three more hops from the source that it would on branch (b), but the move would prune two nodes from the multicast tree. The data rebroadcasts by nodes N1 and N2 are inefficient and, given the opportunity, R1 should move to branch (a). This would allow nodes N1 and N2 to leave the tree, eliminating unnecessary rebroadcasts.

In its simplest incarnation, a TREE-REFRESH message tells members of the multicast tree whether a session is active or inactive. To facilitate route optimization NON_RECEIVER and CAN_PRUNE fields are added to the TREE-REFRESH packet and to the multicast table. The value of the NON_RECEIVER field represents the number of upstream nodes that have only one downstream node. The value stored in the CAN_PRUNE field advertises the state of the node forwarding the TREE-REFRESH.

The CAN_PRUNE field takes on one of five values. A node that is not interested in the multicast session and has only one downstream node sets its CAN_PRUNE field to TRUE. A node that is a multicast receiver and has one downstream node sets the

CAN_PRUNE field to JOINED_1. Similarly, a node that is interested in the multicast session and does not have any children sets its CAN_PRUNE field to JOINED_0. The sender of the multicast session sets the CAN_PRUNE field to SENDER. All other nodes set the value of CAN_PRUNE to FALSE.

A source node begins the process by sending a TREE-REFRESH message with the NON_RECEIVER field set to 0 and the CAN_PRUNE flag set to SENDER. A node that receives a TREE-REFRESH message with the CAN_PRUNE set to TRUE increments the message's value of NON_RECEIVER by two. If the CAN_PRUNE field of the message is set to JOINED_1, the node increments the value of NON_RECEIVER by one. If the CAN_PRUNE message is set to FALSE, the node then sets NON_RECEIVER field to 0. A node should never receive a TREE-REFRESH from its upstream node with the CAN_PRUNE field set to JOINED_0. The node then increments the hop count of the message, decrements the TTL value, stores the value of NON_RECEIVER and forwards the message with this new value of NON_RECEIVER.

If a node has a NON_RECEIVER value that is non-zero and receives a TREE-REFRESH packet from a node that is not its parent, nor member of its downstream list, it checks to see if the new node offers a more efficient route to the source. If the new TREE-REFRESH message has a NON_RECEIVER value that is less than or equal to the node's current value and a CAN_PRUNE value that is not equal to JOINED_0, the receiver MAY leave its parent and become a child node of the last hop of the new TREE-REFRESH message. In the case of a non-zero tie, the node with the larger hop count SHOULD change parents. In the case that the hop count values are equal, the node with the smaller address SHOULD perform the route optimization. If a node changes parents it MUST set its NON_RECEIVER value to 0. This prevents the node from performing another route optimization before it receives information about its new branch from the next TREE-REFRESH message.

In the case that a node receives a TREE-REFRESH with CAN_PRUNE set to JOINED_0, the node MUST increment the message's value of NON_RECEIVERS before making a comparison. Doing this prevents a node from moving from an upstream multicast receiver with one downstream node to a multicast receiver with no downstream nodes. Such a move would not improve the fitness of the network, and would lead to an

oscillation in which the node would move back and forth between the two upstream nodes.

If a node that is performing the route optimization has a hop count that is larger than or equal to the hop count of the new parent, it unicasts a JOIN-OPT message to the new upstream node and PRUNE-OPT message to its former parent. Nodes that receive a JOIN-OPT message add the last hop address to their downstream list. Nodes that receive a PRUNE-OPT message remove the last hop address from their downstream lists. Before sending the JOIN-OPT message, the node sets its NON_RECEIVER value to zero to prevent the node from attempting a new route optimization before it receives a new TREE-REFRESH message from its new parent that accurately represents the status of the network.

If the node that is performing the route optimization has a hop count that is less than that of the node that it wants to join, it MUST first verify that the new parent-to-be is not downstream of the initiator. To do this the initiator of the route optimization sends a JOIN-OPT-REQUEST to the potential parent. The recipient of the JOIN-OPT-REQUEST then sends a JOIN-OPT-NACK message to its parent nodes with a TTL equal to the difference between its hop count and the initiator's hop count. If the recipient of the JOIN-OPT-NACK did not initiate the JOIN-OPT-REQUEST, it decrements the TTL and forwards the message to its upstream node. If the initiator of the JOIN-REQUEST receives a JOIN-OPT-NACK from its potential new parent before JOIN_NACK_TIME seconds, it stops the route optimization. If not, it sends a JOIN-OPT packet to the new parent and a PRUNE-OPT to its former upstream node. If a node that has received a JOIN-REQUEST does not receive a JOIN-OPT message within JOIN_OPT_TIME, it assumes that it is downstream of the initiator of the JOIN-REQUEST. It then sends a JOIN-OPT to the initiator of the JOIN-REQUEST and a PRUNE-OPT to its previous upstream node. This final action accomplishes two goals. First, it improves the node's fitness by reducing its distance to the source. Second, it keeps the initiator of the original JOIN-REQUEST from continuing to request a new route from the originator of the JOIN-OPT-NACK.

A node MUST NOT participate in more than one route optimization per multicast session at any given time. If a node receives a JOIN-OPT or JOIN-OPT-REQUEST for a

multicast session for which it is currently performing a route optimization, it SHOULD ignore the message. To prevent such mistakes, a node that is performing route optimization SHOULD set its NON_RECEIVER value to 0 to prevent itself from initiating a route optimization and advertise a NON_REFRESH value of 255 to prevent other nodes from joining this node.

The JOIN-OPT and PRUNE-OPT messages allow a node to join or leave a specific multicast branch, i.e. a source specific multicast session, and the messages may be used in the future to allow the multicast algorithm to conform to source specific multicast [34].

Because they do not have any downstream routers, leaf nodes do not normally rebroadcast TREE-REFRESH messages. However, by not forwarding TREE-REFRESH messages, leaf nodes prohibit their one-hop nodes from joining their branch via route optimization. Therefore, a leaf node that is performing route optimization SHOULD forward these messages, albeit, less frequently than if they did have a set of downstream nodes.

4.4.6.2. Route Optimization Using JOIN Messages

A node MAY also choose to select “the best” branch during branch reconstruction. It is not uncommon for a node to receive several JOIN-ACK messages in response to its JOIN broadcast. Instead of responding to the first JOIN-ACK message that it receives, it can wait JOIN_ACK_RESPOND seconds and respond to the JOIN-ACK that offers the best path to the multicast source. The best path to the source can be measured by the number of hops between the initiator of the JOIN and the initiator of the JOIN-ACK. These intermediate nodes are not multicast receivers, since if they were, they would have responded to the JOIN. Extra nodes only introduce inefficiencies into the network. Because the node with the least number of intermediate non-receivers is, by definition, the closest node to the initiator of the JOIN, a JOIN-ACK from this node should be received first. Because of this, the JOIN_ACK_RESPOND time SHOULD be short (e.g. 1 or 2 seconds). To facilitate this functionality the JOIN-ACK message MUST include a HOP-COUNT field, which measures the number of hops between the initiator of the JOIN and the initiator of the JOIN-ACK. The initiator of the JOIN-ACK

sets the HOP-COUNT field to 0, and each intermediate node increments the HOP-COUNT field. After waiting JOIN_ACK_RESPOND seconds after it receives the first JOIN-ACK the node chooses to respond to the JOIN-ACK with the smallest hop count.

4.5. Sender Notification

One of the problems that nodes face when creating on-demand source-specific multicast trees is how to determine when a node wants to start sending data. One way to do this is to accept and buffer multicast data packets until a valid tree has been created. Another method is to require the application to notify the router when it wants to start sending data. Using this method an application that wants to start a session sends a SENDER message to the router. SENDER messages contain the source address of the sender, the group address to which the application wants to send data, and the TTL that should be assigned to the TREE-CREATE packet. The source field is necessary to allow operation when applications do not reside on the multicast router. It is possible for hosts without multicast routing capabilities to tunnel data to a multicast router, which then acts as the multicast source. A node that receives a SENDER message initiates multicast tree creation only if it does not already have a multicast session open. A SENDER-FORCE message forces the node to send a TREE-CREATE message regardless the state of the router. These messages are exactly the same, save the message type field.

4.6. Message Formats

The following section shows the structure of the messages described in the previous sections. Most of the message formats are based on packet structures found in the MZR draft [22]. Some messages have been altered to include extra information, or to reduce the size of the packet. Each packet uses the OLSR packet header as shown in the OLSR draft [14]. Note that each message has a message type integer associated with it. However, this number will be stored in the OLSR packet header. Message types are assigned so that they follow the OLSR message type sequence.

4.6.1. OLSR Packet Header

For convenience the OLSR packet header is shown below.

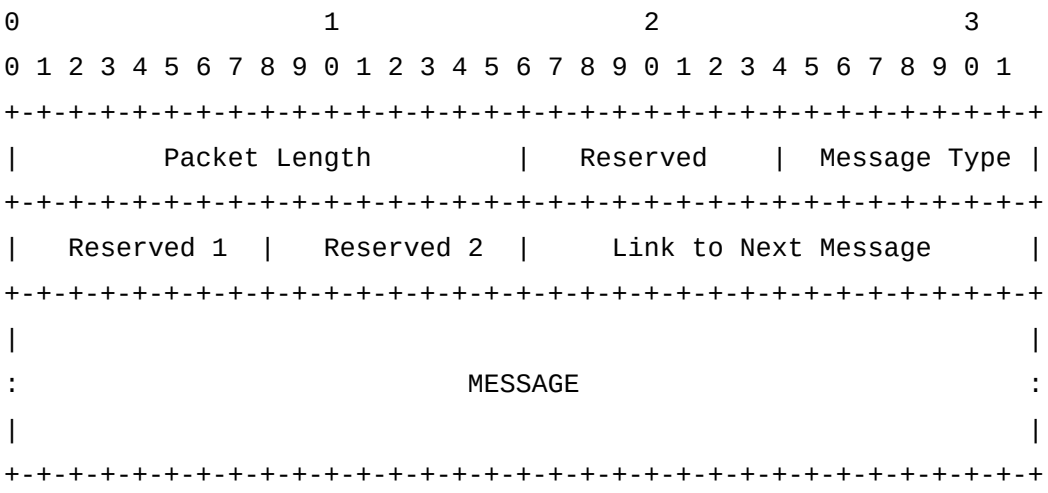


Figure 4.2. OLSR packet header.

Packet Length:

In bytes, the length of the entire packet.

Message Type:

The integer representation of the message type.

Reserved Field:

Used for padding.

Reserved 1 Field:

Used to store various fields that do not fit in messages.

Reserved 2 Field:

Used to store various fields that do not fit in messages.

Link to Next Message:

Used if multiple messages are stored in the packet.

4.6.2. TREE-CREATE Message

The TREE-CREATE message is used by a source to advertise the beginning of a multicast session. These messages are broadcast to the entire network. The message format is as follows.

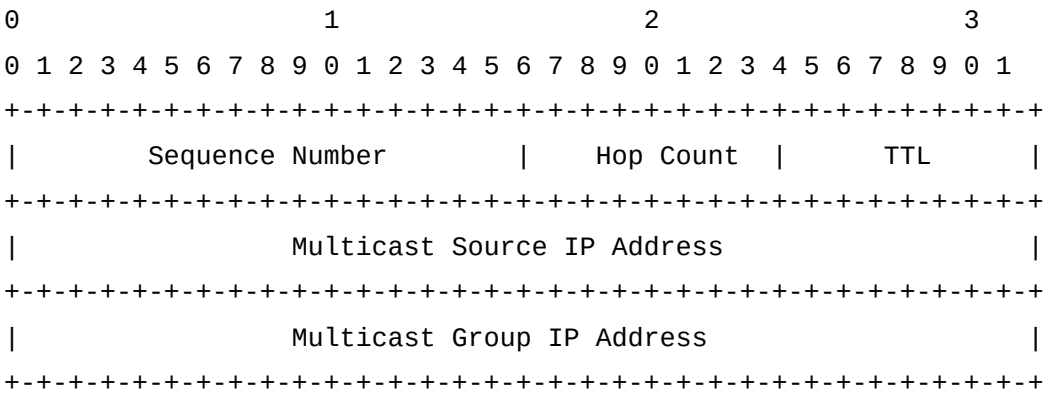


Figure 4.3. TREE-CREATE message.

Message Type: 9

Sequence Number:

16-bit integer uniquely identifying the advertisement for this multicast session.

Hop Count:

Number of hops away from multicast source.

Time to Live (TTL):

This number is decremented by each intermediate node. The message is not forwarded if the TTL reaches 0.

Multicast Source IP Address:

Identifies the source address of the TREE-CREATE message.

Multicast Group IP Address:

Identifies the group address to which data will be addressed. Along with the multicast source IP address uniquely identifies the multicast session

4.6.3. TREE-CREATE-ACK Message

This message is unicast to an upstream router by a node that wishes to participate in a multicast session. The unicast destination address is not included in the message body, as it is provided in the IP header.

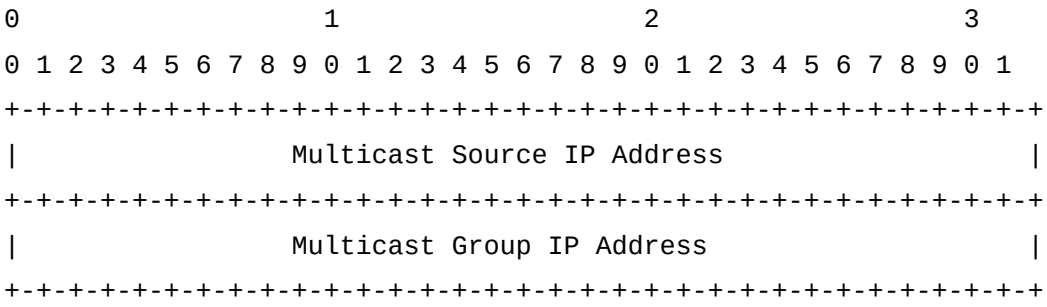


Figure 4.4. TREE-CREATE-ACK message.

Message Type: 10

Multicast Source IP Address:

Identifies the source address of the TREE-CREATE message.

Multicast Group IP Address:

Identifies the address to which data will be addressed. Along with the multicast source IP address, uniquely identifies the multicast session.

4.6.4. JOIN Message

The JOIN message is used by a node that has lost its upstream router or wishes to join an existing session. The format of the message is given below.

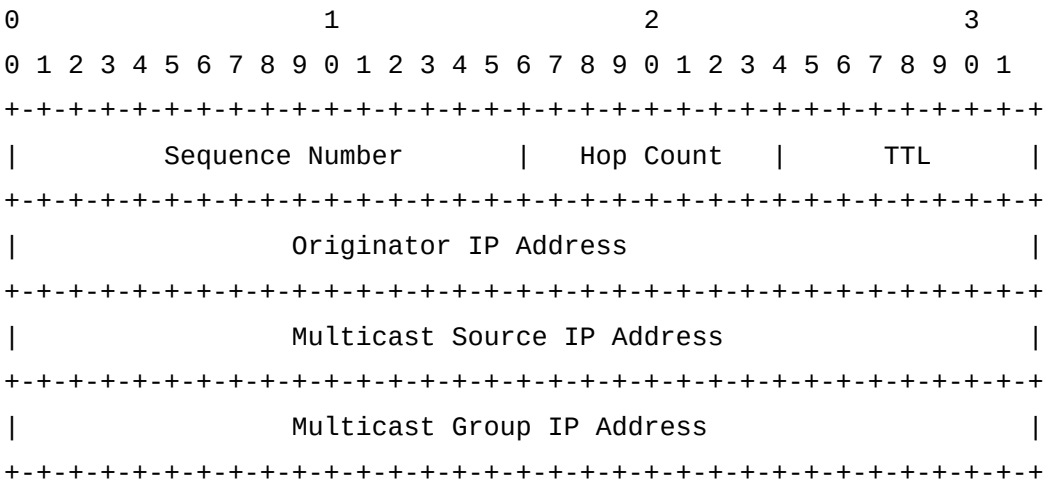


Figure 4.5. JOIN message.

Message Type: 11

Sequence Number:

16-bit integer uniquely identifying the JOIN.

Hop Count:

The minimum hop count that a node must have from the source in order to respond to the JOIN.

Time to Live (TTL):

This number is decremented by each intermediate node. The message is not forwarded if the TTL reaches 0.

Originator IP Address:

The IP address of the node that initiated the JOIN search.

Multicast Source IP Address:

Identifies the multicast source address.

Multicast Group IP Address:

Identifies the address to which data will be addressed. Along with the multicast source IP address, uniquely identifies the multicast session.

4.6.5. JOIN-ACK Message

This message is a response to a JOIN request. These messages are unicast to the last-hop of the JOIN message. The unicast destination address can be found in the IP header.

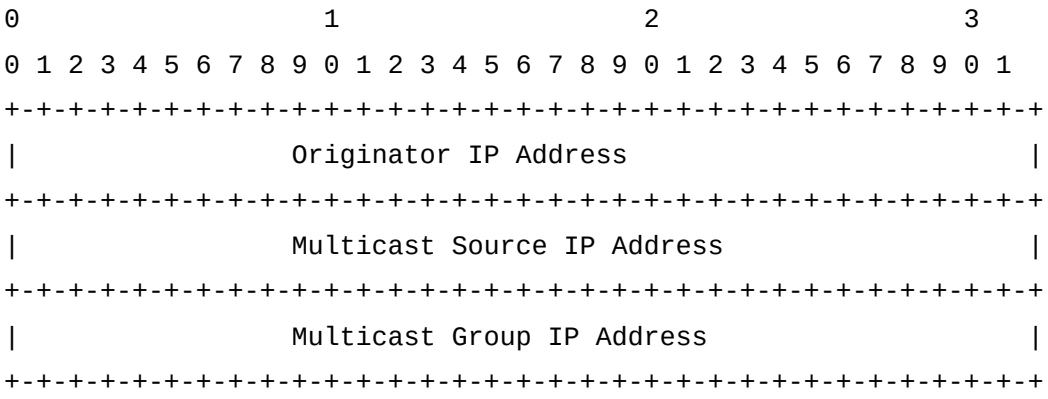


Figure 4.6. JOIN-ACK message.

Message Type: 12

Originator IP Address:

Identifies the node that initiated the JOIN.

Multicast Source IP Address:

Identifies the source address of the group that the JOIN-ACK is responding to.

Multicast Group IP Address:

Identifies the address of the group that the node wants to join. Along with the multicast source IP address, uniquely identifies the multicast session.

Time to Live (TTL):

Time to live. This value will be stored in the packet header.

Hop Count:

Hops to the source. This value will be stored in the packet header.

4.6.6. PRUNE Message

A PRUNE message is sent by a node that has determined that it no longer wishes to receive or forward multicast traffic from a particular multicast group. These messages indicate a node's desire to leave a particular multicast group and are not source specific.

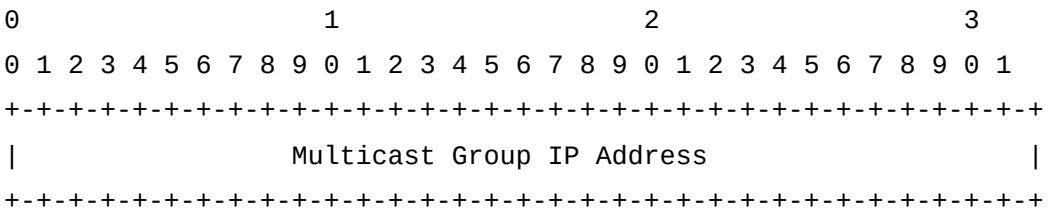


Figure 4.7. PRUNE message.

Message Type: 13

Multicast Group IP Address:

Identifies the group address that the node wants to leave.

4.6.7. TREE-REFRESH Message

The TREE-REFRESH message is used to disseminate state information to nodes in the multicast tree. The message format is as follows.

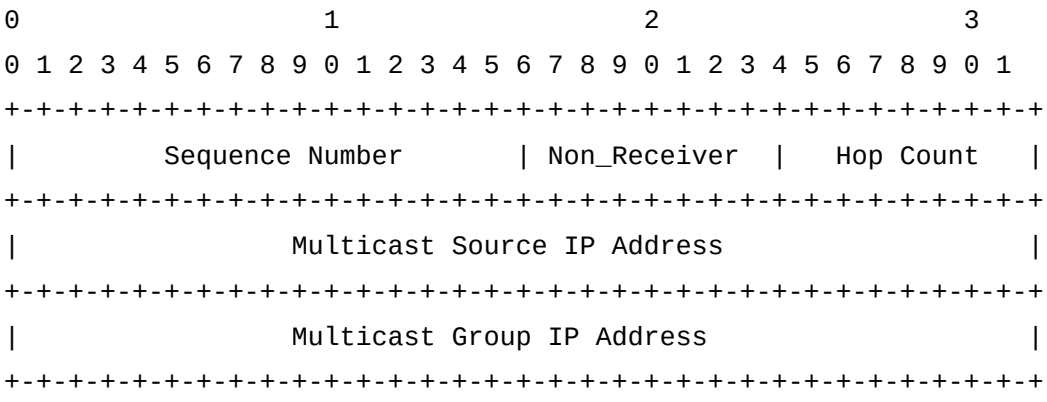


Figure 4.8. TREE-REFRESH message.

Message Type: 14

Sequence Number:

16-bit integer uniquely identifying the TREE-REFRESH.

Non Receiver:

The number of consecutive non-receiving nodes with one downstream node.

Hop Count:

The number of hops away from the source.

Time to Live (TTL):

Time to Live. This value is stored in the reserved field of the packet header.

Multicast source IP address:

The IP address of the multicast source.

Multicast group IP address:

The IP address of the multicast group.

Can_Prune:

This value indicates the status of the initiating node. This value is stored in the reserved field of the OLSR packet header.

4.6.8. SENDER Message

The SENDER message is used by a host to notify the router that an application would like to start a new multicast session. If a router receives a SENDER message and a session is already open, it does not initiate a new multicast tree creation.

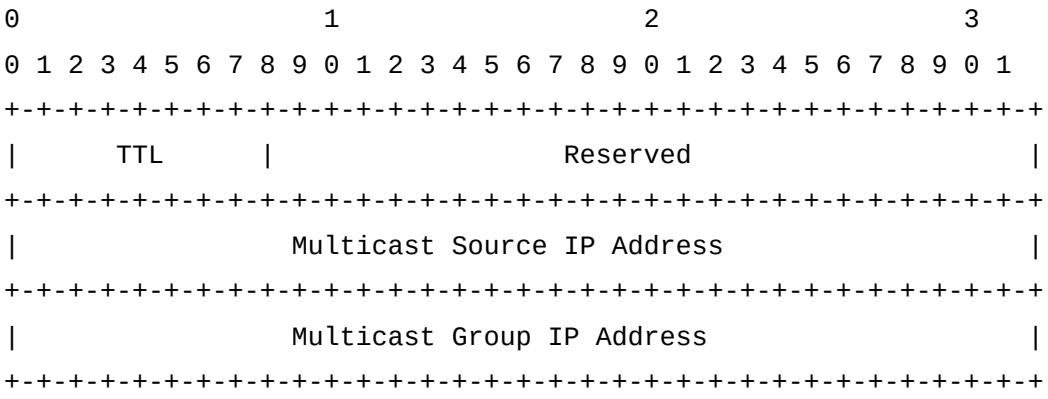


Figure 4.9. SENDER message.

Message Type: 15

Time to Live (TTL):

This value is used to limit the range of the resulting TREE-CREATE message.

Multicast Source IP Address:

The IP address of the multicast source.

Multicast Group IP Address:

The IP address of the multicast group.

4.6.9. SENDER-FORCE Message

A SENDER-FORCE message is used by a host when it wants to force a router to broadcast a TREE-CREATE message. The only difference between the SENDER and SENDER-FORCE messages are the message types associated with them.

Message Type: 16

4.6.10. JOIN-OPT Message

This packet is used by nodes to inform a new node that it has been chosen as an upstream node during a route optimization. This message is unicast to the recipient and the destination address is found in the IP header.

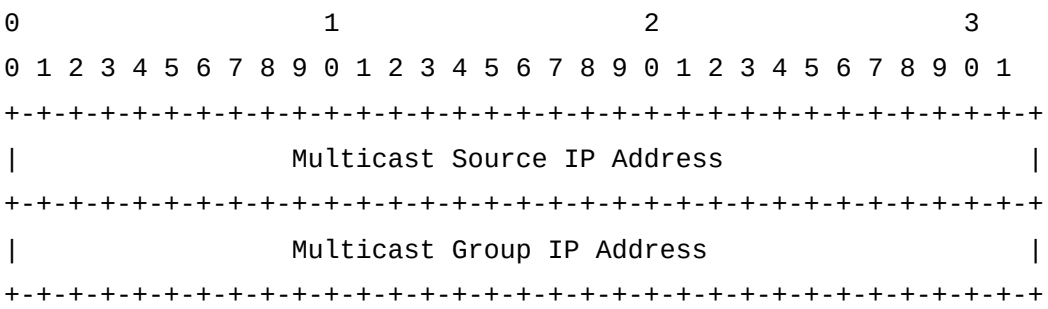


Figure 4.10. JOIN-OPT message.

Message Type: 17

Multicast Source IP Address:

Identifies the source address of the multicast tree.

Multicast Group IP Address:

Identifies the address to which data will be addressed. Along with the multicast source IP address uniquely identifies the multicast session.

Time to Live (TTL):

Time to live. Used when enabling routes in response to JOIN-ACK. Value is stored in the packet header.

Hop Count:

Used when enabling routes in response to JOIN-ACK. Value is stored in the packet header.

4.6.11. PRUNE-OPT Message

Used during route optimization, this packet informs the previous upstream node that it MUST remove the sender from its list of downstream nodes. This message is unicast to a destination that is addressed in the IP header.

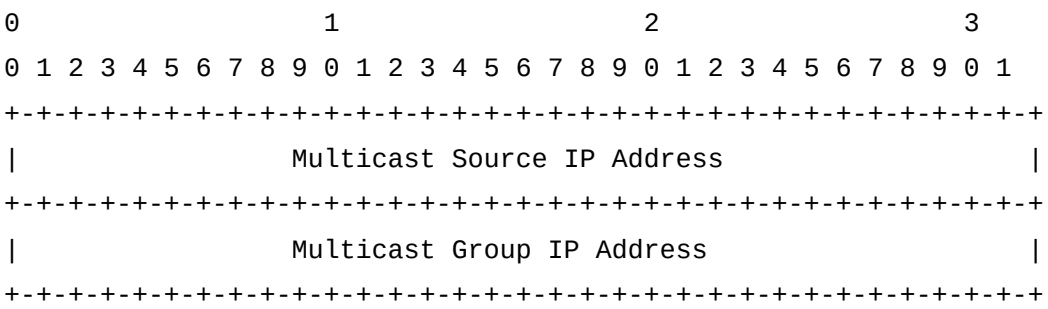


Figure 4.11. PRUNE-OPT message.

Message type: 18

Multicast Source IP Address:

Identifies the source address of the multicast tree.

Multicast Group IP Address:

Identifies the address to which data will be addressed. Along with the multicast Source IP address uniquely identifies the multicast session

4.6.12. JOIN-NOTIFY Message

This message has the same format at the JOIN-OPT message, save for the message type.

Message type: 19

4.6.13. END-SEND Message

This message has the same format at the SENDER message, save for the message type.

Message type: 20

4.6.14. JOIN-FIND-ACK Message

This message has the same format at the JOIN-ACK message, save for the message type.

Message type: 21

4.6.15. JOIN-OPT-REQUEST Message

This message has the same format at the JOIN-OPT message, save for the message type.

Message type: 22

4.6.16. JOIN-OPT-NACK Message

This message is a response to a JOIN-OPT-REQUEST message.

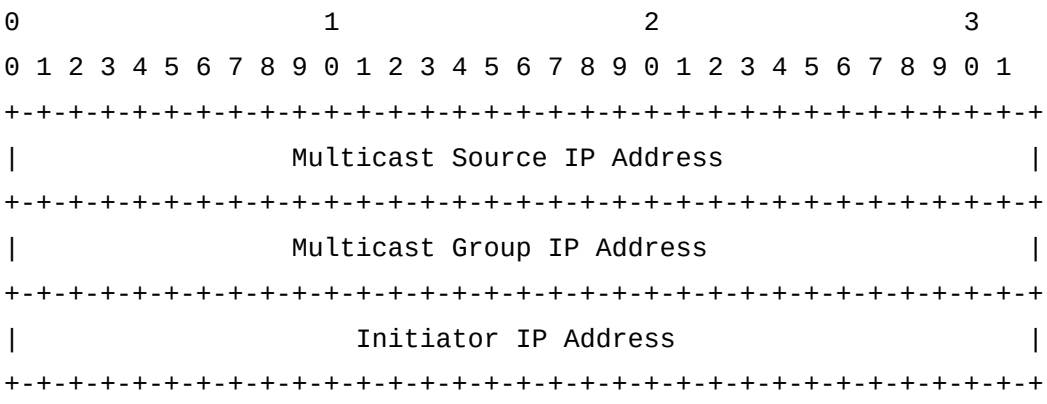


Figure 4.12. JOIN-OPT-NACK message.

Multicast Source IP Address:

Identifies the source address of the multicast tree.

Multicast Group IP Address:

Identifies the address to which data will be addressed. Along with the multicast source IP address uniquely identifies the multicast session.

Initiator IP address:

The IP address of the node that initiated the JOIN-OPT-REQUEST.

TTL:

Difference in the hop count of the initiator of the JOIN-OPT-REQUEST and the potential new upstream node. This value is stored in the packet header.

Message type: 23

4.7. Summary

This chapter presented in detail a protocol for a source based, on-demand, multicast routing algorithm. This protocol conserves bandwidth by creating and maintaining a multicast tree only when absolutely necessary, and by performing route optimizations when possible. A reduction of redundant control and data packets is

possible by using the set of MPR nodes made available from the OLSR [14] routing protocol. By reducing the number of JOIN messages, routers are able to reduce overhead during link recovery and to eliminate advertisements for faulty routes. Chapter 5 describes the implementation of this routing algorithm.

Chapter 5. Implementation

5.1. Introduction

The multicast protocol was implemented using the C programming language in the Linux operating system environment. This software router was developed under Red Hat Linux version 7.2 using the 2.4 kernel. The information provided in this chapter includes, but is not limited to the development and manipulation of the data structures, neighbor sensing, communication, and routing table manipulation. Of utmost importance, the reader should be reminded that for multicasting to work, the Linux kernel MUST be configured to forward packets and with the IP Multicasting flag set.

5.2. Overview

While the mechanism that actually forwards data packets works at the kernel level, the multicast router works completely at the application layer. Linux, and other operating systems, provide an Application Programming Interface (API) that allows users to manipulate the kernel layer routing tables from application space. As the router receives control message from other nodes the application is able to learn enough about the network to make changes to the routing table.

5.3. Data Structures

Control messages with sequence numbers are recorded in a lookup table so that routers can identify the reception of duplicate packets. Messages are stored using a least recently used (LRU) scheme so that the data structures do not grow uncontrollably. Applications that use this method delete messages from the table after a certain period of time.

A data structure similar to the one used to store control messages is used to store multicast routing information, the list of one-hop neighbors, and the list of MPR nodes, if applicable. With the exception of a few modifications, the data structures and the functions that are used to manipulate these tables were taken directly from the original OLSR implementation code [41].

5.3.1. Basic Structure

The router implements the tables as a hash table, which is shown in Figure 5.1. The figure illustrates the structure used to store multicast group routing information.

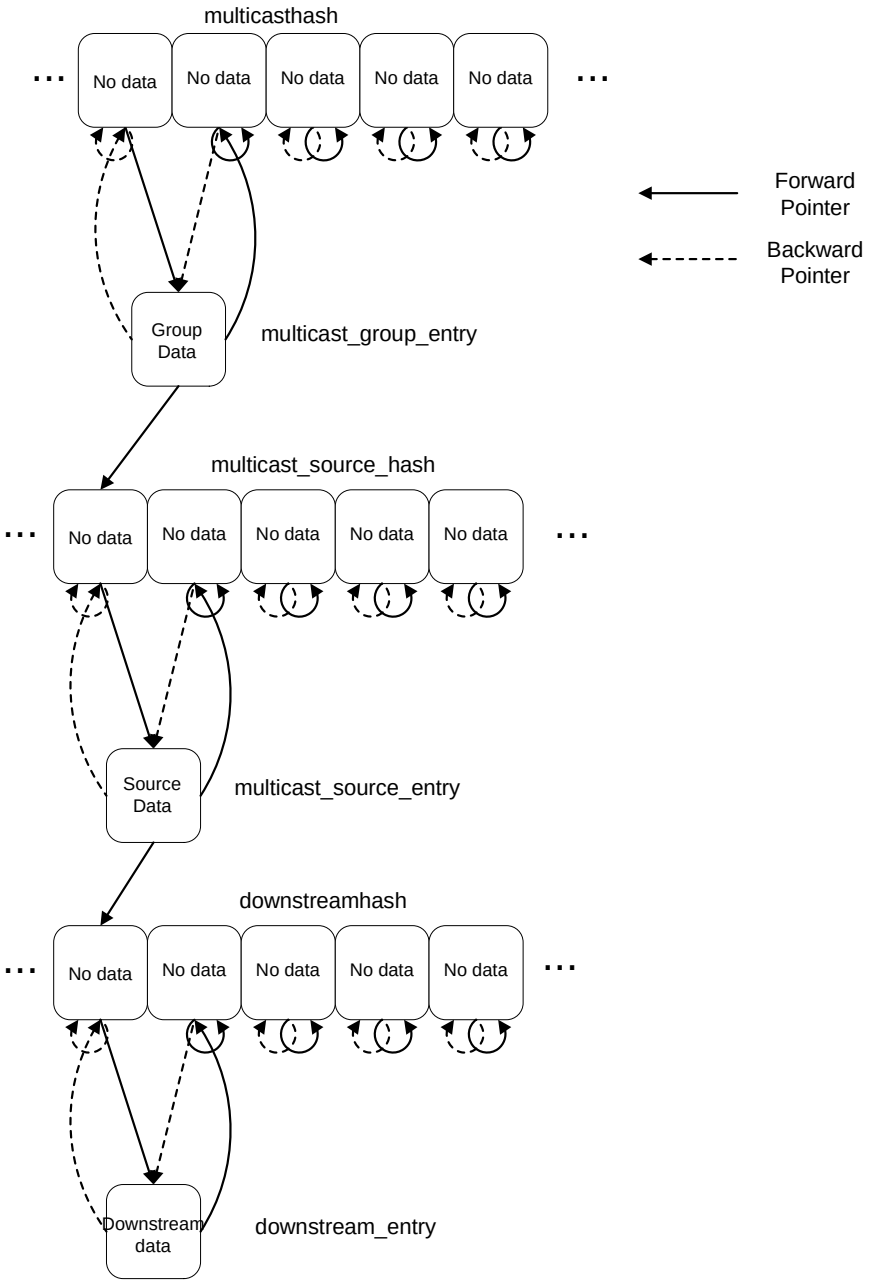


Figure 5.1. The multicast routing table data structure.

From Figure 5.1 the reader can see that the multicast routing table is simply a combination of three nested hash tables. The `multicasthash`, `multicast_source_hash`, and `downstreamhash` hash tables are arrays of pointers. The hash tables do not store any actual information about the multicast routes. The multicast entries, `multicast_group_entry`, `multicast_source_entry`, and `downstream_entry`, are used to store information about the entry. The data structure is organized in this nested manner because of the way in which multicast sessions are identified. Multicast routing entries are identified by unique a [source, group] pair. Because of this, any group can have a number of sources, and visa versa. Consequently, the information of most importance is stored in the source hash table. For example, the `multicast_group_entry` structure contains the following information.

- multicast group address
- hash number
- pointer to the multicast source hash table

In contrast, the `multicast_source_entry` structure contains the following information.

- multicast source address
- hash number
- address of the upstream node
- virtual interface (VIF) towards the origin (for kernel level manipulation)
- TREE-CREATE sequence number
- JOIN sequence number
- TREE-REFRESH sequence number
- hop count to the source
- minimum TTL that a packet must have to be forwarded
- state of the router
- TREE-REFRESH timer

- JOIN-WAIT timer
- NON_RECEIV (number of upstream nodes with only one downstream node)
- vector of TTLs, for kernel level manipulation
- a pointer to the table of downstream nodes, i.e. `downstreamhash`

Retrieving only the `multicast_group_entry` structure would leave the user devoid of the most useful information concerning the multicast session. For this reason the user must perform two lookups when obtaining multicast information. The first lookup returns the `multicast_group_entry` structure, from which, one can perform a second lookup and find the `multicast_source_entry` structure. For most of the tables a two-tiered data structure is sufficient to store the necessary data. Only the multicast routing table and the JOIN duplicate table use three-tiered systems. As shown in Figure 5.1, the multicast routing table depends on the group and source hash tables and a third table for the list of downstream nodes. The JOIN duplicate table contains a group and source hash table, in addition to a table that indicates the node that initiated the JOIN message. This is necessary because JOIN messages are identified by a [source, group, origin] triple. The tables that store the one-hop information, the MPR data, and the multicast state data use only a one-tiered system. Table 5.1 lists each data structure used by the router, the information that it stores and the number of tiers that it has. The C code for one example data structure is given in Appendix A.

Table 5.1. Tables Used by the Multicast Router

NAME	TIERS	INFORMATION
`neighbor_table`	1	One hop neighbors
`mpr_selector_table`	1	List of MPR nodes (optional)
`memberlist_hash`	1	Groups in which the router is interested
`refreshGrpHash`	2	TREE-REFRESH table
`createGrpHash`	2	TREE-CREATE duplicate table
`joinGrpHash`	3	JOIN duplicate table
`multicast_table`	3	Multicast routing information

5.3.2. Manipulating the Data Structures

The use of hash tables allows the user to quickly and efficiently store and retrieve data. Each hash table uses the IP address of the entry as a key. Because most entries are defined by a [source, group] pair, nested hash tables are necessary to store the data. All of the data structures use a basic set of functions, which allow the user to perform the following functions.

- insert an entry
- delete an entry
- lookup an entry
- delete all of the entries (in the case that the program ends)
- time out the entry (in the case that a timer expires)

Other functions calls used to manipulate the tables are simply derivations of these basic functions. For example, a function that takes the group and source addresses as input returns a pointer to a `multicast_source_entry` after performing two consecutive lookup calls. The first lookup is performed on the group table, and the second on the resulting source table.

5.4. State of the Router

Each router can take on several different roles in the network. Given a specific multicast session, a node can be a source, a receiver, or an intermediate node. Table 5.2 shows the states that a router can have for a given session.

Table 5.2. Multicast Route States

State	Description
`NOT_WAITING`	Not interested in the session and is not an intermediate node
`NOT_JOINED`	Interested in the session, but is not currently a part of the tree
`JOINED`	interested in the session and is currently a part of the tree
`SOURCE`	A multicast source for this session and is the root of a tree
`WAIT_SOURCE`	A multicast source for this session, but is not connected
`NOTJOINED_MID`	Not interested in the multicast data, forwarding data, but lost upstream link
`JOINED_MID`	Not interested in the multicast data, but currently a part of the tree

In addition to the state of the session, routers keep a list of multicast groups in which they are interested. This list is used to join existing sessions when an advertisement has not been received.

5.5. Neighbor Sensing

The multicast algorithm does not independently determine its one-hop neighbors. Instead, it depends on an underlying unicast routing algorithm to tell it which nodes are in its immediate vicinity. The multicast daemon retrieves the unicast routing information in the same way that the Linux command `route` [42] retrieves routes. Both `route` and the multicast router check a user space file created by the operating system to get routing information. This method of obtaining routes may prove to be problematic with another version of Linux, but it provides an easy way to gather topology information. This information is located in the file `/proc/net/route` and reflects the routing changes made by the unicast router. Part of the code from the `route` program [42] was used to extract this information from the file. The `/proc/net/route` file is checked frequently to ensure that the router has the most current routes that are available.

5.6. Communication and Sockets

A total of three sockets may be active at any one time in this multicast implementation. These sockets are used to send and receive control packets, to receive IGMP packets and manipulate the multicast kernel routing tables, and to communicate with a modified version of the OLSR routing daemon. The third socket is optional and can be used to retrieve the MPR selector set from the OLSR router. The MPR selector set [10] is the set of nodes that have selected this node as a MPR node. The multicast router can then use this information to optimize the flooding technique used to disseminate control messages.

A multicast router sends and receives multicast control packets using the User Datagram Protocol (UDP) on port 1026. Information concerning the set of MPR nodes is sent using port 1027 and also uses UDP as a transport mechanism. These port numbers were chosen because they have not been assigned to an application by the Internet Assigned Numbers Association (IANA) [43].

The IGMP socket is used to receive multicast requests and to manipulate the multicast routing table and MUST be of type RAW. This socket is used for two purposes. First, multicast routing information is sent to the kernel using this socket. Secondly, it is used to receive IGMP traffic [44]. Many of the functions used to interpret IGMP messages and to manipulate the kernel space multicast routing tables were adapted from the `mrouted` DVRMP routing daemon [45].

5.7. Manipulating Kernel Level Multicast Routing Tables

Luckily for the multicast developer, Linux provides an easy-to-use API that allows the user to add or delete multicast routes in the kernel. An online overview of this API and Linux multicast is given in a "how-to" document [44] that was used as a reference when implementing the multicast router. As mentioned before, manipulation of the multicast routing tables at the kernel level is done through the use of a RAW socket of type `IPPROTO_IGMP` [44]. The manipulations are carried out by sending a combination of routing information and flags to the kernel through the function `setsockopt()`[44]. This is the same function that is used to add and delete multicast membership, and to change multicast parameters, such as the TTL. The flags that most multicast applications, that is, those applications that receive and/or send data but do not worry about routing, use are as follows.

- `IP_MULTICAST_LOOP` (does the host receive data sent from itself?)
- `IP_MULTICAST_TTL` (TTL for sending data)
- `IP_MULTICAST_IF` (which interface to send on)
- `IP_ADD_MEMBERSHIP` (join a group)
- `IP_DROP_MEMBERSHIP` (leave a group)

Before it can forward multicast traffic, a router must first initialize the session by sending the flag MRT_INIT to the kernel. Next, the router must initialize the virtual interfaces. The use of VIFs is necessary because some multicast routers or hosts may use tunneling, which virtual interfaces take into account. The functions used to initialize and maintain the VIFs can be found in the file `molsr_vif.c`. The multicast implementation does not allow the use of multiple VIFs or interfaces. Only after the socket and VIF have

been initialized can the router begin to make changes to the routing table. As before, this is done by setting a flag to the `setsockopt()` function and passing a structure with the appropriate information. Some of the flags used to change the routing tables are as follows.

- `MRT_DONE` (stop multicast routing)
- `MRT_ADD_VIF` (add a VIF)
- `MRT_DEL_VIF` (remove a VIF)
- `MRT_ADD_MFC` (add a multicast route)
- `MRT_DEL_MFC` (delete a multicast route)

To add or delete a multicast route, information describing the route is passed to the `setsockopt()` function via a structure of type `struct mfcctl`. This data structure contains the following information.

- `struct in_addr mfcc_origin` (the multicast source)
- `struct in_addr mfcc_mcastgrp` (the multicast group)
- `vifi_t mfcc_parent` (VIF that receives data)
- `unsigned char mfcc_ttls[MAXVIF]` (the minimum allowable TTL)

The functions used to manipulate the routing tables can be found in the file `kern.c`.

Similar to adding a multicast route, a structure must be passed to `setsockopt()` when adding a VIF. This structure is of type `struct vifctl` and contains the following fields.

- `vifi_t vifc_vifi` (the index of this VIF)
- `unsigned char vifc_flags` (flags)
- `unsigned char vifc_threshold` (the TTL limit)
- `unsigned int vifc_rate_limit` (Rate limiter values)
- `struct in_addr vifc_lcl_addr` (local address on interface)
- `struct in_addr vifc_rmt_addr` (address of tunnel)

Deleting a VIF entry only requires that the user know the index of the VIF that is of interest.

5.8. Limiting Unnecessary Data Rebroadcasts

In wired networks routers forward data packets based on the interface upon which they received the packet. For example, a router may choose to forward every packet for a given [source, group] session that it received on interface A onto interface B. In no case does the router forward data onto the same interface from which it receives the data packet. Because of the broadcast nature of wireless interfaces, nodes receive and forward data on the same interface. This poses an interesting problem as the current multicast API cannot distinguish between a packet whose last hop was the upstream node and one whose last hop was a downstream node. A node A may receive a data packet from a downstream node B when node B rebroadcasts the data for its downstream nodes. Left unchecked, node A will forward the retransmission by B, which will then forward that packet, which node A will hear and rebroadcast. This can potentially lead to an infinite loop. One way to prevent a node from forwarding packets unnecessarily is to assign a sequence number to data packets and require a node to keep track of the data packets that it has already forwarded. However, this method would require that some alterations be made to the kernel.

Rather than make modifications at the kernel level, packets can be limited through the use of the time to live value of the packet. Each routing entry allows the user to specify the minimum TTL that a packet can have in order for the node to forward the data. By keeping track of the distance (number of hops) of the node from the source, the node can limit, but not necessarily eliminate, redundant rebroadcasts.

5.9. Summary

One advantage of this multicast implementation is that it operates completely in user space. This makes it simple to use and allows it to be ported to future Linux kernel versions more easily. Kernel layer modification may be needed for this implementation to fully eliminate redundant data rebroadcasts. However, the experiments with this router described in Chapter 6 focus on the performance optimizations within this implementation. Kernel-level inefficiencies should not significantly alter the relative performance of the optimizations.

Chapter 6. Performance Results

6.1. Introduction

This chapter discusses the validation and verification of the router implementation and the performance of the XMMAN protocol. The performance analysis compares most of the optimizations discussed in Chapter 4. The experiments were performed on a wired test bed which, through the use of a dynamic switch [37], simulated a mobile wireless environment while allowing the accurate replication of network topologies. The duplication of network topologies would have been impossible if the experiments were performed using a true wireless network. Initial results show that the optimizations described in Chapter 4 helped to create and maintain a more efficient multicast tree with a small increase in routing overhead. An efficient multicast tree is one that reduces the consumption of bandwidth by reducing redundant data rebroadcasts. This is also referred to as improving the fitness or quality of the multicast tree.

6.2. Validation and Verification

In verifying the operation of the routing protocol, one effectively answers the question, "Does the multicast router behave in the desired manner?" During the process of verification, perhaps better known as "debugging," the programmer makes sure that he or she has implemented the router so that it accurately follows the protocol specifications. An extensive series of tests were necessary to confirm the proper operation of the various functionalities of the XMMAN routing daemon. As the coding of the routing daemon proceeded, each procedure was carefully tested before continuing on to the next phase.

This was a "real" implementation, rather than a simulation, and the verification of the router was, at times, relatively simple. For example, to verify the proper operation of the multicast tree creation mechanism one simply needed to follow the path of the TREE-CREATE and TREE-CREATE-ACK packets. The developer could confirm that the packets contained the proper data and that the router responded appropriately through the use of debugging messages that were printed either to the screen or to a log file. Each function was tested in a number of network topologies that were designed to challenge and confuse the router. As the functionalities that were introduced became more and more complicated, so did the testing. It became increasingly difficult to test every

possible network topology that a router may have encountered. As a result some errors were not discovered until preliminary experiments had begun.

Usually simulations, not implementations, are put through the process of validation. Validation of a simulation is often achieved by comparing experimental and theoretical results. A theoretical description of the routing protocol does not exist and so it is difficult to make such a comparison. In general, it was assumed that the optimizations should improve the fitness of the multicast tree. Observing an improvement in the quality of the multicast tree during experimentation indicated the proper operation of the routing daemon.

6.3. Experimental Setup

6.3.1. Scenario

The experimental test bed consisted of eleven computers. Ten of these machines were used to form the "mobile" network and each ran a copy of the multicast routing daemon. The remaining computer was used as a dynamic switch. Table 6.1 lists the configuration of each of the machines used in the test bed.

Table 6.1. Machines Used in the Test Bed

Machine	Memory (MB)	Processor (MHz)
1	128	Pentium III 566
2	64	Pentium MMX 200
3	128	Pentium Pro 200
4	256	Celeron 600
5	192	Pentium II 266
6	256	Celeron 600
7	256	Celeron 600
8	64	Pentium II 333
9	128	Pentium III 566
10	64	Pentium Pro 200
Dynamic Switch	256	Pentium III 1000

Because multicast applications are often run with fewer senders than receivers the experiments were run with one sender and a variable number of receivers. Experiments were run using 1, 3, 5, 7, and 9 receivers. For a given experiment, the network had a low,

medium, or high mobility. For each scenario, a total of ten different network topologies were tested, with each run lasting five minutes. The network topologies were created using a program that simulated the pseudo-random movement of mobile nodes. As input, the program was given a number of parameters that described the movement of the nodes and a fixed radio transmission range. The output of the program was a file that described the coverage of each of the nodes. The format of the file was readable by the dynamic switch and indicated when to bring links up or down. This was done automatically by the dynamic switch. The mobility emulator was developed specifically for the purpose of controlling the type of experiment described in the following section. The dynamic switch was configured to not drop packet or constrain bandwidth.

6.3.2. Mobility Model

The mobility model that describes the pseudo-random motion of the nodes was taken from [46]. Using this model, the movement of nodes can be described as "jerky." A node moves at a random speed in a random direction for a random amount of time. Once this interval ends the node pauses for a set period of time, at the end of which it chooses a new direction, speed and time interval. This process is repeated until the end of the experiment. The motion of the node is described by four parameters, α, μ, *pauseTime* and *maxSpeed*. The parameter α quantifies the variability of the direction of the node. A node chooses a new direction by selecting a uniformly random number, in radians, between $-\alpha$ and $+\alpha$. The new direction of the node is the sum of this new random number and the old direction of the node. The time that the node spends moving is exponentially distributed. Parameter μ is the mean movement interval time. The exponentially distributed time interval, t, is generated using $t = -\mu \times ln(x)$, where x is a uniformly distributed random number from the interval [0, 1]. Parameter *pauseTime* quantifies the amount of time that a node pauses after each interval of movement. The pause time is constant and does not change throughout the course of the node's movement. A node's new speed is a uniformly distributed random number from the interval [0, *maxSpeed*].

For the purpose of these experiments, mobility was determined to be a function of *pauseTime*. A node with a large value of *pauseTime* is less mobile than a node with a

small *pauseTime*. Experiments were run for low, medium and high mobility networks. The initial placement of the nodes was determined by a random circular distribution. Using this method, a random angle, uniformly distributed over the interval [0, 2π], and a random radius, uniformly distributed over the interval [0, *maxRadius*], determined the initial position of the node. The motion of the nodes was confined to a finite rectangular area. Nodes began the experiments facing the same direction to simulate the motion of a group of ships that were traveling together. A node that reached the end of the rectangular area "bounced" off of the edge of the rectangle. The parameters used to determine the mobility and initial location of the nodes for each experiment are given in Table 6.2. Note that only the *pauseTime* parameter changes for different network mobilities.

Table 6.2. Mobility Model Parameters

Mobility	α (radians)	μ (secs)	*pauseTime* (secs)	Radio Range (meters)	*MaxSpeed* (m/s)	*MaxRadius* (meters)	Area (km)
low	1	40	**20**	2000	30	5000	50 x 50
medium	1	40	**10**	2000	30	5000	50 x 50
high	1	40	**2**	2000	30	5000	50 x 50

6.3.3. Multicast Data Transmission

For the duration of the experiment, the node designated as the sender would periodically transmit a UDP datagram that contained a sequence number and a time stamp to the multicast address 234.5.6.7. Receivers joined this multicast group appropriately. Lost packets indicated the occurrence and length of a link failure. For these experiments packets were sent every 500 ms. These messages were sent using a `sender` program that was developed specifically for these experiments.

6.3.4. Routing Protocol Optimizations

The experiments tested a number of routing optimizations. To provide a baseline, or control data set, a network of routers without any optimizations was used. This Basic routing protocol did not take advantage of MPR nodes, use route optimizations, or use the incremental JOIN or JOIN-NOTIFY method for repairing link failures. The performance of MPR and/or TREE-REFRESH optimizations was also measured. Several versions of

the Refresh optimizations were tested. One system allowed a node to attempt to join a node with a larger hop count, the other did not. These two systems are referred to as using NACK's or not using NACK's, respectively.

Route optimization through the use of JOIN messages was not fully tested because preliminary results showed that it did not improve the performance of the protocol. The Join route optimization was developed to allow nodes to respond to the JOIN-ACK message that adds the fewest number of intermediate non-receiving nodes to the multicast branch. This is accomplished without the use of the Join route optimizations, as a node should receive a JOIN-ACK from the closest node, i.e. fewest hops, first.

The optimizations were tested individually before any combinations of optimizations were attempted. The MPR_Refresh0N optimization is an example of such a combination. These two optimizations were combined because they provided the "best" individual results, in terms of a reduction of control and data overhead. Table 6.3 shows a complete list of the optimizations that were tested.

Table 6.3. Routing Protocol Optimizations Tested Table

Optimization	**Description**
Basic	Use no optimizations.
MPR	Use MPR nodes.
Refresh0N	TREE-REFRESH (T-R) opt with NACK. Leaf nodes do not rebroadcast T-R messages.
Refresh0	T-R opt without NACK. Leaf nodes do not rebroadcast T-R messages.
Refresh1N	T-R opt with NACK. Leaf nodes rebroadcast every T-R messages.
Refresh1	T-R opt without NACK. Leaf nodes rebroadcast every T-R messages.
Refresh2N	T-R opt with NACK. Leaf nodes rebroadcast 2nd T-R message.
Refresh2	T-R opt without NACK. Leaf nodes rebroadcast every 2nd T-R message.
Refresh3N	T-R opt with NACK. Leaf nodes rebroadcast every 3rd T-R message.
Refresh3	T-R opt without NACK. Leaf nodes rebroadcast every 3rd T-R message.
MPR_ Refresh0N	Use MPR nodes, with T-R opt, with NACK. Leaf nodes do not rebroadcast T-R messages.
Increase	Use increasing JOIN in branch rediscovery.
Notify	Use JOIN-NOTIFY in branch rediscovery.

6.4. Results

6.4.1. Parameters

Both the OLSR unicast routing protocol and the multicast routing protocol use timers in their operation. Timer values, along with a brief description of their purposes, are given in Table 6.4 and Table 6.5.

Table 6.4. OLSR Unicast Routing Protocol Parameters

Parameter	Value	Description
HELLO_INTERVAL	3 seconds	Send HELLO message
TC_INTERVAL	6 seconds	Send TC message
MIN_TC_INTERVAL	2 seconds	Minimum interval to broadcast topology changes
MAX_TC_INTERVAL	MIN_TC_INTERVAL	Maximum time to delay topology changes
TOP_HOLD_TIME	16 seconds	Time to delete entry in topology table

Table 6.5. Multicast Routing Protocol Parameters

Parameter	Value	Description
ROUTE_INTERVAL	2 seconds	Update one-hop neighbors
NEIGHB_HOLD_TIME	2 × ROUTE_ INTERVAL	Delete entry in one-hop table
DUPLICATE_HOLD_TIME	6 seconds	Delete TREE-REFRESH message
JOIN_HOLD_TIME	2 × DUPLICATE_ HOLD_TIME	Delete JOIN message
REFRESH_INTERVAL	6 seconds	Send a TREE-REFRESH message
REFRESH_HOLD_TIME	3 × REFRESH_ INTERVAL	Delete TREE-REFRESH message
CREATE_HOLD_TIME	20 seconds	Delete TREE-CREATE message
DOWN_HOLD_TIME	5 × NEIGHB_ HOLD_TIME	Delete downstream node
JOIN_WAIT_TIME	3 seconds	Wait for JOIN-ACK before sending JOIN
JOIN_NOTIFY_HOLD_TIME	6 × JOIN_ WAIT_TIME	Time out JOIN-NOTIFY message
FIND_TREE_TIME	45 seconds	Proactively look for the tree by sending a JOIN
FIND_ENTRY_TIME	2 × FIND_ TREE_TIME	Delete JOIN with source address 255.255.255.255
JOIN_SENT_REQUEST_TIME	3 seconds	Have not heard a JOIN-OPT-NACK (the node must not be downstream and I should join it)
JOIN_GOT_REQUEST_TIME	2 × JOIN_SENT_ REQUEST_TIME	Have not heard JOIN-OPT from initiator of JOIN-NOTIFY (I should join him)

6.4.2. Presentation of Results

Data was collected using a modified version of `tcpdump` [47] and `listen`. The `tcpdump` program was modified so that it would run for a specified interval and then exit. The `listen` program was created specifically for the experiments and recorded the timestamp and sequence number of received data packets. Control and data packets were captured by distinguishing between those packets found on port 1026 and those destined for the multicast address 234.5.6.7, respectively. The packet type of the control messages was stored in the data payload and was easily determined. By checking the Ethernet source MAC address one could distinguish between an incoming and an outgoing packet. A packet with an Ethernet source MAC address that was different from the address of the node was treated as an incoming packet.

There are obvious advantages to a "real" router implementation over a simulation. For one, an implementation can forward real data packets and allow real applications to run on mobile nodes. However, one encounters some interesting problems while attempting to quantify performance using a "real" router. Unfortunately, the router cannot distinguish between a new data packet that it should forward and one that it has already received and should not forward. As mentioned in Chapter 5, unnecessary data rebroadcasts can be reduced by limiting forwarding based on the TTL field of the incoming packet. However, such redundant rebroadcasts cannot be fully eliminated without some kernel layer modifications. This causes problems when examining the number of data packets in the system. The number of data packets may not always accurately reflect the fitness of the multicast tree.

A metric that more accurately reflects the fitness of the multicast tree is the number of TREE-REFRESH messages in the network. TREE-REFRESH messages are only sent to those nodes that are a part of the multicast tree and follow the same path that multicast traffic should take. Recall that TREE-REFRESH messages should be piggybacked on data packets. However, this method of measuring multicast tree fitness only works when nodes do not use a TREE-REFRESH route optimization where leaf nodes forward TREE-REFRESH messages.

The following section presents the results of the experiments. Multicast tree fitness is most often measured in terms of the number of TREE-REFRESH messages in

the network. Most of the results in this section are presented in graphical form, but a complete set of results in tabular form is available in Appendix B.

6.4.2.1. Medium Mobility

The results from the medium mobility experiments are presented first to provide baseline results with which to compare the low and high mobility results. The data presented in Tables 6.6 through 6.10 show the total average number of data packets in the network. That is, the result is the sum of the average number of packets that each node receives over each of the ten runs. Included in the table is the standard deviation of each of these trials. Also shown is the difference between the number of data packets for the given optimization and the basic multicast routing protocol. Improvements to the multicast tree are indicated by negative differences.

Table 6.6. Data Packets with Medium Mobility for Basic, MPR, and Refresh0

	Basic			**MPR**			**Refresh0**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	6014.2	2530.5	0.0	5710.2	2743.1	-5.1	6057.5	2951.5	0.7
3	**7648.9**	**1803.9**	**0.0**	**6741.4**	**1833.3**	**-11.9**	**7093.8**	**1899.1**	**-7.3**
5	8819.6	2192.9	0.0	8608.9	2130.7	-2.4	8479.6	2678.5	-3.9
7	**10492.0**	**3726.4**	**0.0**	**9427.2**	**2335.2**	**-10.1**	**9273.5**	**2359.6**	**-11.6**
9	11378.1	4131.5	0.0	9440.7	2658.1	-17.0	9288.8	2515.6	-18.4

Table 6.7. Data Packets with Medium Mobility for Refresh0N, Refresh1 and Refresh1N

	Refresh0N			**Refresh1**			**Refresh1N**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	5430.0	2951.5	-9.7	5716.2	2341.2	-5.0	5687.2	2343.9	-5.4
3	**6804.5**	**2669.5**	**-11.0**	**7064.3**	**2019.8**	**-7.6**	**6956.8**	**1563.3**	**-9.0**
5	7942.6	2428.8	-9.9	8339.7	2452.4	-5.4	8646.6	2580.7	-2.0
7	**9874.7**	**2864.6**	**-5.9**	**10162.9**	**3605.3**	**-3.1**	**9596.2**	**2949.9**	**-8.5**
9	9532.5	2437.3	-16.2	9967.6	2513.4	-12.4	9503.0	2551.8	-16.5

Table 6.8. Data Packets with Medium Mobility for Refresh2, Refresh2N and Refresh3

	Refresh2			Refresh2N			Refresh3		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	5669.8	2341.1	-5.7	5682.0	2342.6	-5.5	5939.5	2732.2	-1.2
3	**7631.8**	**2329.8**	**-0.2**	**7413.2**	**2191.4**	**-3.1**	**6871.5**	**1546.5**	**-10.2**
5	8457.4	2377.6	-4.1	8557.2	2537.2	-3.0	8353.1	2225.3	-5.3
7	**10171.6**	**3210.6**	**-3.1**	**9661.0**	**2717.5**	**-7.9**	**9597.8**	**2700.4**	**-8.5**
9	9411.4	2439.7	-17.3	9746.9	2524.2	-14.3	9159.7	2515.7	-19.5

Table 6.9. Data Packets with Medium Mobility for Refresh3N and MPR-Refresh0N

	Refresh3N			MPR-Refresh0N			Notify		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	5523.0	2300.8	-8.2	5847.5	2971.7	-2.8	5937.4	2452.5	-1.3
3	**7122.9**	**1953.0**	**-6.9**	**6660.9**	**1649.9**	**-12.9**	**7962.9**	**2432.6**	**4.1**
5	8654.7	2639.1	-1.9	8527.8	2898.0	-3.3	9381.1	3294.0	6.4
7	**9593.7**	**2982.8**	**-8.6**	**9102.1**	**2751.8**	**-13.2**	**9701.5**	**2748.8**	**-7.5**
9	9749.3	2534.4	-14.3	9424.1	2655.5	-17.2	9927.3	2756.7	-12.8

Table 6.10. Data Packets with Medium Mobility for Increase

	Increase		
Recv	**Packets**	**Stdv.**	**Diff. (%)**
1	6159.5	2255.3	2.4
3	**7380.9**	**2345.4**	**-3.5**
5	8719.8	2160.3	-1.1
7	**9518.0**	**2071.3**	**-9.3**
9	10465.0	2195.3	-8.0

To better illustrate these results, Figure 6.1 presents the data in graphical form. The reader should note that the optimizations perform well in dense multicast networks.

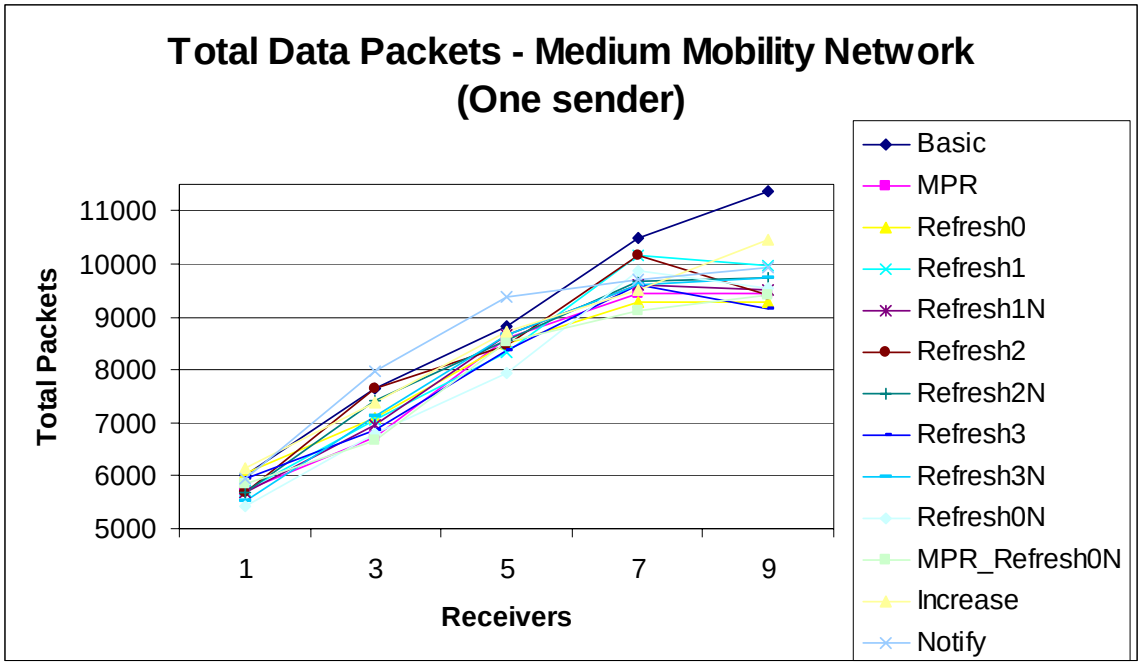


Figure 6.1. Total number of data packets in a network with medium mobility.

In terms of the reduction of data packets, the Refresh0 and Refresh0N systems performed relatively well compared to the other Refresh optimizations. Furthermore, leaf nodes in these systems do not rebroadcast TREE-REFRESH messages. The absence of superfluous TREE-REFRESH control messages is advantageous for two reasons. First, this reduction in control overhead conserves bandwidth. Figure 6.2 illustrates the control overhead used by each of the optimizations. As mentioned in Section 6.4.2, not having leaf nodes forward TREE-REFRESH messages provides an alternative way of measuring the fitness of the multicast tree. For these two reasons the Refresh0 and Refresh0N systems are the only Refresh optimizations presented from this point forward.

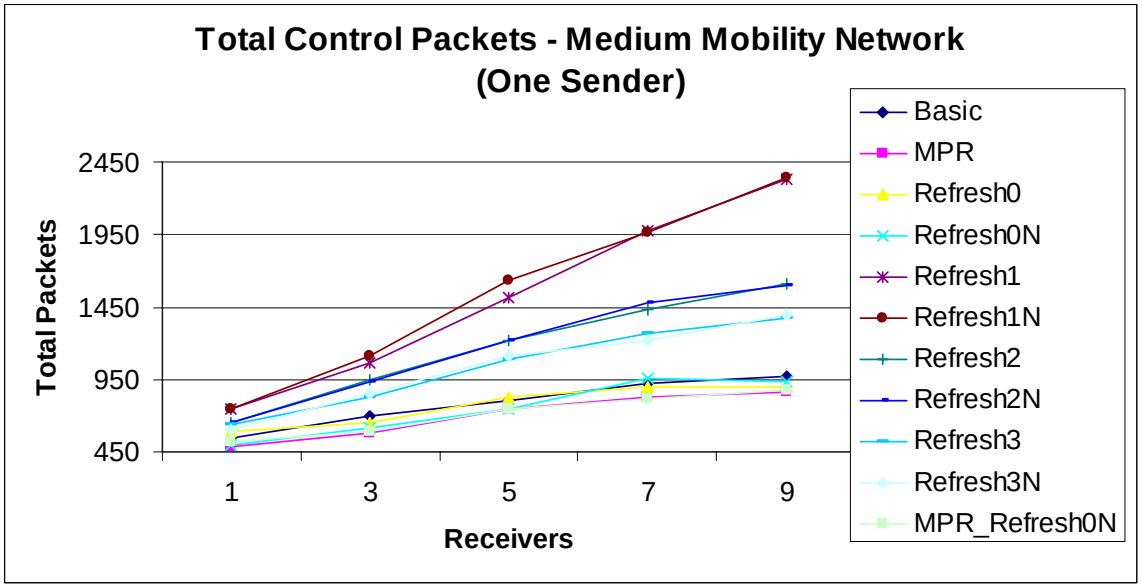


Figure 6.2. Total number of control packets in a network with medium mobility.

To improve clarity, Figure 6.3 presents the total number of data packets for only the Basic, MPR, Refresh0, Refresh0N, Notify, Increase and MPR-Refresh0N optimizations.

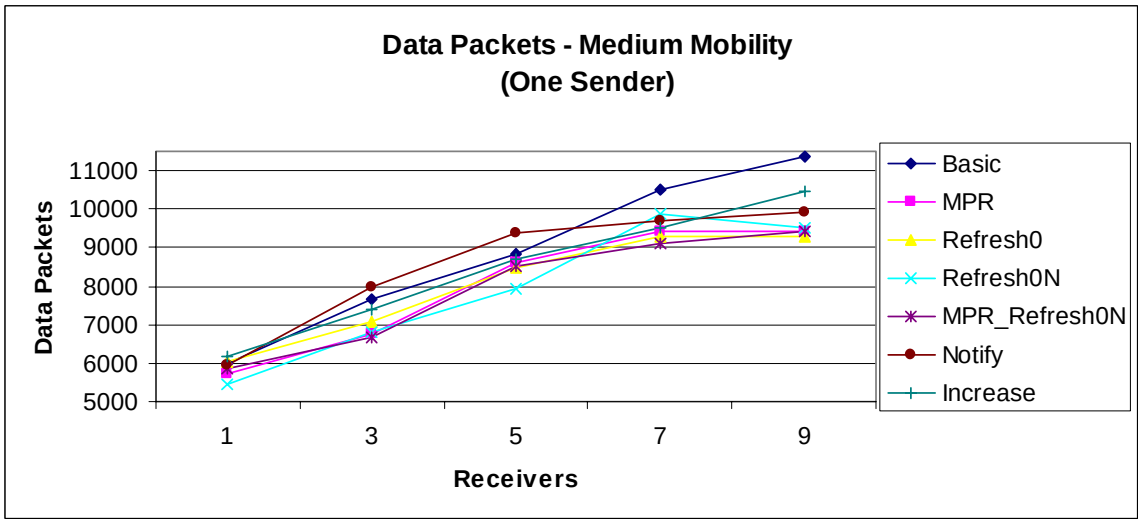


Figure 6.3. Total number of data packets in a network with medium mobility.

Close inspection reveals that the behavior of TREE-REFRESH messages closely mimics the behavior of the data packets. In terms of TREE-REFRESH messages, the improvements made between the Basic routing protocol and the various optimizations are

similar to the improvements seen in the total number of data packets. The graphs in Figure 6.4 and Figure 6.5 present the total number of TREE-REFRESH messages in the network and exhibit nearly identical behavior as seen in Figure 6.3. This evidence reinforces the assertion that the behavior of TREE-REFRESH messages reflects the fitness of the multicast tree. One should pay close attention to the behavior of the multicast protocol in dense networks. The results indicate that the protocol performs well as the number of receivers in the network increases. In all of the experiments large improvements were seen in the network with nine receivers. This hints at the protocol's ability to scale well in dense networks. Tables 6.11 through 6.13 show the number of TREE-REFRESH messages in tabular form.

Table 6.11. TREE-REFRESH Messages in Medium Mobility Network for Basic, MPR and Refresh0 networks

	Basic			**MPR**			**Refresh0**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	469.6	199.7	0.0	445.1	211.8	-5.2	480.7	244.0	2.4
3	**613.9**	**143.5**	**0.0**	**522.0**	**145.0**	**-15.0**	**563.5**	**143.4**	**-8.2**
5	707.6	182.4	0.0	671.1	161.7	-5.2	659.1	224.7	-6.9
7	**815.1**	**240.9**	**0.0**	**751.3**	**191.1**	**-7.8**	**741.9**	**180.9**	**-9.0**
9	859.1	248.5	0.0	758.6	222.8	-11.7	729.6	191.6	-15.1

Table 6.12. TREE-REFRESH Messages in Medium Mobility Network for Refresh0N, MPR-Refresh0, and Notify optimizations

	Refresh0N			**MPR-Refresh0**			**Notify**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Variance**	**Diff. (%)**
1	430.2	173.1	-8.4	454.1	236.7	-3.3	473.3	201.0	0.8
3	**532.3**	**206.3**	**-13.3**	**511.0**	**120.8**	**-16.8**	**627.4**	**193.3**	**2.2**
5	621.4	192.0	-12.2	677.0	222.6	-4.3	706.6	199.5	-0.1
7	**778.9**	**208.4**	**-4.4**	**711.4**	**218.3**	**-12.7**	**755.0**	**188.9**	**-7.4**
9	763.9	193.3	-11.1	751.4	209.0	-12.5	785.5	209.1	-8.6

Table 6.13. TREE-REFRESH Messages in Medium Mobility Network for the Increase optimizations

	Increase		
Recv	**Packets**	**Stdv.**	**Diff. (%)**
1	485.0	173.6	3.3
3	**584.4**	**192.3**	**-4.8**
5	683.2	182.5	-3.4
7	**765.3**	**164.5**	**-6.1**
9	829.9	185.4	-3.4

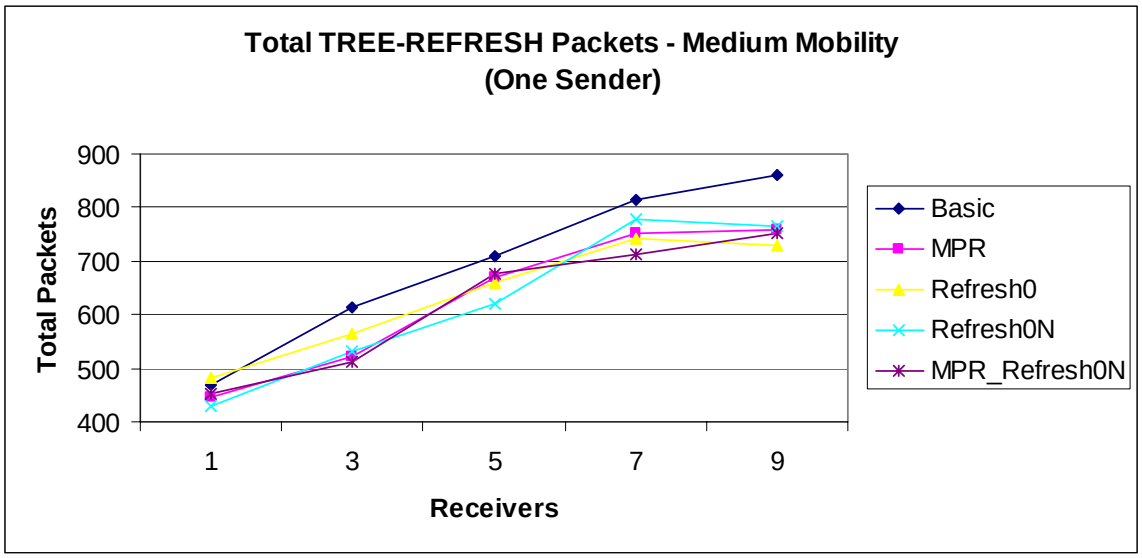


Figure 6.4. Total TREE-REFRESH packets in the network.

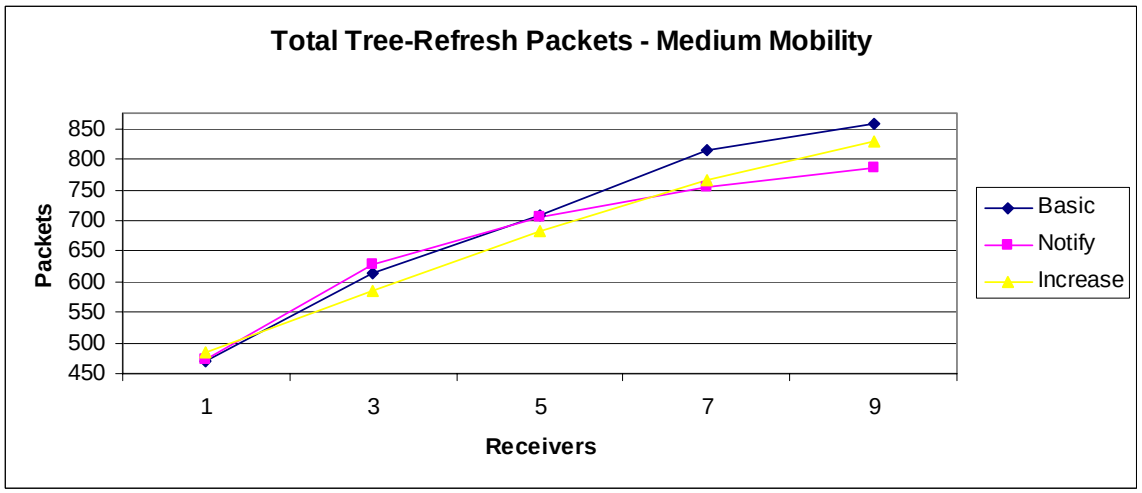


Figure 6.5. Total TREE-REFRESH packets in the network.

Unexpectedly, the Increase and Notify branch rediscovery mechanisms improved multicast tree fitness. The Increase method showed improvements in all five scenarios, while the Notify optimization worked well in dense multicast trees. The improvements made in the Increase network may occur when a node attempts to repair a link failure. The Increase mechanism attempts to fix a link failure locally by finding a new path to a branch that is only a few hops away, thus, minimizing data rebroadcasts.

The Notify optimization allows the lost host to respond to a JOIN-ACK message from a node with a larger hop count to the source. This may allow the node to find efficient routes that were not available while using other link repair methods. As the density of the tree increases this mechanism becomes more useful. It is more likely that in a dense tree a node with a larger hop count provides an efficient route to the source.

Improvements that are made to the multicast tree not only reduce data rebroadcasts, but also have the effect of reducing the amount of control overhead in the network. The reason for this is that most of the control overhead is caused by TREE-REFRESH messages. As discussed in Section 6.4.2, the optimizations effectively reduce the total number of TREE-REFRESH messages in the network. Figure 6.6 shows the total number of control messages in the network. It should be noted that in some cases the Refresh0 and Refresh0N optimizations use more overhead than the Basic router. This is due to an increase in control messages other than TREE-REFRESH messages.

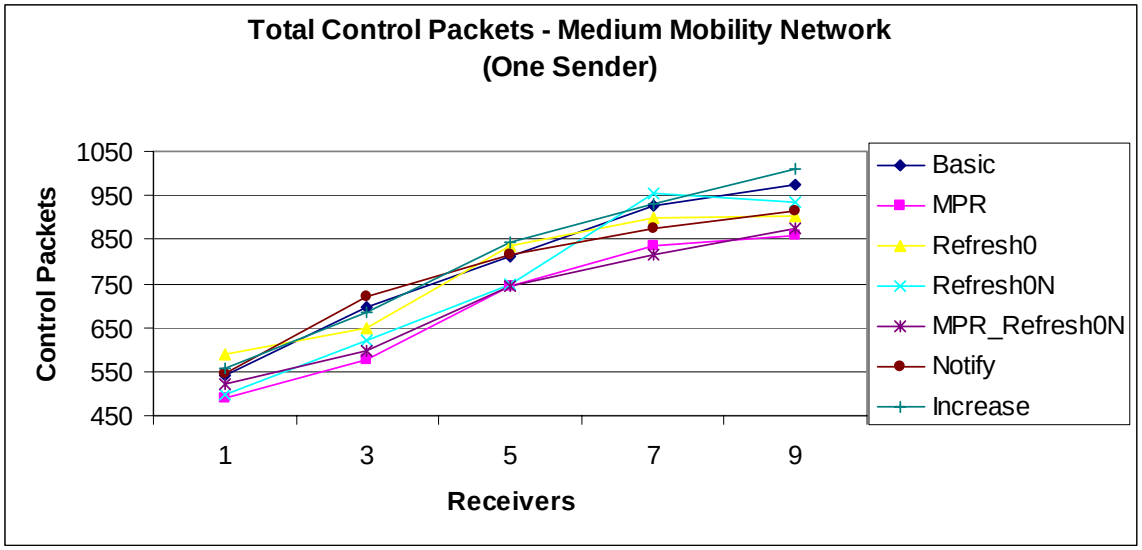


Figure 6.6. Total control packets in a medium mobility network.

While optimizations may reduce the total number of control packets in the network, Refresh, Increase and Notify optimizations may increase the number of non-TREE-REFRESH control messages. Figure 6.7 shows this phenomenon graphically. One can see that the Refresh0 and Refresh0N optimizations use more non-TREE-REFRESH control messages than the MPR or Basic optimizations. This is likely from the additional control overhead involved in the Refresh route optimizations. Additionally, the MPR-Refresh0N optimization shows an increase in data packets compared to the MPR optimization, except for the situation with five receivers, where an almost equal number of control packets were used.

The Increase optimization also uses a large number of non-TREE-REFRESH control messages. This is likely a result of the persistent nature of the Increase mechanism. Using the Increase method, a lost node sends JOIN messages with increasing TTL values until it receives a valid JOIN-ACK or the TTL exceeds a maximum value. This introduces a large amount of control overhead when a node has been partitioned from the network. In this case the node will broadcast JOIN messages until the maximum TTL value has been reached.

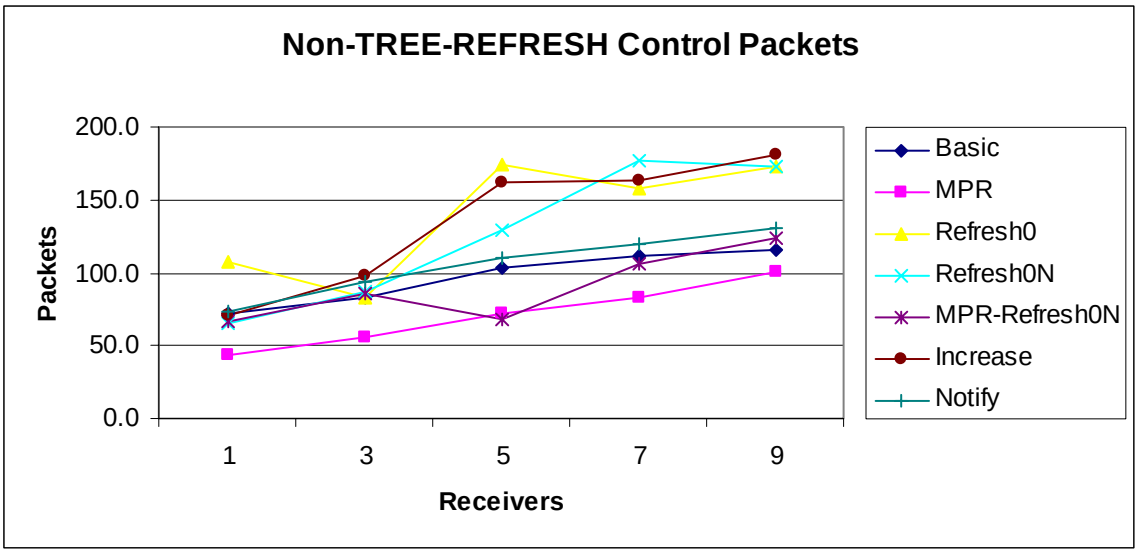


Figure 6.7. Total non-TREE-REFRESH control messages.

It can be helpful to investigate the behavior of the routing protocol as a function of time. Take for example the situation shown in Figure 6.8. This figure shows the

number of TREE-REFRESH packets in the seven receiver network as a function of time for a single run. Each point is a compilation of the total number of packets for a 15-second period. As the experiment proceeds, the difference in the number of TREE-REFRESH packets between the Basic and Refresh optimizations improves as compared to the Basic protocol and MPR optimization. The reason for this is that the Refresh optimization updates the multicast tree throughout the course of the experiment. MPR nodes, on the other hand, are only beneficial to the system during the initial tree creation phase. Figure 6.9 shows the differences in the number of TREE-REFRESH packets between the Basic scheme and the optimizations as a function of time. Note that the more negative the number, the greater the improvement in the fitness of the tree.

One should note that a slight reduction in the number of TREE-REFRESH messages occurs in the Basic and MPR networks as the experiment proceeds. This reduction is likely caused by improvements made when a link failure occurs. When a link repair occurs a node joins the first JOIN-ACK that it receives, which should advertise the path with the fewest number of intermediate non-receiving nodes. The Notify and Increase optimizations also improve the fitness of the tree as the experiment continues. This seems to suggest that branch rediscovery mechanisms can have a positive influence on multicast tree fitness.

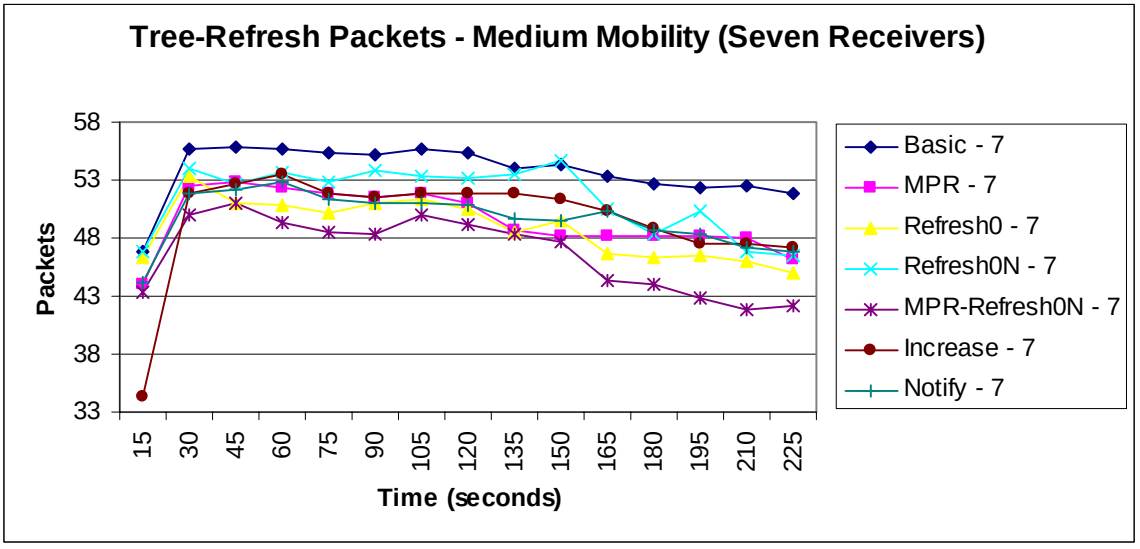


Figure 6.8. TREE-REFRESH packets as a function of time.

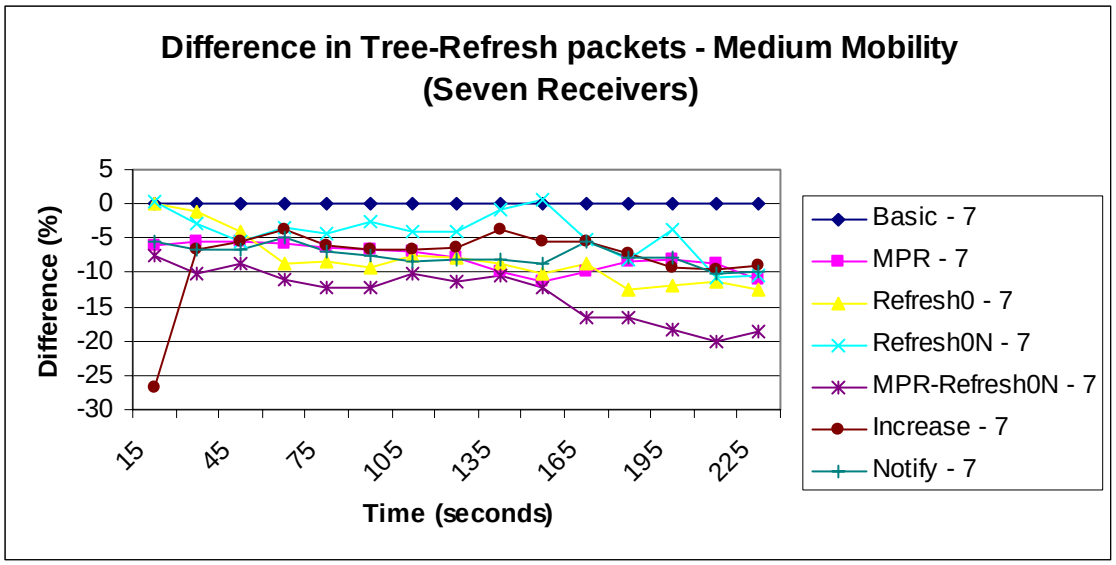


Figure 6.9. Difference between Basic network in TREE-REFRESH packets.

The Refresh optimizations improve the fitness of the multicast tree over the course of the experiment, but at the cost of additional overhead. Figure 6.10 shows the number of non-TREE-REFRESH messages as a function of time. Extra control packets can be seen as small spikes in the graph. These small spikes indicate points at which routing updates have taken place. During the first 15 seconds of the experiment one might notice three distinct starting points. Those networks using MPR nodes start with the lowest number of control packets. This is evidence that MPR nodes aid in the creation of the multicast tree. The large number of control packets in the Increase network may indicate that a link failure has occurred early in the experiment and that a node is searching for a route. This large number of control packets is only seen in the networks with five, seven and nine receivers, which reinforces the link failure hypothesis. A link failure that affects a receiver in the five node network will surely affect the seven and nine receiver networks. This theory is corroborated by data shown in Figure 6.8 and Figure 6.9. In these graphs the Increase network has a low number of TREE-REFRESH packets in the first 15 second interval. This would suggest that a link failure has occurred and has not been repaired preventing TREE-REFRESH messages, or data, from traveling though the network.

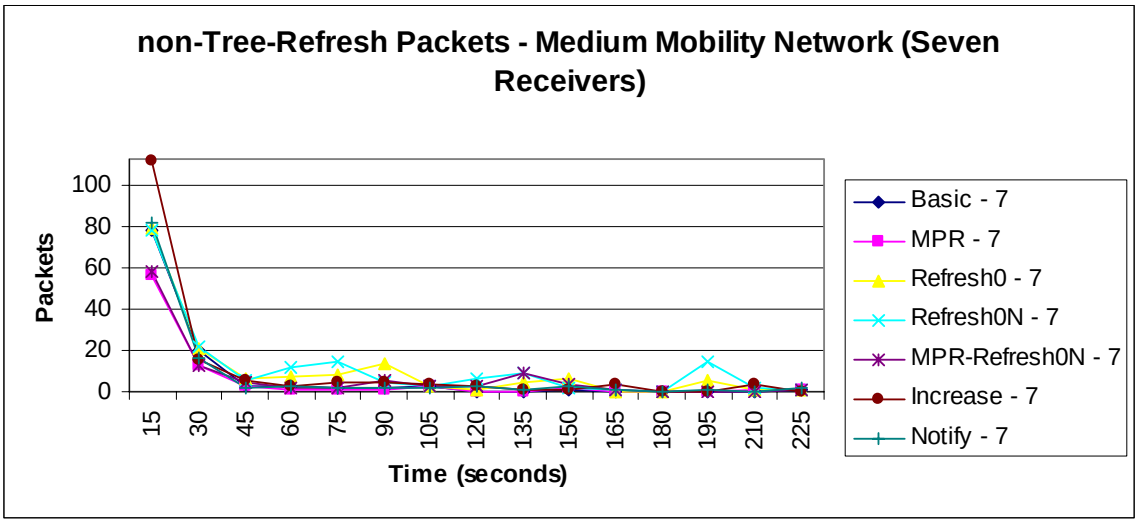


Figure 6.10. Number of non-TREE-REFRESH packets over time.

The following two subsections discuss results for high and low mobility scenarios and compare these results to the results for medium mobility network.

6.4.2.2. High Mobility

As the mobility of the network increases, the data forwarding problem becomes more apparent. For this reason, we must increasingly rely on the number of TREE-REFRESH messages to measure the fitness of the multicast tree. As evidence of this problem consider Figure 6.11. This is one example of the erratic behavior that is caused by the data forwarding problem.

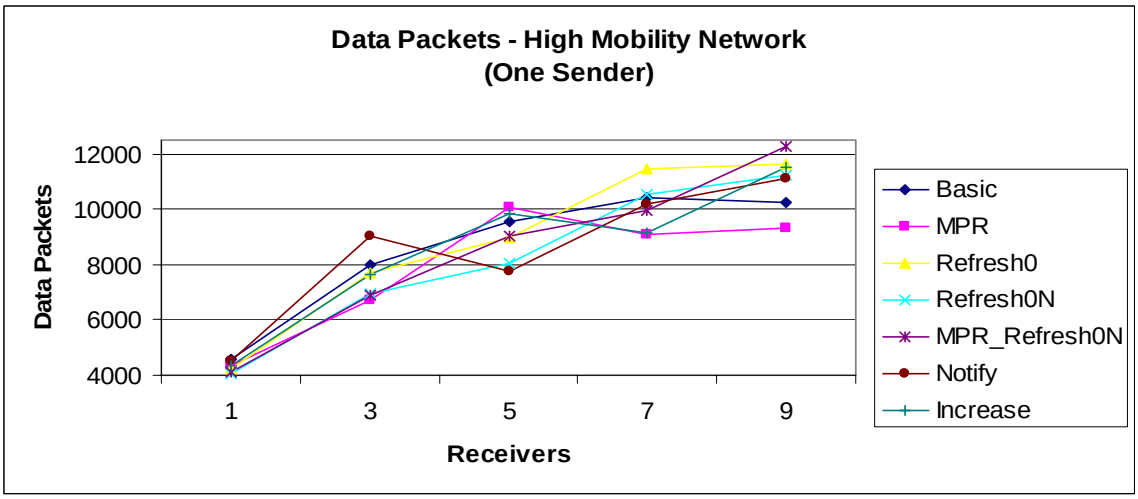


Figure 6.11. Total data packets in the high mobility network.

As discussed earlier, the number of TREE-REFRESH messages in the network more accurately portrays multicast tree fitness than the number of data packets. Figure 6.12 shows the number of TREE-REFRESH packets in the network. As in the case of medium mobility, the MPR and Refresh optimizations reduce the number of TREE-REFRESH packets in the network.

Results suggest that the difference between the MPR and Refresh optimizations may not be as exaggerated in the high mobility network. The MPR, Refresh0 and Refresh0N optimizations seem to perform equally as well in the high mobility network. The combination consisting of the MPR and Refresh0N optimizations show improvement in dense networks. This seems to indicate that MPR and Refresh optimizations alone cannot keep up with rapid changes in network topology. Combining the two optimizations allows the Refresh mechanism to make routing changes in a multicast tree that already has improved fitness. The improvements made by using MPR nodes during the initial tree creation phase can be lost when link failures occur. Rapid changes in topology may even lead to a lower quality MPR multicast tree than Basic multicast tree, as is evident in the scenario with five receivers. The high mobility data is presented in Tables 6.14 through 6.16 and Figures 6.12 and 6.13.

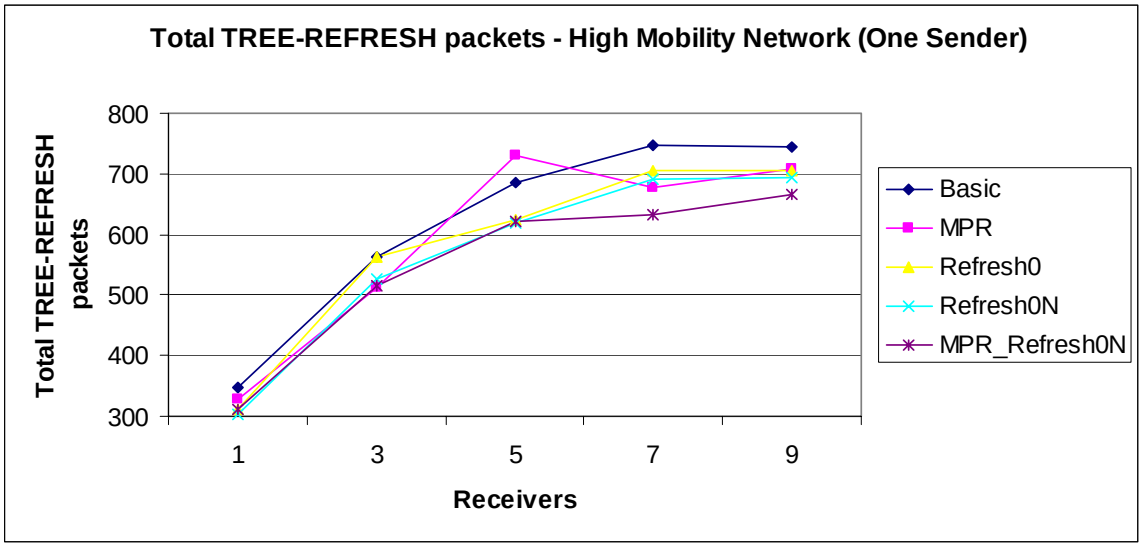


Figure 6.12. Total number of TREE-REFRESH messages in a high mobility network.

The Increase and Notify optimizations did not exhibit the same improvements seen in the medium mobility network. In the high mobility network the behavior of the two systems is erratic, as improvements are seen in some cases but not in others. This seems to indicate that while these two branch rediscovery mechanisms can affect the performance of the multicast protocol, their influence is not always a positive one. These two repair mechanisms take longer to perform than the basic link repair mechanism. These link repair methods may not be able to keep up with rapid changes in network topology.

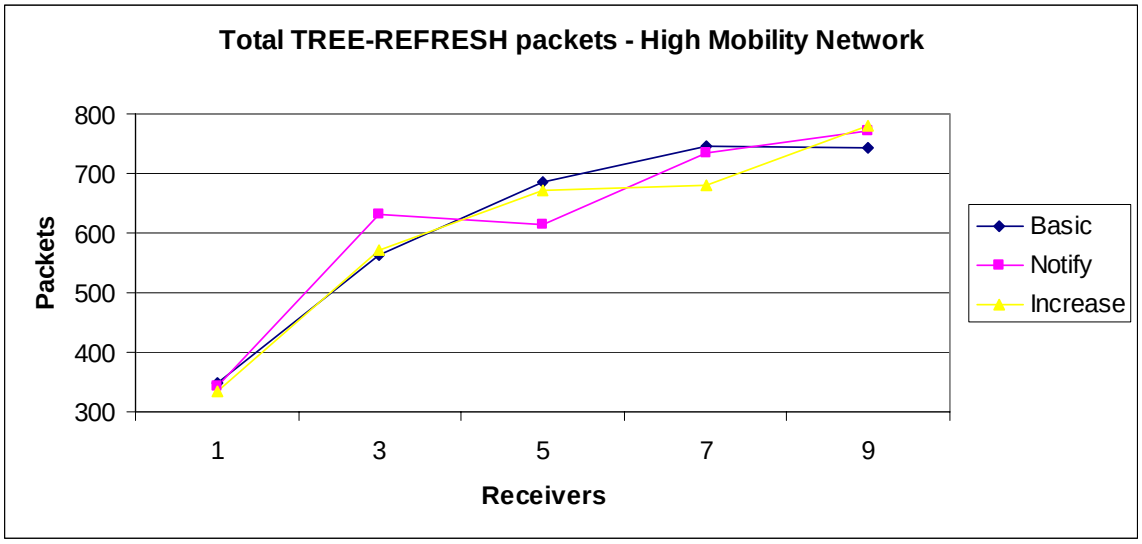


Figure 6.13. Total number of TREE-REFRESH messages in a high mobility network.

Table 6.14. TREE-REFRESH Packets in High Mobility Network for Basic, MPR and Refresh0 optimizations

	Basic			MPR			Refresh0		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	348.1	267.1	0.0	328.1	240.3	-5.7	310.8	240.2	-10.7
3	**561.6**	**205.1**	**0.0**	**511.3**	**196.4**	**-9.0**	**562.3**	**253.4**	**0.1**
5	685.4	267.2	0.0	729.5	354.4	6.4	624.8	279.2	-8.8
7	**745.9**	**372.1**	**0.0**	**675.9**	**323.4**	**-9.4**	**703.8**	**325.7**	**-5.6**
9	743.9	353.2	0.0	708.8	341.4	-4.7	705.2	323.1	-5.2

Table 6.15. TREE-REFRESH Packets in High Mobility Network for Refresh0N, MPR-Refresh0N and Notify optimizations

	Refresh0N			MPR-Refresh0N			Notify		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	302.8	221.5	-13.0	312.0	225.4	-10.4	343.0	270.6	-1.5
3	**525.1**	**202.7**	**-6.5**	**515.8**	**201.0**	**-8.2**	**630.2**	**275.7**	**12.2**
5	617.4	275.7	-9.9	622.0	262.9	-9.3	613.9	245.0	-10.4
7	**690.7**	**318.0**	**-7.4**	**632.1**	**290.7**	**-15.3**	**735.7**	**401.2**	**-1.4**
9	694.8	314.1	-6.6	667.0	288.8	-10.3	772.8	371.6	3.9

Table 6.16. TREE-REFRESH Packets in High Mobility Network with Increase Optimization

	Increase		
Recv	**Packets**	**Stdv.**	**Diff. (%)**
1	335.0	232.2	-3.8
3	**570.2**	**256.6**	**1.5**
5	672.0	278.5	-2.0
7	**679.4**	**300.4**	**-8.9**
9	779.3	366.9	4.8

Figure 6.14 shows the reduction of control packets that occurs in the high mobility network. As was true in the medium mobility network, the improvement in control overhead is mostly due to the reduction of TREE-REFRESH messages.

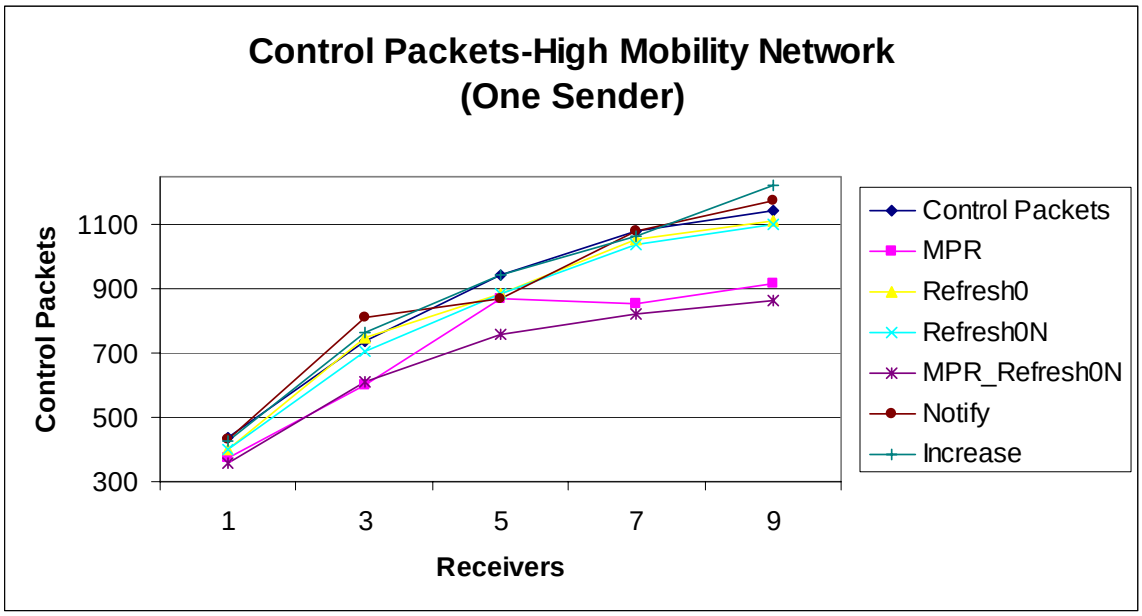


Figure 6.14. Total number of control packets in high mobility network.

Figure 6.15 illustrates the behavior of the non-TREE-REFRESH messages. Most of the non-TREE-REFRESH overhead is incurred during the tree creation and branch rediscovery phases. Routers that use MPR nodes can significantly reduce control packets during these phases. Note that the Refresh optimizations showed a slight increase in the number of non-TREE-REFRESH control packets. The Increase optimization leads to an exponential increase in control overhead as the number of receivers increases. This is to be expected, as more nodes are affected by link failures and participate in branch rediscovery. If these nodes happen to become partitioned from the network or a new route is far away, they add a large amount of overhead to the network from the multiple JOIN rebroadcasts.

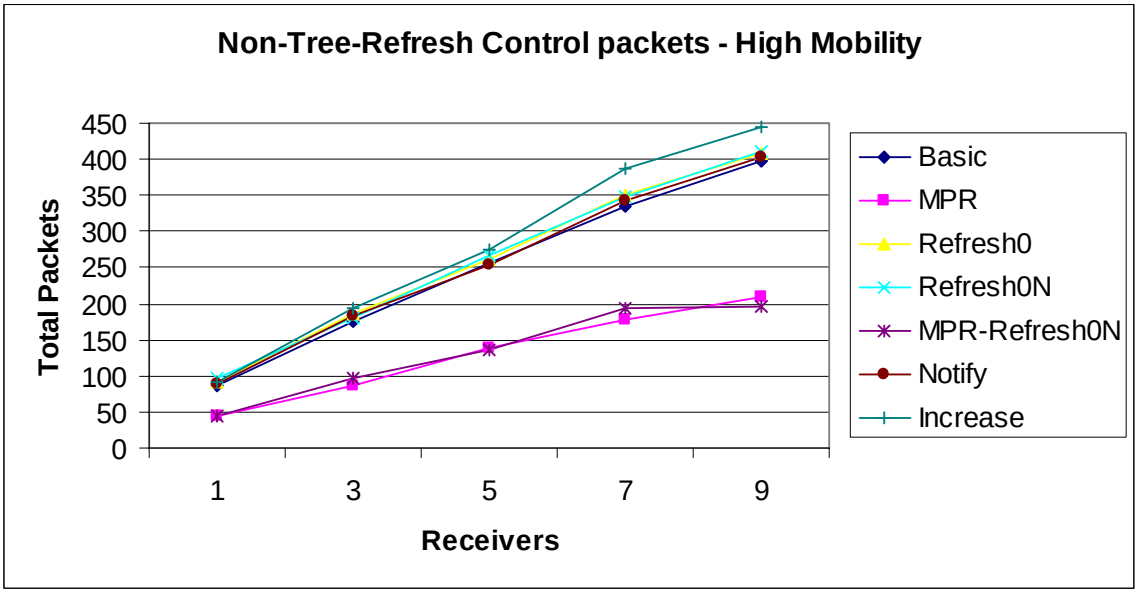


Figure 6.15. Total non-TREE-REFRESH control packets in high mobility network.

The next section presents the results of the experiments with low mobility.

6.4.2.3. Low Mobility

Improvements in the quality of the multicast tree were least dramatic in the low mobility network. Here, Refresh optimizations had a noticeably smaller impact on the fitness of the network. This is because the Refresh optimizations are most advantageous when many network changes occur. In fact, at some points, the Refresh optimizations

reduced the fitness of the network. Inefficiencies in the multicast tree are introduced when the network topology changes and the multicast tree does not change with it. The Refresh optimization eliminates some of these inefficiencies when changes occur, but, it is not used as often in a low mobility network. Figures 6.16 and 6.17 present the number of TREE-REFRESH packets in the network. We see in Figure 6.17 that the Notify and Increase mechanisms provide little or no advantage in terms of the number of TREE-REFRESH messages. Link failures in the low mobility network are rare and branch rediscovery is not often used. Tables 6.17-6.19 present the TREE-REFRESH messages in tabular form.

Table 6.17. TREE-REFRESH Packets in Low Mobility Network for Basic, MPR and Refresh0 optimizations

	Basic			**MPR**			**Refresh0**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	135.4	165.9	0.0	152.8	199.9	12.9	114.2	145.1	-15.7
3	**196.5**	**235.6**	**0.0**	**185.0**	**200.0**	**-5.9**	**178.2**	**192.9**	**-9.3**
5	206.2	236.9	0.0	207.1	235.0	0.4	174.9	168.6	-15.2
7	**247.2**	**238.7**	**0.0**	**202.6**	**175.1**	**-18.0**	**228.0**	**195.2**	**-7.8**
9	243.0	213.8	0.0	221.9	176.2	-8.7	240.4	205.0	-1.1

Table 6.18. TREE-REFRESH Packets in Low Mobility Network for Refresh0N, MPR-Refresh0N and Notify optimizations

	Refresh0N			**MPR-Refresh0N**			**Notify**		
Recv	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**	**Packets**	**Stdv.**	**Diff. (%)**
1	113.6	144.7	-16.1	118.1	142.4	-12.8	135.5	164.9	0.1
3	**164.6**	**161.8**	**-16.2**	**159.0**	**160.7**	**-19.1**	**192.6**	**218.4**	**-2.0**
5	191.2	189.8	-7.3	202.5	204.4	-1.8	198.3	209.1	-3.8
7	**210.0**	**176.8**	**-15.0**	**223.6**	**194.1**	**-9.5**	**258.0**	**241.1**	**4.4**
9	238.8	196.2	-1.7	231.1	173.2	-4.9	268.8	232.5	10.6

Table 6.19. TREE-REFRESH Packets in Low Mobility Network with Increase Optimization

	Increase		
Recv	**Packets**	**Stdv.**	**Diff. (%)**
1	131	166.6	-3.2
3	**202.2**	**234.5**	**2.9**
5	211.5	233.7	2.6
7	**234.5**	**216.2**	**-5.1**
9	268.7	233.8	10.6

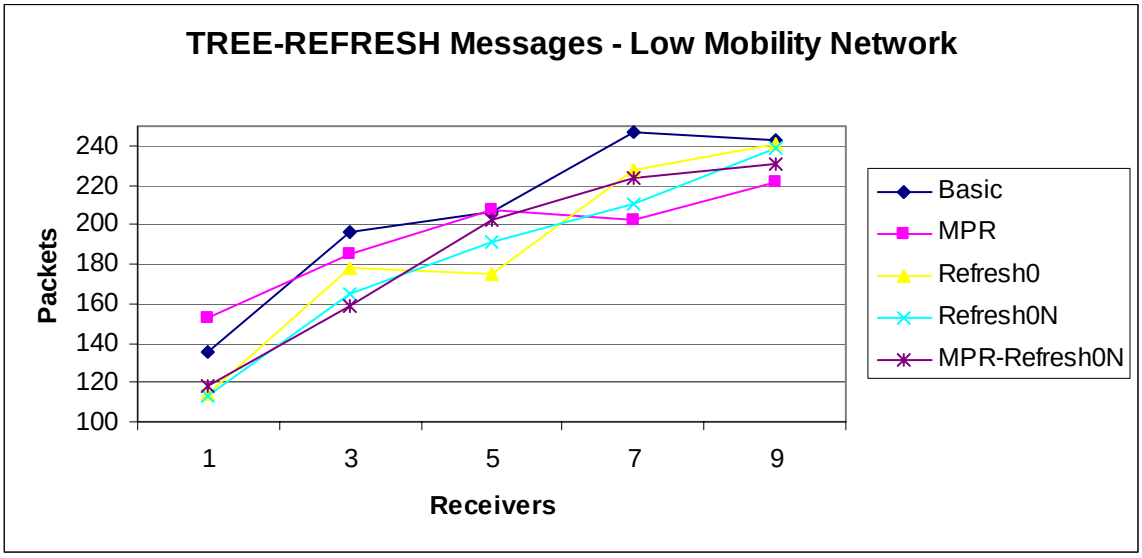


Figure 6.16. TREE-REFRESH messages in low mobility network.

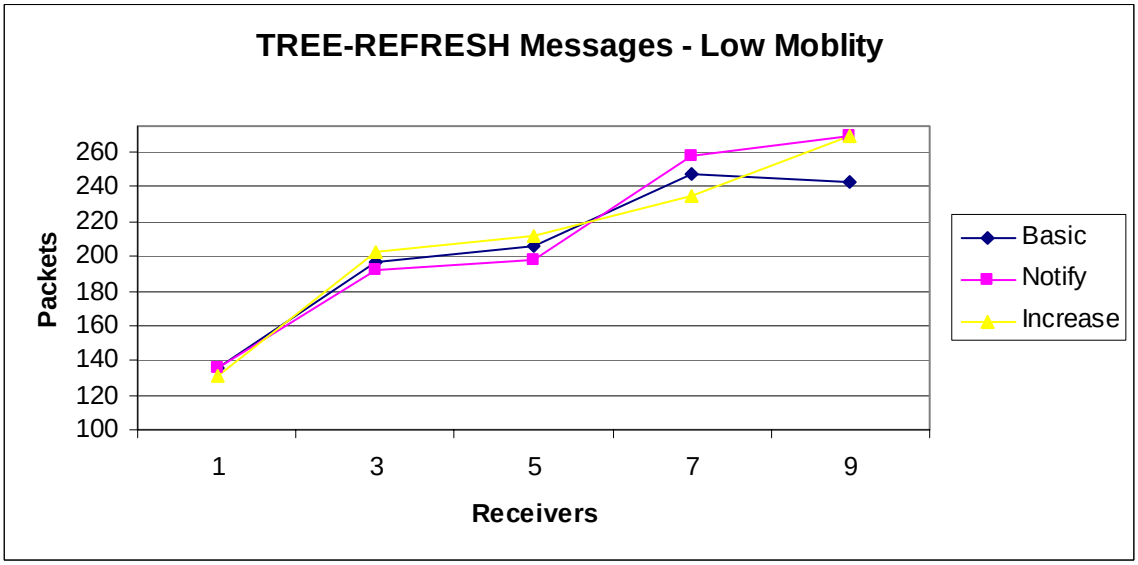


Figure 6.17. TREE-REFRESH messages in low mobility network.

As in the previous experiments, the Refresh and MPR optimizations result in a reduction of control packets. This is illustrated in Figure 6.18.

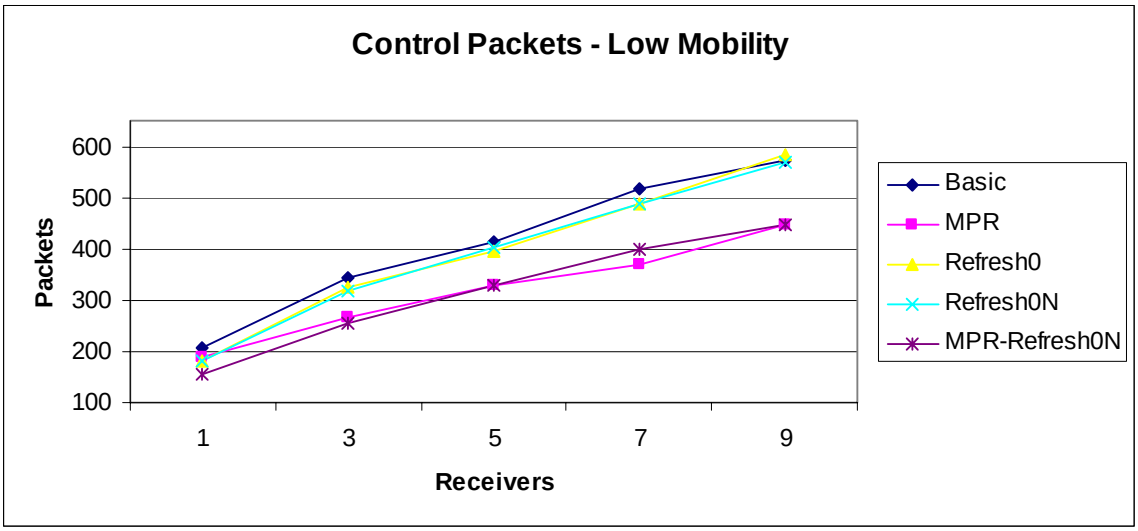


Figure 6.18. Control packets in low mobility network.

As one might expect, there are fewer non-TREE-REFRESH control messages in the low mobility network. Lower mobility leads to a reduction in the number of link failures and the need for branch rediscovery. Lower mobility also translates to fewer route optimization control messages. This is evident in the slight increase or reduction in non-TREE-REFRESH control messages in the experiments using Refresh optimizations. This is shown in Figure 6.19.

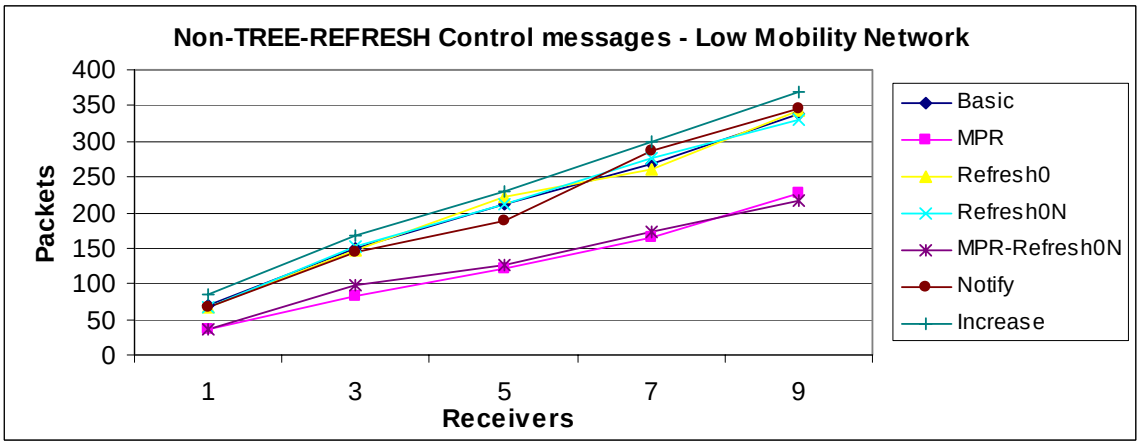


Figure 6.19. Non-TREE-REFRESH control messages in low mobility network.

6.5. Summary

Through the use of innovative routing techniques that take advantage of the broadcast nature of the wireless environment, the XMMAN protocol is able to reduce

both control and data overhead. The results presented in this chapter suggest that the Refresh and MPR optimizations described in Chapter 4 are especially adept at improving the performance of the protocol. The results also show that combining the two optimizations can further improve the multicast tree with minimal cost in terms of control overhead. While this implementation is merely a proof-of-concept prototype, it shows that when properly implemented, these extensions can improve performance.

The optimizations are economical in that they take advantage of existing mechanisms to improve fitness. Most reactive multicast protocols require that sources remind the network that they are still interested in the multicast session through some sort of signaling mechanism [20, 22, 27]. The Refresh optimization uses this mechanism not only to inform nodes that the multicast session is still active, but also to give nodes a sense of the environment around them. It was also shown that OLSR is especially suited to the task of multicast tree creation. By reducing unnecessary rebroadcasts, OLSR is able to aggregate multicast data flows and reduce both data and control overhead. By "borrowing" the set of MPR nodes from the OLSR unicast routing protocol the XMMAN router was able to provide this extension without introducing any additional overhead into the network. With the goal of reducing data overhead, these optimizations are able to improve the fitness of the multicast tree with minimal increases in overhead.

The MPR and Route optimizations were shown to improve the performance of the XMMAN protocol at different stages of the multicast session's life cycle. The use of MPR nodes is beneficial during the tree creation phase of the multicast session, while the Refresh optimization is able to make adjustments for the entire session. The multicast algorithm MOLSR [23] takes advantage of MPR nodes for the duration of the multicast session, but at the cost of bandwidth-hungry network broadcasts. It may be beneficial to investigate a mechanism through which MPR nodes could be exploited throughout the entire life of the network without the use of network broadcasts. High mobility causes degradation in the advantages gained by using MPR nodes during the tree creation phase. In one example, it was even shown that the use of MPR nodes could reduce the fitness of the multicast tree. Through a combination of the MPR and Refresh optimizations, the multicast routers are able to keep up with rapid changes network topology.

The MPR and Refresh optimizations have a smaller impact on the low mobility network. Refresh optimizations are designed to make improvements when changes in network topology cause inefficiencies in the multicast tree. With low mobility, changes in network topology are rare and Refresh optimizations are used infrequently. In fact, it was shown that in a low mobility network the Refresh optimization can reduce the fitness of the tree. In general, the MPR and Refresh optimizations work best in high density multicast trees. As the number of multicast receivers increases, so do the number of overlaps in coverage and the chance that redundant rebroadcasts can be reduced.

Surprisingly, the Increase and Notify optimizations affect the fitness of the multicast tree. Unfortunately, the effect is not always an improvement. The branch rediscovery systems improved multicast tree fitness in the medium mobility network, but had mixed results in the high mobility network. In the case of the low mobility network, the Increase and Notify modifications showed a small difference in a low density multicast tree, and had a slight degradation in performance in the highest density networks.

Two promising techniques, one old and one new, were shown to reduce data overhead in multicast trees. While originally developed to reduce the number of rebroadcasts of control messages during the maintenance of unicast routes, it was shown that MPR nodes can improve multicast tree fitness by aggregating data flows. Through the exploitation of TREE-REFRESH messages, nodes are capable of learning enough about the network so that they can make routing changes that improve the multicast tree.

Further investigation should look into how to disseminate MPR information for the duration of the entire multicast session and into the improvement of the Refresh route optimization method. Another topic of interest is the method by which link repairs are performed. It was shown that branch rediscovery mechanisms can affect performance. Further research should look into how to force these changes to be beneficial. In general, future multicast algorithms for mobile *ad hoc* networks should consider the reduction of data overhead.

Chapter 7. Conclusion

7.1. Summary

Existing multicast protocols for MANETs strive to improve the reliability of mobile networks while reducing routing overhead, in order to relieve the burden on low data rate wireless links. Through these routing techniques, designers of MANETs hope to help provide high speed communication between wireless mobile nodes. Despite these innovative approaches, current MANET technologies are lacking. This thesis proposed a new approach to building and maintaining multicast routes, one that dictates that the creation of efficient multicast trees should be considered when developing wireless multicast algorithms.

Previously, engineers ignored the fact that it is data traffic, not control traffic that usually consumes most of the bandwidth in a network. Efficient routes could help to free up some of this bandwidth for use by other applications. Unlike a unicast route, where only one path between a sender and a receiver exists, multicast trees must maintain several branches between a sender and multiple receivers. An efficient multicast tree would be one in which some of these branches were combined. This would reduce the number of data packet rebroadcasts needed to reach every multicast receiver. The failure of existing protocols to address the issue of efficient multicast tree creation was the main motivation for this thesis.

The methods used to create and maintain multicast trees proposed in this thesis take advantage of the broadcast nature of wireless links and attempt to aggregate data flows. However, rather than present a completely new multicast protocol, this thesis built on existing protocols and offered optimizations in line with this new multicast philosophy. By “borrowing” the set of MPR nodes from a OLSR unicast router, nodes were able to control the number of redundant rebroadcasts during network broadcasts. This not only limited the routing overhead in the network, but also helped to create a multicast tree in which data flows were aggregated. With MPR nodes, TREE-CREATE messages, used to advertise a new multicast tree, followed an efficient path during the tree creation phase network broadcast. Because multicast routes were built in a backward learning manner, efficient routes were created. Although MPR nodes were first developed to aid in the maintenance of unicast routes, it was shown that they could

effectively reduce data and control overhead by aggregating both data and routing messages.

Including topology information on TREE-REFRESH messages allows a node to learn enough about its network such that it can make routing changes to reduce data overhead. From this information, a node knows the status of its upstream parent and the fitness of the branch that it lies on. Because wireless links broadcast messages, nodes often overhear TREE-REFRESH messages from routers that are not their parents. When this situation occurs, a host can compare the fitness of the new branch with the fitness of the branch on which is currently resides. If this new branch offers a more efficient route the node may then make an appropriate routing change.

7.2. Conclusions and Contributions

The research in this thesis was done as a part of the Navy Collaborative Integrated Information Technology Initiative (NAVCIITI) project [48], and was funded under a grant from the Office of Naval Research (ONR) [49]. The goal of this project was to provide seamless communication across a heterogeneous mobile wireless network. This would be beneficial to an international naval task force, such as was described in Chapter 2, Section 2.1.2. This MANET multicast algorithm could facilitate the use of multicast applications for communication between ships that do not have a direct line of sight or the benefit of a satellite link.

Perhaps one of the most useful contributions to the scientific community is that this router was implemented as a “real” software multicast router, and not as a simulation. In general, implementations are more useful than a simulation. This implementation allows a host to forward multicast data packets in a wireless mobile network. Even though the data forwarding problem exists, this implementation provides a foundation upon which other researchers can use to begin to perform experiments in a real world environment.

One disadvantage of having a real implementation is that experiments are often more limited in size than are simulations. Even though these experiments used a wired dynamic switch to control the network topology, a limited number of nodes could be used in the network. However, the results seem to suggest that the protocol will scale well in a

large network. From the tables and figures presented in Chapter 6 one sees that the number of TREE-REFRESH packets increases more or less linearly. This would indicate that the optimizations will continue to work well in a large network. The results of the high mobility network tapered a bit as the number of receivers increased, but this could be caused by experimental error. Other results that were not presented in this thesis did not show this asymptotic behavior. The reader may also be interested to know what happens when multiple senders are present in the multicast tree, as this situation was not tested in this thesis. Multicast trees are source-specific, and as such, are rooted at the source. Adding a multicast source would require that a new multicast tree be created. This new tree would be entirely independent of the original multicast session, and optimizations would not be shared across multicast trees. An increase in the number of multicast sources should bring a proportional increase in control and data overhead.

The XMMAN protocol shares the same advantages and pitfalls present in any reactive MANET algorithm. As mentioned in Chapter 2, reactive MANET protocols can conserve bandwidth by using routing control messages only when a routing change is explicitly needed. Proactive protocols, on the other hand, actively look for changes in topology and constantly update the routing tables. This liberal use of routing packets clearly will cause an increase in control overhead. On the other hand, proactive routing protocols repair link failures more quickly, and generally, packet loss is less common in a proactive network.

More than being just another MANET multicast algorithm, this protocol proposes a new way of approaching the problem of multicast in wireless networks. Developers of MANET multicast protocols are usually concerned with improving reliability and reducing overhead and seem to ignore the problem of efficient multicast tree maintenance. New techniques that reduce data overhead were proposed and were shown to be successful. Existing techniques, when modified to work in a multicast tree, were also shown to improve multicast fitness. If nothing else, this thesis will hopefully make developers aware of this important dimension of multicast routing and will provide a foundation upon which they can develop new multicast routing techniques.

7.3. Future Work

Further research into this protocol should include an attempt to improve the Refresh optimization and should involve the development of a mechanism that can take advantage of MPR nodes for the duration of the entire multicast session. It was observed that branch rediscovery mechanisms could affect the performance of the multicast protocol. It would be interesting to explore a mechanism that could guarantee that efficient routes were created during link recovery. Just as multicasts routes are created in an efficient manner, broken links be repaired efficiently. When nodes respond to a JOIN message it may be possible to include some information that would allow lost nodes to choose the "best" new route.

It would also be useful to eliminate the data forwarding problem. The development of a new API by which to manipulate multicast routes in wireless networks would be beneficial to all of those working in the wireless multicast field. Unicast routers are able to eliminate this problem by using the address resolution protocol (ARP) tables to compare the hardware address, or MAC address, of the last hop of the packet. The data forwarding problem could be solved if the multicast API allowed the user to modify routes according to [source, group, last hop] triples rather than [source, group] pairs.

The development of a simulation for the ns/2 simulation would be extremely useful. Many of the proposed multicast routing protocols have been simulated using ns/2, and a simulation of XMMAN would allow one to make performance comparisons with these other protocols. Because of the data forwarding problem, such metrics as throughput, end-to-end delay and the percentage of received packets are difficult to measure. A simulation would allow a user to simulate these results and to make comparisons with other protocols.

References

[1] S. Corson and J. Macker, “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations,” IETF Request for Comments 2501, January 1999.

[2] C. Perkins, “IP Mobility Support for IPv4,” IETF Request for Comments 3220, January 2002.

[3] IEEE Std 802.11-1997, Wireless LAN Medium Access Control and Physical Layer Specifications, IEEE, Piscataway, NJ, 1997.

[4] IEFT MANET workgroup webpage, http://www.ietf.org/html.charters/manet- charter.html.

[5] F. Templin, IETF MANET Working Group Mailing List archive, ftp://manet.itd.nrl.navy.mil/pub/manet/2001-12.mail.

[6] J. Markoff, “The Corner Internet Network vs. the Cellular Giants,” New York, NY, *The New York Times*, March 4, 2002: C1.

[7] C. von Clausewitz, “On War,” Bibliomania, http://www.bibliomania.com/2/1/61/108/frameset.html.

[8] E. M. Royer and C.-K. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," *IEEE Personal Communications*, Vol. 6, No. 2, pp. 46-55, Apr. 1999.

[9] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. G. Jetcheva, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks,” IETF MANET Working Group Internet Draft (draft-ietfmanet-dsr-05.txt), March 2001, Work in Progress.

[10] P. Jacquet, A. Qayyum, T. Clausen, A. Laouiti, L. Viennot, P. Minet, and P. Muhlethaler, “Optimized Link State Routing Protocol,” IETF MANET Working Group Internet Draft (draft-ietf-manet-olsr-05.txt), November 2001, Work in Progress.

[11] J. Moy,” OSPF Version 2,” IETF Request for Comments 2328, April 1998.

[12] G.Holland and N. Vaidya, “Analysis of TCP Performance over Mobile Ad Hoc Networks,” *Proceedings of MobiCom '99*, Seattle, WA, August, 1999, pp. 219--230.

[13] E. Dijkstra, “A note on two problems in connection with graphs”, *Numerische Mathematik*, Vol. 1, pp 269-271, 1959.

[14] A. Qayyum, L. Viennot, and A. Laouiti, "Multipoint Relaying: An efficient technique for flooding in mobile wireless networks," French National Institute for Research in Computer Science and Control (INRIA), Research Report No. 3898, March 2000.

[15] J. Moy, "Multicast extentions of OSPF," IETF Request for Comments 1584, March 1994.

[16] T. Pusateri, "Distance Vector Multicast Routing Protocol," IETF Internet draft (draft-ietf-idmr-dvmrp-v3-10), August 2000.

[17] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering,M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification," IETF Request for Comments 2362, June 1998.

[18] A. Adams, J. Nicholas and W. Siadak "Protocol Independent Multicast - Dense Mode (PIM-DM):Protocol Specification (Revised)," IETF Internet Draft (draft-ietf-pim-dm-new-v2-01.txt), February 2002 , Work in Progress.

[19] L. Peterson, and B. Davie, "Computer Networks: A Systems Approach, Second Edition," San Francisco, CA, Morgan Kaufmann Publishers, 2000.

[20] E. Royer and C Perkins, "Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol," In *Proceedings Of the 5th Iinternational Conference on Mobile Computing and Neworking (MobiCom99)*, pp 207-208, Seattle, WA, USA, August 1999.

[21] S.-J Lee, C. Chiang and M. Gerla, "On-Demand Multicast Routing Protocol," In *Proceedings of IEEE WCNC'99,*New Orleans, LA, September 1999, pp. 1298-1304.

[22] V. Devarapalli, "MZR: A Multicast protocol for Mobile Ad Hoc networks," IETF MANET Working Group Internet Draft (draft-vijay-manet-mzr-00.txt), November 2000, Work in Progress.

[23] P. Jacquet, P. Minet, A. Laouiti, L. Viennot, T. Clausen, and C. Adjih, "Multicast Optimized Link State Routing," IETF MANET Working Group (draft-ietf-manet-olsr-molsr-01.txt), November 2001, Work in Progress.

[24] M. Liu, R. Talpade, A. McAuley, and E. Bommaiah, "AMRoute: Adhoc Multicast Routing Protocol," *Center for Satellite and Hybrid Communication Networks Technical Report: CSHCN T.R. 99-1.*

[25] C.W. Wu, C-K, Toh, "Ad hoc Multicast Routing protocol utilizing Increasing id-numbers (AMRIS) Functional Specification," IETF Internet Draft, (draft-ietf-manet-amris-spec-00.txt), November 1998, Work in Progress.

[26] J.J. Garcia-Luna-Aceves and E.L. Madrgua, "The Core-Assisted Mesh Protocol," *IEEE Journal on Selected Areas in Communications,* vol. 17, no 8, August 1999, pp. 1380-1394.

[27] J. Jetcheva and D. Johnson. "Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks," P*roceedings of the ACM Symposium on Mobile Ad Hoc Networking and Computing, (MobiHoc 2001),* pp. 33-44, ACM, Long Beach, CA, October, 2001.

[28] L. Ji and M. S. Corson. "A Lightweight Adaptive Multicast Algorithm," *Proceedings of IEEE GLOBECOM '98*, pp. 1036-1042, December 1998.

[29] L. Ji, and M.S. Corson. "Differential Destination Multicast (DDM) Specification," Internet Draft (draft-ietf-manet-ddm-00.txt) July 2000. Work in Progress.

[30] P. Sinha, R. Sivakumar, and V. Bharghavan. "MCEDAR: Multicast Core Extraction Distributed Ad-Hoc Routing," In *Proceedings of the Wireless Communications and Networking Conference, WCNC '99*, pp. 1313-1317, September 1999.

[31] Z.J. Haas and M.R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks," IETF Internet Draft, (draft-ietf-manet-zone-zrp-02.txt,) June 1999, Work in Progress.

[32] C. Perkins, E. Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF Internet Draft (draft-ietf-manet-aodv-10.txt), January 2002, Work in Progress.

[33] S. Deering, "Host Extensions for IP Multicasting," RFC 1112, August 1989.

[34] H. Holbrook and B. Cain. "Source- Specific Multicast for IP," Intenet-Draft, (draft-holbrook-ssm-arch-01.txt), November 2000, Work in Progress.

[35] A. Laouiti, A. Qayyum, and L. Viennot, "Multipoint relaying: An efficient technique for flooding in mobile wireless networks," Technical Report RR-3898, INRIA, 2000.

[36] Redhat Linux, "Redhat Linux Distribution Site," http://www.redhat.com.

[37] T. Lin, S.F. Midkiff, and J.S. Park, "A Dynamic Topology Switch for the Emulations of Wireless Mobile Ad Hoc Networks," accepted at the Workshop for Wireless Local Networks, November 2002.

[38] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," IETF RFC 2119, March 1997.

[39] M. Handley, C. Perkins, and E. Whelan, “Session Announcement Protocol,” IETF RFC 2974, October 2000.

[40] S. Hanna, B. Patel, and M. Shah, “Multicast Address Dynamic Client Allocation Protocol (MADCAP),” IETF RFC 2730, December 1999.

[41] A. Plakoo and A. Laouiti, “OLSR Routing Code,” INRIA Rocquencourt, 2000.

[42] F. Kempen, and B. Eckenfels, “Route code,” 1998.

[43] IANA, “Port Numbers,” http://www.iana.org/assignments/port-numbers, May 2002.

[44] J. M. de Goyeneche, “Multicast over TCP/IP HOWTO v1.0,” http://www.tldp.org/HOWTO/Multicast-HOWTO.html, March 20, 1998.

[45] S. Deering, “Mrouted 3.8 Code,” Stanford University, November 1995.

[46] C. Bettstetter, “Mobility Modeling in Wireless Networks: Categorization, Smooth Movement, and Border Effects,” *ACM Mobile Computing and Communications Review*, vol. 5, no. 3, pp. 55-67, July 2001.

[47] J. Wells, “Tcpdump 3.7.1 Time Patch,” Virginia Tech, 2002.

[48] Navy Collaborative Integrated Information Technology Initiative (NAVCIITI), “NAVCIITI website,” http://www.irean.vt.edu/navciiti/ .

[49] Office of Naval Research (ONR), “Office of Naval Research website,” http://www.onr.navy.mil/ .

Appendix A: JOIN Table Data Structure

```
/*
 * BIG join duplicate table
 * "origin" (who is looking for the new path)
 * "source" (who is sending the data)
 * "group" (where are they sending the data)
 */

struct joinOriginDuplInfo
{
  olsr_u32_t              joinorigin_hash;
  olsr_ip_addr            joinorigin_addr;
  olsr_u16_t              join_seq;
  olsr_ip_addr            join_lasthop;  //send the JOIN-ACK here
  struct timeval          join_timer;
  olsr_u8_t               joinorigin_ttl;
  olsr_u8_t               join_hop_count;
 };

struct joinOriginDupl_entry{
  struct joinOriginDupl_entry *duplicate_forw;
  struct joinOriginDupl_entry *duplicate_back;
  struct joinOriginDuplInfo   join_origin_infos;
};

#define joinorigin_hash      join_origin_infos.joinorigin_hash
#define joinorigin_addr      join_origin_infos.joinorigin_addr
#define join_seq             join_origin_infos.join_seq
#define join_lasthop         join_origin_infos.join_lasthop
#define join_timer           join_origin_infos.join_timer
#define joinorigin_ttl       join_origin_infos.joinorigin_ttl
#define join_hop_count       join_origin_infos.join_hop_count

struct joinOriginHash {
  struct   joinOriginDupl_entry *duplicate_forw;
  struct   joinOriginDupl_entry *duplicate_back;
};

struct joinSrcDuplInfo
{
  olsr_u32_t              joinsrc_hash;
  olsr_ip_addr            joinsrc_addr;
  struct joinOriginHash      *joinoriginhash;
};
```

```
struct joinSrcDupl_entry{
  struct joinSrcDupl_entry *duplicate_forw;
  struct joinSrcDupl_entry *duplicate_back;
  struct joinSrcDuplInfo   joinsrc_infos;
};

#define joinsrc_hash         joinsrc_infos.joinsrc_hash
#define joinsrc_addr         joinsrc_infos.joinsrc_addr
#define joinoriginhash       joinsrc_infos.joinoriginhash

#defineHASHSIZE    32                /* must be a power of 2 */
#defineHASHMASK  (HASHSIZE - 1)

struct joinSrcHash {
  struct   joinSrcDupl_entry *duplicate_forw;
  struct   joinSrcDupl_entry *duplicate_back;
};

struct joinGrpDuplInfo {
  olsr_u32_t                joingrp_hash;
  olsr_ip_addr              joingrp_addr;
  struct joinSrcHash          *source_info;
};

struct joinGrpDupl_entry{
  struct joinGrpDupl_entry *duplicate_forw;
  struct joinDrpDupl_entry *duplicate_back;
  struct joinGrpDuplInfo   join_infos;
};

#define joingrp_hash           join_infos.joingrp_hash
#define joingrp_addr           join_infos.joingrp_addr
#define joinsrchash            join_infos.source_info

struct joinGrpHash {
  struct   joinGrpDupl_entry *duplicate_forw;
  struct   joinGrpDupl_entry *duplicate_back;
};
```

Appendix B: Results of High Mobility Network

Table B.1. Average Control Packets as Function of Time for High Mobility

	Secs	Basic -						MPR -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	69.2	97.5	124.2	136.7	147.1		44.7	67.0	91.3	99.6	105.1
	30	29.5	55.2	75.7	95.1	104.5		24.0	42.1	59.1	62.6	70.1
	45	27.3	48.7	68.4	92.0	97.2		23.1	39.6	58.5	61.1	63.7
	60	27.2	50.4	69.1	85.7	97.1		22.5	38.9	57.5	59.4	68.8
	75	26.7	49.9	67.5	84.7	95.2		22.2	38.6	56.9	59.2	65.0
	90	26.2	49.0	66.3	82.7	92.7		22.0	38.0	56.7	57.8	64.4
	105	21.6	34.1	42.6	51.4	55.1		20.4	32.2	45.9	44.0	47.3
	120	21.1	32.1	38.7	44.3	44.7		19.8	31.3	42.8	40.4	42.2
	135	21.0	37.3	47.3	51.6	50.0		19.8	33.8	45.2	43.2	45.2
	150	24.6	38.7	44.9	48.4	47.9		21.3	32.4	47.1	44.8	46.0
	165	20.9	34.8	42.1	45.0	46.2		19.8	31.7	45.3	42.5	45.1
	180	20.9	35.1	42.5	45.5	46.1		19.8	31.3	44.4	41.7	44.0
	195	21.0	35.1	42.7	45.7	46.2		19.8	31.2	44.9	42.3	45.0
	210	21.0	34.8	42.0	44.6	45.5		19.8	31.0	44.3	41.5	44.1
	225	21.0	34.4	41.2	43.8	48.7		19.8	31.2	43.0	41.5	46.1
	240	1.6	7.3	8.8	10.7	9.5		3.1	4.3	7.3	7.7	12.6
	255	0.0	0.0	0.2	0.2	0.2		0.0	0.1	0.3	0.1	0.7

Table B.2. Average Control Packets as Function of Time for High Mobility

	Secs	Refresh0N -						MPR-Refresh0 -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	68.8	95.6	121.5	142.8	154.1		40.0	69.0	86.8	100.7	102.7
	30	28.8	54.8	74.8	92.8	103.0		23.5	42.7	55.8	64.3	67.9
	45	26.0	50.7	67.7	91.9	98.1		22.3	41.6	50.3	63.1	64.8
	60	26.1	50.1	66.0	82.8	93.8		22.9	41.4	51.2	58.5	59.6
	75	24.4	48.5	65.1	81.1	91.5		21.1	39.6	51.3	59.8	59.7
	90	24.0	47.0	63.7	79.3	88.6		21.4	38.4	49.5	57.3	59.5
	105	18.9	32.6	39.8	48.0	50.4		18.7	32.1	41.5	43.3	44.5
	120	18.5	30.4	36.4	40.5	41.0		18.2	30.3	37.4	37.4	39.0
	135	18.0	36.9	42.7	45.9	47.6		18.3	31.3	38.3	39.6	44.1
	150	21.7	36.3	41.0	47.0	43.6		20.0	33.8	39.0	38.6	41.9
	165	20.0	34.0	41.8	44.9	43.0		19.2	34.6	40.3	40.7	43.3
	180	19.7	32.8	37.6	43.7	43.1		19.0	33.1	39.0	39.3	42.1
	195	19.1	32.4	37.4	43.4	42.7		17.8	32.5	38.4	41.2	42.1
	210	17.1	32.1	36.5	43.0	42.4		20.1	32.5	38.3	38.7	41.5
	225	17.8	32.1	37.1	41.9	44.7		18.1	31.4	37.5	40.4	43.3
	240	1.4	5.0	5.4	6.8	8.2		1.0	3.7	5.1	8.2	8.5
	255	0.0	0.0	0.1	0.1	0.2		0.0	0.0	0.0	0.1	0.3

Table B.3. Average Control Packets as Function of Time for High Mobility

	Secs	**Refresh0 -**				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	68.6	100.6	121.3	121	149.4
	30	28.6	58.3	73.7	83.3	103.4
	45	26.4	53.1	67.4	61.9	100.3
	60	26.4	51.9	67.4	74.9	93.2
	75	24.5	51.6	66	56.6	91.8
	90	24.4	49.9	63.9	74.9	89.3
	105	21	34.9	40.1	49.9	52.9
	120	19.1	33.4	36.1	66.9	42.4
	135	19	39.9	43.7	54.6	47.7
	150	22.3	41.5	42.3	69.5	45.9
	165	21.1	39.3	40.5	52.1	44.9
	180	19.4	35.2	39.2	68.8	44.8
	195	19.5	35.1	39.1	47.2	44.9
	210	19.5	34.8	38.6	47.5	44.4
	225	19.4	35.1	37.3	42.7	46
	240	3.3	4.1	7	9.3	11.7
	255		0.0	0.0	0.1	0.5

Table B.4. Average TREE-REFRESH Packets as Function of Time for High Mobility

	Secs	**Basic -**						**MPR -**				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	19.4	30.2	34.3	37.4	36.9		17.4	27.2	35.1	32.9	31.8
	30	21.9	32.5	39.4	45.9	43.4		20.7	31.5	42.2	39.7	40.3
	45	21.1	31.8	40.1	44.9	43.1		20.2	31	41.9	39.1	41.1
	60	21.4	32.5	41.1	46.5	46.3		20.1	32	45.1	42.5	43.4
	75	21.5	33.4	41	46.1	46.5		20.4	31.7	44.8	42.3	44.3
	90	21.5	32.9	40.4	45.9	46		20.3	31.4	44.6	41.8	44
	105	21.5	32.6	39.2	44.9	45		20.3	31.5	43.8	41.4	43.2
	120	21.1	31.8	38.5	44	44.1		19.8	31.3	42.6	40.1	41.8
	135	21	31.2	38.5	43.6	44.1		19.8	30.6	42.4	39.8	41.9
	150	21.2	33.7	42.2	45.3	46.5		19.9	31.5	44.8	42.3	44.6
	165	20.9	34.8	42.1	45	46.2		19.8	31.7	45	42.2	44.6
	180	20.9	35.1	42.2	45	45.7		19.8	31.3	44.4	41.7	44
	195	21	35.1	42.2	44.9	45.7		19.8	31.2	44.4	41.8	44.2
	210	21	34.8	41.9	44.4	45.4		19.8	31	44.2	41.4	43.9
	225	21	34.4	41.2	43.8	45.6		19.8	31.2	43	41	43
	240	1.6	7.3	8.8	10.7	9.5		3.1	4.3	7.3	7.7	12.6
	255			0.2	0.2	0.2			0.1	0.3	0.1	0.7

Table B.5. Average TREE-REFRESH Packets as Function of Time for High Mobility

	Secs	Refresh0N -						MPR-Refresh0 -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	18.5	28.3	33.3	37.8	36.6		16.1	29.7	32	33.2	32.8
	30	20	31.9	38.5	43.2	43.5		19.9	32	38.5	39.5	40.7
	45	19.4	32.3	38.2	43.5	42.8		20	32.1	38.8	40.7	41.1
	60	19.5	32.8	38	43.1	42.5		19.8	32.5	38.5	40	40.9
	75	18.8	31.2	37.2	42.2	42.7		19.2	31.2	38.2	39.5	40.8
	90	18.9	31.2	36.8	42.2	41.6		19.4	31.1	38.3	39.9	41.3
	105	18.8	31.1	35.9	41.3	40		18.6	31.1	38.5	39.6	40.8
	120	18.5	30.3	36.1	40.1	40.5		18.2	30.2	36.6	36.9	38.7
	135	18	30.8	36.1	40	41.1		18.3	29.5	36.2	36.4	40.7
	150	18.3	31.8	36	42	42.3		18.6	31.8	38.1	38.2	41.5
	165	18.3	32.3	37.7	42.9	42.4		18.7	32.6	38.2	38.4	41.5
	180	17.8	32.4	36.9	42.9	42.3		18.7	32.6	38.8	38.6	41.7
	195	17.1	32.4	36.8	42.8	42.1		17.4	32.5	38.4	38.5	41.5
	210	17.1	32.1	36.3	41.8	41.5		18.4	31.7	37.3	37.6	40.4
	225	17.1	30.4	36.6	41	40.5		17.6	30.3	36.5	36.9	39.2
	240	1.4	4.8	5.4	6.8	8.2		1	3.7	5.1	8.2	8.4
	255	0		0.1	0.1	0.2		0			0.1	0.3

Table B.6. Average TREE-REFRESH Packets as Function of Time for High Mobility

	Secs	Refresh0 -				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	18.7	30.8	32	37.1	36.6
	30	20.7	35.3	38.3	43.2	43.3
	45	19.7	34.7	38.9	41.3	44
	60	19.6	34.8	38.9	43	43.8
	75	19	33.9	37.7	43.6	42.7
	90	19.3	33.7	37.3	41.8	42.5
	105	19.2	33.7	36.4	41	42
	120	19.1	33.1	35.6	40.2	41.6
	135	19	33.7	36.8	40.9	41.6
	150	18.9	35.5	37.3	42.4	44.3
	165	19.1	36.4	38.3	42.5	44.2
	180	19.4	35.1	38.4	42.9	43.8
	195	19.5	35.1	38.4	42.7	43.9
	210	19.5	34.8	37.5	43	42.7
	225	19.4	33.4	36.4	41.1	41.9
	240	3.3	3.9	7	9.3	11.3
	255	0		0	0.1	0.4

Table B.7. Average Data, Control, and Refresh Packets and Standard Deviations

	Basic					
Rcvr	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4553.2	435.1	348.1	3365.7	252.3	267.1
3	8011.1	738.1	561.6	3839.4	120.2	205.1
5	9561.9	942.5	685.4	4078.7	217.2	267.2
7	10423.2	1081.5	745.9	5736.4	415.2	372.1
9	10243.9	1142.3	743.9	5463.1	548.9	353.2

Table B.8. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0N					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4046.5	399.7	302.8	2759.3	212.3	221.5
3	6955.4	705.2	525.1	2575.0	112.1	202.7
5	8069.2	883.2	617.4	3251.0	238.1	275.7
7	10506.6	1039.5	690.7	5303.7	380.9	318.0
9	11200.2	1104.5	694.8	6618.5	536.3	314.1

Table B.9. Average Data, Control, and Refresh Packets and Standard Deviations

	MPR					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4329.9	371.3	328.1	2969.7	239.3	240.3
3	6698.7	598.8	511.3	2379.6	122.7	196.4
5	10100.3	868.9	729.5	5431.0	266.9	354.4
7	9073.6	853.1	675.9	4827.5	230.2	323.4
9	9336.6	919.1	708.8	4718.8	262.7	341.4

Table B.10. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4222.5	402.1	310.8	2905.1	231.4	240.2
3	7688.5	747.7	562.3	3142.1	172.1	253.4
5	8991.1	887.3	624.8	4620.7	238.9	279.2
7	11435.1	1054.6	703.8	7333.3	373.6	325.7
9	11615.5	1112.3	705.2	7154.8	535.1	323.1

Table B.11. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4222.5	402.1	310.8	2905.1	231.4	240.2
3	7688.5	747.7	562.3	3142.1	172.1	253.4
5	8991.1	887.3	624.8	4620.7	238.9	279.2
7	11435.1	1054.6	703.8	7333.3	373.6	325.7
9	11615.5	1112.3	705.2	7154.8	535.1	323.1

Table B.12. Average Data, Control, and Refresh Packets and Standard Deviations

	Notify					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4541.8	431.7	343.0	3360.5	257.2	270.6
3	9035.8	814.1	630.2	4213.9	190.0	275.7
5	7782.9	867.4	613.9	2944.2	222.1	245.0
7	10165.2	1079.4	735.7	6151.9	447.1	401.2
9	11130.1	1176.7	772.8	6365.7	548.7	371.6

Table B.13. Average Data, Control, and Refresh Packets and Standard Deviations

	Increase					
	Data Pkts	Cntl Pkts	Rfrsh Pkts	Data Stdv	Cntrl Stdv	Rfrsh Stdv
1	4337.2	426.0	335.0	2978.7	217.3	232.2
3	7658.9	763.4	570.2	3931.9	183.5	256.6
5	9822.0	946.0	672.0	4894.9	231.0	278.5
7	9134.3	1067.3	679.4	4339.8	406.0	300.4
9	11504.7	1222.9	779.3	6312.5	548.5	366.9

Appendix C: Results of Medium Mobility Network

Table C.1. Average Control Packets as Function of Time for Medium Mobility

	Secs	Basic -						MPR -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	87.3	100.5	115.5	124.8	132.2		60.6	73.1	90.4	100.7	105.9
	30	37.9	55.4	63.6	75.2	78.4		32.8	39.7	57.5	65.2	62.0
	45	33.4	43.2	51.5	58.2	61.0		31.6	38.0	50.0	55.1	54.6
	60	33.1	42.8	49.9	58.6	61.1		31.5	38.0	48.9	53.7	53.5
	75	33.0	42.4	50.2	57.3	62.2		31.2	37.9	47.8	53.1	53.6
	90	32.9	42.3	51.2	56.8	60.0		31.2	37.3	47.8	52.7	55.1
	105	33.0	43.5	50.4	58.5	60.5		31.3	37.6	49.0	54.9	52.7
	120	31.8	41.6	48.8	55.3	59.8		30.4	35.4	46.0	51.4	51.8
	135	30.0	39.4	47.9	54.2	59.1		28.2	34.5	43.6	48.7	54.4
	150	30.0	40.7	49.2	55.6	59.0		28.2	34.6	44.6	49.7	57.5
	165	30.0	39.7	46.9	53.3	57.1		28.2	32.6	43.1	48.1	50.2
	180	29.9	39.8	45.1	52.7	54.1		28.1	33.2	42.1	48.2	48.3
	195	29.9	39.8	44.8	52.4	53.8		28.1	32.9	41.9	48.2	47.9
	210	29.7	39.1	43.9	52.5	53.3		32.0	33.9	43.7	49.3	49.6
	225	35.4	40.8	44.0	52.4	53.0		31.0	34.7	42.2	46.6	48.6
	240	4.3	6.4	7.8	8.7	10.4		3.6	4.3	4.8	7.8	13.1
	255	0.0	0.1	0.2	0.1	0.4		0.0	0.0	0.0	0.3	0.6

Table C.2. Average Control Packets as Function of Time for Medium Mobility

	Secs	Refresh0N -						MPR-Refresh0 -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	82.5	95.1	116.3	125.7	132.8		59.0	70.6	89.6	101.4	107.8
	30	33.4	44.8	55.8	75.9	80.5		40.8	54.3	54.7	62.9	69.6
	45	30.4	40.5	48.8	58.5	60.1		34.2	40.2	52.4	55.7	57.9
	60	30.1	38.8	46.3	65.1	56.4		32.7	37.4	48.3	51.5	56.0
	75	30.0	38.1	50.7	67.1	66.3		34.2	37.2	47.3	50.7	57.6
	90	30.0	38.7	52.6	58.5	62.6		33.3	42.1	49.5	53.5	58.9
	105	30.1	41.8	45.3	56.4	55.4		31.7	38.6	48.8	52.6	52.9
	120	29.6	37.2	42.9	59.9	52.3		31.5	38.0	46.8	52.0	54.9
	135	27.7	34.4	43.8	62.4	61.0		32.2	35.3	45.0	57.9	59.5
	150	27.6	38.1	45.3	56.6	53.4		30.6	32.0	46.2	51.7	52.2
	165	27.6	33.7	40.8	51.2	48.8		28.8	33.0	44.0	45.2	47.9
	180	27.5	32.9	37.8	48.5	47.5		28.7	32.4	42.5	44.1	46.8
	195	27.5	33.0	37.3	65.3	52.9		33.2	33.5	42.6	42.9	52.4
	210	27.3	32.6	43.2	48.6	48.1		38.8	35.5	42.3	41.9	45.6
	225	32.3	34.0	38.3	47.0	47.2		28.5	33.5	41.3	43.1	47.0
	240	2.3	5.1	4.7	8.1	10.3		2.9	2.9	3.9	9.8	7.9
	255	0.0	0.1	0.1	0.3	0.7		0.0	0.0	0.0	0.5	0.1

Table C.3. Average Control Packets as Function of Time for Medium Mobility

	Secs	Refresh0 -				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	84.0	99.7	119.7	126.0	136.8
	30	53.6	46.0	66.6	74.4	70.0
	45	36.3	40.3	53.0	57.1	56.4
	60	35.1	41.8	50.5	58.2	56.2
	75	35.0	45.0	59.4	58.8	61.0
	90	38.9	39.8	55.4	64.6	63.9
	105	33.7	39.7	46.7	53.9	54.5
	120	32.4	38.9	47.9	51.0	52.0
	135	36.6	37.4	55.3	53.0	58.1
	150	32.8	36.2	49.4	55.9	55.1
	165	31.2	35.7	42.4	47.1	47.3
	180	31.6	35.5	40.0	46.7	46.8
	195	36.8	34.9	60.2	52.0	49.2
	210	30.5	34.7	42.6	46.6	43.0
	225	36.5	36.5	39.8	46.0	42.8
	240	3.3	4.5	4.8	8.4	8.6
	255	0.0	0.1	0.0	0.4	0.2

Table C.4. Average Control Packets as Function of Time for Medium Mobility

	Secs	Increase -						Notify -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	92.8	100.5	132.9	146.8	158.8		83.6	98.0	111.2	126.4	129.3
	30	35.5	59.2	68.7	67.3	78.8		40.4	57.0	62.4	68.1	71.3
	45	34.2	41.9	54.8	58.6	63.5		33.3	45.2	51.5	54.2	58.4
	60	33.7	41.3	51.1	56.2	61.5		32.5	44.1	49.9	55.6	57.6
	75	33.8	44.5	48.8	56.2	59.5		32.4	46.1	49.7	53.6	57.0
	90	33.6	41.1	50.0	55.9	59.6		32.6	44.6	51.9	53.1	56.3
	105	33.7	43.3	53.8	55.4	59.8		32.8	45.2	51.1	52.6	57.3
	120	34.0	39.2	51.3	54.6	58.9		32.2	51.7	54.5	53.9	62.4
	135	31.5	38.7	47.1	52.4	56.5		31.3	41.7	50.3	50.3	55.1
	150	31.4	39.7	46.7	52.5	57.5		31.5	41.5	50.4	51.8	54.0
	165	31.2	37.9	51.4	54.1	55.7		31.5	40.6	48.3	51.0	52.5
	180	31.2	37.8	42.9	49.0	53.8		31.2	39.8	45.8	48.9	47.5
	195	31.2	37.5	42.6	47.5	50.2		31.1	39.2	44.6	49.6	47.5
	210	30.6	37.0	42.3	51.2	56.2		31.2	38.7	43.5	47.2	48.1
	225	34.5	38.5	44.5	47.4	52.7		36.5	40.5	43.8	48.8	47.8
	240	3.5	4.4	16.2	23.7	27.9		2.8	7.2	7.9	9.7	13.0
	255	0.0	0.0	0.0	0.1	0.2		0.0	0.2	0.1	0.1	0.6

Table C.5. Average TREE-REFRESH Packets as Function of Time for Medium Mobility

	Secs	Basic -						MPR -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	27.2	37.6	43.7	46.8	49.9		27.2	31.9	39.0	44.0	44.0
	30	33.1	42.1	46.9	55.6	58.1		31.4	36.2	47.5	52.5	49.8
	45	33.0	42.2	49.0	55.8	58.3		31.2	37.0	48.0	52.8	51.0
	60	32.9	42.1	47.9	55.6	58.4		31.1	37.0	47.3	52.4	51.1
	75	32.8	42.1	47.8	55.3	58.3		31.0	36.7	46.8	51.8	50.8
	90	32.7	41.9	47.7	55.2	58.2		31.0	36.8	46.8	51.5	50.2
	105	32.8	42.2	48.2	55.6	58.9		31.1	37.1	46.8	51.8	50.9
	120	31.8	41.4	48.6	55.3	59.5		30.4	35.4	46.0	51.0	51.8
	135	30.0	39.4	47.9	54.0	58.9		28.2	33.0	43.4	48.7	50.8
	150	30.0	39.4	47.9	54.3	59.0		28.2	32.5	43.7	48.2	51.7
	165	30.0	39.7	46.8	53.3	57.1		28.2	32.6	43.0	48.1	50.2
	180	29.9	39.8	45.1	52.7	54.1		28.1	33.2	42.0	48.2	48.3
	195	29.9	39.8	44.8	52.4	53.8		28.1	32.9	41.9	48.2	47.9
	210	29.7	39.1	43.9	52.5	53.3		28.1	32.9	42.4	48.0	48.3
	225	29.5	38.7	43.4	51.9	52.5		28.2	32.6	41.7	46.1	48.1
	240	4.3	6.3	7.8	8.7	10.4		3.6	4.2	4.8	7.7	13.1
	255	0.0	0.1	0.2	0.1	0.4		0.0	0.0	0.0	0.3	0.6

Table C.6. Average TREE-REFRESH Packets as Function of Time for Medium Mobility

	Secs	Refresh0N -						MPR-Refresh0N -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	27.3	32.3	41.8	46.9	46.3		27.3	30.9	40.9	43.3	42.9
	30	30.0	37.8	43.9	54.0	53.4		33.2	36.9	48.7	50.0	53.3
	45	30.0	38.1	44.1	52.7	52.8		32.3	37.2	49.1	51.0	53.3
	60	29.9	37.9	43.4	53.7	51.6		31.6	36.1	46.8	49.4	52.3
	75	29.8	37.7	42.7	52.9	51.6		31.5	35.8	46.3	48.5	51.6
	90	29.8	37.4	42.8	53.8	52.7		32.3	35.7	46.3	48.4	52.6
	105	29.9	37.3	42.7	53.3	52.2		31.5	36.9	46.7	50.0	51.2
	120	29.6	36.4	42.1	53.1	52.1		31.5	34.3	46.6	49.1	52.6
	135	27.7	34.3	40.7	53.5	51.0		28.9	32.4	45.0	48.4	50.3
	150	27.6	34.0	41.9	54.6	51.9		29.8	32.0	44.9	47.7	51.1
	165	27.6	33.6	39.7	50.5	48.4		28.8	32.3	43.6	44.4	47.6
	180	27.5	32.9	37.7	48.4	47.3		28.7	32.4	42.5	44.0	46.5
	195	27.5	33.0	37.3	50.4	48.1		29.4	31.8	42.6	42.8	46.8
	210	27.3	32.6	38.1	46.9	47.3		28.1	32.1	42.3	41.9	45.5
	225	26.4	31.9	37.7	46.5	46.2		26.3	31.4	40.8	42.2	45.8
	240	2.3	5.0	4.7	7.4	10.3		2.9	2.8	3.9	9.8	7.9
	255	0.0	0.1	0.1	0.3	0.7		0.0	0.0.	0.0	0.5	0.1

Table C.7. Average TREE-REFRESH Packets as Function of Time for Medium Mobility

	Secs	**Refresh0 -**				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	28.0	37.0	41.4	46.3	44.7
	30	34.6	39.4	47.2	53.3	52.2
	45	33.2	39.3	46.2	51.0	51.4
	60	32.7	39.2	45.6	50.9	50.1
	75	32.6	39.4	45.6	50.2	49.9
	90	33.0	39.3	45.6	51.0	51.5
	105	32.7	39.4	44.0	51.3	50.7
	120	32.4	38.9	45.3	50.5	50.8
	135	31.5	36.1	44.2	48.5	49.5
	150	32.0	35.8	46.0	49.5	50.0
	165	31.2	35.7	41.3	46.7	46.9
	180	31.1	35.5	40.0	46.4	44.1
	195	31.7	34.9	41.6	46.5	44.2
	210	30.5	34.7	41.0	46.0	43.0
	225	30.2	34.4	39.3	45.0	41.8
	240	3.3	4.4	4.8	8.4	8.6
	255	28.0	37.0	41.4	46.3	44.7

Table C.8. Average TREE-REFRESH Packets as Function of Time for Medium Mobility

	Secs	**Increase -**						**Notify-**				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	30.9	35.3	33.0	34.3	33.0		27.0	35.2	41.2	44.2	44.6
	30	33.8	40.5	46.5	51.9	57.1		31.2	42.0	46.7	51.9	55.0
	45	33.6	40.5	47.1	52.7	58.6		32.7	43.9	48.1	52.1	55.3
	60	33.5	40.4	47.6	53.5	58.6		32.3	43.7	47.8	52.8	54.3
	75	33.4	40.5	46.3	51.9	56.7		32.2	44.0	46.8	51.4	53.6
	90	33.4	40.6	45.8	51.5	57.0		32.4	43.8	47.3	51.0	53.4
	105	33.5	40.5	47.0	51.9	56.4		32.6	44.0	48.4	51.0	53.6
	120	32.5	39.2	46.8	51.8	56.5		32.2	44.4	48.6	50.8	54.6
	135	31.2	38.5	46.5	51.9	56.5		31.3	41.7	49.8	49.6	53.5
	150	31.2	38.6	46.4	51.3	56.1		31.5	40.8	49.6	49.5	53.1
	165	31.2	37.9	45.2	50.4	53.1		31.5	40.6	48.3	50.3	52.0
	180	31.2	37.8	42.9	48.9	52.6		31.2	39.8	45.8	48.6	47.5
	195	31.2	37.5	42.6	47.5	50.2		31.1	39.2	44.1	48.3	47.3
	210	30.5	37.0	42.3	47.5	50.6		31.2	38.7	43.4	47.2	46.9
	225	30.4	35.6	42.3	47.2	51.3		30.1	38.3	42.8	46.8	47.2
	240	3.5	4.0	14.9	21.0	25.4		2.8	7.1	7.8	9.4	13.0
	255	0.0	0.0	0.0	0.1	0.2			0.2	0.1	0.1	0.6

Table C.9. Average Data, Control, and Refresh Packets and Standard Deviations

	Basic					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	6014.2	541.6	469.6	2530.5	224.5	199.7
3	7648.9	697.5	613.9	1803.9	163.7	143.5
5	8819.6	810.9	707.6	2192.9	181.3	182.4
7	10492.0	926.6	815.1	3726.4	247.7	240.9
9	11378.1	975.4	859.1	4131.5	248.7	248.5

Table C.10. Average Data, Control, and Refresh Packets and Standard Deviations

	MPR					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5710.2	488.0	445.1	2743.1	230.7	211.8
3	6741.4	577.7	522.0	1833.3	167.6	145.0
5	8608.9	743.4	671.1	2130.7	166.9	161.7
7	9427.2	833.7	751.3	2335.2	197.6	191.1
9	9440.7	859.4	758.6	2658.1	241.0	222.8

Table C.11. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	6057.5	588.3	480.7	2951.5	287.7	244.0
3	7093.8	646.7	563.5	1899.1	161.0	143.4
5	8479.6	833.7	659.1	2678.5	212.3	224.7
7	9273.5	900.1	741.9	2359.6	181.2	180.9
9	9288.8	901.9	729.6	2515.6	220.6	191.6

Table C.12. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5430.0	495.9	430.2	2308.8	192.7	173.1
3	6804.5	618.9	532.3	2669.5	208.7	206.3
5	7942.6	750.0	621.4	2428.8	180.7	192.0
7	9874.7	955.1	778.9	2864.6	249.4	208.4
9	9532.5	936.3	763.9	2437.3	193.5	193.3

Table C.13. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh1					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5716.2	747.1	638.8	2341.2	324.8	248.1
3	7064.3	1068.2	966.7	2019.8	270.8	253.8
5	8339.7	1516.3	1396.8	2452.4	309.7	289.7
7	10162.9	1972.6	1798.1	3605.3	430.4	338.4
9	9967.6	2329.1	2165.1	2513.4	461.4	396.1

Table C.14. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh1N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5687.2	743.3	635.5	2343.9	347.8	265.5
3	6956.8	1117.9	963.6	1563.3	344.3	231.2
5	8646.6	1637.2	1464.0	2580.7	378.5	314.2
7	9596.2	1958.9	1809.8	2949.9	381.3	333.8
9	9503.0	2343.3	2173.4	2551.8	478.5	407.1

Table C.15. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh2					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5669.8	646.4	547.9	2341.1	263.5	209.9
3	7631.8	947.9	801.1	2329.8	242.7	188.8
5	8457.4	1224.2	1051.9	2377.6	258.9	220.0
7	10171.6	1430.5	1271.3	3210.6	266.3	200.1
9	9411.4	1608.1	1451.2	2439.7	269.3	248.2

Table C.16. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh2N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5682.0	646.1	547.9	2341.1	263.5	209.9
3	7413.2	939.1	801.1	2191.4	254.5	212.0
5	8557.2	1225.0	1051.9	2537.2	251.3	221.3
7	9661.0	1481.0	1271.3	2717.5	327.8	222.0
9	9746.9	1598.7	1451.2	2524.2	264.3	236.9

Table C.17. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh3					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5939.5	635.2	530.6	2732.2	282.2	229.8
3	6871.5	833.9	682.0	1546.5	219.9	133.8
5	8353.1	1094.2	919.9	2225.3	225.7	186.3
7	9597.8	1262.2	1108.8	2700.4	249.3	197.1
9	9159.7	1373.0	1223.1	2515.7	231.1	208.7

Table C.18. Average Data, Control, and Refresh Packets and Standard Deviations

	MPR_Refresh0N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5847.5	521.1	454.1	2971.7	286.1	236.7
3	6660.9	596.5	511.0	1649.9	142.7	120.8
5	8527.8	745.2	677.0	2898.0	226.9	222.6
7	9102.1	817.4	711.4	2751.8	235.5	218.3
9	9424.1	875.0	751.4	2655.5	210.7	209.0

Table C.19. Average Data, Control, and Refresh Packets and Standard Deviations

	Notify					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	5937.4	546.9	473.3	2452.5	227.6	201.0
3	7962.9	721.3	627.4	2432.6	211.1	193.3
5	9381.1	816.9	706.6	3294.0	211.1	199.5
7	9701.5	874.9	755.0	2748.8	203.4	188.9
9	9927.3	915.7	785.5	2756.7	226.4	209.1

Table C.20. Average Data, Control, and Refresh Packets and Standard Deviations

	Increase					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	6159.5	556.4	485.0	2255.3	188.7	173.6
3	7380.9	682.5	584.4	2345.4	222.6	192.3
5	8719.8	845.1	683.2	2160.3	193.1	182.5
7	9518.0	928.9	765.3	2071.3	187.8	164.5
9	10465.3	1011.1	829.9	2195.3	198.7	185.4

Appendix D: Results of Low Mobility Network

Table D.1. Average Control Packets as Function of Time for Low Mobility

	Secs	Basic -						MPR -				
Receivers		1	3	5	7	9		1	3	5	7	9
	15	28.2	37.6	47.9	62.4	64.0		19.0	30.2	31.7	37.7	41.7
	30	17.8	28.2	34.0	36.8	41.1		14.0	22.7	25.6	28.8	37.1
	45	17.1	25.2	28.1	40.1	48.3		13.7	21.8	23.1	30.0	35.8
	60	17.2	28.3	33.1	37.3	42.0		14.0	22.4	24.7	27.9	37.9
	75	17.1	24.0	28.6	38.6	44.1		13.6	21.7	24.2	29.8	33.2
	90	16.7	27.2	34.0	42.1	46.4		12.2	20.3	22.6	31.0	34.8
	105	8.7	23.3	22.2	51.8	55.4		9.8	15.7	23.7	41.6	25.9
	120	8.2	29.6	33.3	25.7	31.7		9.2	17.0	22.8	19.9	35.7
	135	7.9	15.4	30.4	32.1	50.3		9.3	13.4	19.6	26.0	41.8
	150	8.4	17.0	23.2	23.6	30.0		9.3	12.7	20.2	18.9	26.7
	165	8.3	13.7	20.7	25.1	21.8		9.3	10.9	15.2	17.4	27.6
	180	7.9	15.1	19.2	23.4	20.9		9.1	9.8	13.5	12.7	12.4
	195	7.5	13.9	10.6	16.8	16.6		8.7	9.6	10.4	8.0	10.3
	210	7.3	10.5	10.5	15.5	15.0		8.5	8.8	9.7	6.3	8.1
	225	7.3	9.7	9.8	12.4	15.1		8.4	8.8	9.6	6.3	7.0
	240	0.8	1.8	2.3	3.7	4.6		2.1	1.5	2.3	1.2	2.5
	255	0.0	0.0	0.0	0.0	0.7		0.0	0.0	0.1	0.0	0.1

Table D.2. Average Control Packets as Function of Time for Low Mobility

	Secs	MPR-Refresh0N -						Refresh0 -				
Receivers		1	3	5	7	9		1	3	5	7	9
	15	18.3	28.4	35.4	37.0	41.3		28.1	37.5	44.4	55.6	61.4
	30	13.6	23.4	23.2	31.1	34.0		18.2	27.9	32.3	38.0	55.0
	45	11.8	21.9	24.3	29.4	32.9		16.2	23.4	26.3	39.1	55.7
	60	11.4	20.0	22.2	32.1	34.1		16.2	27.0	30.3	38.0	46.6
	75	10.5	20.9	24.4	31.5	34.3		16.0	23.2	31.5	38.9	43.6
	90	10.0	19.5	19.7	31.2	31.6		12.0	26.0	32.9	40.0	38.1
	105	7.8	17.1	24.4	29.1	36.7		7.9	22.5	23.8	51.6	56.5
	120	7.1	18.7	26.3	30.4	45.3		7.1	28.5	32.4	24.1	61.5
	135	7.3	14.4	24.6	23.3	28.1		7.4	14.8	38.3	28.4	27.9
	150	7.3	12.8	18.5	22.9	27.1		7.2	15.5	21.5	25.9	28.6
	165	7.2	11.3	16.5	18.3	22.5		7.2	13.3	20.2	21.5	17.2
	180	6.9	8.0	13.0	19.8	17.1		6.9	14.7	17.9	20.7	14.8
	195	6.6	6.9	10.5	11.0	11.2		6.6	12.4	7.8	11.4	12.0
	210	6.4	6.0	9.5	7.8	8.2		6.4	9.2	5.7	10.2	10.2
	225	6.3	5.8	9.4	7.2	7.6		6.4	8.5	5.7	9.1	10.0
	240	0.7	1.7	1.7	1.2	2.4		1.5	1.6	1.2	1.8	3.8
	255	0.0	0.0	0.0	0.0	0.1		0.0	0.0	0.0	0.1	0.4

Table D.3. Average Control Packets as Function of Time for Low Mobility

	Secs	Refresh0N -						Increase -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	27.5	36.3	44.7	58.7	64.9		28.0	40.6	52.7	51.8	68.6
	30	18.7	28.8	31.9	39.8	44.0		28.7	37.3	37.0	47.6	43.7
	45	16.3	25.3	27.3	41.7	41.0		17.3	23.3	32.3	43.6	45.1
	60	16.3	25.3	28.1	34.9	42.1		17.4	26.2	31.3	47.5	46.3
	75	16.0	22.8	28.7	41.0	44.4		24.6	21.9	31.5	45.1	46.2
	90	12.0	25.3	29.6	36.4	46.2		12.9	22.7	28.6	47.5	42.3
	105	7.8	22.7	22.4	51.4	32.8		9.1	26.8	30.7	51.0	62.1
	120	7.1	25.5	32.2	22.8	54.4		7.9	26.9	32.2	42.3	58.3
	135	7.3	16.2	38.8	36.0	43.2		8.1	17.8	20.3	22.5	44.7
	150	7.3	17.6	22.4	22.6	34.1		8.4	20.4	30.0	23.7	29.8
	165	7.3	15.8	18.4	24.9	24.6		8.1	25.9	22.4	22.4	38.1
	180	6.9	13.0	17.0	19.8	26.4		8.0	17.7	23.9	14.6	24.7
	195	6.6	9.9	10.5	10.7	13.2		7.8	13.2	13.3	15.2	18.4
	210	6.4	6.4	9.0	7.7	11.2		7.4	10.1	10.4	13.8	13.3
	225	6.4	5.7	8.9	6.5	11.0		7.3	9.8	12.1	9.7	12.8
	240	1.3	0.8	2.7	2.3	2.4		1.5	4.1	2.2	4.9	4.8
	255	0.0	0.0	0.2	0.0	0.2		0.0	0.0	0.0	0.1	0.0

Table D.4. Average Control Packets as Function of Time for Low Mobility

	Secs	Notify -				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	28.2	41.5	47.2	54.3	55.2
	30	17.8	32.1	29.2	40.2	46.6
	45	17.1	28.8	28.3	38.0	41.4
	60	17.4	29.4	29.3	38.4	46.2
	75	17.1	28.5	28.1	38.8	44.5
	90	13.1	28.7	27.0	43.0	46.0
	105	8.6	30.5	23.2	32.2	57.3
	120	8.0	14.0	35.0	25.9	64.1
	135	8.3	15.6	19.1	46.2	37.8
	150	8.1	14.5	27.9	31.6	34.2
	165	8.2	11.8	21.5	36.2	32.6
	180	7.8	9.8	21.8	36.4	26.5
	195	7.5	9.0	10.6	22.5	16.0
	210	7.4	9.0	8.9	12.1	12.0
	225	7.3	8.8	8.8	11.0	10.9
	240	0.5	1.6	1.9	4.5	4.3
	255	0.0	0.0	0.1	0.1	0.2

Table D.5. Average TREE-REFRESH Packets as Function of Time for Low Mobility

	Secs	Basic -						MPR -				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15	5.8	8.3	8.2	9.6	8.7		5.6	10.1	8.0	8.5	8.6
	30	8.1	13.0	13.0	14.1	12.6		9.0	12.2	12.8	12.1	12.7
	45	8.2	13.5	13.5	16.1	14.0		9.3	12.5	13.1	14.3	14.9
	60	8.4	14.5	14.6	17.4	18.3		9.5	13.3	14.4	16.2	17.0
	75	8.2	14.2	14.3	17.3	18.6		9.4	13.3	14.7	16.3	17.1
	90	7.8	11.7	11.3	15.0	15.9		9.1	10.6	12.3	14.5	14.2
	105	7.7	11.5	11.2	15.9	14.4		9.1	10.3	11.2	15.7	14.3
	120	8.0	11.6	11.3	17.4	14.7		9.2	11.9	11.9	14.2	14.6
	135	7.9	11.5	12.6	15.9	14.3		9.1	12.5	14.6	14.2	18.2
	150	8.2	11.8	11.9	16.8	14.4		9.3	11.5	13.1	13.2	15.9
	165	8.3	11.2	11.8	14.9	13.4		9.3	10.7	11.5	12.1	13.7
	180	7.8	10.1	10.8	12.8	10.7		9.0	9.8	10.5	8.8	9.5
	195	7.5	10.2	10.3	11.4	9.4		8.7	9.0	9.9	7.3	7.7
	210	7.3	9.7	9.8	11.5	7.7		8.5	8.8	9.7	6.3	7.5
	225	7.3	9.7	9.7	10.3	6.9		8.4	8.8	9.6	6.3	6.7
	240	0.8	1.8	2.3	3.7	2.7		2.1	1.5	2.3	1.2	2.5
	255	0.0	0.0	0.0	0.0	0.0		0.0	0.0	0.1	0.0	0.1

Table D.6. Average TREE-REFRESH Packets as Function of Time for Low Mobility

	Secs	MPR -Refresh0-						Refresh0-				
Receivers		**1**	**3**	**5**	**7**	**9**		**1**	**3**	**5**	**7**	**9**
	15.0	5.8	8.2	7.4	10.2	10.2		5.4	7.1	7.4	7.4	10.2
	30.0	7.6	11.9	10.7	14.6	14.7		7.5	11.7	10.9	12.8	14.1
	45.0	7.2	12.4	9.2	15.2	15.7		7.2	10.1	11.5	14.6	15.3
	60.0	7.3	13.4	10.4	16.4	16.8		7.3	12.1	13.3	16.0	16.8
	75.0	7.2	13.3	13.3	16.3	16.7		7.2	11.8	13.6	16.5	16.9
	90.0	6.9	10.6	11.2	14.0	15.0		6.9	10.2	12.3	14.5	13.8
	105.0	7.0	10.4	10.7	16.2	13.6		7.0	10.1	10.1	13.8	13.3
	120.0	7.1	10.7	11.2	13.0	13.8		7.1	11.8	10.4	13.9	14.1
	135.0	7.1	10.9	15.0	14.1	13.0		7.0	12.6	13.4	13.1	15.4
	150.0	7.2	10.5	13.0	12.9	13.5		7.3	11.9	11.5	16.4	14.5
	165.0	7.1	10.5	11.8	13.1	13.8		7.2	10.7	10.7	12.3	14.0
	180.0	6.9	9.8	10.6	10.4	12.9		6.9	7.9	9.8	9.6	13.3
	195.0	6.6	9.3	6.9	9.6	11.0		6.6	6.9	9.3	8.1	12.0
	210.0	6.4	8.8	5.7	9.4	10.0		6.4	5.9	8.9	6.5	10.8
	225.0	6.4	8.5	5.7	9.1	10.0		6.4	5.7	8.9	6.3	10.1
	240.0	1.5	1.6	1.2	1.8	3.8		1.3	0.8	2.7	2.3	2.4
	255.0	0.0	0.0	0.0	0.1	0.4		0.0	0.0	0.2	0.0	0.2

Table D.7. Average TREE-REFRESH Packets as Function of Time for Low Mobility

	Secs	Refresh0N -				
Receivers		**1**	**3**	**5**	**7**	**9**
	15	5.4	7.1	7.4	7.4	10.2
	30	7.5	11.7	10.9	12.8	14.1
	45	7.2	10.1	11.5	14.6	15.3
	60	7.3	12.1	13.3	16.0	16.8
	75	7.2	11.8	13.6	16.5	16.9
	90	6.9	10.2	12.3	14.5	13.8
	105	7.0	10.1	10.1	13.8	13.3
	120	7.1	11.8	10.4	13.9	14.1
	135	7.0	12.6	13.4	13.1	15.4
	150	7.3	11.9	11.5	16.4	14.5
	165	7.2	10.7	10.7	12.3	14.0
	180	6.9	7.9	9.8	9.6	13.3
	195	6.6	6.9	9.3	8.1	12.0
	210	6.4	5.9	8.9	6.5	10.8
	225	6.4	5.7	8.9	6.3	10.1
	240	1.3	0.8	2.7	2.3	2.4
	255	0.0	0.0	0.2	0.0	0.2

Table D.8. Average TREE-REFRESH Packets as Function of Time for Low Mobility

	Secs	Increase-						Notify-				
Receivers		**1.0**	**3.0**	**5.0**	**7.0**	**9.0**		**1.0**	**3.0**	**5.0**	**7.0**	**9.0**
	15.0	5.7	6.9	7.5	8.7	8.3		5.8	7.5	10.4	9.3	10.4
	30.0	7.3	12.9	11.8	15.0	15.1		8.1	12.4	12.3	14.3	14.6
	45.0	8.2	13.5	13.4	15.1	16.7		8.2	13.2	12.5	16.0	16.7
	60.0	8.4	14.3	14.6	16.1	18.2		8.4	14.0	13.7	17.3	17.8
	75.0	8.2	13.3	14.8	16.1	19.0		8.2	13.9	13.8	17.3	18.2
	90.0	8.0	12.2	13.2	14.4	16.6		7.8	12.1	12.3	15.4	17.0
	105.0	7.7	11.0	11.4	14.0	14.9		7.9	13.1	10.4	14.3	15.0
	120.0	7.9	12.1	11.9	14.0	16.3		8.0	12.5	12.0	14.8	17.3
	135.0	8.0	13.8	13.7	19.1	20.5		8.0	11.9	13.3	15.3	21.5
	150.0	8.2	13.5	14.9	15.7	18.9		8.1	11.6	13.9	18.0	18.7
	165.0	8.1	11.8	13.7	13.2	14.2		8.1	10.9	16.4	19.4	14.9
	180.0	7.8	10.7	11.5	12.7	13.7		7.8	9.1	10.0	18.5	11.9
	195.0	7.5	9.9	9.9	9.7	12.1		7.5	9.0	9.1	11.2	11.0
	210.0	7.4	9.7	10.2	9.6	11.7		7.4	8.8	8.9	10.6	11.0
	225.0	7.3	9.8	9.7	9.4	10.9		7.3	8.8	8.8	10.3	10.9
	240.0	1.5	4.1	2.2	4.9	4.8		0.5	1.6	1.9	4.4	4.3
	255.0	0.0	0.0	0.0	0.1	0.0		0.0	0.0	0.1	0.1	0.2

Table D.9. Average Data, Control, and Refresh Packets and Standard Deviations

	Basic					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2126.6	206.0	135.4	2113.8	172.8	165.9
3	3107.7	345.0	196.5	3456.6	245.2	235.6
5	3241.1	414.7	206.2	3464.9	300.6	236.9
7	3707.3	516.8	247.2	3426.7	321.6	238.7
9	3614.0	571.5	243.0	3036.2	326.4	213.8

Table D.10. Average Data, Control, and Refresh Packets and Standard Deviations

	MPR					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2408.8	189.8	152.8	2714.2	203.7	199.9
3	2717.9	266.9	185.0	2415.5	196.6	200.0
5	3150.5	328.4	207.1	3436.5	246.1	235.0
7	3001.9	368.0	202.6	2171.6	204.4	175.1
9	3191.3	448.0	221.9	2159.5	248.6	176.2

Table D.11. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	1867.1	181.1	114.2	1748.8	146.6	145.1
3	2661.4	325.6	178.2	2379.4	213.0	192.9
5	2630.0	396.7	174.9	2232.7	268.4	168.6
7	3105.9	488.7	228.0	2401.0	269.3	195.2
9	3344.9	582.5	240.4	2373.6	338.6	205.0

Table D.12. Average Data, Control, and Refresh Packets and Standard Deviations

	Refresh0N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2083.5	181.0	113.6	2037.4	146.5	144.7
3	2509.1	317.0	164.6	2018.8	200.5	161.8
5	3037.2	402.2	191.2	3202.9	263.8	189.8
7	3045.2	486.6	210.0	2206.4	277.3	176.8
9	890.1	570.4	238.8	691.2	312.9	196.2

Table D.13. Average Data, Control, and Refresh Packets and Standard Deviations

	MPR-Refresh0N					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2083.2	153.9	118.1	2037.4	143.1	142.4
3	2430.4	256.4	159.0	2005.2	180.0	160.7
5	2814.6	328.1	202.5	2439.9	228.8	204.4
7	2983.3	397.6	223.6	2226.2	227.3	194.1
9	3380.2	448.8	231.1	2276.4	234.1	173.2

Table D.14. Average Data, Control, and Refresh Packets and Standard Deviations

	Notify					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2123.4	202.0	135.5	2108.2	169.6	164.9
3	3259.1	338.1	192.6	3812.3	238.2	218.4
5	2880.6	387.5	198.3	2513.6	262.1	209.1
7	3880.9	545.7	258.0	3508.7	356.2	241.1
9	4173.7	615.0	268.8	3569.4	386.8	232.5

Table D.15. Average Data, Control, and Refresh Packets and Standard Deviations

	Increase					
Receivers	Data Pkts	Control Pkts	Tree Refresh	Data Stdv	Control Stdv	Refresh Stdv
1	2123.9	217.2	131.0	2110.1	185.7	166.6
3	3159.9	369.2	202.2	3445.0	282.9	234.5
5	3205.8	440.3	211.5	3470.4	318.4	233.7
7	3260.7	532.7	234.5	2493.9	337.6	216.2
9	4070.4	638.4	268.7	3465.8	398.4	233.8

Vita

Michael Christman was born in Jacksonville Florida in 1978, but, as a "military brat" he has lived in a number of states. He graduated from Hayfield High School in Alexandria, Virginia in 1996 and was the captain of the track and cross country. Although he did dabble in a bit of programming in the classroom, his interest in computers was limited while in high school.

He joined an undergraduate engineering program at Columbia University only because "I would rather finish a problem set than write a paper". He received his B.S. in Electrical Engineering in 2000. While in New York City, he was the captain of the track and cross country programs and received Second Team All-Ivy honors.

Michael became interested in pursuing an advanced degree in engineering after spending two summers at the University of Colorado in Boulder, Colorado. There he participated in a pair of undergraduate research projects and conducted high altitude training. He found that he enjoyed the flexibility and challenges offered in the university setting. Not only that, but he soon realized that as a graduate student he would be allowed to show up to work in shorts and a t-shirt.

Michael began his Master's work in the fall of 2000 at Virginia Tech in Blacksburg, Virginia. While at Virginia Tech he was funded under the Office of Naval Research's NAVCIITI project. He continued his collegiate running career as a member of the Virginia Tech Cross Country team and was awarded the Coach's Award. He has run two marathons as a post collegiate athlete. He finished 15th in the Philadelphia Marathon, his first try at that distance, and hopes to qualify for the Olympic Marathon trials.

Following graduation, Michael will begin work in the Northern Virginia area. Ready to return to an urban setting, he will live within the Washington DC city limits.