2009-07-29

A Hop-by-Hop Architecture for Multicast Transport in Ad Hoc Wireless Networks

Manoj Kumar Pandey
Brigham Young University - Provo

A HOP BY HOP ARCHITECTURE FOR MULTICAST TRANSPORT IN AD HOC WIRELESS NETWORKS

by

Manoj K Pandey

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2009

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Manoj K Pandey

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Dr. Daniel Zappala, Chair
Date	Dr. Charles Knutson
Date	Dr. Dan Olsen
Date	Dr. Parris Egbert
Date	Dr. Kent Seamons

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Manoj K Pandey in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Dr. Daniel Zappala Chair, Graduate Committee

Accepted for the Department

Date	Dr. Kent Seamons Graduate Coordinator

Accepted for the College

Date	Dr. Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

A HOP BY HOP ARCHITECTURE FOR MULTICAST TRANSPORT IN AD HOC WIRELESS NETWORKS

Manoj K Pandey

Department of Computer Science

Doctor of Philosophy

Ad hoc wireless networks are increasingly being used to provide connectivity where a wired networking infrastructure is either unavailable or inaccessible. Many deployments utilize group communication, where several senders communicate with several receivers; multicasting has long been seen as an efficient way to provide this service. While there has been a great deal of research on multicast routing in ad hoc networks, relatively little attention has been paid to the design of multicast transport protocols, which provide reliability and congestion control. In this dissertation we design and implement a complete multicast transport architecture that includes both routing and transport protocols.

Our multicast transport architecture has three modules: (a) a multicast routing and state setup protocol, (b) a mobility detection algorithm, and (c) a hop-by-hop transport protocol.

The multicast routing and state setup protocol, called ASSM, is lightweight and receiver-oriented, making it both efficient and scalable. A key part of ASSM is its use of Source Specific Multicast semantics to avoid broadcasting when searching for sources. ASSM also uses routes provided by the unicast protocol to greatly reduce routing overhead.

The second module, MDA, solves the problem of determining the cause of frame loss and reacting properly. Frame loss can occur due to contention, a collision, or mobility. Many routing protocols make the mistake of interpreting all loss as due to mobility, resulting in significant overhead when they initiate a repair that is not required. MDA enables routing protocols to react to frame loss only when necessary.

The third module is a hop-by-hop multicast transport protocol, HCP. A hop-by-hop algorithm has a faster response time than that of an end-to-end algorithm, because it invokes congestion control at each hop instead of waiting for an end-to-end response. An important feature of HCP is that it can send data at different rates to receivers with different available bandwidth.

We evaluate all three components of this architecture using simulations, demonstrating the improved performance, efficiency and scalability of our architecture as compared to other solutions.

ACKNOWLEDGMENTS

I owe my gratitude to a large number of people who have inspired me and helped me make this dissertation possible. These people have taught me, developed my research abilities, and assisted me in developing the right temperament to identify and solve problems.

I would like to begin by thanking my PhD adviser, Professor Daniel Zappala for supporting my research. Dr. Zappala is someone who has always inspired me and is one of the smartest people I know. Without an exception, I have always found our weekly meetings to be stimulating, relevant and motivating.

It is a great opportunity to express my respect and gratitude to Professor Charles Knutson for all his valuable guidance, inspiring discussions, and overall motivation during my PhD. He is the second member of my PhD committee and has always provided me with insightful comments and constructive criticisms.

I would also like to thank Professors Dan Olsen, Parris Egbert, and Kent Seamons for taking out time to be a part of my PhD committee and going over my research areas, dissertation proposal, and dissertation drafts to provide valuable feedback.

I am pleased to thank my manager at Cisco, Siva Yaragalla, and our team's Technical Leader, Anantha Ramaiah. In spite of our aggressive work-related deadlines, both of them have always stressed that I finish my research. I owe a lot to them for their sincere encouragement.

I am also grateful to my manager at Cypress Semiconductor, Ryan Woodings, for all his help and encouragement during my two summer-internships. Ryan and I worked on several interesting problems related to wireless networking.

I would also like to thank my lab members: Roger Pack, Lei Wang, Daniel Scofield, Qiuyi Duan, and Daniel Delorey. I cannot thank them enough for all the meaningful (and occasionally not so meaningful!) discussions; these discussions were both enjoyful and often provided the much needed food for thought.

Additionally, I would also like to thank our Graduate Program Assistants Shannon Källåker and Mindy Varkevisser. I have bothered them on a constant basis and they have tirelessly helped me with all my queries related to course-work, registration, scheduling of presentations etc.

I would like to dedicate this dissertation to my parents. They have always valued education and have worked very hard to ensure that I get the best possible education. I just cannot thank them enough!

Most importantly, I would like to thank my wife and our son. On countless occasions, when I should have spent time with them, I instead chose to work on my dissertation! Without their love, understanding and sacrifice, it would not have been possible for me to complete my dissertation.

Contents

Chapter 1

Introduction

The future of communication networks lies in providing reliable and dependable access to the Internet anywhere, anytime. Today, the Internet is comprised mainly of computers connected to each other using wires, which poses an obvious limitation on the reach of the Internet. To overcome this limitation, wireless access is increasingly used to connect computers. A wireless access point is a computer simultaneously connected to the Internet via wires and to Internet users via a wireless radio (Figure 1.1). These access points typically allow connection to the Internet in public places, such as a university campus or inside a bookstore. Although access points extend the reach of the Internet, wireless devices still need to be within the radio range of an access point to use the Internet.

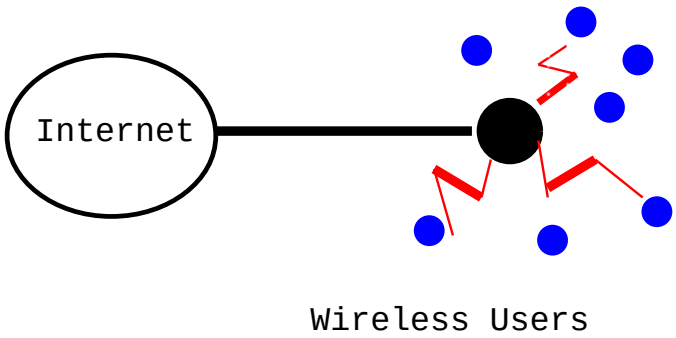


Figure 1.1: Wireless Access Point

To extend the reach of the Internet beyond wireless access points, research is being done on ad hoc networks [1], which consist of a group of mobile wireless nodes without any fixed infrastructure. Ad hoc devices typically use batteries so that they can be mobile without depending upon electricity. To overcome the limitations of a wired infrastructure, they form network connections (or links) among themselves as they move. In an ad hoc network, the path from a source to a destination often uses several intermediate nodes, which allows communication even if the source and destination are not within the radio range of each other (Figure 1.2).

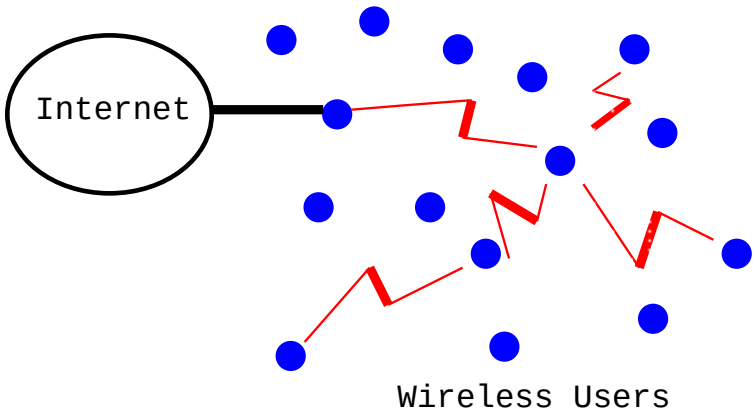
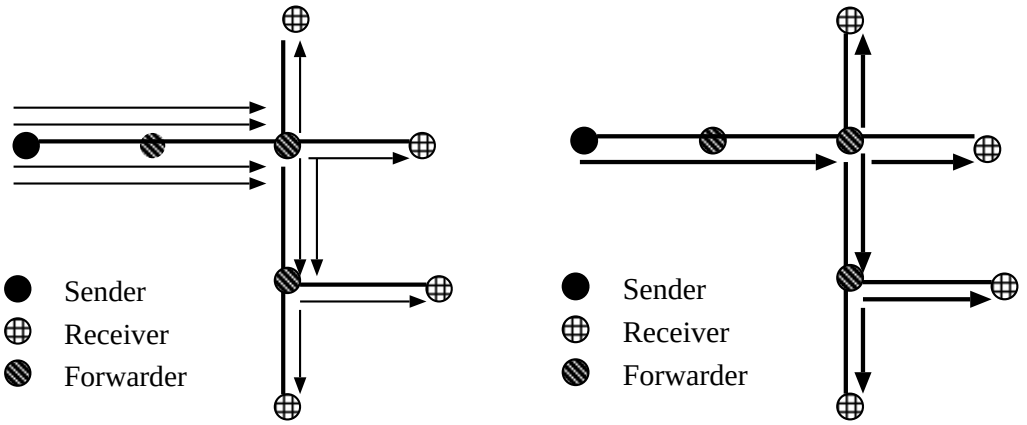


Figure 1.2: Ad Hoc Wireless Networks

Another motivation for ad hoc networks is the case when a network needs to be established on the fly, when existing resources are disabled or destroyed, such as fire or earthquake. In these cases, group communication becomes important as a means of providing search-and-rescue or other communications. Examples of such deployments include military situations, disaster management, ad hoc conferences (e.g., at an airport or in a restaurant), classroom deployments or at social networks (e.g., a sports event). Group communication is achieved efficiently using a mechanism known as multicast. Consider the case of a person talking to four colleagues (Figure 1.3). Without multicast (Figure 1.3)(a) this person must send a separate audio stream to four other computers, which wastes valuable network resources. On



(a) Unicast: Packet is replicated at the sender

(b) Multicast: Packet is replicated by the network

Figure 1.3: Unicast and Multicast

the first part of the path, the sender must transmit four copies of each packet. With multicast,(Figure 1.3)(b) this person needs to send only one copy of each packet and the network replicates it as necessary. Multicasting is especially important in ad hoc networks because it allows an ad hoc network, which is typically power-constrained, to be more efficient.

The goal of multicasting in an ad hoc network is to provide services that are indistinguishable from those available in the wired Internet. From a routing perspective, the primary challenge is that when nodes move, the links forming the multicast tree can break. Multicast needs efficient mechanisms that can detect and repair these route breaks. From a transport perspective, multicast must be reliable, use congestion control, and provide fairness to all data flows. Reliability is especially important in wireless networks, which often suffer from a high loss rate due to mobility and congestion; reliability ensures that when data is lost, then it gets retransmitted. Congestion control is equally important because of the limited bandwidth of wireless networks; congestion control ensures that each flow estimates the correct sending rate and thereby fully utilizes the available bandwidth. Lastly, fairness ensures that competing flows get equal treatment.

1.1 Motivation for Research

The problem we solve in this dissertation is the ability to provide services of reliability and congestion control efficiently for multicast flows in ad hoc networks. In absence of reliability, group communications like battlefield communication and disaster management may drop critical information that would never get retransmitted. In absence of congestion control, group communication like multiple co-existing conferences running in the same geographical area, say an airport, may not estimate the correct sending rate and hence either may not fully utilize the available bandwidth or may not fairly share the available bandwidth among themselves.

Existing approaches fail to provide reliability and congestion control in an efficient manner. First, network layer multicast can be used with *a transport layer of reliable and congestion controlled multicast* built above it [2, 3]. Using network layer multicast means that the network provides best-effort delivery of data to all group members, but packets are not sent reliably. Building a transport protocol on top of this multicast service is complex because different group members receive different sets of packets and have different levels of congestion on their multicast branch. Experience with the Internet has shown that this approach can have a high control overhead, and typically the source can only send data at the rate of the slowest group member. Furthermore, such a transport protocol is slow to respond to congestion due to an end-to-end approach.

Another existing approach is to build an *application layer multicast* or *peer-to-peer multicast* using TCP connections [4, 5]. The advantages of this approach are that it is easy to deploy and avoids the complexity of the network layer multicast. Further, this approach uses TCP, which is a proven solution for reliability and congestion control. However, the downside of this approach is that it suffers from two problems of stress and stretch. Stress occurs when the same network link is used more than once for the same packet and stretch occurs when multicast chooses a path that is

not the shortest. Because capacity and power are constrained in an ad hoc network, inefficiencies due to both stress and stretch should be avoided.

Our alternative to these approaches is to build a hop-by-hop multicast transport protocol. Unlike the above schemes, where congestion control is invoked only at the end node of the flow, a hop-by-hop scheme invokes congestion control at each hop and adjusts the rate appropriately at each hop. At each node, the rate at which it receives packets can be different than the rate at which it can send packets. In order to accommodate this rate differential, each intermediate node also maintains a buffer to store packets that are waiting to be sent. The size of this buffer limits the rate differential, since an upstream node must stop transmitting when the downstream buffer is full.

Our hop-by-hop style congestion control offers several advantages over the above two approaches of network layer or application layer multicast. The hop-by-hop approach can overcome the disadvantages of other network-layer solutions by simplifying reliability and congestion control, since it is invoked at each hop. This type of congestion control mechanism has a faster response time to congestion as compared to an end-to-end style congestion control offered by both native multicast and application layer multicast [6]. Further, unlike a network layer multicast, this approach also allows members to receive data at different rates rather than enforcing the entire group to receive data at a single rate, which is often determined by the slowest member. If different branches of the multicast tree support different rates, a hop-by-hop scheme can send data at these rates, provided each hop has enough buffer space. The hop-by-hop approach also overcomes the limitations of peer-to-peer multicast. Each packet is sent once per link, eliminating stress, and packets can be sent on the shortest path, eliminating stretch. Thus, a hop-by-hop multicast transport architecture for ad hoc wireless networks can potentially improve multicast

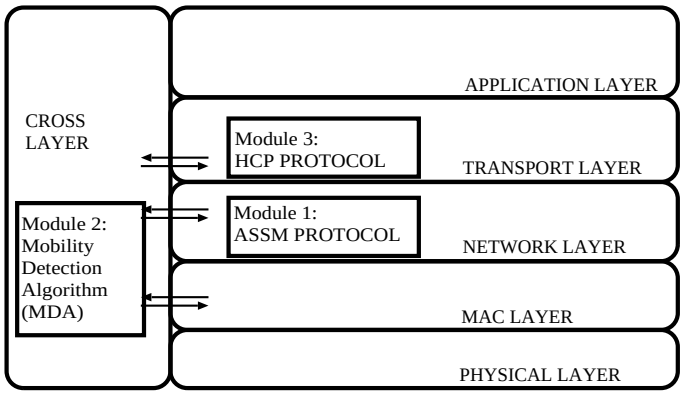


Figure 1.4: Hop-by-hop Multicast Transport Architecture

application performance. Further, it can also provide fast response to congestion, allow for bandwidth heterogeneity, and be more efficient than peer-to-peer multicast.

1.2 Overview of Research: ASSM, MDA, and HCP

In this dissertation, we build a hop-by-hop multicast transport architecture, where several modules sitting at different layers work in tandem to provide routing, reliability, and congestion control. Our research can be divided into four phases.

In the first phase, we begin with a rigorous evaluation of existing ad hoc multicast routing protocols. This phase motivates our subsequent research by providing valuable insight into the design of multicast routing protocols; later, we use these insights as we build our multicast architecture. Further, this chapter provides an outline of various factors that can affect multicast performance in ad hoc networks and emphasizes the importance of a sound performance evaluation methodology that takes into account the unique characteristics of ad hoc networks.

The next three phases of our research design a hop-by-hop multicast transport architecture, shown in Figure 1.4. The architecture consists of three modules working together to provide reliability and congestion control for multicast flows.

The first module is a simple multicast state setup protocol; we call this protocol the *Ad hoc State Setup Multicast Protocol (ASSM)*. A state setup protocol is used to

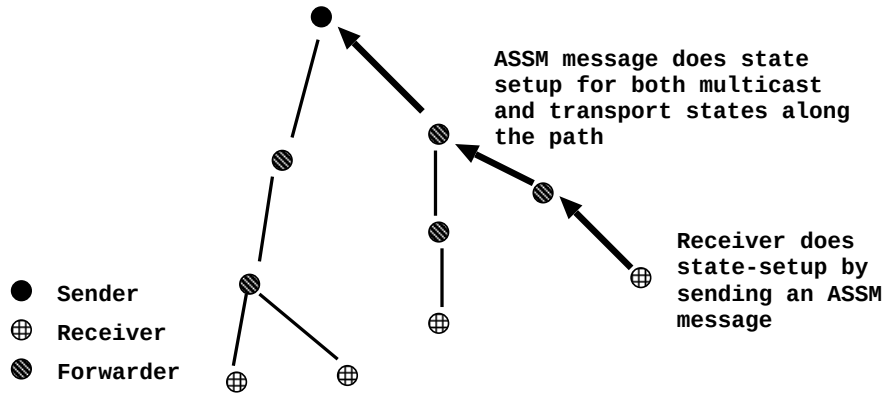


Figure 1.5: State setup using ASSM

setup multicast forwarding and transport states in the multicast tree. As shown in Figure 1.5, ASSM uses one of its messages to do the state setup. An ASSM receiver sends this message, which travels towards the multicast sender. As the message travels, all the routers along the path configure their multicast and transport states. The multicast state is setup by creating a forwarding entry in the routing table so that the node can forward future data coming from the sender. The transport state is setup by adding an entry for the flow at the transport layer and by allocating buffer to hold data segments.

One of the advantages of ASSM is that it does not do any route discovery of its own, instead it uses routes provided by the underlying unicast routing protocol. Traditionally, route discovery is done with the help of a network-wide route discovery broadcast, where a route discovery packet is sent to each node in the network. Using existing routes provided by underlying unicast routing protocol reduces the network overhead. Further, ASSM is designed to be independent of the underlying unicast routing protocol.

The second module solves one of the difficult problems facing mobile ad hoc networks, which is determining the cause of frame loss and reacting properly. Most ad hoc routing protocols assume that any lost frame indicates that the destination node for the frame has moved, and hence the route is broken. This assumption

can be faulty; it is possible that contention or collision may have caused the loss, rather than mobility. In certain scenarios, routing protocols can incorrectly mark 80% of contention or collision losses as mobility losses (Section 6.6). Routing protocols making this mistake suffer from significant overhead when they initiate the repair of routes that have not been broken, resulting in decreased throughput for affected applications. Hence, the second module, which is a *mobility detection algorithm (MDA)*, properly detects the cause of a lost packet. Routing protocols first consult with MDA and they discover new routes only when MDA confirms that the current route is truly broken.

MDA is an integral module of our multicast architecture since our architecture consists of a multicast state setup protocol, ASSM, which is dependent upon underlying unicast routing protocols. Without MDA, these unicast protocols fail to distinguish congestion based losses from mobility based losses during congestion. This failure can lead to an inefficient behavior, which is undesirable for our multicast architecture.

The third module is a *hop-by-hop congestion control (HCP)* algorithm. This module addresses several problems that are present when one attempts to provide congestion control for ad hoc networks. First, an end-to-end congestion mechanism can increase the signaling time for indicating congestion occurring at receivers. HCP uses a hop-by-hop mechanism and selects a rate based on the state of local congestion; this way, it provides a faster response time to congestion. Second, a network layer multicast typically does not allow individual members to receive data at rates suitable to them, instead it enforces the entire group to receive data at a single rate. With HCP, each intermediate node buffers its packets and selects its own sending rate; this method does not impose the constraint of maintaining a global sending rate for the entire group. Third, end-to-end multicast transport often send acknowledgments on a per-packet basis, which is an inefficient behavior. HCP uses a periodic acknowledg-

ment and piggybacks this feedback on top of an ASSM message and hence provides a more efficient behavior.

Our research provides several contributions. First, we conduct evaluation of existing multicast protocols; this study provides valuable recommendations, which should be considered in the design of multicast routing protocols. Second, we show that ASSM greatly simplifies the steps required to establish and maintain a multicast tree and delivers high throughput with low overhead. Third, we show that a Mobility Detection Algorithm, MDA, significantly reduces routing overhead and can increase throughput by 10 to 100%, depending on the routing protocol and the mobility scenario. Finally, we show that HCP delivers high throughput and ensures fairness if there are multiple co-existing multicast groups. Furthermore, HCP provides better throughput as compared to an optimal peer-to-peer overlay multicast under conditions of high traffic and mobility.

Chapter 2

Related Work

In this chapter we present an overview of some of the important research related to multicast routing and transport in ad hoc networks. Since our dissertation proposes an architecture that provides multicast transport for ad hoc networks, these related research provide an important background. We begin by discussing several multicast routing protocols for ad hoc networks. We supplement this discussion by presenting overview of the major Internet multicast routing protocols, because more work has been done on multicasting in the Internet. Since our multicast state setup protocol uses a unicast routing protocol, we also present a section on the state of the unicast routing protocols. Next, we examine research related to distinguishing the nature of packet loss; this discussion is important to our research since one of the main modules of our architecture is a mobility detection algorithm. We then examine the state of multicast transport in ad hoc networks. The foundation of this work lies in unicast congestion control, so we review improvements to TCP in wireless networks. Finally, since cross-layer information sharing can be a powerful way to enhance performance of wireless networks, this section concludes by describing several advances in this area.

2.1 Multicast Routing

Multicasting enables a group of users to communicate efficiently with each other. Users belonging to these group are called multicast members; a member can be either a source or a receiver or both. Since one of the components of our proposed architecture is a multicast routing protocol for ad hoc networks, we begin by providing an overview existing multicast routing. In the beginning, multicasting was designed for the Internet [7]; more recent work involves multicasting in ad hoc networks. Most of the multicasting in ad hoc networks seems to draw inspiration from that of the Internet. Hence, in this section, we present a discussion on some of the important multicast routing protocols not only for ad hoc networks, but also for the Internet. However, none of the multicast protocols discussed in this section provide any form of reliability of congestion control; we revisit those protocols that aim to provide these services in Section 2.3. Lastly, since ASSM runs on top of an ad hoc unicast routing protocol, we also describe two of the important unicast routing protocols proposed for ad hoc networks.

2.1.1 Multicast Routing in Ad Hoc Networks

In an ad hoc network, multicast routing protocols can be divided into two categories. In the first category, multicast members communicate by building a *tree* structure for doing routing. In the second category, members communicate by generating a *mesh* among them. Both of these approaches need help from intermediate nodes, often known as *forwarding nodes*, to provide routes between the members. These intermediate nodes may or not be a member of the group.

The mechanism of multicast routing can be divided broadly into three phases – *route discovery*, *data transfer*, and *route maintenance*. In the *route discovery* phase, which is usually the beginning phase, sources or receivers (or both) discover routes. After this phase, a routing structure (tree or mesh) is established. Once this phase is

successful, multicast sources begin to send data to group members. This leads to the *data transfer* phase. However, during the data transfer phase, mobility may cause the routing structure to break. This is the third phase, called *route maintenance* phase. The burden of route maintenance may lie with either the source or the receivers. For example, a source could periodically send a broadcast message to refresh the routes (tree or mesh) and to repair any broken links. On the other hand, if the responsibility of route repair lies with each receiver, a receiver can rejoin if it detects its disconnection from the other members.

ODMRP: On Demand Multicast Routing Protocol

Gerla et. al. have developed ODMRP, which belongs to the category of mesh based multicast routing protocols [8]. Like DVMRP [7] (used in the Internet), ODMRP is a very simple protocol, and avoids excessive control messages. In fact, the onus of *route discovery* lies only with the source. When a source has data to send, it sends a network-wide flood of a JOIN QUERY; when receivers listen to this Query, they join the group by replying to the source. This JOIN REPLY message travels in the reverse direction of the Query message, a mechanism known as Reverse Path Forwarding [7]. As the reply message travels towards the source, the intermediate nodes establish (or refresh if it existed already) forwarding states. When an intermediate node receives multiple requests for the same group, for which it has already established a forwarding state, it suppresses any further JOIN REPLY forwarding to reduce channel overhead. Each member (source, receiver, and forwarder) stores information about the multicast mesh in a soft state and expires it after certain time duration.

With ODMRP, the responsibility of the *route maintenance* phase also lies with the source. A source uses the periodic flood of JOIN QUERY messages for maintaining routes as well. This flood refreshes routes and repairs any broken links. Therefore, if a link breaks in between two JOIN QUERY messages, the source does not take any

action; instead, it merely waits for the next JOIN QUERY flood to heal the mesh. In conditions of high mobility, where a large fraction of nodes are mobile and link breaks frequently, this can lead to reduced performance [9].

ADMR: Adaptive Demand Driven Multicast Routing

Jorjeta and Johnson designed ADMR, which unlike ODMRP, belongs to the category of tree based multicast routing protocols [10]. ADMR uses a special type of multicast tree, known as a source specific tree, in which a separate tree is built for each source, with the source at the root of the tree. Unlike ODMRP, here the responsibility of *route discovery* lies with both the source and the receivers. A source sends a network-wide flood at regular intervals for discovering receivers that wish to join the source. Receivers respond by sending a RECEIVER JOIN message, using Reverse Path Forwarding. If a new receiver misses these route discovery messages, it can send a network-wide flood packet, called MULTICAST SOLICITATION message, to discover the source.

As compared to ODMRP, ADMR has a more sophisticated approach during the *route maintenance* phase. Each receiver/forwarder monitors the status of the link to their upstream member by maintaining a counter of recently received packets. If a certain number of consecutive packets (for example 3) are not received by a member, then it concludes that the link is broken and initiates a repair process. The nature of the repair action depends upon the node. If the node is a pure receiver (a receiver that is not a forwarder for that source/group), it rejoins the group by sending another multicast solicitation message. If the node is a forwarder, then it rejoins by sending a RECONNECT message to the source. A source replies to the RECONNECT message using a RECONNECT REPLY reply message, informing the forwarder that it is now reconnected to the tree. The disadvantage to this approach is that it can trigger a lot of control overhead, making ADMR inefficient [9].

During the route maintenance phase, ADMR uses an interesting adaptive approach to handle mobility. If a receiver makes multiple unsuccessful attempts to join the source, it indicates to the source that the mobility of the system has increased. ADMR then concludes that instead of doing excessive route repair, it can simply revert to flooding on a temporary basis; every data packet is forwarded to all the nodes in the network.

MAODV: Multicast Ad hoc On Demand Distance Vector Routing

Royer and Perkins design a novel multicast routing protocol, MAODV, which uses routes provided by the underlying unicast routing protocol [11]. This protocol, which is named as Multicast AODV or MAODV, does minimal additional routing efforts of its own and relies primarily on routes provided by AODV [12]. The unicast protocol, AODV consists of RREQ and RREP messages, to broadcast a request for a route and to send back a route reply to that request respectively [12].

With MAODV, multicast *route discovery* is done by building a tree with a group leader, which is the first node to start the group. A new node wishing to join the group broadcasts a RREQ message to find the group leader. However, if the node wishing to join the group already has a valid unicast route entry in its AODV routing table for the group leader, then instead of broadcasting, the node unicasts the RREQ directly to the group leader. Member nodes of the tree respond by sending RREP message, which contains the address of the group leader.

MAODV's *route maintenance* mechanism takes into account not only link breaks but also network partitioning[1]. Similar to AODV, link break repair is based on a periodic HELLO message sent by the group leader, which is meant to announce that the members are still connected. If a member forwards a data packet, it does not need to send a HELLO message, since neighbors already know its presence when they

[1]If one or more nodes become unreachable from the rest of the network, they are referred to as partitioned.

receive data packets. If a node on the multicast tree fails to receive any HELLO messages or data packets, it starts link repair by sending a TTL limited local RREQ with the join flag set; the disconnected node expects to hear an RREP. If it fails to receive an RREP within a certain amount of time, it tries to resend RREQ for an additional certain number of times. If it still does not receive any RREP, the disconnected node concludes that network partitioning has occurred. In that case, a new group leader is selected for the newly detected partition.

2.1.2 Multicast Routing in the Internet

Multicast routing was originally designed for the Internet and has been developed for a long time and thus is more mature than multicasting in ad hoc networks. Research for multicasting in ad hoc networks has often borrowed mechanisms from those of the Internet-based multicast routing protocols and adapted them for ad hoc networks. In this section, we describe three of the popular Internet-based multicast routing protocols and aim to highlight some of their important mechanisms: DVMRP, PIM, and SSM. However, all of these protocols are not designed to provide any form of reliability of congestion control and hence, cannot be used for our proposed multicast transport architecture.

Distance-Vector Multicast Routing Protocol (DVMRP)

DVMRP is the Internet's first multicast routing protocol [7]. In DVMRP, a multicast source periodically broadcasts data to the network. Nodes receiving the broadcast forward only those packets that arrive from the shortest path to the source; data coming from other interfaces is considered duplicate and hence dropped. When a router does not need the multicast data, it prunes itself from the tree, provided all of its children in the tree also do not want to be a part of the group. Thus, if all the children of a node send a prune message, the parent node prunes itself. The prune

requests travel upstream to the source till all the nodes that have no child members are pruned. In the pruned state, if a node wishes to (re)join the group, it does so by sending a *graft* message, which cancels the previous prune request.

Core Based Trees (CBT)

CBT addresses one of the central problems of multicasting in the Internet – source discovery - by creating a special tree that has a common rendezvous point (RP) or core for all the members belonging to a group. All branches leading to various members emanate from this core. Unlike DVMRP, CBT is a member-driven protocol, where the onus of joining the tree lies with the sender or receiver. A member joins the tree by sending a JOIN-REQUEST message towards the RP. When the RP or a node already on the tree receives this JOIN-REQUEST, it sends a JOIN-ACK message back to the requesting node. Reception of a JOIN-ACK message indicates that the member is now connected to the group. To forward data, a member sends packets to its neighbors on the tree and this process repeats until everyone has the data.

Protocol-Independent Multicast (PIM)

PIM is another widely used multicast routing protocol in the Internet. PIM is called so because it is not dependent upon the underlying unicast routing protocol. Unlike DVMRP, PIM works in two modes – *dense* and *sparse*, depending upon the multicast application model. The *dense mode* application model assumes that most of the routers in the network belong to the multicast group [13]. In this mode, data forwarding is done using a protocol similar to DVMRP. The *sparse-mode* application model is applicable to scenarios where members are sparsely distributed in the Internet, but members are not necessarily small in number. Like CBT, this mode uses a rendezvous point (RP) [14]. The RP is used for learning about sources; after that a member joins using the shortest path tree for each source.

Source Specific Multicast (SSM)

In the IP multicast model, any sender can send at any time. This causes the source discovery problem, where members must constantly locate new sources. This also causes a shortage of multicast addresses, since the group address must be globally unique. SSM solves both of these problems by using a combination of the source and group address to identify the multicast group. This eliminates the source discovery problem and provides each host with its own multicast address space. SSM allows permanent multicast addresses, which is useful for applications like Internet TV, and distance learning where a single sender serves numerous receivers.

2.1.3 Unicast Routing Protocols for Wireless Ad Hoc Networks

Our multicast transport architecture uses a multicast state setup protocol, ASSM, which uses routes provided by a unicast routing protocol; hence, we present a section on the state of the art for unicast routing protocols in ad hoc networks.

Dynamic Source Routing (DSR)

Dynamic Source Routing (DSR) is an on-demand source routing protocol [15]. Each node stores routes to several destination nodes in its route cache. When a node needs to send data, it uses these routes to forward data; however, if it does not have a route for a destination, then it broadcasts a network-wide *Route Request* broadcast. Each of the nodes that receive this Route Request, broadcasts it further; this forwarding is done until the request flood reaches the destination node or an intermediate node that has a route for that destination.

Each Route Request carries with it a unique identifier created by the initiator of the request, which is stored by all the nodes that process the request. Nodes that hear the broadcast for the second time discard it to avoid excessive overhead. If a node fails to hear a Route Reply within the expected time, it sends the request

again. This behavior is limited to a minimum interval between two consecutive Route Requests; this interval increases with each request.

All source routes learned by nodes are kept in a route cache, which decreases the overhead of the route discovery step. In route maintenance, broken links are detected and the routes containing them are removed from use. All the forwarding nodes in the route must verify that the data has been received by the next node in the route; if the subsequent next hop fails to receive the data packet successfully, the node generates a *Route Error* packet and sends it back to the source. Further, DSR can also use promiscuous listening to overhear the Route Reply messages from other nodes and add these routes in its cache.

Ad hoc On Demand Distance Vector (AODV)

Ad hoc On Demand Distance Vector (AODV) is a distance vector routing protocol. Like DSR, the source first generates a network-wide Route Request (RREQ) broadcast. The RREQ propagates till it reaches the destination or an intermediate node, which has 'fresh enough' route for the destination. In either case, they attach the destination route to the header and send a Route Reply (RREP) message back to the source. After broadcasting a RREQ, the node starts its timer and waits for RREP-WAIT-TIME. If the timer expires before it gets a RREP, it sends another RREQ, upto a maximum of RREQ-RETRIES. However, unlike DSR, AODV employs a ring search algorithm to find a route (while sending RREQ), where the ring keeps expanding until it reaches a threshold. The intermediate nodes that forward the RREQ packet also insert the address of their previous node, thereby building the reverse path which might be used ultimately when the RREP is to be sent back to the source.

Each route in the routing table has a lifetime after which it is expunged from the table. AODV may broadcast HELLO messages every HELLO-INTERVAL

to ensure that the neighbors are still alive. If a node in the neighborhood fails to broadcast a HELLO message in the expected time interval, then others conclude that its link is down. Each forwarding node also watches its active next hops (those nodes which have been used by the node to send packets recently) using link layer or network layer mechanisms. Hence, the detection of a link breakage is done regularly in AODV.

2.2 Mobility Detection

One of the difficult problems in mobile ad hoc networks involves finding if a frame loss has occurred because of mobility or congestion, and then having the transport and routing protocols react properly. Traditional transport protocols, like TCP and many of its variants, assume all packet loss is a sign of congestion, and they needlessly decrease the sending rate when mobility causes loss. Similarly, if the routing protocol assumes all packet loss is a sign of mobility, it may initiate an expensive route discovery or repair process when it should instead do nothing and let the transport protocol adjust its rate.

An ad hoc network can loose a frame primarily due to two reasons. First, when the forwarding queue at a node overflows. In this case, the loss is clearly due to congestion, so there is no ambiguity as to the appropriate response. Second, when transmission of a frame fails at the MAC layer, in which case, the cause for a failed transmission could be due to mobility, contention, or interference. Loss due to mobility occurs when a node moves, so that the packet cannot possibly be delivered. Loss due to contention may occur when two or more wireless nodes are transmitting at the same time, causing collisions. Interference also causes loss due to collisions, but the competing source uses a different technology in the same spectrum, such as a microwave oven or cordless phone [16, 17]. In this dissertation, we focus on distinguishing between mobility and contention.

2.2.1 Mobility Detection for Routing Protocols

It is necessary for routing protocols to determine the nature of frame loss and react only when the loss is due to mobility. If routing protocols initiate the repair of routes that have not been broken, it can result in excessive overhead and can lead to decreased throughput for affected applications. In this section, we outline two of the solutions suggested for enabling routing protocols to determine the true nature of frame loss.

Preemptive Routing in Ad Hoc Networks

In this approach, the signal strengths are monitored for existing links. When neighbors move, the signal strength fades and this fading can be taken as a sign of an impending link break. Intermediate nodes monitor the signal strength and in the case of a likely route break, they inform the source by sending a warning message to it. The source can then trigger a new route selection or use other existing routes that it might have.

Using an early warning before the link break occurs enables the protocols to achieve a faster response time for broken links. However, temporary signal fading can occur due to several reasons, even if two nodes are within the radio range of each other. For example, if one of the users walks and temporarily goes behind a physical obstacle, like a wall or tree, this triggers a false warning. Care must be taken to avoid such false warnings. Thus, the mechanism must compensate for variations in signal strength due to fading, multi-path effects, or power conservation mechanisms.

Using Signal Strength for Detecting Link Breaks

Klemm et al. propose another approach that uses signal strength for determining the nature of packet loss. If the received signal strength from a neighbor drops, they prompt the routing layer to trigger a route discovery process. In the meantime,

they ramp up the transmit power of the device to communicate with the neighbor. Like the earlier approach, this approach can also trigger false warnings. Further, it also requires that the power level of the received signal be monitored continuously. Received signal strength is usually a hardware property and hence would involve additional interactions for porting the signal strength output from a specific neighbor and informing the routing layer about it when the signal drops.

2.2.2 Mobility detection for TCP

While our work focuses on routing protocols, there has also been a great deal of work on helping TCP react properly to lost frames. In TCP, reliability is implemented with the use of Acks, which are sent by a receiver upon successful reception of a packet. If a sender fails to receive an Ack or receives duplicate ACKs, it retransmits the lost packets. If an Ack is lost or is received with information about packet loss, it reflects that some part of the network has become congested. Most versions of TCP assume that in the Internet losses (or delay) are primarily due to congestion, and this leads TCP to cut down its sending rate. This behavior makes TCP send data at a much slower rate than what can be sent.

In this section, we present work for wireless ad hoc networks that describe this problem and provide a solution for TCP to work well. We organize this review into two sections. First, we discuss solutions that enable TCP to determine the nature of packet loss before reacting to the loss. Second, we discuss solutions that enable TCP to react to mobility based packet loss.

Enabling TCP to Determine Nature of Packet Loss

Research in this area has focused an end-to-end mechanism for distinguishing between congestion-based losses from mobility-based losses. When a loss occurs, TCP invokes

the differentiating algorithm to verify that the loss is due to congestion and not due to mobility; TCP reacts only when the loss is due to congestion.

For wireless ac hoc networks, Wang and Zhang propose a scheme where out of order packet delivery is used to differentiate packet losses [18]. For a network with little mobility and no congestion, data packets and ACKs arrive in order at the receiver and sender respectively. Hence, the authors assume that any out of order packet delivery is an indication of mobility based route change. Upon detecting that the route has changed, TCP avoids invoking its congestion control algorithm if it sees a packet reordering.

However, like TCP, routing protocols also need to differentiate mobility losses from congestion losses; in this case, the routing protocols should react only when the loss is due to mobility. Both of the above approaches are limited to TCP and do not provide this service of loss differentiation to routing protocols.

Enabling TCP to react to mobility based Packet Loss

Besides enabling TCP to distinguish congestion based losses from mobility losses before reacting to it, TCP has also been modified to mitigate mobility based link loss.

Sundaresan et al. propose *ATP*, which is a rate based transport protocol [19]. Every node tracks the average of the queuing delay and transmission delay. Whenever a data packet is forwarded, the information about average delay is passed to the downstream node. This way it reaches the receiver, which in turn maintains an average of all the delays received. At regular intervals, the receiver uses this information to estimate the sending rate for the sender and sends it back to the sender. When a mobility induced link break happens, the routing layer informs ATP. ATP temporarily pauses sending new data and instead sends a probe message to the receiver to get an estimate of the bandwidth available on the new route.

Chandran et al. propose *TCP-F*, where intermediate nodes send information whenever a link failure occurs[20], using a *Route Failure Notification* (RFN) message. When a TCP sender receives an RFN, it enters into a *snooze* mode, where it freezes all its timers and stops sending any packets. This is done till it receives a *Route Re-establishment Notification* (RRN) message. However, the source does not remain in the snooze mode forever and runs a timer to get out of this mode. RRN can be forwarded by any intermediate node that has previously forwarded the RFN, once it discovers that a new route has been established.

Similar to TCP-F, Holland and Vaidya also propose to send an *Explicit Link Failure Notification* (ELFN) to the TCP sender when a link failure occurs [21]. The authors suggest sending ELFN in two ways. First, it can be sent using an ICMP "host unreachable" message. Second, it can be piggybacked with a route error message (if generated by the routing algorithm). TCP's response to ELFN is to disable the congestion control mechanism and to wait for the route to get repaired before re-invoking congestion control.

2.3 Multicast Transport

Multicasting in ad hoc networks needs to be supported by two services – *reliability* and *congestion control*. Wireless networks incur significant packet loss due to collisions, mobility, and interference. Reliability must be built for multicasting to recover these losses. Besides significant packet loss, an ad hoc network also has limited bandwidth. If a node receives data at a faster rate than the rate at which it can forward, it drops packets from its buffer; congestion control is required to avoid this loss. In this section, we discuss approaches that address the problem of providing reliable multicast, both with and without congestion control.

We first examine multicast transport protocols developed for ad hoc networks. Because this area is new, we also review earlier work on multicast transport for

the Internet. Finally, we explore work on application layer multicast as a realistic alternative to network-layer multicast.

2.3.1 Multicast Transport for Ad Hoc Networks

Reliable Multicast

Chandra et al. use a gossip-based approach to provide multicast reliability in an ad hoc network [22]. In gossiping, each node propagates a message by forwarding it to a randomly-selected neighbor. The neighbor, in turn, selects another random neighbor for forwarding; thus, the message is propagated recursively. *Anonymous Gossiping* (AG) is built on top of a network-layer multicast routing protocol. It is called anonymous since a gossiping node does not need to know about the group membership. Member nodes implement a bounded buffer to store packets; they can retransmit these packets upon a request from any downstream node.

AG proceeds in two phases. The first phase does regular multicasting of data packets. The second phase provides reliability by recovering packets lost in the first phase. For reliability, each receiver node maintains a list of sequence numbers of lost packets. This node randomly selects a neighbor to gossip and sends a gossip message. Upon receiving this gossip message, if the neighbor is a member of the group (forwarder, receiver, or source) and has those lost packets, it sends back a gossip reply and also the lost packets. Else, it propagates this gossip message further, till lost packets are recovered.

Gossiping provides inspiration to another protocol for providing multicast reliability. Luo et. al. propose *Route Driven Gossiping* (RDG), which also uses gossiping to propagate information about lost packets [23]. Unlike AG, RDG maintains current routes and gossiping is done only with those members for which routes are available. A node joining a group broadcasts a GROUP-REQUEST message, and group members respond by sending a GROUP-REPLY message. These messages help RDG establish

views of the multicast members of the group. Based on this member information, a node sends out a gossip periodically with information about the missed packets. Using gossiping, this message is retransmitted to the receiver that missed the packets. RDG uses two parameters to modify the gossip behavior, namely fanout and quiescence threshold. Fanout refers to the number of neighbors that are selected for gossiping. The quiescence threshold refers to the number of times a particular packet is sent out as a gossip by the same node.

Both of the above approaches address only the problem of reliability but do not determine the appropriate sending rate. In the next section, we discuss two approaches that address both of these problems together.

Congestion Control

Tang et. al. design the *Reliable Adaptive Lightweight Multicast* (RALM) protocol to provide both reliability and congestion control for ad hoc networks [2]. In RALM a source maintains a *Receiver List* to store information about all the receivers. For each window worth of data, the source provides reliability and congestion control by picking a receiver, called the *feedback* receiver, from the Receiver List. Each feedback receiver also sees if it has missed any previous data besides the current data. The feedback receiver replies with an ACK if it receives the current window worth of data and all the previously missed data successfully, otherwise it sends a NACK. When the source receives a NACK, RALM adjusts its sending rate and multicasts the lost packets again. If an ACK is received from a feedback receiver, the transmission is considered complete for that receiver. For the next window worth of data, RALM selects a new feedback receiver for receiving ACK. Since the source specifies the address of the feedback receiver in the packet header, other receivers need not respond.

In addition to reliability, RALM also provides congestion control. Like TCP, RALM uses a window based congestion control scheme. However, the congestion

control for RALM differs from that of TCP in two ways. First, a global window is maintained for all the receivers. Thus, RALM assumes that all the receivers are experiencing identical levels of congestion. Second, only the last packet from the current window needs to be acknowledged.

Rajendran et. al. design another protocol, *Reliable, Adaptive, Congestion Controlled multicasT* (ReACT), which also provides reliability and congestion control [3]. ReACT works by providing reliability at two levels – *local* and *global*. The decision to mark a loss as global or local is done by a receiver, which analyzes the loss pattern. If the loss is persistent, it is considered due to congestion or global otherwise it is considered local. If the loss is considered local, a local recovery is initiated and the source is not informed about it. For recovering lost packets, each receiver maintains a list of upstream nodes from whom local retransmission of lost packets can be made.

However, if the loss pattern is global, the receiver informs the source by sending a NACK. The source maintains a list of all the receivers, who have sent a NACK and their packets have not been transmitted yet. This phase is referred to as congestion control phase. During this phase, ReACT uses a stop-and-wait algorithm to adjust the sending rate; when congestion occurs, the source does not send any new data. Instead, it attempts to deliver lost data to receivers present in the above mentioned list. Once the source leaves the congestion control phase, it reverts to its original sending rate.

2.3.2 Multicast Transport in the Internet

Having discussed multicast protocols for ad hoc networks, in this section, we discuss some of the important Internet based multicast routing protocols.

Reliable Multicast

Scalable Reliable Multicasting (SRM) is one of the leading solutions for a reliable multicast protocol [24]. One of the main strengths of SRM lies in its scalability, which enables it to scale to large networks and large sessions. SRM provides reliability by retransmitting lost packets. If a member fails to receive a multicast packet, it sends a negative acknowledgment (NACK) to the group. The sender or any other member can re-send the lost packet.

Since there can be many multicast members that can request and retransmit the lost packets, SRM uses request and repair timers to achieve scalability. When a member loses a packet, it sets a repair timer and listens for NACKs for that packet. If another member sends a NACK before the timer expires, other members suppress their own. This NACK message is sent by the node for which the request timer expires first. The retransmission is done as a multicast and all the receivers will receive the retransmitted packet. Similar to the request timer, SRM also supports a repair timer. When members receive a NACK request timer and have the packet requested, they set a repair timer. The packet is sent by the node for which the repair timer expires first; the rest of the nodes hear this repair and suppress their own repair timers.

Besides SRM, RMTP is another approach for providing reliability in the Internet [25]. Unlike SRM, RMTP uses a hierarchical approach to provide scalability. A receiver sends a selective ACK with a bitmap, which contains information for missed packets. The problem of ACK implosion[2] is avoided by the hierarchical design. Each group of receivers is divided into a region, which is headed by a Designated Receiver (DR). The ACKs are sent periodically to the respective DR, which can in turn, send these ACKs to its own DR; the source being the root DR. DRs buffer received data

[2]When a lot of nodes send ACK at the same time, this can overwhelm the receiver of the ACK. This is referred as an ACK implosion

and can respond to retransmission requests of the receivers in their corresponding local regions. This decreases the end-to-end latency and improves resource usage.

Congestion Control

Providing congestion control for multicasting in the Internet can be both source-based and receiver-based. In a source-based approach, the responsibility of providing congestion control lies with the source. However, source based rate-adaption schemes can scale poorly when the number of receivers increases. Hence, these schemes must solicit feedback from the multicast receivers in a scalable fashion [26, 27, 28]. In receiver-based approaches, it is the receiver that is responsible for adjusting the rate. These approaches may not require a sender to maintain per-receiver information, making them more scalable [29].

Bhattacharya et. al. use a source-based approach for providing congestion control [27]. They argue that as the number of receivers increases, the likelihood of a receiver reporting a loss increases. If the source were to reduce its rate for all such loss notifications, the performance would severely degrade. Further, it is important that all the receivers should get a fair share of the bandwidth. Hence, they propose that the source should regulate its rate according to the receiver that has the smallest bandwidth.

Rhee et al. propose Multicast TCP (MTCP), a scalable approach for providing congestion control and reliability when there is large number of receivers [28]. Like RMTP, they use a hierarchical style of congestion control, where designated multicast tree members take the responsibility of providing reliability to their local downstream members. Each of these designated intermediate routers store the packets that they receive from the upstream sender. Once it receives an ACK from all the downstream receives, the designated node deletes the copy from its buffer. On the other hand, if it receives a NACK, then it unicasts the packet to that receiver. Thus, the source

is relieved from the burden of retransmitting lost packets. Further, this information is also used for congestion control. These designated nodes keep track of how many packets are outstanding and send this count along with the size of their congestion window as a congestion report. The source compares all of these congestion reports and uses the minimum rate to transmit data.

Receiver-driven Layered Multicast (RLM) is one of the novel solutions for congestion control for multicast [30]. RLM is designed for applications using layered encoding for a video source, and each layer is sent to a separate group. Congestion control is achieved by having receivers add and drop layers. The basic idea for congestion control is that on congestion, drop a layer and when there is spare capacity, add the next layer. Congestion control is also facilitated by the use of join experiments. Each member maintains a separate join timer for each layer. If the timer expires without congestion, that layer is added. If congestion happens, that layer is dropped and the duration of the timer is doubled. When other members observe that a particular member has dropped a layer due to congestion, they use this information rather than joining themselves and noticing the congestion. Thus, everyone learns from the outcome of join experiments of their neighbors.

Like RLM, Vicisano et. al. also propose another receiver based layered encoding solution for multicasting. Receivers join and leave groups, which represent each layer, as congestion decreases and increases respectively. However, each receiver anonymously takes its decision to join or leave groups; authors provide techniques so that all receivers that are facing the same bandwidth bottleneck in any subtree region, can use synchronization among themselves for joining and leaving a layer. This approach differs from RLM in that receivers do synchronization without any explicit group membership protocols and their congestion control mechanism is more fair to nearby TCP flows.

2.3.3 Application Layer Multicasting

An alternative approach to providing reliability and congestion control is to use application layer multicast. The key idea of application layer multicast is to build a virtual network of TCP connections between members. By using TCP, this solution provides reliability and congestion control without any additional cost. Application layer multicast is also a necessity since IP Multicast has not been deployed fully.

However, application layer multicasting pays a price for the benefit of using underlying TCP connections. Since connections are setup at application layer, members often do not use the shortest path, which is usually available when one uses network layer routing. Thus, with relative to network layer multicasting, application layer multicast can have longer paths; this inefficiency is referred to as stress. Further, due to longer path, application layer also incurs higher delay, an inefficiency referred to as stretch.

Narada

Narada is the first proposed protocol for application layer multicasting [4]. Narada is a self-organizing overlay structure connecting all the multicast members in the Internet. The overlay nature comes from the fact that each of the members belonging to the group are connected using a unicast path. In Narada, the construction of the overlay is achieved in two steps. In the first step, a mesh is generated, which connects all the members of the group. In the second step, a variant of a standard distance vector routing protocol is used to construct a shortest-path tree on the top of this mesh.

Bayeux

Zhuang et. al. built Bayeux, which is a large-scale application layer multicast protocol for streaming multimedia applications [31]. Bayeux is built using the peer-to-peer

architecture provided by Tapestry [32]. Bayeux scales well to a large number of receivers, incurs minimum delay and bandwidth overhead, and gracefully handles faults due to routing changes. Multicast members become nodes in the Tapestry overlay and Bayeux builds a multicast distribution tree on top of this Tapestry overlay. Tapestry uses longest prefix match for routing messages from the sender to the destination node. Furthermore, the design of Bayeux incorporates load-balancing into it so that network bandwidth is used efficiently. However, because Bayeux is an application layer multicast, it still suffers from the problems of stress and stretch.

Multicast Overlay using CAN

Similar to Bayeux, Ratnasamy et. al. propose a multicast application layer overlay using the peer-to-peer structure, Content-Addressable Networks, CAN [33, 34]. As is the case with Bayeux, building a multicast application on top of an existing overlay amounts to minimum additional effort, a trend usually true for application layer overlays. However, unlike Bayeux, which builds a source-specific tree for each multicast sender, this approach builds a single tree for all members to share.

2.4 Cross Layer Information Sharing

Ad hoc networks must cope with mobility and limited battery life. One of the likely ways to overcome these limitations is to exploit cross-layer feedback mechanisms. In the Internet, with ossification of the networks at the core, more and more emphasis is being given to designing application layer protocols; it is difficult to build mechanisms that would help various layers interact in the Internet. Because wireless networks are still in the deployment phase, exploring cross-layer mechanisms presents a good opportunity to optimize their performance. In this section, we present some of this work, which aims to use cross-layer information sharing.

Raisinghani and Iyer describe an architecture for cross-layer sharing [35]. They list information at each layer that can potentially prove advantageous for other layers. They consider the user to be the top-most layer and other layers benefit from information provided by the user. Besides user constraints, they believe that battery life is another most important motivation for building a cross-layer scheme. The authors discuss a preliminary architecture where each device has a Device Management Entity, DME (similar to a vertical cross-layer spanning across all the layers). Each layer has its own Layer Management Entity, LME, storing all the relevant information about that layer.

Akyildiz et al. design *Adaptnet*, which consists of protocol solutions at different layers of the protocol stack addressing several problems like rate adaptation, congestion control, and mobility support [36]. At the transport layer, Adaptnet uses a transport layer protocol, Radial Reception Control Protocol, R^2CP that has an adaptive congestion control mechanism. R^2CP monitors the network loss rate and delay at lower layers and adjusts its sending rate at the transport layer. At the link layer, they design an A-MAC [adaptive-MAC], which integrates seamlessly over heterogeneous networks (TDMA/CDMA/widebandCDMA/CSMA). A-MAC is capable of providing various flows with different QoS requirements on various links simultaneously. The link layer also has an adaptive error correcting system. At the application layer, the sending of the video rate is adapted according to the data rate available by different (heterogeneous) technologies at the MAC layer.

W. Yuen et al. use information at the radio layer to enhance the route discovery mechanism in DSR [37]. At the core of their solution lies the fact that SNR (signal to noise ratio) information provided by the radio layer can be used to estimate achievable transmission rate in the physical medium. When a station receives an RTS (Request To Send), it sends the value of this transmission rate along with the CTS (Clear To Send) packet. The (data) sending station, instead of transmitting at a standard rate,

transmits data packets at the rate provided by the CTS header. Other nodes, by looking at the CTS packet become aware that the transmission rate for data packets is different. They use the new transmission rate provided in the CTS header in order to adjust their NAV[3]. In DSR, when a node sends RREQ to find new routes, nodes that have routes, send back RREP. Based on the transmission rate at each hop, message delay is successively appended in the RREP message. At the node, which originated RREQ messages, all the RREP messages are gathered and the one with the least delay is selected rather than the one with the shortest path.

2.5 Conclusion

In this chapter, we present important research related to providing routing, reliability and congestion control for multicasting in ad hoc networks. Most of the research related to multicasting in ad hoc networks seems to draw inspiration from the Internet. Hence, to gain more insight, we present similar work done for the case of Internet. TCP has been extensively used to provide the services of reliability and congestion control.

In spite of previous work done in ad hoc networks, the problem of providing an efficient transport architecture for multicast is still an open problem. Given the nature of ad hoc networks, where each node can be in a heterogeneous neighborhood, it is advantageous to design a transport protocol that can adjust the rate for each group member. To make this work, we also need to design a state-setup protocol to establish transport connections at each hop and an algorithm for distinguishing mobility losses from congestion losses. We believe that these components can work together to provide efficient multicast routing and transport for ad hoc wireless networks.

[3]NAV stands for Network Allocation Vector. NAV is the time for which other nodes that do not participate in the transmission must wait (in addition to their current back-offs) before they can access the medium again

Chapter 3

Motivation: Performance Study of Multicast Routing Protocols

This chapter was published as "A Scenario-Based Performance Evaluation of Multicast Routing Protocols for Ad Hoc Networks", by Manoj Pandey and Daniel Zappala, IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), June 2005,Italy, Pages 31-41.

Current ad hoc multicast routing protocols have been designed to build and maintain a tree or mesh in the face of a mobile environment, with fast reaction to network changes in order to minimize packet loss. However, the performance of these protocols has not been adequately examined under realistic scenarios. Existing performance studies generally use a single, simple mobility model, with low density and often very low traffic rates. In this paper we explore the performance of ad hoc multicast routing protocols under scenarios that include realistic mobility patterns, high density and high traffic load. We use these scenarios to identify cases where existing protocols can improve their performance. Based on our observations, we make a series of recommendations for designers of multicast protocols.

3.1 Introduction

Mobile ad hoc networks have numerous practical applications, such as emergency and relief operations, military exercises and combat situations, and conference or class-

room meetings. Each of these applications can potentially involve different *scenarios*, with movement pattern, density and traffic rate dependent on the environment and the nature of the interactions among the participants. For example, in a search-and-rescue operation, individuals may fan out to search a wide area, resulting in a fairly regular pattern of movement, low density, and low traffic rate. In a battlefield scenario, the movement of soldiers may be heavily influenced by the movements of their commander, with higher density and a higher traffic rate. In other cases, the environment itself may give rise to movement patterns and density, such as patrons visiting an exhibit hall and moving among a selected group of displays. In addition, depending upon the communication need, applications can be very demanding, requiring the system to support very high traffic rates.

To enable group communication in these scenarios, a number of ad hoc multicast routing protocols have been proposed [8, 10, 11, 38, 39, 40, 23, 41, 42]. In making the transition from wired to wireless networking, protocol designers have focused on the obvious challenge of designing a multicast routing protocol that can cope with a mobile environment. As a result, the main goal of most ad hoc multicast protocols is to build and maintain a multicast tree or mesh in the face of a mobile environment, with a fast reaction to network changes so that packet loss is minimized.

While most ad hoc multicast protocols have met this basic design goal, their performance has not been adequately examined under realistic scenarios. Existing performance studies in this area suffer from three common flaws:

- *Simplistic mobility models.* Existing studies use either a Uniform mobility model [43] or the Random Waypoint model [10, 11, 38]. It is well known that because these models utilize random, independent movements they do not reflect realistic usage patterns. A number of researchers have evaluated unicast routing performance under a variety of mobility patterns [44, 45, 46], using more elaborate models and metrics to capture their effect on routing performance. Our

study is the first to apply this methodology to examine multicast routing performance.

- *Low density.* Many evaluations use only 50 mobile nodes in a $1000m^2$ square field and a $250m$ radio range, which often leads to a density of less than 10 nodes within radio range [43, 38, 39, 40]. However, there are many common scenarios in which a network may have many more users in a small area – any situation in which there is a planned gathering or a crowd. This density will likely result in congestion and packet loss; because some protocols use packet loss as an indicator for mobility they may react in precisely the wrong way. Moreover, multicast protocols are often built on top of broadcast, such as for source discovery or tree repair, and these mechanisms will cause excessive overhead in high density scenarios.
- *Low traffic load.* Current evaluations generally employ a very low data rate of 2 to 20 Kbps [10, 39, 23, 41, 42], with only a few using rates as high as 80 to 200 Kbps [38, 47, 8]. Clearly, a protocol that performs very well at lower traffic rates may not be able to handle higher traffic rates efficiently. In addition, while some theoretical work has examined the capacity of ad hoc networks [48, 49], no one has examined the fundamental question of whether multicast routing protocols can actually reach these limits.

In our study, we try to rectify these shortcomings by examining the performance of two common multicast routing protocols – ODMRP [8] and ADMR [10] – under scenarios that include *realistic mobility patterns*, *high density*, and *high traffic load*. We use a simulation-based performance evaluation so that we can thoroughly examine protocol behavior in a controlled environment. Our goal in this work is to impact how future multicast routing protocols are designed by identifying general cases where existing protocols can improve their performance. Accordingly, we

identify specific mechanisms that cause performance bottlenecks, then generalize our experiences into a set of recommendations for multicast protocol designers.

We first explore multicast routing performance using realistic mobility models, similar to those described in the IMPORTANT framework [46] and other unicast evaluations. To the best of our knowledge, this is the first time that multicast routing protocols have been examined over a wide range of mobility patterns. Our goal is to sort through the overwhelming number of models and metrics and identify the key areas where protocol designers should focus their attention.

In studying this scenario, we show that multicast performance is largely predicted by two key mobility metrics – the frequency of link breaks and the density of the mobility pattern. This enables protocol designers to focus on optimizing performance with respect to these metrics, regardless of the particular usage scenario. We find that ODMRP does not react quickly to link breaks because it must rely on a periodic broadcast from the source in order to find members that have moved. ADMR reacts more quickly, but at the cost of much higher transmission and control overhead. There is certainly room for a protocol that achieves better tradeoffs.

Our second scenario examines multicast performance under high density, which we believe will be a common case for ad hoc networks. While ODMRP handles this scenario very well, ADMR throughput is severely impacted by density. In this scenario, ADMR experiences congestion collapse at about 25 Kbps, performing even worse than flooding, while ODMRP can achieve rates in excess of 200 Kbps. We analyze ADMR's poor performance and show that it is due to its use of explicit acknowledgments and its assumption that all packet loss is a sign of mobility. We design several modifications of ADMR that correct this problem and dramatically increase throughput in this scenario without affecting its performance in other situations.

Our last scenario examines multicast performance under high traffic load. We find that ODMRP performs significantly better than ADMR, but both experience

high packet loss as the traffic rate increases. Packet size plays an important role in reaching high capacity – sending fewer large packets is generally better than sending many small packets. We then extend recent theoretical results for the capacity of ad hoc networks to the case where multicast traffic is transmitted, and find that the number of hops in the multicast tree is a critical factor. We argue that for this scenario it is better to use a multicast tree rather than a mesh.

3.2 Background

We have chosen to study the performance of ODMRP and ADMR in these scenarios because they operate *on-demand* rather than proactively maintaining routes. Several performance studies indicate that these protocols perform well [43, 10]. In addition, ODMRP is mesh-based whereas ADMR is tree-based, providing us a perspective on both types of protocols.

3.2.1 ODMRP

ODMRP is a mesh-based demand-driven multicast protocol, similar to DVMRP [7] for wired networks. A source periodically builds a multicast tree for a group by flooding a control packet throughout the network. Nodes that are members of the group respond to the flood and join the tree. Nodes that are on the tree use *soft state*, meaning their status as forwarders for a given group times out if not refreshed. Because the forwarding state is shared among all sources for a given group, the set of forwarders for a group forms a mesh, providing robustness for the mobile receivers.

In particular, an active ODMRP source periodically floods a JOIN QUERY message throughout the entire network. Each node receiving this message stores the previous hop from which it received the message. When a group member receives the JOIN QUERY, it responds by sending a JOIN REPLY to the source, following the previous hop stored at each node. Nodes that forward a JOIN REPLY create soft

forwarding state for the group, which must be renewed by subsequent JOIN REPLY messages. If the node is already an established forwarding member for that group, then it suppresses any further JOIN REPLY forwarding in order to reduce channel overhead.

The basic trade-off in ODMRP is between throughput and overhead. A source can increase throughput by sending more frequent JOIN QUERY messages. Each message rebuilds the multicast mesh, repairing any breaks that have occurred since the last query, thus increasing the chance for subsequent packets to be delivered correctly. However, because each query is flooded, increasing the query rate also increases the overhead of the protocol. ODMRP can also control redundancy via the soft-state timer for node forwarding state. A longer timer will increase the size of the mesh and hence provide more redundant paths for packets to be delivered. Of course, increasing the soft-state timer also increases overhead since many of the links in the mesh will result in duplicate packets being delivered.

3.2.2 ADMR

ADMR creates source-specific multicast trees, using an on-demand mechanism that only creates a tree if there is at least one source and one receiver active for the group. Unlike ODMRP, receivers must explicitly join a multicast group. Sources periodically send a network-wide flood, but only at a very low rate in order to recover from network partitions. In addition, forwarding nodes in the multicast tree may monitor the packet forwarding rate to determine when the tree has broken or the source has become silent. If a link has broken, a node can initiate a repair on its own, and if the source has stopped sending then any forwarding state is silently removed. Receivers likewise monitor the packet reception rate and can re-join the multicast tree if intermediate nodes have been unable to reconnect the tree.

To join a multicast group, an ADMR receiver floods a Multicast Solicitation message throughout the network. When a source receives this message, it responds by sending a unicast Keep-Alive message to that receiver, confirming that the receiver can join that source. The receiver responds to the Keep-Alive by sending a Receiver Join along the reverse path.

In addition to the receiver's join mechanism, a source periodically sends a network-wide flood of a Receiver Discovery message. Receivers that get this message respond to it with a Receiver Join if they are not already connected to the multicast tree.

Each node begins a repair process if it misses a defined threshold of consecutive packets (2 for our simulations). Receivers do a repair by broadcasting a new Multicast Solicitation message. Nodes on the multicast tree send a Repair Notification message down its subtree to cancel the repair of downstream nodes. The most upstream node transmits a hop-limited flood of a Reconnect message. Any forwarder receiving this message forwards the Reconnect up the multicast tree to the source. The source in return responds to the Reconnect by sending a Reconnect Reply as a unicast message that follows the path of the Reconnect back to the repairing node.

Nodes on the multicast tree also maintain their forwarding state. They expect to receive either passive acknowledgments (if a downstream node forwards the packet) or an Explicit Acknowledgment if it is a last hop router in the tree. If a defined threshold of consecutive acks are missed (15 for our simulations), then the forwarding node expires its state.

Finally, a receiver keeps track of how many times it has had to initiate a repair due to a disconnection timeout. If this number reaches a certain threshold then the receiver asks the source to switch to flooding in its next Receiver Join message. If enough receivers ask, then the source switches to flooding for a limited time. During

flooding, all the data packets are sent as network-wide flood and all repair messages are suppressed.

3.3 Simulation Methodology

For our simulations we use GloMoSim-2.03 [50]. We fixed the ODMRP implementation provided in the GloMoSim distribution because it contained several major bugs that prevented packets from being delivered. Since GloMoSim did not include ADMR, we wrote our own implementation based on the original ADMR publication [10] and a specification published as an Internet draft [51]. We did not implement source pruning (where the source stops sending data if there are no receivers) so that we could study the effects of partitioning on packet loss. We also wrote a simple flooding protocol for GloMoSim.

One important implementation detail for multicast routing protocols is the use of randomization (or jitter) to avoid collisions due to protocol synchronization. Each node that forwards a multicast message adds a random delay between 0 and 10 ms before forwarding the packet. Likewise, at the application layer, we avoid starting sources at the same time, since they use CBR and would thus remain synchronized for the duration of the simulation. Finally, for ADMR we exclude any startup delay caused by buffering packets before sufficient receivers have joined the group.

To verify our protocol implementations, we ran simulations identical to those reported in [10] that compare ADMR and ODMRP. Our results are very close, with slightly higher delay due to the jitter we have added at each node. The results reported in this paper differ more substantially from [10] because we are using a different field size.

In our simulations we collect the following metrics:

- **Throughput (%)**: The ratio of the number of packets received to the number of packets sent.
- **Throughput (rate)**: The rate at which group members receive data, in kilobits per second.
- **Transmission Overhead**: The ratio of the number of data messages transmitted (originated or forwarded) and the number of data messages received. This metric is a measure of the efficiency of a routing protocol – a lower value for transmission overhead indicates that fewer forwarders were needed.
- **Control Overhead**: The ratio of the number of control messages originated or forwarded over the combined total of data and control messages originated or forwarded. This metric indicates the percentage of all messages that are control messages.
- **Delay**: The difference between the time when the packet is sent by the source and when it is received.

Note that previous studies have combined transmission overhead and control overhead into a single metric called normalized overhead, but this can obscure the differences between the two.

For ODMRP, control packets consist of Join Query, Join Reply, and Acknowledgments. For ADMR, control packets consist of Receiver Discovery, Multicast Solicitation, Keep-Alive, Receiver Join, Repair Notification, Reconnect, Reconnect Reply, and also Acknowledgments sent by the end receivers in order to maintain the tree. Since ODMRP Join Queries and ADMR Receiver Discovery messages have data piggybacked with them, we count these packets as both data and control messages.

Except where noted, our simulations use 50 nodes, randomly placed over a square field whose size is $1000m^2$. Nodes communicate using IEEE 802.11 for the

MAC protocol, with a $250m$ radio range, free-space radio signal propagation, and a maximum data rate of 2 Mbps.

3.4 Mobility Scenarios

Our first scenario explores the effects of realistic mobility patterns on multicast routing performance. We use mobility models similar to the IMPORTANT framework [46], but apply this methodology for the first time to multicast. We consider a wide range of mobility models and explain how they impact multicast routing performance. As a result, we are able to identify two key mobility metrics that predict multicast performance. This enables protocol designers to focus on optimizing performance with respect to these metrics, regardless of the particular usage scenario.

3.4.1 Mobility models and connectivity metrics

A number of researchers have developed mobility models to capture the movement patterns of wireless network users. We have chosen a set of models from three different classes of motion – random, path-based, and group-based movements:

- *Uniform*: Each node starts at a random position and moves in a random direction with a constant velocity [43]. This models independent movement with high temporal dependency.
- *Random Waypoint*: Each node chooses a random destination within the simulated field, moves to the destination at a randomly chosen speed, pauses for a fixed period of time, and then chooses a new destination. This model has been widely used in the literature, allowing us to validate our results with other research.
- *Manhattan*: Each node moves along a set of pre-defined streets, which are arranged in a grid pattern. All nodes use the same speed, and each node may

Model	Parameters
Random Waypoint	speed: between max and $max - 5m/s$ pause time: 30 seconds
Manhattan	grid size: 150 meters
Exhibition	centers: 10 minimum distance to center: 20 meters pause time: 30 seconds
Battlefield	leaders: 16 minimum distance to leader: 20 meters pause time for leader: 30 seconds

Table 3.1: Parameters used in Mobility Models

choose any direction when reaching an intersection. This models path-based motion with low spatial dependence [46, 52].

- *Exhibition*: Each node chooses a destination from among a fixed set of exhibition centers and then moves toward that center with a fixed speed. Once a node is within a certain distance of the center it pauses for a given time and then chooses a new center. This model is similar to the event scenario described by Johansson et al. [44] and represents independent movement but with high node density.

- *Battlefield*: Each node follows a group leader by choosing a destination close to where the leader is currently located and then moving there. The group leader uses the Random Waypoint model. This is similar to the RPGM model [45] and represents group-based mobility with high spatial dependence.

Table 3.1 lists the default parameters we use for each of these models. For each of these models we vary the speed at which nodes move. For all models we avoid sharp turns and sudden changes in velocity by using acceleration and deceleration vectors [53]. As part of this work we extended GloMoSim to include the Uniform, Manhattan, Exhibition, and Battlefield mobility models.

To differentiate among these models, we use a set of mobility and connectivity metrics. Of the mobility metrics defined in the IMPORTANT framework [46],

we found that spatial dependency and the average number of link changes are able to differentiate between our mobility models and help to explain multicast routing performance. These are defined informally as:

- *Spatial dependency:* The degree to which two nodes are moving in a similar direction with similar speed.
- *Number of Link Changes:* The number of link changes seen during the course of a simulation. A link change is counted when two nodes come within radio range when previously they had not been able to communicate directly, and vice versa.

In addition, because we are studying multicast routing, we introduce two new metrics:

- *Neighbor Density:* The number of nodes which are within radio range of a given node. We do not distinguish between active and inactive nodes, because with multicast a node that is not transmitting its own data can still act as a forwarder.
- *Reachability:* The number of nodes that are reachable via forwarding through the ad hoc network. We measure this using a recursive coloring algorithm, so that nodes in the same partition have the same color.

We used a set of simulations to confirm that each of these metrics is able to distinguish between the mobility models. Of the most importance to us, the number of link changes clearly differentiates among the models (Figure 3.1). This behavior was not seen with the IMPORTANT framework [46], but it is significant because the number of link changes can have a significant impact on a multicast routing protocol – each link change may potentially break the multicast tree or mesh and cause a receiver or intermediate node to attempt a repair. Both spatial dependency and

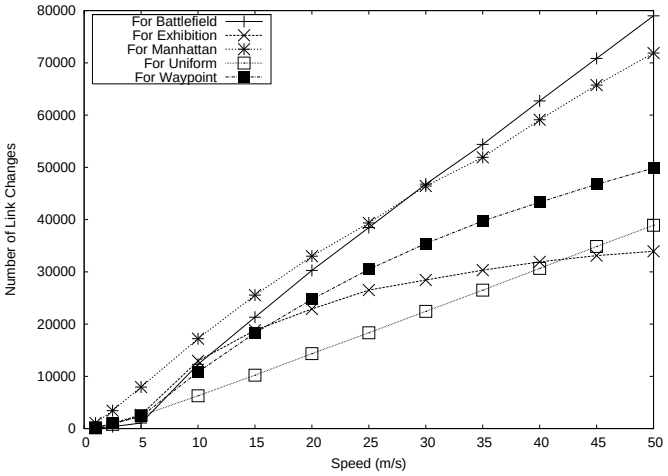


Figure 3.1: Mobility model link changes

density are higher for the group-based mobility models (Exhibition and Manhattan), as is expected, though the difference is clearer with density (Figure 3.2). Note that density is initially low for Battlefield and Exhibition because at low speeds the nodes do not have time to form groups. Another significant difference among the protocols is that all of them maintain high reachability except for Battlefield [54]. This means that the Battlefield model is a good test for determining how well a protocol reacts to a partition in the network.

Mobility metrics explain routing performance

For the mobility scenarios, we try to isolate the pattern of node movement from other factors, such as the traffic rate. Hence we use only three multicast groups, each consisting of 1 source and 7 receivers. The multicast groups are not overlapping, which means that with the 3 senders and 21 receivers combined about half of the nodes are participating in a multicast session. Each multicast source uses a Constant Bit Rate (CBR) flow, transmitting a 64 byte packet every 250 milliseconds (6 Kbps). We run each simulation for 600 seconds and we average the results of 25 simulations.

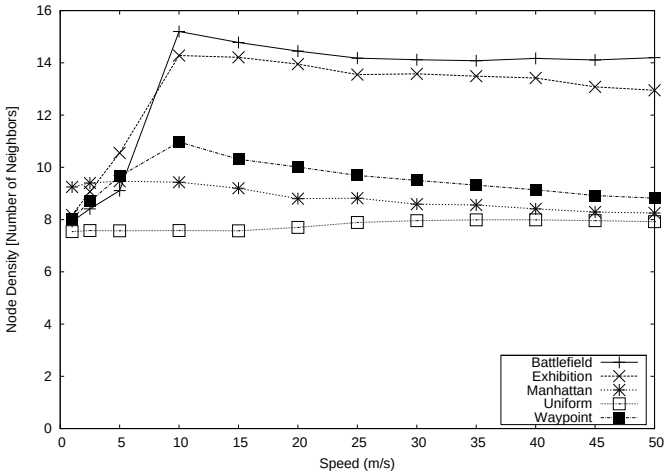


Figure 3.2: Mobility model density

For both ODMRP and ADMR, throughput depends on the model (Figures 3.3 and 3.4) and the ordering from worst to best can be explained by both the number of link changes and the density. Throughput is worst for Battlefield because it creates the most link changes, to the point that the network frequently becomes partitioned. At moderate to high speeds, approximately 10 nodes are separated from the rest of the network with the Battlefield model. The Manhattan model has a similar large number of link changes and also low density, so it is the next worst. Uniform and Random Waypoint result in similar performance; although Uniform has fewer link changes this is balanced by Random Waypoint having higher density. Finally, the Exhibition model results in the best performance because the number of link changes is relatively low and it creates high density.

Note that for ODMRP our results show lower throughput than with previous ODMRP simulations [43] because these previous simulations are at low speed ($20m/s$), use a lower data rate, and try only the Uniform mobility model. Likewise, we show lower throughput for ADMR than with previous ADMR simulations [10] because the previous simulations use a elongated field ($1500m\ x\ 300m$) that results in higher density and are limited to the Random Waypoint model.

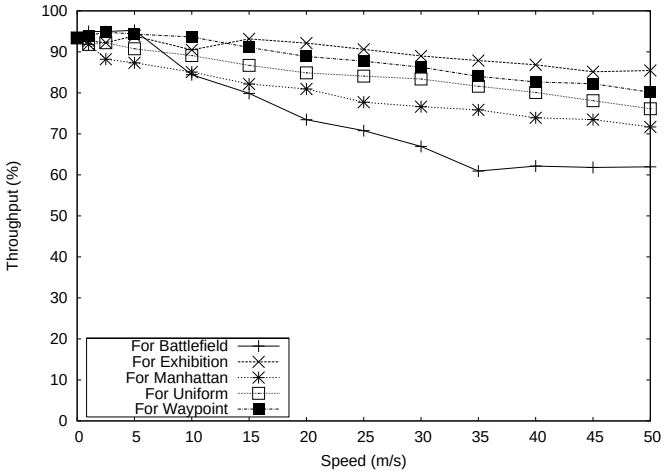


Figure 3.3: ODMRP throughput

Regardless of the mobility model, ODMRP performance degrades as speed increases, whereas ADMR is able to maintain throughput greater than 80%. Based on our results, we calculated Pearson's correlation coefficient and confirmed that there is a strong negative linear relationship between the number of link changes and throughput for ODMRP. Note that ODMRP could obtain better performance by sending Join Query messages more frequently, and hence reacting to broken links more quickly, but this would in turn incur more overhead. Recent revisions of ODMRP use GPS to track location and adaptively increase the refresh interval as mobility increases. ADMR is able to maintain high throughput because (a) forwarding nodes are able to initiate local repair of the multicast tree and (b) receivers experiencing high packet loss can ask ADMR to switch to flooding. However, the cost of these actions are higher control and transmission overhead.

One of our important findings is that the density created by each of the mobility models explains the performance of the rest of our performance metrics. For both ODMRP and ADMR, the transmission overhead, control overhead, and delay varies according to the mobility model, with the ordering among the models correlates exactly with the ordering for density. We show representative graphs for ADMR

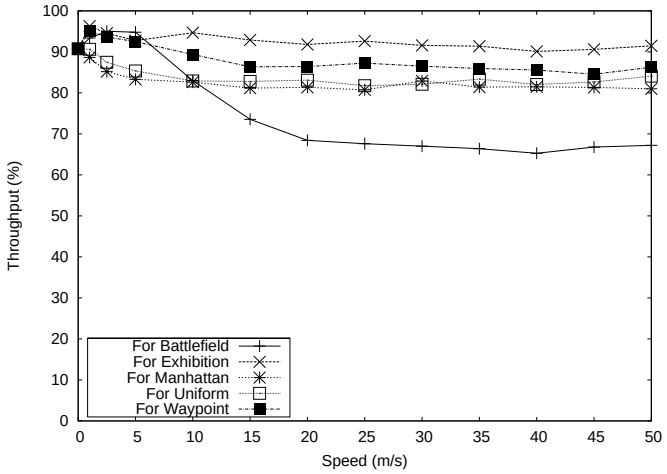


Figure 3.4: ADMR throughput

in Figures 3.5 and 3.6. Group-based mobility models, which lead to higher density, result in a greater chance that multicast group members will be located near the source. This leads to a savings in transmission overhead and delay. High density also decreases control overhead for ODMRP, since JOIN REPLY messages travel fewer hops. For ADMR, however, control overhead increases with density (Figure 3.6). This happens because ADMR switches to flooding more frequently when density is low [54]; during these times ADMR has no control overhead.

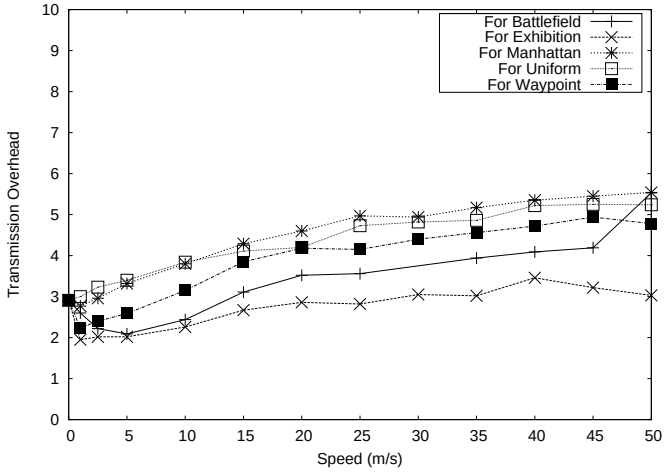


Figure 3.5: ADMR transmission overhead

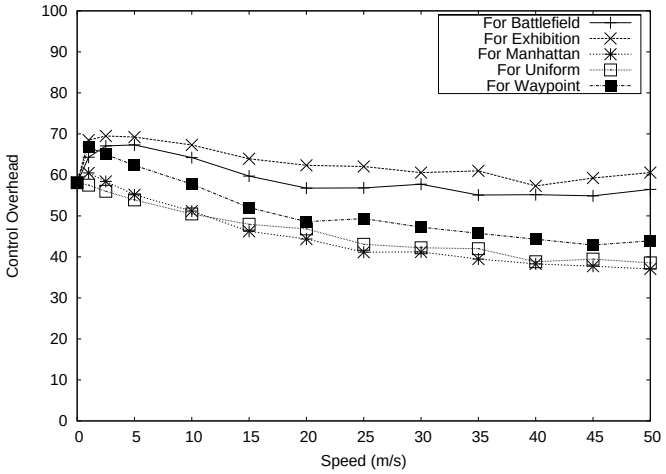


Figure 3.6: ADMR control overhead

These results confirm that characterizing link variations and density fluctuations for any user movement is crucial towards understanding routing performance. We also include reachability as a key metric since partitioning is a special case of a link breaking. Additional metrics defined in the IMPORTANT framework, such as temporal dependence, spatial dependence, and relative velocity, are important only in that they induce link changes and density. It is possible, for example, to construct a model that creates spatial dependence and yet few link changes. Each node in this strawman model simply vibrates in place, with the amplitude of vibration significantly smaller than the radio range. Given sparse placement, density variations are negligible and since movements are small no link breaks occur. However, depending upon the alignment of the vibrations and relative velocities, spatial dependence between two nodes can be varied arbitrarily. Hence, we can conclude that protocol designers should concentrate on optimizing multicast protocols for both frequent mobility and density.

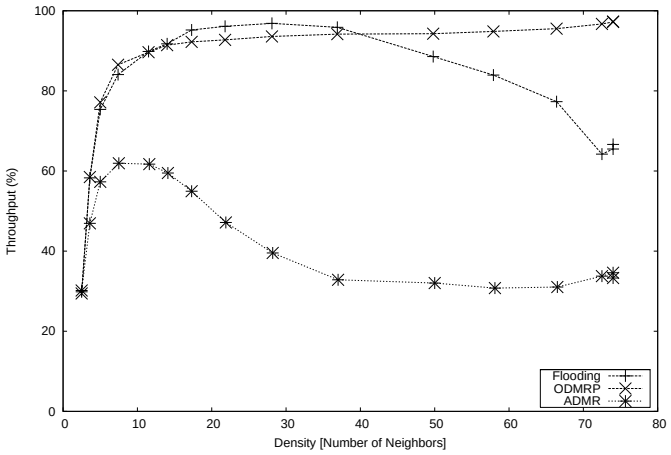


Figure 3.7: Variation of throughput with density

3.5 High Density Scenario

Our second scenario explores the effects of high density on multicast routing performance. For this scenario, we study density without any mobility. Accordingly, we statically place a set of 75 nodes on a field and vary the density by varying the size of the field. We use three multicast groups, each having 1 source and 22 receivers. To see the effects of density, we increase the traffic rate of each source slightly; each source sends a 100 byte packet every 100 milliseconds (24 Kbps). We run each simulation for 100 seconds and we average the results of 25 simulations.

Surprisingly, ADMR performs very poorly in this scenario (Figure 3.7). Both ODMRP and ADMR initially receive low throughput as the density of the network is so small that it is partitioned. As the density increases, ODMRP achieves very high throughput once the network is connected, while ADMR never delivers more than 60% of the packets. In fact, ADMR does much worse than a simple flooding protocol.[1] This indicates that ADMR's ability to switch to flooding is not causing this problem.

[1]For our implementation of flooding, we use a simple protocol in which each node receiving a packet for a group first checks whether it is a duplicate and, if not, forwards the packet by retransmitting it.

To explore this problem further, we fix the density at a high value and then vary the traffic rate to determine when ADMR encounters a problem. To accomplish this, we randomly place 50 nodes within 20 meters of a central point, again with no mobility. We use only a single multicast group, with 1 sender and 30 receivers. We vary the traffic rate for this source by keeping the packet size fixed at 100 bytes and adjusting the number of packets sent per second. We run each simulation for 60 seconds and we average the results of 25 simulations.

In this scenario, ADMR's throughput begins to collapse when the sending rate is only about 25 Kbps (Figure 3.8). This is even worse than flooding, which must forward each packet 50 times! Because of its simplicity, ODMRP operates extremely well, even at extremely high rates. ODMRP's only control traffic is a periodic Join Query; while this message is flooded, it is transmitted once every 3 seconds, regardless of the sending rate. The subsequent Join Reply messages, one per receiver, form a tree shaped like a star. This means that each data packet is simply broadcasted by the source and then immediately received by all group members.

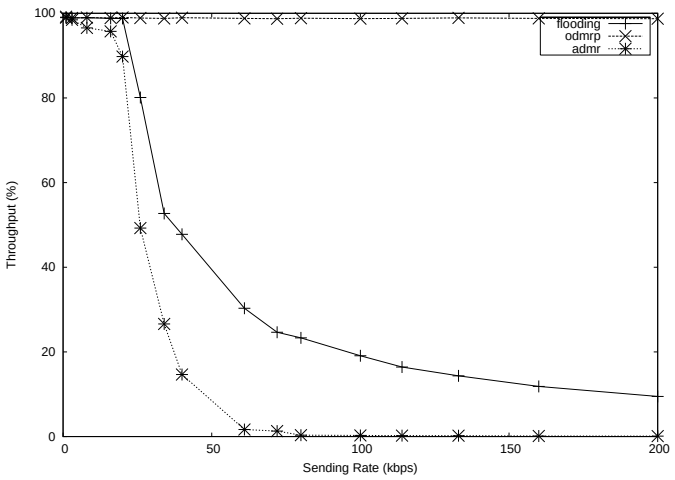


Figure 3.8: High density throughput

To pinpoint ADMR's problems with high density, we plot a trace of the major network activity when the traffic rate is 200 Kbps (Figure 3.9). The graph is divided into three sections, with each section being a different version of ADMR. The bottom

section of this plot shows the standard version of ADMR, before we have fixed any of ADMR's problems under high density. Each symbol on the graph represents a packet being transmitted, with the y-axis indicating which node sent the packet. Node 0 is the sender for the group. To make the trace readable, we show only the first 7 seconds of network activity; the rest of the 60 second trace continues with exactly identical patterns to those shown here.

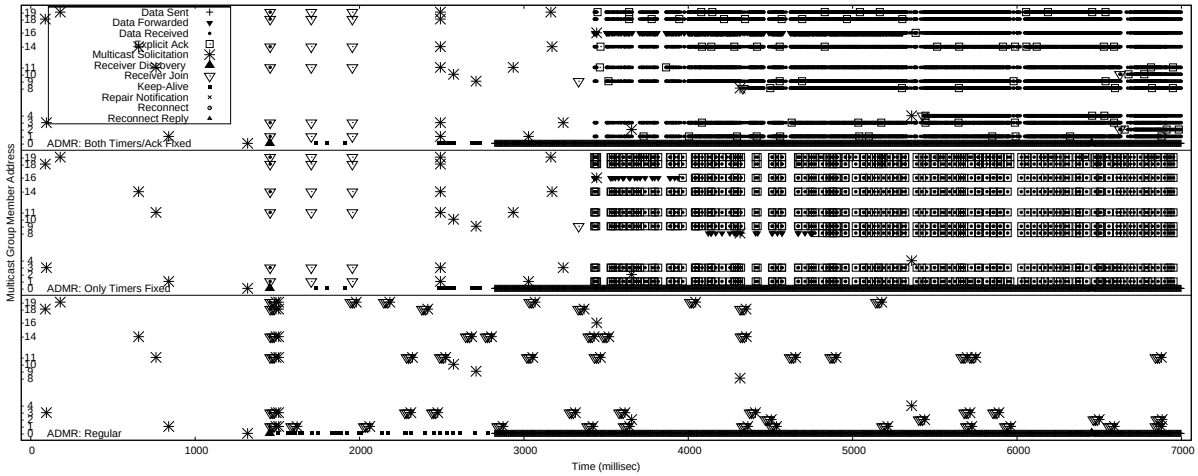


Figure 3.9: High density ADMR packet trace

The first problem evident from this trace is that ADMR suffers from Receiver Join implosion. When the source starts at approximately 1300 ms, it transmits a Receiver Discovery message. All existing group members immediately respond with a Receiver Join message, which overwhelms the source. The source responds with a separate Unicast Keep-Alive message for each receiver, but these messages are delayed due to congestion.

However, the primary problem in this case is that ADMR sets a timer for each Receiver Join message that is based on the inter-packet gap advertised by the source. Because the inter-packet gap is so small in this simulation (4 ms), the timer fires faster than the source can respond and ADMR assumes the join attempt has failed. Hence, when the source sends its Receiver Discovery message, each receiver sends three Receiver Join messages, then gives up and sends a Multicast

SOLICITATION message. Eventually, the source delivers a UNICAST KEEP-ALIVE message to a receiver, and this causes another round of triple RECEIVER JOIN messages followed by solicitations. Because the solicitations are flooded and sent using broadcast at the MAC layer, they cause severe congestion; 10 solicitations in a second result in 500 control packets transmitted during that time. This congestion forces the source to continually backoff. Hence, even though the source queues packets in the MAC layer at about 2800 milliseconds, many of them are never sent. Even when packets are sent, few of the group members successfully join for any length of time – the throughput is 0.07%. Later in the trace (not shown), Repair timers have the same problem as the Join timers, resulting in additional MULTICAST SOLICITATION messages.

Fixing this problem requires only a small adjustment to the ADMR Join and Repair timers. We establish a *RepairWaitTime*, so that the calculated timer can never be less than this value. At large inter-packet gaps (low sending rates), the original timer value is used, but at higher sending rates the timers are set to their minimum values. By setting the *RepairWaitTime* to a reasonably short time (e.g. 500 ms), we can ensure that the timer never fires too soon (this is the equivalent of 125 missed packets for our packet trace), but is fast enough to adapt to mobility-induced losses.

The middle section of the packet trace in Figure 3.9 shows network activity for ADMR with the Join and Repair timers fixed, using a value of $500ms$ for the *RepairWaitTime*. This dramatically reduces the number of RECEIVER JOIN messages that are sent, which allows data to be transmitted. Note that most group members can now receive data, though some members that join later have difficulty joining because of the congestion in the network.

This reveals a second serious problem with ADMR in the high density scenario: each receiver transmits explicit acknowledgments to the source, resulting in ack implosion. Like any ADMR forwarder, the source must receive either a passive or

active acknowledgment to maintain its forwarding state. Since the forwarding tree is actually a star at high density, all receivers are the last hop in the tree and all must send an EXPLICIT ACK for each packet. Hence, the throughput for this case improves to only 12.92% as many packets are either delayed or lost due to the congestion from explicit acks. Moreover, some nodes actually become forwarders for a short time, as seen for nodes 8 and 16 in the middle section of the trace. This occurs when an intermediate node receives a MULTICAST SOLICITATION before the source.

Our solution for the ack implosion problem is based on the observation that this problem is very similar to the ack implosion problem in reliable multicast [55]. In the wireless case, a source (or any other forwarder in a dense region) only needs to receive one acknowledgment for a packet in order to maintain its forwarding state. Even better, if the source (or forwarder) can receive one ack for every k packets, then it can maintain its forwarding state with a minimum of overhead.

Damping the explicit acks in ADMR is simple, since each ack is broadcasted. Each group member sets an ack timer to a random value between zero and *MaxAck-Time*. If the timer expires and the member has received data during this interval then it sends an EXPLICIT ACK. However, if the group member hears an EXPLICIT ACK during this interval, it cancels its timer and then waits the remainder of *MaxAckTime* before it sets a new ack timer. The source (or forwarder) sets its forwarding state timer to *AckWaitTime*, which is equal to $m * MaxAckTime$. This ensures that the source does not time out its forwarding state unless it misses m acks in a row.

The top section of the packet trace in Figure 3.9 shows network activity for ADMR with both explicit acks and the Join and Repair timers fixed. We use a value of 66 ms for *MaxAckTime*, 2 seconds for *AckWaitTime*, and $500ms$ for the *RepairWaitTime*. As can be seen from this trace, explicit acks are sent regularly, but at a much reduced rate. This relieves the congestion in the medium, allowing a throughput of 95.86%!

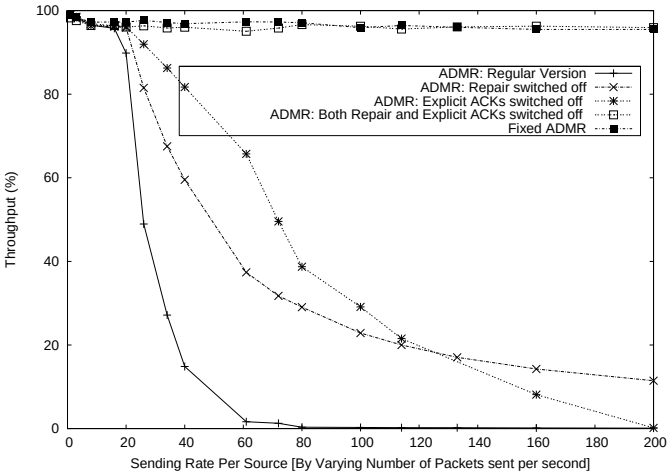


Figure 3.10: ADMR fixed for high density

To verify that our two solutions work, we repeated the simulation of Figure 3.8 using five versions of ADMR: the regular version, ADMR with the Join and Repair timers disabled, ADMR with explicit acks disabled, ADMR with timers and explicit acks disabled, and finally ADMR with our two solutions. As shown in Figure 3.10, our solutions enable ADMR to scale with offered load during the high density scenario. In fact, they work just as well as completely disabling the problematic mechanisms, though this is not a viable alternative.

We do not claim that our solutions enable ADMR to obtain high throughput for all scenarios. We did, however, test our solutions with the mobility scenario in the previous section and found that we were able to obtain nearly identical results in these cases. We do note that ADMR's performance is sensitive to the timer settings we have adjusted. Using larger values for the *RepairWaitTime* cause ADMR to react too slowly to high mobility conditions, and using larger values for *AckWaitTime* cause excessive pruning and low throughput for high mobility conditions.

Our primary purpose in this exercise is to demonstrate the pitfall of testing multicast routing performance in only low density situations. Our experiments with high density have identified several flaws in ADMR that can be generalized to any multicast routing protocol. Explicit acknowledgments should be avoided if possible,

and control messages that are flooded should be rate limited to avoid broadcast storms. While we have implemented changes to ADMR that solve these problems for a single group, dampening these messages across multiple groups is a more difficult problem. Our results also question the practice of building a routing protocol that reacts to packet loss. If the protocol interprets all packet loss as a sign of mobility, then it will misinterpret congestion as a sign of mobility and potentially cause a congestion collapse.

3.6 High Traffic Scenario

Our third scenario explores the effects of high traffic rates on multicast routing performance. To isolate the effects of load from other factors we keep density and mobility low. In these simulations we use the same parameters as the mobility scenario, except we keep the speed constant at 1 m/s. We then vary the traffic rate in two ways: using a fixed packet size of 64 bytes and varying the inter-packet gap, and using a fixed inter-packet gap of 250 ms and varying the packet size. For this scenario use the Exhibition model with 10 centers; results for other mobility models are similar.

Our results indicate that ODMRP is able to achieve a maximum throughput of about 55 Kbps, while ADMR can receive at most 40 Kbps when varying the packet size (Figure 3.11). Both protocols obtain significantly lower throughput when varying the inter-packet gap (Figure 3.12). In both figures we show the average rate at which data is received by the group members, as well as the average transmission rate (at the MAC layer) for each node in the network and the average rate at which packets are received by each node (also at the MAC layer). The data reception rate indicates how successful the routing protocol is; for example, when the packet size is varied, ODMRP receives about 30 Kbps when the sources send at 50 Kbps, or about 60% of the throughput. The rate transmitted per node and the rate received per node indicate the load imposed on the network by the routing protocol. The rate received

is always higher than the rate transmitted because each node has more than one neighbor.

Our second purpose in exploring this scenario is to examine a fundamental question: how effectively can a multicast routing protocol use the available capacity of an ad hoc network? It clear from our results that multicast can use more of the available capacity by transmitting large packets rather than small packets. For example, when varying the packet size ODMRP can receive about 55 Kbps, but can only receive about 20 Kbps when varying the inter-packet gap. But how much of the available capacity can a multicast routing protocol actually use?

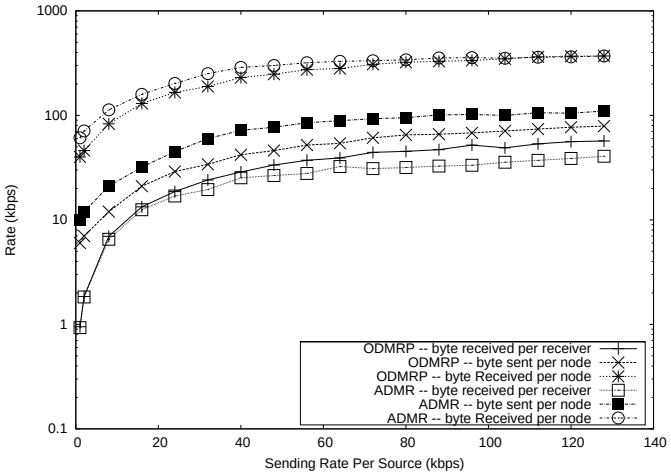


Figure 3.11: High traffic: packet size

From work on the capacity of ad hoc networks [48, 49], we know that the unicast sending rate, λ_u available to a node is bounded by:

$$\lambda_u < \frac{C/n}{\overline{L_u}/r}, \tag{3.1}$$

where C is the capacity of the network, n is the number of nodes that are transmitting, $\overline{L_u}$ is the expected path length in meters, and r is the radio range in meters. Because $\overline{L_u}/r$ is the expected unicast path length in hops, we can translate this equation into

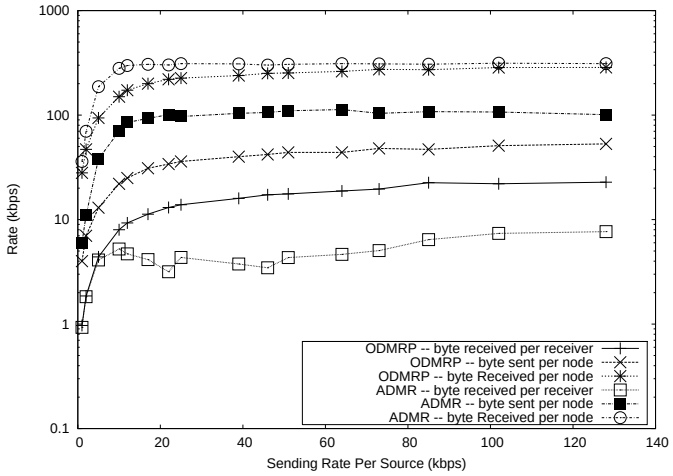


Figure 3.12: High traffic: inter-packet gap

an available sending multicast rate by substituting the expected number of hops in a multicast tree. Hence, for multicast:

$$\lambda_m < \frac{C/n}{\overline{L_m}}, \tag{3.2}$$

where $\overline{L_m}$ is the expected number of hops in the multicast tree. This bound indicates that one of the critical factors for a multicast tree is its efficiency, with a lower L_m enabling a higher transmission rate. This is exactly what the bandwidth efficient multicast protocol [47] tries to accomplish, by having receivers join along the shortest path to the existing tree, rather than using the shortest path to the source. This indicates that in a high traffic scenario it is better for a multicast protocol to use a tree instead of a mesh.

To determine L_m for our scenarios, we ran an experiment that measured this value for both ODMRP and ADMR as we vary the number of group members for a single group with one source (Figure 3.13). In both cases, this measurement excludes any forwarding done by flooding of control or data packets. The average L_m increases slowly for both protocols when the group includes only a single source. This measurement appears to follow the Chuang-Sirbu law, in which $L_m/\overline{L_u} = m^k$, where m

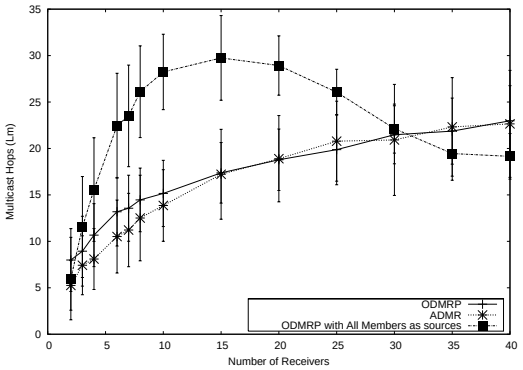


Figure 3.13: L_m as a function of group size

is the number of receivers [56, 57]. This law fits our data when $\overline{L_u}$ is 3 (the measured value for our simulations) and k is 0.58. When we allow all members to be sources for the multicast group, ADMR is unaffected (since it builds a tree per source), but L_m grows significantly for ODMRP since it uses a mesh. The number of links in the mesh falls at very large group sizes for ODMRP, but this is only because the number of nodes and the area of the network is limited. Hence, we would expect ODMRP to not be bandwidth efficient as the number of sources grows.

Returning to Equation 3.2, we can now calculate the sending rate available to a node. We use a measured L_m of 12 for a group size of 7 (Figure 3.13), 2000 Kbps for C [49], and we have 3 active sources. This yields $\lambda_m = 55Kbps$ for ODMRP, which is what it does obtain when using large packets. Since ADMR has a lower L_m, it should be able to achieve a higher sending rate. However, ADMR switches to flooding once packet loss begins to occur, which is exactly the wrong thing to do during a period of congestion. ADMR misinterprets packet loss as a sign of mobility and thus makes the situation even worse. This problem is particularly acute when varying the inter-packet gap, because increasing the sending rate in this case increases the number of packets that are flooded.

It is an open question as to whether other multicast routing protocols better utilize network capacity. We believe a tree-based protocol (with lower overhead than ADMR) could do better than ODMRP, particularly for multiple sources, because its

L_m will be lower. On the other hand, it is possible that the theoretical limits are too high, and that ad hoc networks using IEEE 802.11a are simply unable to support high levels of multicast traffic. We are currently designing a new multicast routing protocol that uses a tree and reliable broadcast at the MAC layer in an attempt to achieve high utilization.

3.7 Conclusion

Our goal in this work has been to identify cases where ad hoc multicast routing protocols can improve their performance, resulting in a set of recommendations to multicast protocol designers.

Our first recommendation is that designers focus on optimizing for both mobility and density. Protocols should react to link breaks, but with lower overhead than ADMR does. Designers should also be aware of the pitfalls associated with high density situations, avoiding problems such as ack implosion and broadcast storms. While these problems have been seen in wired networks, it is apparent that these lessons have not always carried over into wireless networking. Protocols should instead take advantage of the density created by group-based mobility patterns. Recent work by Yi et al. does this by treating groups of users as a team and then multicast a single packet to each team [58].

We strongly recommend that routing protocols should not attempt to monitor packet loss and repair routes when loss is high. As we have seen with ADMR, such loss may in fact be due to congestion and the increased repair traffic can lead to congestion collapse. Rather, loss monitoring should be done by the transport layer, which can then use input from the MAC layer to determine if the loss is due to congestion or mobility. The transport layer can then request that the routing protocol take appropriate action when the loss is due to mobility (and suspend sending any more data until a new route is found).

To achieve high capacity, protocols should use a bandwidth-efficient tree rather than a mesh and should have very low control overhead. Mesh-based multicast protocols increase the number of hops in the multicast tree and hence cannot support high traffic rates. This argues for a simple, end-to-end protocol design, with receivers in charge of joining groups and reacting to route changes. Asking multicast forwarders to maintain the tree results in too much control traffic, particularly when broadcast is used to repair the tree.

Finally, we suggest that future multicast protocol evaluations – both in simulations and in testbeds – need to be more comprehensive. Evaluations should consider a range of realistic mobility models and should include special cases, such as high density and high traffic rates.

Chapter 4

Multicast Architecture

4.1 Introduction

Our findings from the earlier evaluation chapter motivates us to build a multicast transport architecture for providing reliability and congestion control. Ad hoc networks can suffer from packet losses due to congestion, mobility, and interference. Hence, for multicasting, reliability must be provided to recover these lost packets. Besides unreliability, these networks can also be bandwidth-constrained, which necessitates a congestion control mechanism to estimate the correct sending rate. Further, the bandwidth-constraint can vary from region to region, hence the congestion control should be able to estimate the correct reception rate for each receivers.

In this dissertation, we build a hop-by-hop multicast transport architecture, which operates at the network layer to ensure that packets are sent reliably along each hop in the multicast tree. Using a hop-by-hop approach allows us to avoid problems present in the existing approaches to building multicast. A network layer approach to proving reliability and congestion control to multicast is complex to build. An application layer approach suffers from the twin problems of excessive overhead (stress) and high latency (stretch).

Our architecture uses a receiver-driven approach since a receiver-based approach is more scalable and efficient. Each receiver can discover and maintain its own route for joining the multicast tree [14, 59]. Further, each receiver can also request for

retransmission of packets that it loses [24]. Hence, unlike TCP, the source is relieved of the responsibility of the book-keeping for reliable transmission of each packet and thus provides more scalability.

Therefore, the architecture needs three components: (a) an efficient receiver-driven multicast routing and state-setup protocol; this protocol establishes the branch, (b) a mechanism for determining when a node has moved, distinguishing this from congestion; this mechanism maintains the branch, and (c) a mechanism for reliable transport and congestion control at each hop along the tree.

This architecture is comprised of three modules: (a) a multicast state setup protocol that builds the multicast tree and sets up both transport forwarding state at each node in the tree, (b) a plug-in module which allows routing protocols to make a more informed decision about the nature of loss before they react to it, and (c) a multicast transport protocol, which uses application layer buffering to provide reliability and congestion control. In this chapter, we present an overview of all the modules in our multicast architecture and their interaction with each other. Figure 4.1 shows a schematic diagram of this multi-modular architecture. The multicast architecture uses input from several layers – transport, network and MAC layers. In subsequent chapters, we provide an in-depth description and evaluation of each of these modules.

4.2 ASSM

The first module is a multicast state setup protocol, *Ad hoc State Setup Multicast Protocol (ASSM)*. This protocol is used for setting up the multicast forwarding state from a receiver along a branch of the multicast tree and establishing transport state at each hop. To find the path from a receiver to the rest of the tree, ASSM uses the route to the multicast source that is provided by the unicast routing protocol. Traditionally, multicast protocols discover their own routes using a network-wide broadcast, where

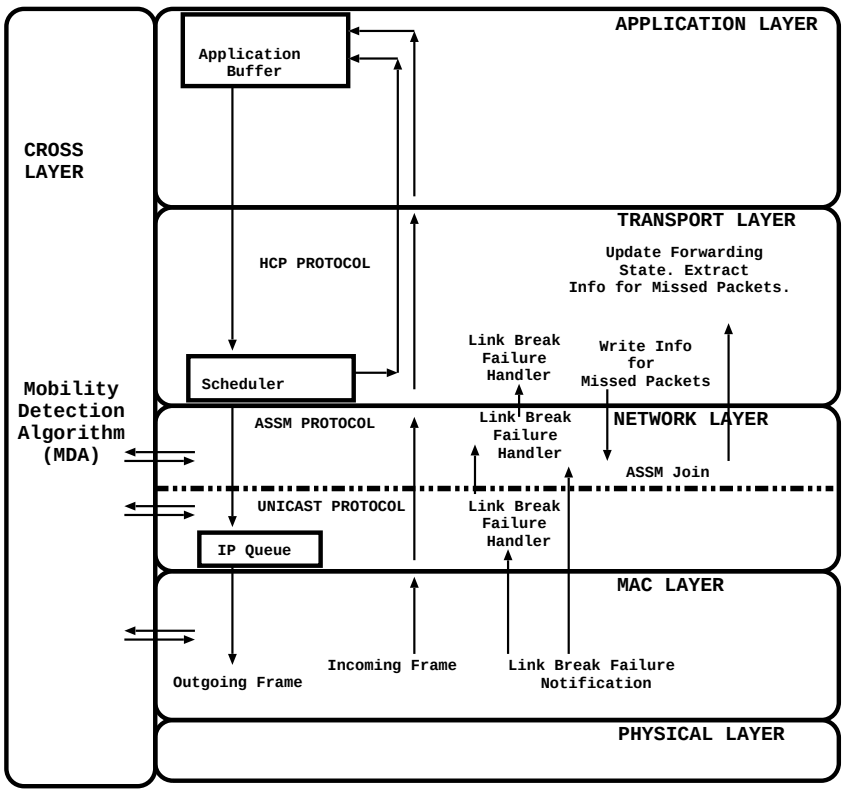


Figure 4.1: Multicast Transport Architecture

a route discovery packet is sent to each node in the network. By using routes that are already present in the unicast routing table, ASSM can greatly reduce its overhead.

The operation of ASSM is show in Figure 4.2, where a new HCP receiver sends an ASSM Join message to join a multicast tree. When an intermediate node (or the source itself) receives an ASSM Join, it establishes multicast forwarding state for the new branch. After that, ASSM passes the Join packet to the HCP layer, which establishes transport state. Whenever the Join message reaches an existing node on the tree, the Join message may be suppressed if there is no need to send it upstream towards the source.

Another interaction between ASSM and HCP is to carry information about missed packets. After the multicast and transport state is established, the source transmits data for the receivers. However, due to several reasons, receivers might miss some of these packets. An HCP receiver periodically resends the ASSM Join

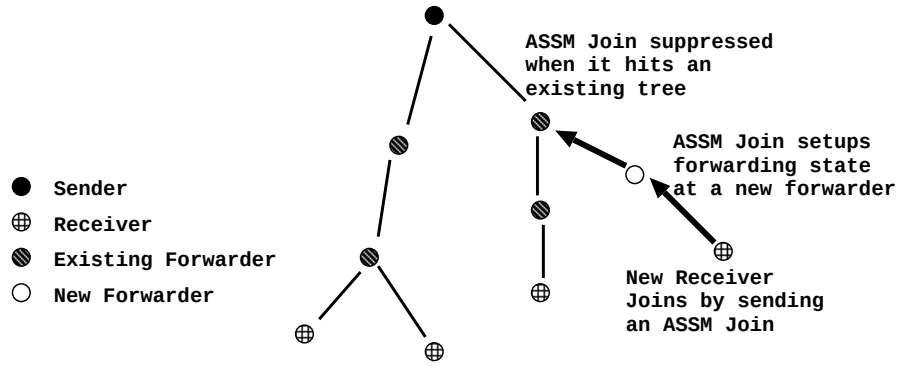


Figure 4.2: ASSM assists an HCP Receiver to join the multicast tree

both to refresh the forwarding state and to inform upstream nodes about any lost packets. The receiver builds a list of missed packets and includes this list in the Join message. As the Join message travels to upstream nodes, each node checks if it has some of these missed packets in its application buffer; if it does, then it queues those packets for retransmission. Any time a node has those missing packets and queues them for retransmission, then that node deletes those packets from the Join message so that its own upstream nodes avoid a duplicate retransmission. Ultimately the Join message reaches the source; the source retransmits whatever packets remain in the list.

In Chapter 5, we explore and verify ASSM's independence of the underlying unicast routing protocol by running it over top of two popular unicast routing protocols. Our results indicate that ASSM is able to deliver high throughput with low overhead.

4.3 MDA

The second module is a Mobility Detection Algorithm (MDA), which enables the multicast architecture to determine the cause of packet loss and react properly. A link break is typically detected by the MAC layer, when it fails to send a frame after trying for a certain number of times. The MAC layer propagates this failure

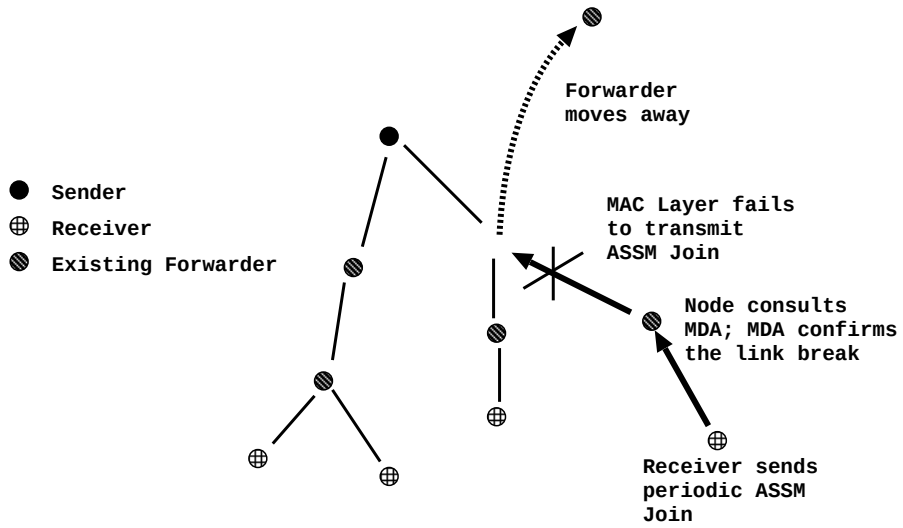


Figure 4.3: MDA: Link Break Due to Mobility

notification to the routing layer. When a routing protocol receives this information, it usually makes an assumption that the cause of the lost packet must be mobility. However, this assumption is not always correct, since packets may get dropped due to collision or congestion. Making this faulty assumption comes with a price; routing protocols making this mistake suffer from significant overhead when they initiate the repair of routes that have not been broken, resulting in decreased throughput for affected applications.

We have designed this cross-layer MDA to detect the cause of a lost packet. When ASSM sends a Join message and the MAC layer reports a link break, ASSM first consults with MDA; new routes are discovered only when MDA indicates that the current route is truly broken (Figure 4.3). However, if the packet loss is due to ongoing congestion, MDA does not confirm the link break and ASSM does not trigger any route repair activity (Figure 4.4).

In Chapter 6, we describe and evaluate MDA. Our results show that MDA dramatically reduces routing protocol overhead and significantly increases application throughput.

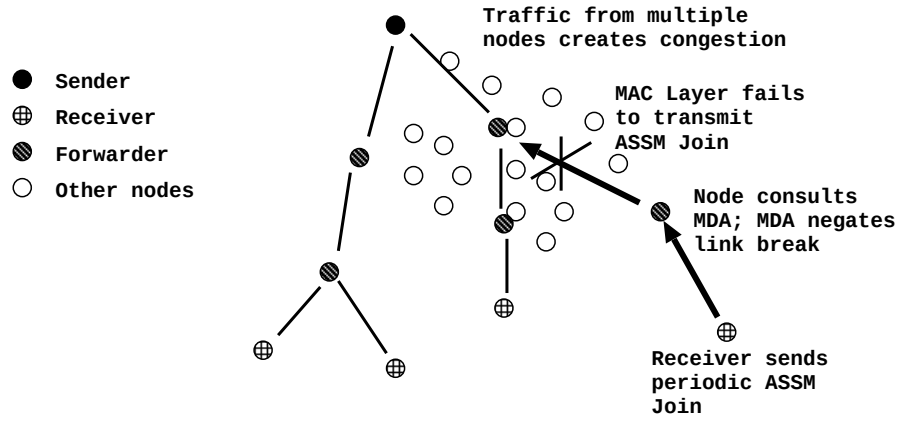


Figure 4.4: MDA: False Link Break Due to Congestion

4.4 HCP

The third module is a *hop-by-hop congestion control (HCP)* algorithm, which provides reliable transmission and congestion control at each hop in the multicast tree. Nodes in the tree buffer packets so they can retransmit them if needed. One of the main goals of HCP is to be able to send faster to those receivers who have more bandwidth, while still accommodating slower receivers. Further, by design HCP uses application layer buffering since an application layer can typically access the disk space and is not constrained by the sizes of buffer space available at the transport layer (Figure 4.5). When a sender or a receiver application is started, it initializes by allocating a pre-specified amount of disk space. A forwarder also has access to a buffer on disk and when a forwarding state is set, it also calls an initialization and allocates from this buffer.

HCP achieves reliability in two phases. In the first phase, each node in the tree chooses one of its children and sends data to it reliably, using the MAC layer. Other children try to receive the packet by listening to this transmission. However, this kind of snooping is not completely reliable, so some children may lose some packets. To cope with this loss, HCP uses a second phase of end-to-end reliability. Each receiver keeps track of all the packets it has received, and if it misses any packet, it asks

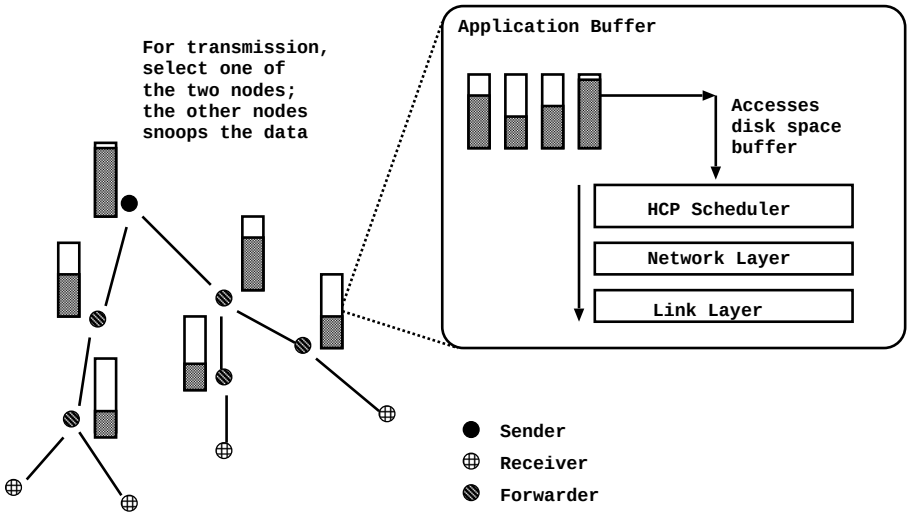


Figure 4.5: HCP: Forwarding of a new packet

its upstream nodes to resend those packets. This request is added to the periodic ASSM Join message. HCP stores recent packets in its application buffer; when it receives a request for retransmission, it pulls that packet from the application buffer and schedules it for transmission by queuing the packet in its retransmission queue (Figure 4.6). Next time, when a packet is transmitted, HCP uses packet from its retransmission queue, removes the packet from this queue, and then retransmits it.

Besides reliability, HCP also provides a *congestion control algorithm* which estimates the correct sending rate. Congestion control takes help from an internal module: a *Scheduler* that maintains information about all the flows passing through the node. The scheduler uses a fair policy and sends data to each flow by taking turns. The correct sending rate depends upon MAC layer; whenever MAC layer transmits a frame, it informs HCP and HCP queues its next packet.

HCP also uses flow-control, which avoids sending data to a downstream node, if the buffer at the downstream node is full. This mechanism is also achieved by snooping. When a downstream node forwards data to its own downstream node, then it updates a field reserved for available buffer in the HCP packet; the upstream node snoops at the HCP packet and comes to know about the available buffer at its

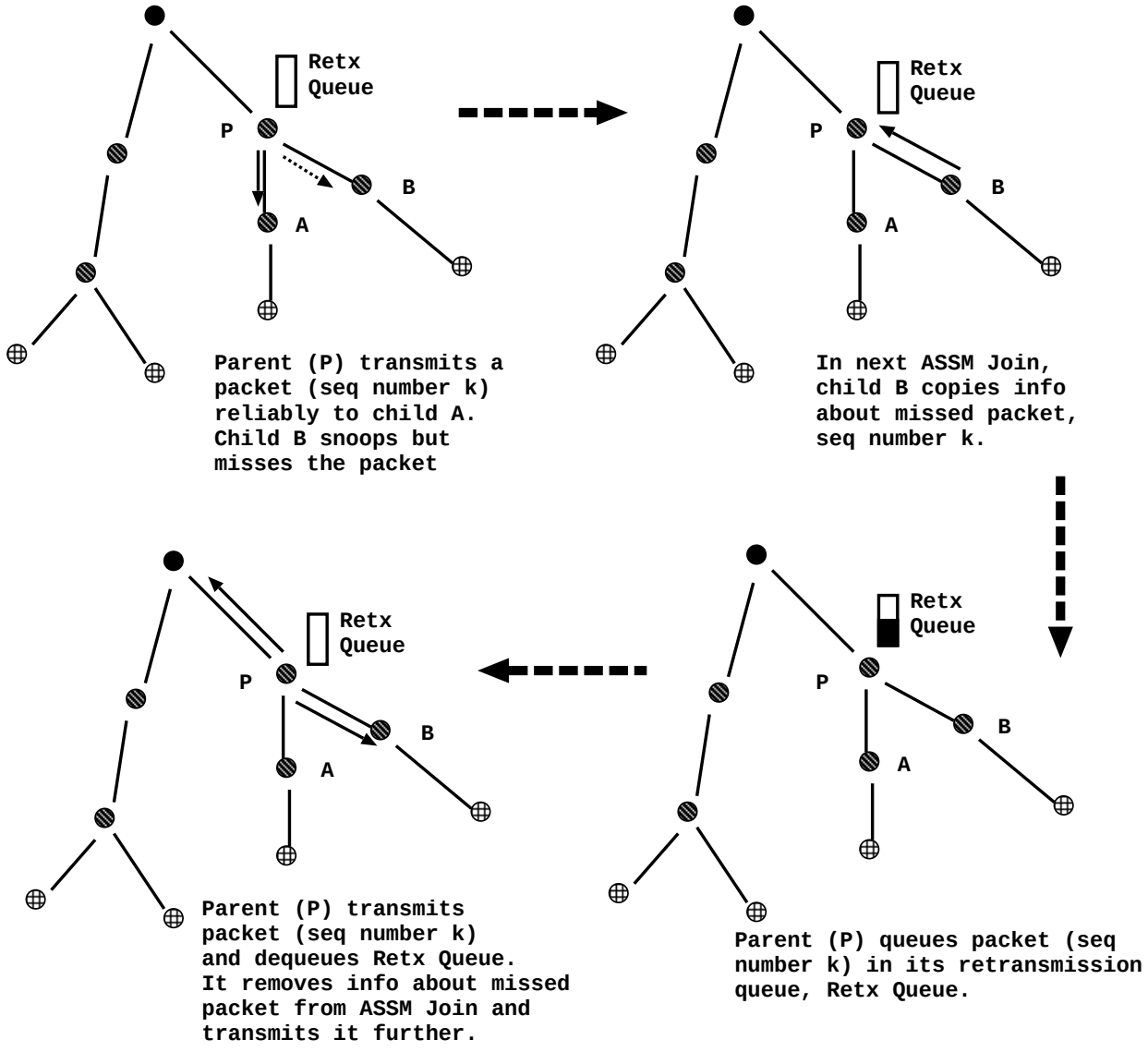


Figure 4.6: HCP: Retransmission of a lost packet

downstream node. Data is forwarded to that neighbor, only when that the buffer becomes available at that neighbor.

Chapter 5

Ad Hoc State Setup Multicast Protocol (ASSM)

This chapter is an extended version of the paper published as "Scalable Multicast Routing for Ad Hoc Networks", by Manoj Pandey and Daniel Zappala, The Fourth International Workshop on Localized Communication and Topology Protocols for Ad Hoc Networks (LOCAN), 2008, Atlanta.

Routing in a mobile ad hoc network is challenging because nodes can move at any time, invalidating a previously-discovered route. Multicast routing is even more challenging, because a source needs to maintain a route to potentially many group members simultaneously. Providing scalable solutions to this problem typically requires building a hierarchy or an overlay network to reduce the cost of route discovery and maintenance. We show that a much simpler alternative is possible, by using source specific semantics and relying on the unicast routing protocol to find all routes. This separation of concerns enables the multicast routing protocol to focus on minimizing join latency, repair latency, and control overhead. Using these principles, we build a new multicast state setup protocol – Ad Hoc State Setup Multicast Protocol, ASSM. Our simulations demonstrate that ASSM has a low overhead and provides a high throughput in most of the simulation scenarios.

5.1 Introduction

Group communication is an important service for mobile ad hoc networks, because many proposed applications involve coordination among groups of people, such as for emergency or military operations. To provide efficient group communication, a number of multicast routing protocols, like ODMRP and ADMR, have been proposed for ad hoc networks, usually building either a tree or a mesh. The focus of the first generation of protocols was to have a fast reaction to network changes, so that packet loss could be minimized [8, 10, 11, 41, 42]. More recent work in this area addresses the scalability of multicast routing, in terms of network size, group size, and the number of groups [60].

Two approaches have been used to provide scalable multicast routing in ad hoc networks. One approach is to create a hierarchy by partitioning the network into regions, with a group leader elected in each region [61]. Multicast packets are then sent among group leaders, and each leader forwards packets to members within its region. Another approach is to build a peer-to-peer overlay among the nodes, and then construct a tree based on the overlay [62, 63, 64, 65, 66]. In this case, multicast is provided entirely at the application layer, and packets are sent among overlay nodes using TCP connections.

While these two approaches have been shown to scale well, they both have the drawback that additional structure, either a hierarchy or an overlay, must be maintained to provide multicast. This incurs additional overhead, which increases as the nodes move. Both of these approaches may also increase delay, since packets are not necessarily delivered along the shortest or fastest paths between the source and the group members. Since overlay multicast is built out of unicast connections, it also introduces additional *stress*, or excess packets, as compared to multicast forwarding in the network layer.

What these approaches overlook is that scalable multicast routing has already been designed on much larger scale for the Internet, using the Source Specific Multicast (SSM) [59] protocol. In SSM, each multicast tree has a single source, the group owner, which is the root of the tree. Using a single source eliminates the need to perform an Internet-wide search for any user that wants to be a source for the group. Group members join a multicast tree by sending a *join* message that follows the unicast route to the source for that tree. If the join reaches a router already on the tree, then it *merges* at that point. If any group member wants to send data, it can either relay its data through the source, or it can notify the group members and build a new tree rooted at that group member. This design scales well because it separates group advertisement and source discovery (which operate at the application layer) from tree maintenance and data forwarding (which operate at the network layer).

In this chapter we apply this principle of separation of concerns to the design of an ad hoc multicast routing protocol, called ASSM. As with SSM, the ASSM protocol assumes that trees are identified by a group owner. This means that, in order for a group member to join a multicast tree, it only needs to know the unicast route from itself to the group owner. Since scalable unicast routing protocols have already been developed for ad hoc networks, ASSM can leverage this work to provide light weight, scalable, and receiver-oriented multicast routing for ad hoc networks. With ASSM, multicast functionality is divided into the following components:

- *group advertisement and source discovery*: As with SSM [59], groups are identified by a combination of the source and group address (S, G), rather than by the group address (G) alone. Essentially, every source is the group owner for its own collection of multicast addresses, which allows addresses to be assigned permanently to groups. The group and owner identifiers can then be advertised via any application, on a web page, or even configured directly into an application. This means that when a group member wants to join a multicast tree,

ASSM already knows the identity of the root of the tree. This avoids using a network-wide broadcast to discover multicast sources, as is necessary in many other multicast routing protocols designed for ad hoc networks [8, 10, 11].

- *unicast routing*: A unicast routing protocol finds and maintains routes to all nodes in the ad hoc network. Whenever ASSM needs to send a message, it sends it using a unicast route. This design concentrates route discovery in a single component, which eliminates duplicate broadcasts that are often performed by multicast routing protocols. This design also allows multicast to take advantage of any optimizations applied to the unicast routing protocol; making unicast routing more scalable directly benefits multicast routing as well.
- *multicast routing*: ASSM builds and maintains a separate tree for each multicast group, which reduces forwarding overhead as compared to a mesh. ASSM uses a receiver-oriented design, so that join latency is small. ASSM also localizes the repair decision, so that only the group members rebuild the tree, rather than asking all nodes on the tree to repair it when it breaks. This minimizes repair overhead, particularly during periods of high mobility.

Due to this separation of concerns, ASSM is able to provide scalable multicast routing for large networks, large groups, and large numbers of groups. In addition, this design greatly simplifies the multicast routing protocol, which only needs to install and maintain multicast forwarding state using existing unicast routes.

To test the effectiveness of ASSM, we investigate two fundamental questions: (1) *Can a purely receiver-oriented multicast routing protocol provide low repair latency while also minimizing overhead?* (2) *Does this design scale as well in ad hoc networks as it does in the Internet?* Repair latency is a critical measure because it indicates how quickly the multicast routing tree can be rebuilt when nodes move. Scalability is important because, while ad hoc network deployments are typically not large, network bandwidth is often very limited. As the number of groups increases, more

flows compete for this limited bandwidth, and it is important that the network provide efficient delivery, with low loss rates and low control overhead.

Using simulations, we show that ASSM repairs broken links faster than ADMR and ODMRP when a receiver moves, and is comparable to these protocols when a source moves. Our results also show that ASSM provides a high packet delivery ratio and low overhead as the number of groups increases, both for localized and co-located groups. ASSM also scales well in situations with high mobility, high density, and high traffic. Lastly, we run a simulation where we place additional sources in the network for the same group; our results indicate that using the first source to relay incoming data from newer sources, leads to better packet delivery than building source-specific trees for all sources.

5.2 Related Work

When designing a multicast routing protocol for a mobile ad hoc network, there are two main concerns that arise due to mobility: source discovery and tree maintenance. Because nodes may be mobile, a group member does not know the location of the nodes who want to send data to the group. Once the sources are found, the group members join a tree or mesh that connects them to the sources. The routing protocol must then repair the tree or mesh whenever any node on the tree moves. In this section, we describe three categories of multicast routing protocols and how they address the above two problems: first-generation flat protocols, hierarchical protocols, and peer-to-peer overlays.

5.2.1 First-generation Multicast Routing Protocols

ADMR

ADMR [10] solves the above two problems using a receiver-oriented protocol combined with an aggressive maintenance of the multicast tree. To find multicast sources, a group member floods a MULTICAST SOLICITATION message throughout the network. When a source receives this message, it responds by sending a unicast KEEP-ALIVE message to that receiver; the receiver joins the tree by sending a RECEIVER JOIN along the reverse path. When multiple sources are present, ADMR forms a separate tree for each multicast source. To ensure the tree is repaired whenever a node moves, each node monitors incoming packets and begins a repair process if it misses some number of consecutive packets. The repair is done by having the node downstream from the failure transmit a hop-limited flood of a RECONNECT message, to reattach to the existing tree.

ODMRP

ODMRP [8] solves these problems using a source-oriented protocol. To provide source discovery, each source for each group floods a JOIN QUERY message throughout the entire network. Each node receiving this message stores the previous hop from which it received the message. When a group member receives the JOIN QUERY, it responds by sending a JOIN REPLY to the source, following the previous hop stored at each node. The collection of state created by this and subsequent *Join Reply* messages forms a mesh that connects all sources to all group members. To repair the tree when nodes move, ODMRP simply re-sends the JOIN QUERY periodically and times out mesh state periodically. Over time, the mesh adapts to any changes in node location.

The basic trade-off in ODMRP is between throughput and control overhead. A source can increase throughput by sending more frequent JOIN QUERY messages.

Each message rebuilds the multicast mesh, repairing any breaks that have occurred since the last query, thus increasing the chance for subsequent packets to be delivered correctly. However, because each query is flooded, increasing the query rate also increases the control overhead of the protocol. ODMRP also can trade off redundancy and forwarding overhead. By increasing the soft-state timer for node forwarding state, ODMRP can increase the robustness of the mesh and hence provide more redundant paths for packets to be delivered. Of course, the richer the mesh, the greater the overhead when forwarding multicast packets.

MAODV

MAODV [11] is similar to ADMR in that it also uses a receiver-oriented protocol. Rather than building a separate tree for each source, however, MAODV builds a single tree shared among all sources. A group member finds the tree by broadcasting a RREQ message; any node currently on the tree responds with a RREP. The group member can then choose the best route and connect itself to the tree. To maintain the tree, each node exchanges HELLO messages with its parent. If a node stops hearing from its parent, it reconnects itself by broadcasting a RREQ.

One of the key advantages of MAODV and ADMR is that they are receiver-oriented, so a new group member can quickly join the multicast tree, rather than waiting for the source to add it. In addition, they both use a tree, rather than a mesh, so they have lower forwarding overhead than ODMRP. In terms of repairing the tree, ADMR can react to broken links more quickly because each node monitors packet reception, rather than waiting for a timer to fire. However, both ADMR and MAODV allow any node to repair the tree. This can lead to very high overhead during periods of high mobility, since many nodes on a single tree will initiate repairs simultaneously, leading to a large number of broadcast messages. Another problem

with ADMR is that it switches to flooding when packet loss is high; if the packet loss is due to congestion, the flooding of packets actually makes congestion worse [9].

5.2.2 Hierarchical Multicast Routing Protocol

MCEDAR (Multicast Core-extraction Distributed Ad hoc Routing) uses a hierarchical approach towards multicast routing [61]. It divides the network into various zones or *subgraphs* and each of these subgraphs elect a central node, called the *core*. These core nodes connect with core nodes of other subgraphs by forming a mesh, referred to as the *mgraph*. Inside each zone, the receivers of the multicast group connect to the respective core node to send control messages when they wish to join or leave the group. MCEDAR uses the mgraph to handle the problems of source discovery and tree maintenance. A new receiver looking for a source simply sends a join-request message to the core node in its zone. If the core node is already a member of the group, then it adds the new receiver. Otherwise, it broadcasts the join information on the mgraph. When the source receives this join message, it responds by sending a message to the receiver to confirm its join operation. Similarly, when a member has to leave the group, then it sends a *leave* message to its core, which forwards the leave message on the mgraph, if the core no longer has any children else the core simply removes the member from the list of its children.

Although, MCEDAR uses a mesh-based mgraph for control traffic, it does build a source-specific tree to forward data. The tree is built when the source responds to the join-request sent by the core members. When the response is sent, the unicast path is also stored for the new member; nodes use this route to forward data instead of mgraph. This way, MCEDAR combines advantages of both tree and mesh based multicast routing.

However, MCEDAR adds a layer of complexity as compared to non-hierarchical routing protocols. The set of core-nodes can potentially become a bot-

tleneck when a large number of groups are present. Further, additional studies need to be done to evaluate and demonstrate the robustness of MCEDAR in the face of high mobility, density, and traffic.

5.2.3 Peer-to-peer Overlay Multicast Routing Protocol

In this approach, group members form a mesh among themselves and a tree is constructed on top of this mesh. Individual links of the tree use TCP connections to forward data. This approach is valuable since the underlying TCP offers services like reliability, congestion-control, and flow-control. However, since a peer-to-peer overlay does not use optimal routes, this approach can be inefficient. In fact, an evaluation of Internet-based peer-to-peer overlay multicast in ad hoc networks reveals that a direct deployment of these protocols can be far from efficient [63].

Gui and Mohapatra design an overlay multicast protocol called, Progressively Adapted Sub-Tree in Dynamic Mesh, PAST-DM [5]. When a new receiver joins the group, it begins by doing a local hop-limited discovery of other members belonging to the overlay. Nearby members share the link state information among themselves periodically, creating a mesh among the members. Next, PAST-DM creates a source-based Steiner Tree on the top of this mesh. The tree-formation begins at the source, which connects itself to all of its next-hop children and divides the remaining members into sub-groups rooted at one of these children. Each of these children are responsible for forwarding packets to their sub-groups. Further, each child node also re-computes a tree for itself and groups its non-children member into sub-groups. This is done until there are no more children left. For tree maintenance, each member keeps track of its neighbors using a periodic neighbor discovery or by sending a query message to its current neighbors.

ALMA (Application Layer Multicast Algorithm) uses a receiver-driven approach to achieve source discovery [67]. Receivers first choose a subset of existing

members of the group and give higher preference to those that are closer. The authors recommend using a rendezvous point or a local search to retrieve the subset of existing members. These existing members are already connected to the source and thus form a logical multicast tree. Next, a new receiver sends a join message to the chosen subset of existing members. Upon receiving the join message, these members can either accept the new receiver or deny it, if they are already supporting a large number of receivers. Sending a leave message is comparatively simpler. The receiver sends a leave message to its parent in the tree as well as to all of its children. Lastly, receivers can also monitor the quality of their current connection to the tree and if the quality of the connection degrades, then they can switch to a new parent. When compared with ODMRP, ALMA starts to suffer when the group-size becomes large since the inherent inefficiencies in overlay make it difficult for ALMA to scale.

Another overlay multicast protocol, Prioritized Overlay Multicast (POM), allows users to join several groups depending upon the priority of the service [65]. POM enables an existing multicast group member to initiate and form another group of higher priority. A typical usage is the case of law-enforcement officials responsible for security of a public event. For a normal scenario, these officials can create an overlay with regular priority. However, the moment a security breach occurs, one of these officials can initiate a more secure overlay and the members of the earlier tree with sufficient security authorization can join the new overlay. Thus, POM allows members to smoothly transition from one priority overlay to another. POM depends upon the unicast protocol to provide it with unicast routes for both source discovery and tree maintenance.

5.3 ASSM

In this section, we describe how ASSM builds and maintains trees, as well as how additional sources can send data to an existing multicast group.

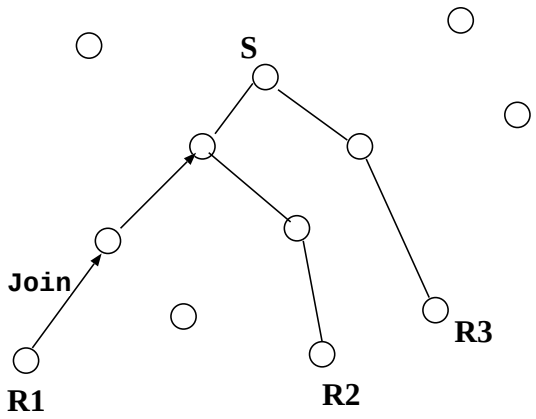


Figure 5.1: ASSM: Joining a Multicast Tree

5.3.1 Building a Tree

When an ASSM receiver joins a group, it already knows the group owner, S, since this information is provided through a group advertisement mechanism. The owner is also the root of the multicast tree. Figure 5.1 illustrates how a new group member (R1) joins a multicast tree. The new member unicasts a JOIN message to the source S. The format of the JOIN consists of following fields:

- *Group Owner Address*: The advertised owner for the group.
- *Group Address*: Address of the multicast group.
- *Receiver Address*: Address of the receiver.
- *Previous Address*: Address of the previous node sending the message. This starts as the address of the receiver and is set at each intermediate node to that node's address.
- *Role Type*: This field is an enumeration of four roles: ROUTE-REFRESH, LINK-REPAIR, PRUNING and RETRANSMISSION. ROUTE-REFRESH is used when a receiver sends a periodic ASSM Join. LINK-REPAIR is used when the ASSM layer has detected a link break. A receiver sends an ASSM Join message with PRUNING option, when it wants to prune itself from the tree. Lastly, RETRANSMIS-

SION is used when the Join message carries container information that controls whether the message is resent at each hop.

- *Protocol*: The protocol that will handle the payload.
- *Payload Length*: Length of the payload in bytes.
- *Payload*: This payload acts as a generic state setup container. Protocols can use this payload and interpret its data depending upon the protocol and role type specified.

A typical ASSM Join has the address set as described and uses a role type of ROUTE-REFRESH. The protocol and payload fields are set by the protocol performing state-setup. For an ordinary join, with no other state setup, the payload length is zero and the protocol is ASSM.

Whenever a node receives a JOIN message, it checks whether a forwarding state already exists for the group (Figure 5.2). If it does, then $needToForwardAssmJoin()$ returns false and the node does not forward the JOIN MESSAGE further. Instead, it simply adds a new child to the forwarding state (the new child is the previous node that sent the JOIN message). If the state does not exist, the node creates it, adds the child, and sets the parent to the next node on the route. The node then forwards the JOIN to its parent; this step is repeated until the JOIN reaches the source. In the example shown, R2 and R3 have previously joined the tree, so when R1 sends a JOIN it stops when it reaches the tree.

ASSM creates "soft state" forwarding states, meaning it times out if it is not refreshed periodically – the timeout being $T_{refresh}$. All group members re-send a JOIN message periodically after every T_{join} seconds, and these JOIN messages are forwarded upstream if the state has not already been refreshed recently. As shown in Figure 5.2, $needToForwardAssmJoin()$ also takes into account the freshness of the state; if the state is fresh, then it returns false. If a group member wants to leave the tree, it can

```
1 bool function needToForwardAssmJoin(packet) :
 1: /*
 2: * Return true if the packet needs to be forwarded */
 3: */
 4: retVal = true
 5: assmHeader = getAssmHeader(packet)
 6:
 7: /* Find the multicast Entry */
 8: multicastEntry = lookupMulticastState(assmHeader);
 9:
10: if multicastEntry == NULL then
11:       multicastEntry = createMulticastState(multicastEntry);
12: else
13:
14:       if (assmHeader.roleType ==
          ROUTE-REFRESH) and multicastEntry.fresh() then
15:             retVal = false;
16:       end if
17:       multicastEntry.refreshMulticastState();
18: end if
19:
20: /* Invoke the appropriate handler to process the packet */
21: handlers_list[assmHeader.protocol].handle(packet);
22:
23: return (retVal);
24:
```

Figure 5.2: ASSM Forwarder or Source receives ASSM Join

stop refreshing the JOIN messages and the state on its branch will eventually time out. To leave the group more quickly, a node can send a PRUNE message upstream to immediately delete the state on its branch.

5.3.2 Repairing a Tree

The advantage of using soft state and refresh messages is that it also rebuilds the multicast tree automatically whenever a node moves. As a JOIN message propagates upstream, it gets forwarded upstream based on the current unicast route. If the route has changed, then ASSM will automatically use a new path. If the JOIN message cannot be transmitted from one hop to the next hop, then this indicates that either the next node has moved or the frame was lost due to contention. We use the MDA protocol [68] to determine the cause of the lost frame, and if the loss is due to mobility, then the unicast routing protocol sends a failure message to the originator of the *Join* message, which is the group member. The group member then sends a new *Join* message, which triggers a new route request with the unicast routing protocol. Once a new route is computed, ASSM can send the JOIN upstream to rebuild the tree.

The process of repairing a tree due to mobility is shown in Figure 5.3. In this case, a node upstream has moved, and previous JOIN messages have caused the downstream receivers to be notified of this failure. When new routes are computed to the source of the tree, ASSM sends a JOIN message that follows the new route. To prevent the JOIN from merging on the old route in this case, ASSM sets a bit indicating that it is repairing the route, so that it can progress further upstream.

However, when using DSR as a unicast routing protocol, ASSM may have a higher repair latency. This high repair latency occurs because DSR maintains multiple routes to the same destination and when the current route is broken, it attempts to use the remaining routes from its route cache for the same destination. We use Figure 5.4 to explain this behavior.

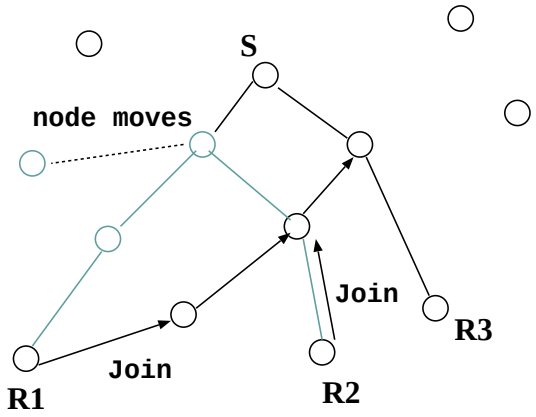


Figure 5.3: ASSM: Repairing a Multicast Tree

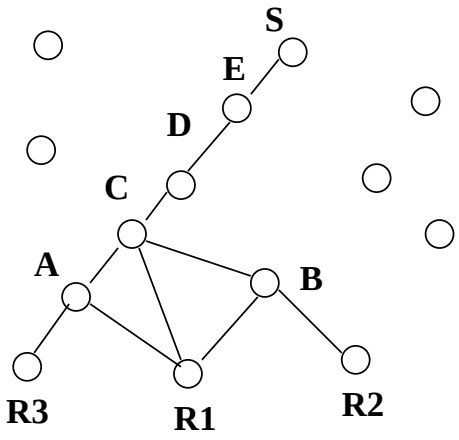


Figure 5.4: ASSM: Interaction with DSR Route Cache

At receiver R1, DSR maintains three routes to the source, S: $\{R1, C, D, E, S\}$, $\{R1, A, C, D, E, S\}$, and $\{R1, B, C, D, E, S\}$. Assuming that DSR uses the first route and sends an ASSM Join message via $\{R1, C, D, E, S\}$. Now, if forwarder D moves away, then the forwarder C can not send the ASSM Join and reports a link failure. This link failure is propagated back to Receiver R1 as a DSR *Route Error*. DSR has no way of knowing that the other routes are invalid since the error message does not carry information about the broken link $C - D$ and so DSR continues to keep these routes. DSR then tries the second and third routes, both of which cause link failures. It is only after all of the routes are exhausted, that DSR sends a new Route Request to find a working route. Thus, maintaining multiple routes for the same destination can lead to a higher repair latency.

5.3.3 Multicast Forwarding

Each node on the multicast tree has a parent node (null for the root of the tree) and a set of children nodes (null for a leaf). To forward multicast packets, the node first checks whether the packet arrived from the parent, and if not it discards the packet. If the packet is not discarded, the node must deliver it to all of its children.

By default, multicast packets are forwarded down the tree using broadcast at the MAC layer. However, broadcast traffic often suffers when it must compete with traffic sent reliably by the MAC layer (e.g. using an RTS/CTS exchange). In this case, the sender of a packet may set a field in each packet asking that forwarding be done using an alternative method. We list some of the methods that ASSM can use for forwarding:

1. Broadcasting [69]: Traditionally, multicast applications use UDP and broadcasting is the preferred style of data forwarding. With broadcasting, data is transmitted with no reliability and some of the children may not receive the data.

2. Reliable Broadcasting (Broadcast Medium Window, BMW [70]): Reliable broadcasting can be used to guarantee that all neighbors receive each multicast packet.

3. Reliable Multicast: This is similar to reliable broadcast, but instead ensures reliable delivery only to those neighbors that are children on the multicast tree.

Excluding the first option of broadcast, data is sent using unicast with one or more children; for these cases, ASSM plays an important role when a link break occurs. When the MAC layer fails to transmit a packet to a moved child, it drops the packet and notifies the node that the link is broken. In this case, ASSM deletes the entry for the downstream node for which data is being sent and it waits for the receiver to rejoin the tree.

5.3.4 Additional Sources

Besides the group-owner, ASSM can also support additional sources. These additional sources can use several different methods to send data to the same multicast group:

- *relaying:* A source can relay data through the group owner by sending it to the owner via unicast. When the group owner receives the data, it multicasts it over the tree. The primary disadvantage of this method is that it adds additional delay. This method can also be inefficient, since some routers may handle the same data twice – once when sending it to the root as a unicast packet, and again when forwarding it as a multicast packet.

- *source-specific groups:* A new source can send a unicast message to the group owner, informing it of its identity. The group owner then relays a multicast message to the group, informing them of the presence of the new source. Finally, the group members join a separate multicast tree rooted at the new source. This

method avoids the delay incurred by relaying all data through the group owner, but adds additional unicast and multicast routing overhead.

- *shared tree:* A new source can send data using the same tree as the group owner, provided multicast forwarding is done using a shared tree. For a shared tree, the parent and child nodes are treated equally when forwarding packets – a packet may arrive from any of these nodes and is then forwarded to the rest of them. Using Figure 5.1 as an example, R1 could send a packet to the group using this same tree, by having its parent accept the packet and send it further up the tree.

To implement multicast using a shared tree, every node must cache data packets to prevent forwarding loops. Imagine node A sends a packet to child node B, which forwards it to child C. In a wireless network, when node B forwards the packet, A will also receive it. In a directional tree, A will discard the packet, since it did not arrive from its parent in the tree. However, in a shared tree, this may be a new packet that B is forwarding on behalf of a downstream source. Node A needs to be able to distinguish between new packets and packets it has already sent. A small cache of packets a node has already forwarded solves this problem.

5.4 Simulation Methodology

We have implemented ASSM in both GloMoSim and ns-2 [50, 71]. GloMoSim is shipped with AODV and DSR. However, we also implemented additional optimizations for DSR, which are mentioned in its IETF draft, but were not implemented in GloMoSim [72]. Unless specified otherwise, the data rate used for our simulations is 11 Mbps.

When evaluating ASSM, we set the T_{join} to 1 second and the $T_{refresh}$ to 500 milli-seconds. We use CBR as the traffic and UDP as the transport protocol. Unless

specified otherwise, we send four 64 Bytes packets per second. For comparison, we use ADMR and ODMRP as two multicast routing protocols [8, 10]. Also, unless specified otherwise, each of the simulations are run for 5 random seeds and we present the average results from these runs.

Our evaluation uses four sets of simulations. The first set of simulations uses controlled mobility, where one of the nodes of the multicast tree is mobile. The second set of simulations tests ASSM using specific scenarios of varying group members and varying number of groups. In the third set of simulations, we subject ASSM to generic tests of high mobility, high density, and high traffic. In the last set of simulations, we examine the effects of additional sources joining an existing group.

Performance Metrics

We use several metrics to evaluate the performance of ASSM along with ADMR and ODMRP. For the controlled mobility simulations, we use three metrics to evaluate the repair response and the amount of overhead incurred during the process.

- *Repair Overhead*: The number of control packets originated or forwarded in the network to repair the broken link. For ASSM, we count packets that are either triggered by ASSM or are sent by the unicast protocol on behalf of ASSM. Since some amount of lingering control data can still be present in the network even after the repair is complete, we consider a fixed time interval (of 3 seconds) after the node moves and count all the overhead during this fixed time interval as overhead.
- *Repair Latency*: The time difference between when a node moves away, breaking the link, and when the affected receiver begins to receive packets again. We move the target nodes at a fast speed, so this accurately reflects the time it takes to repair the link. For the case of mobile receiver, we consider the time difference when the receiver moves away and when it begins to receive data again. For the

case of mobile source, we measure the latency for a randomly chosen receiver. It is worth noting that repair latency depends on the *convergence time* of the unicast protocol; fast convergence of the routing protocol reduces the time it takes to repair a broken ASSM tree.

- *Packet Loss*: The number of packets lost during the repair period. With CBR, packets are sent with a fixed inter-packet gap, hence packet loss is typically proportional to repair latency.

For the remaining sets of simulations, we use the following four metrics:

- *Packet Delivery Ratio*: The ratio of the number of packets received to the number of packets sent.
- *Control Overhead*: The ratio of the number of control messages originated or forwarded over the combined total of data and control messages originated or forwarded. Control overhead indicates the percentage of all messages that are control messages.
- *Transmission Overhead*: The ratio of the number of data messages transmitted (originated or forwarded) and the number of data messages received. This metric measures the efficiency of a routing protocol; the lower the transmission overhead, the fewer forwarders are required.
- *Throughput rate*: This is the number of kilobytes received per second by each receiver.

5.5 Controlled Mobility

In this set of simulations, we move one of the nodes forming the multicast tree in a field of size $1000m^2$ and study ASSM's performance. We first move one of the receivers and then we move the source. When we move a node, other nodes in the network

are kept static, the goal being to understand the effects of link breaks occurring at a receiver and a source. The data rate used for this set of simulations is 2 Mbps.

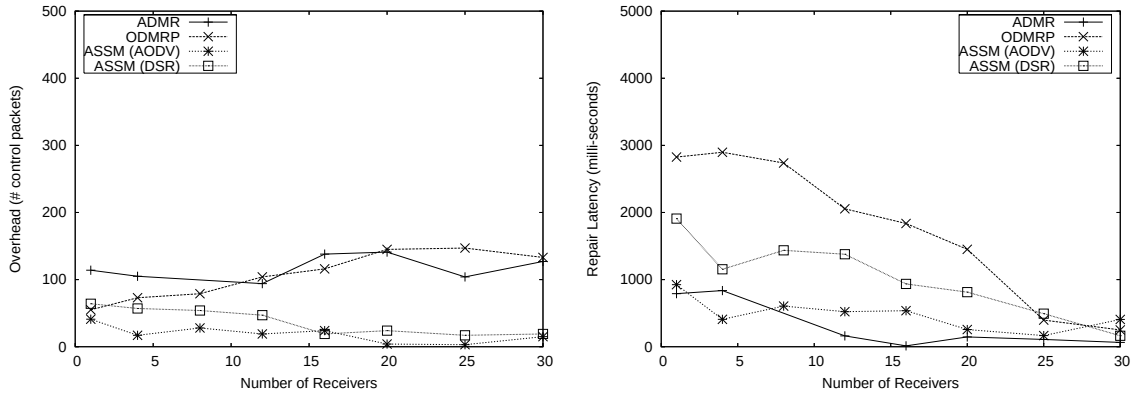
For these simulations, we vary the number of receivers for the multicast group. As the number of receivers increases, the mobile node may move to a region where there is an existing receiver and hence might find an existing route. Thus, increasing the number of receivers, increases the probability of finding a route for the source. Since ASSM uses existing routes, this simulation allows us to test if ASSM is able to benefit from existing routes.

5.5.1 Mobile Receiver

We begin by examining the case when a receiver moves during a multicast session. We randomly choose one of the receivers and make it move to a distant location compared to its current position. To understand the effects better, the receiver moves at a large speed so that it reaches the destination location in a negligible time. Other nodes remain static in the field. We use the three metrics of repair control overhead, repair latency, and packets lost during repair.

In Figure 5.5, we plot repair overhead for all the three protocols. ASSM performs better than both ADMR and ODMRP. With ASSM, only the receivers downstream of a failed link need to repair their branches of the tree; this controlled repair behavior leads to a lower overhead. It should be noted that ASSM also decreases its repair overhead because nodes near the receiver's new location are likely to already have a unicast route to the source.

ODMRP has a higher overhead due to its source-based approach of multicast forwarding state setup. It sends a periodic network-wide flood to refresh the mesh, which causes it to have a higher overhead. For ADMR, when a receiver experiences a link break, it sends a network-wide broadcast in the form of Multicast Solicitation message ; this behavior is similar to what a unicast routing protocol may do. However,



(a) Repair Overhead (b) Repair Latency

Figure 5.5: Controlled Mobility with Mobile Receiver

if the mobile receiver happens to be in the middle of the tree, which has a higher chance when we increase the number of receivers, it can break the middle of a branch of the multicast tree. This link break leads intermediate multicast forwarders to trigger repair efforts – an activity that increases the overhead.

Of all the protocols, ADMR has the lowest repair latency (Figure 5.5); its fast response is because an ADMR receiver continuously monitors incoming packets for link breaks. If it loses a series of consecutive packets, it concludes that a link has broken and immediately sends a Multicast Solicitation packet. ADMR demonstrates an important design element – a smaller repair latency can be achieved when receivers are allowed to independently monitor their connections and trigger a route repair once the link break occurs. However, when load is high, monitoring received packets can be deceiving, since packets can also be lost due to congestion. Hence, we need a mechanism that would instruct ADMR not to react when the loss happens due to congestion. To achieve lower repair-latency, ASSM could also monitor packets, provided there is a mechanism to advise it to not react when packets are lost due to congestion.

ASSM running with AODV performs better as compared to ASSM running with DSR. Using DSR increases repair latency – a behavior that can be easily ex-

plained because DSR maintains multiple routes to the same destination in the route cache.

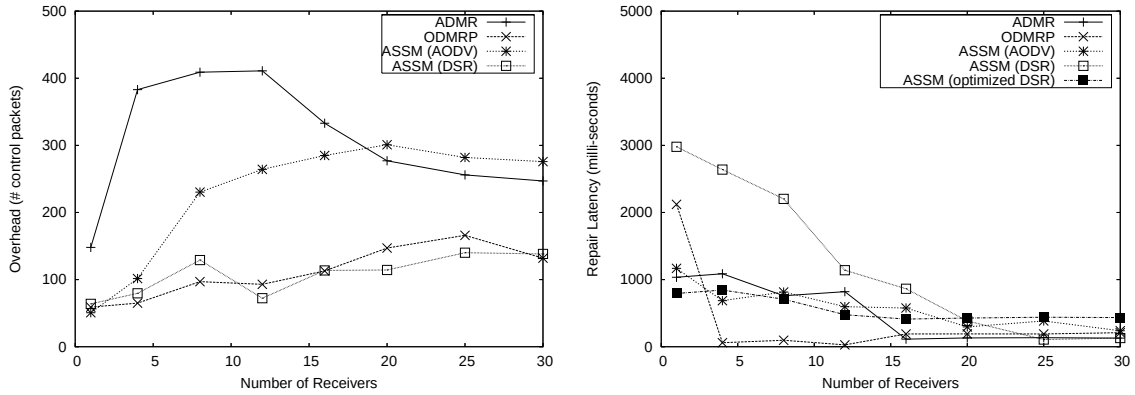
Among the three protocols, ODMRP has the worst repair latency. As explained in Chapter 3, repair latency for ODMRP is high because an ODMRP receiver has to wait for the periodic Join Query to reconnect it to the mesh. The Join Query is sent every 3.0 seconds and as shown in Figure 5.5, the repair latency has a similar value. However, once the number of receivers increases, the latency decreases. With more receivers, a receiver is likely to move in an area that already has a receiver and hence a connection to the multicast mesh. This allows the mobile receiver to receive data immediately, which reduces its repair latency.

For all the three protocols, number of packets lost follows the same pattern as that of repair latency. As explained in the simulation methodology, we use a Constant Bit Rate (CBR) traffic and hence the amount of packets lost is proportional to the repair latency. We omit the graph for the sake of brevity.

5.5.2 Mobile Source

We next consider a source that moves during a multicast session. Like the previous case of a mobile receiver, the source also moves to a random distant location compared to its current position. Once again, to understand the effects better, the source moves at a large speed so that it reaches the destination location in a negligible time. Lastly, the topology is also similar to that of the previous section.

In general, for most of the protocols, overhead is slightly higher for a mobile source than that of a mobile receiver. Figure 5.6 shows the overhead for the scenario of a mobile source. ASSM with DSR and ODMRP both have low overhead, while ADMR and ASSM with AODV fare poorly. For ASSM, the Join must travel till the vicinity of source before discovering that the source has moved; when the receiver moves, the ASSM Join needs to travel only *one* hop to find out that the link is broken.



(a) Repair Overhead (b) Repair Latency

Figure 5.6: Controlled Mobility with mobile source. Varying number of Receivers

.

Upon detection of the link break, a Route Error travels back to the originator. Hence, the Route Error also has to travel more hops as compared to the case when the receiver is mobile, which adds to the repair overhead. The overhead for ODMRP is fairly constant, regardless of whether the source or receiver moves, since it is a source-initiated protocol.

Several problems affect ASSM when the source is mobile. First, both of its underlying unicast routing protocols, AODV and DSR, face difficulties due to caching of stale routes. When a source moves, all of its previous neighbors still believe that they have a valid route to that source. Hence, when a Route Request is sent, the neighbors respond by sending a Route Reply with a stale route, even though the source has moved away! This use of stale routes further adds to the repair overhead. Second, AODV uses an expanding ring search for propagating its Route Request. If the source moves a large distance, the receiver may try multiple Route Requests, leading to a higher overhead. Third, as the number of receivers increases, the repair control overhead increases; this increase is because now due to more receivers, more periodic ASSM Join messages are sent.

ADMR also has a high overhead. With a mobile source, intermediate nodes fail to receive packets and trigger a route repair of their own. This consists of sending

Repair and Reconnect messages. In addition, end receivers also send Multicast Solicitation messages. Since these solicitation messages are sent in the form of network-wide flood, they become expensive. Hence a mobile source leads to a remarkably high overhead for ADMR.

However, for ADMR the overhead decreases when the number of receivers increases. Initially when the number of receivers is low, the tree formed is small and the source is likely to move into a region, where there is no existing multicast receiver. As a result, receivers are more likely to send a multicast Solicitation. However when the number of receivers becomes large, so does the multicast tree and the source is likely to move into an area where a forwarder already exists. Since the tree is bidirectional, receivers will keep receiving data packets even when the source moves to a new region as long as it moves near an existing forwarder. Thus, there is no need to send a multicast solicitation message leading to a lower overhead.

The repair latency for all protocols decreases greatly as the number of receivers grows, since the source is more likely to move near the existing tree (Figure 5.6). The nature of packet loss is similar to that of repair latency; hence, we omit the latency graphs for the sake of brevity.

The combination of ASSM and DSR does not perform as well as others in terms of repair latency – this behavior can be explained due to DSR's use of route cache. To improve the performance of ASSM running with DSR, we implemented several optimizations mentioned in the DSR draft, but not previously implemented in its GloMoSim model. We added the following three enhancements:

- Draft section: 3.1 – *Using exponential back-off to send Route Requests*: When a node sends a Route Request, it waits for a specified amount (ΔT) of time for the Route Reply. If the node does not receive the Route Reply during that time, it re-sends the Route Request. Such subsequent requests should be sent with a new ΔT selected with an exponential back-off.

- Draft section: 3.4.2 – *Queued Packets Destined over a Broken Link*: When a link break happens, the node should also check its IP queue and remove all the packets that use the broken link. This deletion ensures that the MAC layer does not keep forwarding packets for a broken link time and again.
- Draft section: 8.2.5 – *Preventing Route reply storms*: If multiple nodes have routes for a particular destination, they set a random timer, and only the node for which timer expires sends a reply. Other nodes can reply only if they have a better route (with shorter hops) for that destination.

In addition, we also implemented a novel Route Request aggregation optimization for DSR. After a node forwards a Route Request, if it hears another Route Request for the same destination from other nodes, it buffers the new Route Request. Upon receiving a route reply, it sends a route reply back to all the buffered route requests. This way, unnecessary forwarding of route requests for the same destination is reduced and response are aggregated.

These optimizations lead to a significant savings in both overhead and latency. Broadly, these enhancements reduce the number of packets being sent and hence the contention in the network. Reducing the number of unnecessary request and reply packets being sent on broken links reduces contention in the network leading to lower delay for data packets. All of the above three enhancements were helpful in improving the performance. As shown in Figures 5.6, with these added optimizations, DSR was able to perform much better.

Suggested Optimizations for DSR

Nevertheless, we also implemented two additional optimizations suggested in the DSR draft and found that they did not improve DSR's performance. These include:

- Draft section: 3.4.1 – *Packet Salvaging*: When an intermediate node finds that a link is broken to the destination node, it attempts to salvage the packet by

finding a new route from its own routing table. At first guess, salvaging makes intermediate nodes retry other stale routes in the vicinity of S and quickly invalidate them. This could lead to a faster invalidation of the routes than the receiver selecting routes one by one and deleting them. This improvement would occur because an intermediate node is in the vicinity of S and hence a unicast packet would have to travel fewer hops to reach the broken link.

However, this does not work. We use Figure 5.7 to illustrate the failure. Let us assume that Receiver R has three routes: Route1, $\{R, A, B, C, D, S\}$, Route2, $\{R, A, B, C, D, J, S\}$ and Route3, $\{R, A, B, E, G, H, I, S\}$. It starts by using Route1. Since S has moved away, forwarder D finds out that the link DS is broken. It then tries to salvage packet by using route $\{D, J, S\}$. However, when salvaging it replaces the original route, Route1 in the source header by the new route $\{D, J, S\}$. Hence when J triggers a route error, it deletes its link to S and the error is unicasted back to D and not to R! Thus, replacement of the source route during salvaging limits the proliferation of the information about a broken link and does not let the original receiver, R, know about other routes being invalid as well.

Furthermore, when R selects the next route, Route2 and sends data, J attempts to forward data to S, even though it has recently deleted its link JS. The reason for this behavior is that all intermediate nodes simply follow the source route specified in the packet and they do not store any recent link breaks. Hence, even with packet salvaging, the amount of time required to exhaust invalid routes remains the same.

- Draft section: 3.4.4 – *Increased Spreading of Route Error Messages*: The draft suggests that Route Error messages can be piggybacked on a Route Request to spread information about a recent link break and thus stale routing can be potentially reduced. However, this is not completely true. If R uses Route3,

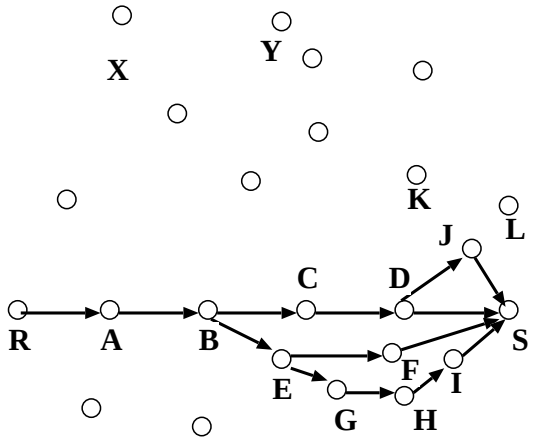


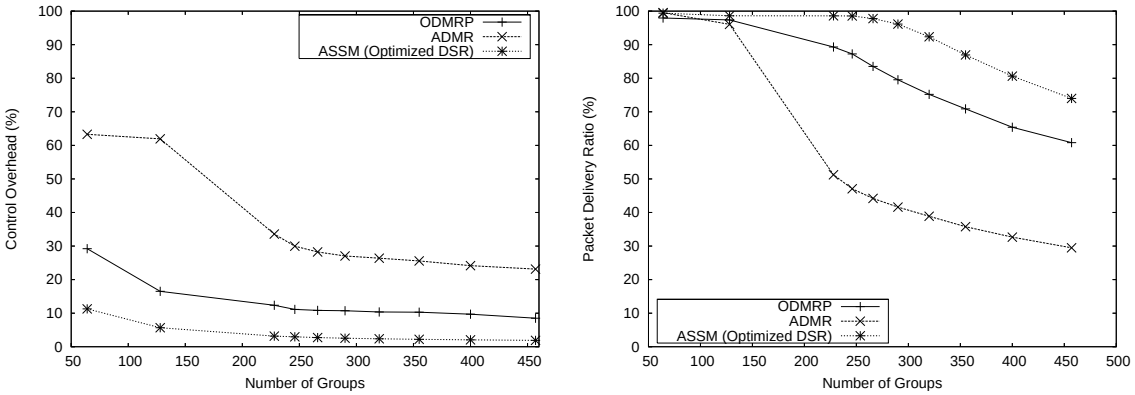
Figure 5.7: Routing in DSR

then the Route Error would carry information only for the broken link IS. All the recent link break information for broken links, DS, JS, and FS is not piggybacked with the route request. Nodes like K, and L could still believe that they have a route to S. Thus, piggybacking does not completely eliminate this stale information.

5.6 Localized Groups

In this section we use a scenario of localized groups to demonstrate the efficiency of ASSM. For this, we place several one-hop groups in a field such that members of each group are within one hop of the source. In real-life deployments, a group could be visiting an exhibition in a museum and receiving a multicast describing the exhibit; in this case, most of the receivers would be in the close vicinity of the source. In our simulation, each multicast group has 3 members. There are 100 static nodes kept in an area of $1000m^2$. During the simulation, we vary the number of these one-hop groups. During the simulation, we create traffic by sending ten 4 kilobyte packets per second. Lastly, we run ASSM with optimized DSR.

The purpose of this scenario is to demonstrate that for communication happening in closely placed groups, traditional multicast protocols that use network-wide



(a) Control Overhead (b) Packet Delivery Ratio

Figure 5.8: Controlled Mobility with varying one-hop Groups

messages can degrade performance. For example, ODMRP sends a periodic network-wide Join Query broadcast; nodes belonging to every group hear Join Queries being sent for all the other groups in the field. Clearly, there is no need for such a network-wide broadcast in this scenario. However, ASSM does not need to use a network-wide broadcast for its periodic route refresh; it simply uses the unicast route to send a one-hop control packet to refresh the forwarding state.

Figure 5.8 shows the control overhead and packet delivery ratio. ASSM has low overhead and high packet delivery ratio. ASSM sends the first request as a one hop broadcast and merely with this first broadcast, it is able to locate the source. AODV, which uses an expanding ring search for sending out Route Requests, is able to have the same benefit. Furthermore, periodic ASSM Joins are sent via the unicast routing protocol, which simply forwards it within one hop. As compared to ASSM, ODMRP does poorly because of two reasons. First, it uses flooding to propagate its Join Query, which consumes valuable bandwidth. Second, unlike ASSM, it uses a mesh to forward data packets and thus consumes additional resources to forward a packet.

With respect to the packet delivery ratio, ADMR's performance is the worst among all the three protocols in this scenario. As described in Chapter 3, this per-

formance degradation is caused due to ADMR's excessive control mechanisms. As stated in the earlier chapter, when ADMR looses a packet due to congestion, it falsely interprets the loss as a link break due to mobility, even though none of the nodes are mobile. Due to this incorrect interpretation, ADMR unnecessarily triggers expensive route repair mechanisms. These repairs add to the overhead and we see a sharp decrease in the observed packet delivery ratio. ODMRP has a lower packet delivery ratio due to use of mesh.

5.7 Co-located Groups

In this section we use a scenario of co-located groups to demonstrate ASSM's use of SSM semantics and unicast routing. For this, we test the three multicast protocols by increasing the number of groups located at the same source. Unlike the previous section, these group-members are not localized and hence could be several hops away from the source. Further, the members of these groups are not identical, however, they all receive multicast data from the same source. In real-life deployments, a source could be providing different multicast streams for different movies; each movie being viewed by a different set of receivers. For this simulation, we place 100 static nodes in an area of $1000m^2$. During the simulation, we vary the number of groups. For each group, both members and group owners are chosen randomly. During the simulation, we also create traffic by sending ten 4 kilobyte packets per second. Lastly, we run ASSM with optimized DSR.

The purpose of this scenario is to demonstrate that when there are multiple co-located groups, traditional multicast protocols will send control messages targeted to the same source over and again. This redundancy occurs because these multicast protocols do not distinguish the fact that all the groups are located at the same physical device.

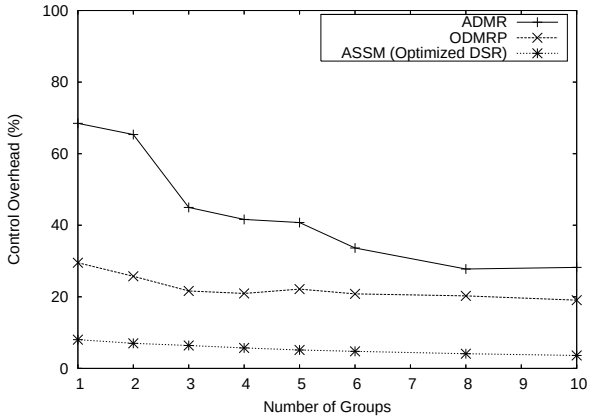


Figure 5.9: Control Overhead with Co-located Groups

As shown in Figure 5.9, ASSM outperforms both ADMR and ODMRP in terms of overhead. Because ASSM uses SSM semantics, it has the ability to use the same route for different groups since the source of the group is the same node. If one receiver searches for a route to the source, then the unicast routing protocols simply caches the route for all the nodes in that neighborhood. If there are additional receivers in the neighborhood, then they do not need to do any additional flooding for finding the source.

However, this is not the case for both ODMRP and ADMR since they both do group-specific network flooding. By design, an ODMRP source uses flooding to search for receivers. Even though the source is the same for all the groups, the source still needs to do flooding for each group, since this allows receivers for each group to receive the query and respond. Thus, for ODMRP, unless the source sends information about all the groups in one single Join Query, there is no way to reduce this overhead.

Likewise, ADMR creates separate trees for all the groups and the source sends a receiver-discovery broadcast. Once again, even if the source is the same, it still needs to do a network-wide flood for each group, leading to no savings.

5.8 Evaluation with high mobility, high density, and high traffic

In Chapter 3, we find that it is important to test a new multicast protocol with scenarios of high mobility, high density, and high traffic. Multicast protocols can suffer from excessive repair and during congestion, this behavior can lead to congestion collapse. Therefore, having explored several facets of ASSM, we next conduct a set of simple generic tests for ASSM, where we subject ASSM to conditions of high mobility, high density and high traffic. The purpose of these tests is primarily to confirm that the performance of ASSM does not degrade during these scenarios. In this section, we run ASSM on top of AODV.

5.8.1 High mobility

We begin by testing ASSM with high mobility. The simulation methodology is same as that of Chapter 3. We place 50 nodes in a field of area $1000m^2$. There are three groups, each with 1 sender and 7 receivers. We use only one mobility model, *random waypoint*, since our purpose is to ensure that ASSM does not fail at high speeds. The random waypoint model uses a pause time of 30 seconds.

We plot the packet delivery ratio for the three protocols in Figure 5.10. ASSM has a similar drop in its packet delivery ratio like ODMRP and a slightly higher drop in its packet delivery ratio at higher speeds as compared to ADMR (Figure 3.4 and 3.3). This similarity with ODMRP is no coincidence! Like ODMRP, ASSM also uses periodic refresh messages to maintain a route. While ODMRP sends a periodic route-refresh flood and receivers reply to that flood to refresh routes, ASSM receivers periodically send a refresh join message. This approach does introduce latency in its repair as the disconnected receiver needs to wait for the next ASSM Join to repair the broken route. It is due to this latency that we see a slightly higher drop in the

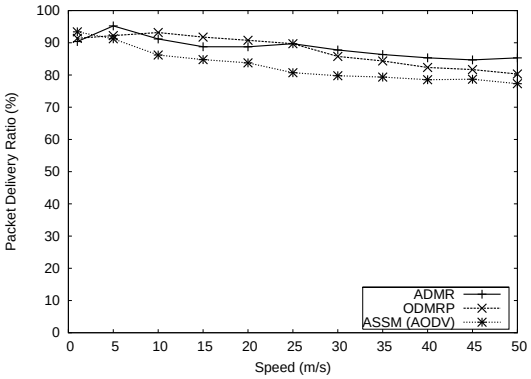


Figure 5.10: Packet Delivery Ratio vs. mobility

packet delivery ratio at higher speeds. It is worth noting that ASSM can potentially benefit from a faster approach towards repairing broken routes by monitoring received packets.

5.8.2 High density

We now consider the scenario of high density. Once again, we use a methodology similar to that of the Section 3.5. We place 75 nodes and vary the size of the field, measuring the average number of neighbors for each node as the neighbor density. We use three multicast groups, each with 1 sender and 22 receivers.

We plot the packet delivery ratio for this simulation in Figure 5.11. Both ODMRP and ASSM perform similar. Unlike ADMR, ASSM's performance does not degrade with increasing density.

As explained in Section 3.5, ADMR suffers from two problems: Receiver Join implosion and ACK implosion. ASSM avoids both of the pitfalls. With ASSM, the source does not use any network-wide flood to discover receivers; instead the receivers join explicitly by sending a unicast message to the source. Since ASSM solicits this route from the unicast routing table and if the route is already available, then no further source discovery flooding is done. Secondly, the receivers do not send any

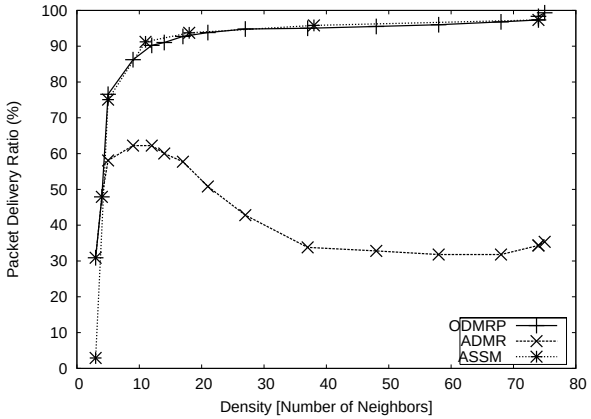


Figure 5.11: Variation of Packet Delivery Ratio with density

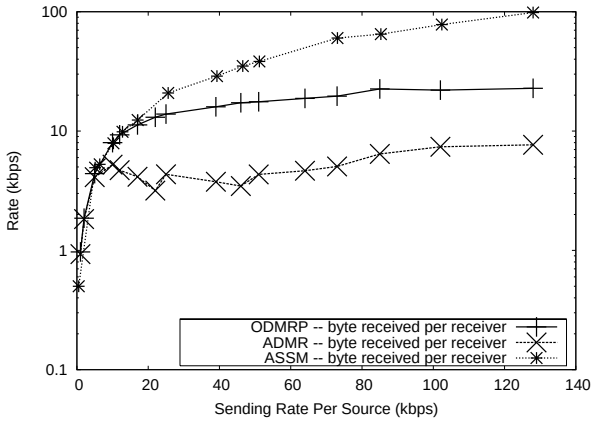


Figure 5.12: Variation of throughput with Traffic

explicit acks when they receive packets. Both of these design choices make ASSM more efficient when handling a high density scenario.

5.8.3 High traffic

Next, we evaluate ASSM using a high traffic scenario. We use a simulation methodology similar to that of the Section 3.6. However, since the results for varying the inter-packet gap and for varying the size of packets are similar, we choose to run ASSM for the case of varying the inter-packet gap. Further, we focus only on the average number of bytes received by each receiver.

As shown in Figure 5.12, both ASSM and ODMRP are able to deliver high throughput. In fact, ASSM performs better than ODMRP, because ASSM avoids

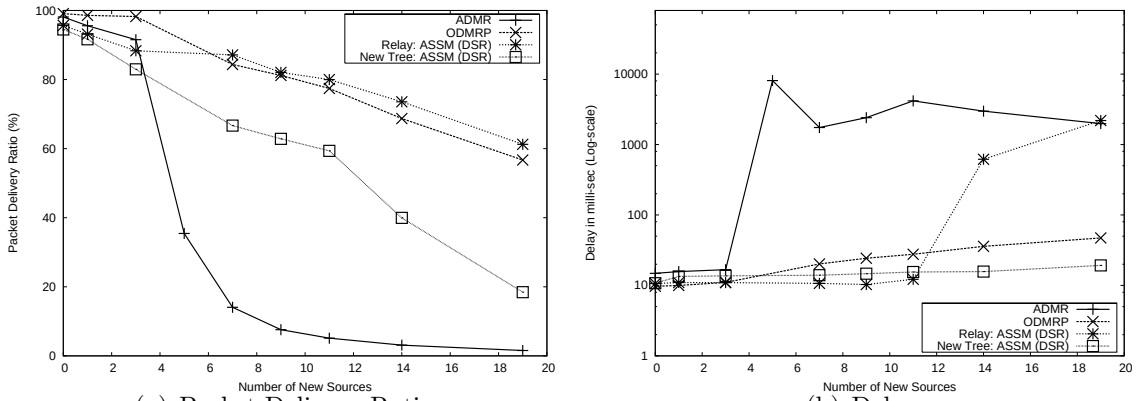
any periodic flooding unlike ODMRP, where each source must periodically flood the network for discovering the receivers. Further, after an ASSM receiver sends its Join message to the source, all the receiver does is wait for the source to transmit data. Hence, except for the case of sending the ASSM JOIN, it is only the sender that transmits and all other nodes merely listen to the data and forward it; this simplicity keeps the network relatively free and hence, enables a higher throughput for ASSM.

5.9 Additional Sources

Finally, we conduct a set of simulations to examine several alternatives for handling additional sources. For this scenario, the group has a total of 7 receivers and we keep the number of receivers same when increasing the number of sources that transmit to the same group. We choose these receivers randomly from a total of 50 randomly placed static nodes. The sources generate CBR traffic and send ten 64 byte packets every second.

We test two methods by which ASSM can handle additional sources. First, they may use the existing tree to *relay* their data; they simply unicast their data to the group owner, which then multicasts the data on the tree. Second, they may send their information to existing receivers via the group owner; when receivers get information about a new source, then they explicitly join the new source thereby creating a *source-specific tree* for each source.

From our results, we find that when using relay method, ASSM has a higher packet delivery ratio that when it builds new trees for each source (Figure 5.13). This trend is not surprising because building a new tree causes additional overhead for sending an extra JOIN message per source. When we plot the number of control packets sent, we find that it is higher for the case of source-specific trees. As expected, for ASSM, the delay incurred, when using a relay is higher (Figure 5.13). It is interesting to note when the number of sources is small, the relay has low delay, as with



(a) Packet Delivery Ratio (b) Delay

Figure 5.13: Additional Sources

all other protocols. The probable cause for this is that with a small number of sources relaying packets to the original source, the original source does not get overwhelmed and is able to relay the packet on the tree with no additional delay. In this case, setting up a tree for each source perhaps consumes enough network bandwidth to introduce additional delay as compared to when using the relay. However, when the number of sources becomes too large, the original source gets overwhelmed and the delay starts to increase sharply since the original source can forward data only at a certain limited rate.

ADMR continues to perform worst among all protocols. ADMR performs worse as compared to ASSM with source-specific case, even though in both cases source-specific trees are built. This behavior can be explained due to the inherent inefficiencies present in ADMR; in fact, when the number of sources is increased beyond four, ADMR's performance decreases sharply both in terms of throughput and delay. ODMRP, like the relay-variant of ASSM, uses one common mesh and avoids a high overhead. Hence, ODMRP's performance is similar to the relay-variant of ASSM.

5.10 Conclusion

In this chapter, we demonstrate that it is possible to design a scalable multicast routing protocol for ad hoc networks without using hierarchy or an overlay. Using SSM semantics eliminates much of the overhead previously required for joining a multicast group. Using existing unicast routing protocols enables protocol designers to focus on making one scalable routing protocol that can be used for both unicast and multicast services.

We evaluate various aspects of ASSM using simulations. Our simulations include scenarios of controlled mobility, high mobility, high density, high traffic, and additional sources. Our results shows that localizing repair decisions at the group members enables ASSM to provide both low repair latency and low repair overhead. Further, due to its simple design, ASSM is also able to deliver high throughput in most of the above tests.

As a future work, we need to improve ASSM's repair latency even further by triggering repairs instead of using a periodic timer. A group member should be able to measure the average inter-packet delay, and then trigger a repair if this delay grows too large, rather than waiting for the next timer expiration. This would allow ASSM to perform better when mobility is high. We also plan to directly compare ASSM to multicast routing protocols that use a hierarchy or overlay to demonstrate that multicast scales better when it builds on top of scalable unicast routing.

Chapter 6

Mobility Detection Algorithm (MDA)

This chapter was published as "To Repair or Not to Repair: Helping Routing Protocols to Distinguish Mobility From Congestion", by Manoj Pandey, Roger Pack, Lei Wang, Qiuyi Duan, and Daniel Zappala, IEEE INFOCOM MiniSymposia, May 2007, Anchorage.

In this chapter, we present the second module of our Multicast Architecture: a **M***obility* **D***etection* **A***lgorithm, MDA, that properly detects the cause of a lost frame. Ad hoc networks are designed for mobile scenarios; when nodes move, established links break. These networks detect link break when they are unable to transmit frame on an established link. However, packets might not be transmitted due to an ongoing congestion as well! Hence, it is easy for protocols to mistake a congestion based loss as a mobility based loss. However, reacting wrongly to a congestion based packet means that a network-wide route discovery flood might be triggered, which is an inefficient behavior. Thus, routing protocols can frequently remove valid routes and keep acquiring newer routes; this mistake can lead to route oscillations leading to a lower performance. MDA is an integral module of our multicast architecture since our architecture consists of a multicast state setup protocol, ASSM, which is dependent upon underlying unicast routing protocols. Without MDA, these unicast protocols can fail to distinguish congestion based losses from mobility based losses during congestion.*

One of the difficult problems facing mobile ad hoc networks is determining the cause of frame loss and reacting properly. Most ad hoc routing protocols typically

assume that any lost frame indicates that the destination for the frame has moved, so all paths using that node are broken. This is a faulty assumption; it is possible that the frame could not be delivered due to congestion, rather than mobility. Routing protocols making this mistake suffer from significant overhead when they initiate the repair of routes that have not been broken, resulting in decreased throughput for affected applications. In this paper, we present a mobility detection algorithm, MDA, that properly detects the cause of a lost frame. It then coordinates with the routing protocol so that new routes are discovered only when the current route is truly broken. This approach dramatically reduces routing protocol overhead and significantly increases application throughput. We use a simulation study to demonstrate the effectiveness of MDA in a variety of scenarios and to determine the proper setting for MDA parameters.

6.1 Introduction

One of the difficult problems in mobile ad hoc networks involves distinguishing whether packet loss has occurred due to mobility or congestion, and then having the transport and routing protocols react properly. If the transport protocol assumes all packet loss is a sign of congestion, then it may needlessly decrease its rate when mobility causes loss. Likewise, if the routing protocol assumes all packet loss is a sign of mobility, it may initiate an expensive route discovery or repair process when it should instead do nothing and let the transport protocol adjust its rate.

There are two instances when packets can be lost in an ad hoc network: when the forwarding queue at a node overflows or when transmission of a frame fails at the MAC layer. In the first case, the loss is clearly due to congestion, so there is no ambiguity as to the appropriate response. In the second case, however, the cause for a failed transmission could be due to either mobility, congestion, or interference. Loss due to mobility occurs when a node moves, so that the packet cannot possibly

be delivered. Loss due to congestion may occur when two or more wireless nodes are transmitting at the same time, causing collisions. Interference likewise causes loss due to collisions, but the competing source uses a different technology in the same spectrum, such as a microwave oven or cordless phone. In this paper we focus on distinguishing between mobility and congestion, leaving interference for future work.

In examining this problem, we focus on the IEEE 802.11 standards, in which the MAC layer uses CSMA/CA to attempt to avoid collisions with other nodes. Prior to transmitting, each node uses an RTS/CTS exchange to obtain the medium, and then uses a DATA/ACK exchange to ensure reception of a transmission [73]. If either the RTS/CTS or DATA/ACK exchange is unsuccessful, it retries the entire transmission up to 7 times for smaller packets or 4 times for larger packets, doubling the time between transmissions for each retry. If the frame cannot be delivered, it notifies the network layer that the link has failed. The exact cause of the dropped frame cannot be determined by the MAC layer.

What exactly should happen when a packet is lost? If the loss is due to mobility, then clearly the current route is broken. In this case, the routing protocol should repair the route, and in the meantime the transport protocol should stop transmitting. Once the route is repaired, the transport protocol may need to restart its congestion control algorithm to determine the appropriate sending rate for the new path.

When loss is due to congestion, the appropriate reaction depends on whether the route is experiencing *transient* or *persistent* congestion. Many transport protocols purposely cause transient congestion as they probe the network for available bandwidth; if just a single packet is lost, the transport protocol should reduce its rate. If a flow experiences persistent congestion, its rate will eventually decrease to a level that the application or user considers too slow; the only remedy in this case is to find a different path, assuming some kind of alternate path routing is available. With per-

sistent congestion, a basic routing protocol cannot be allowed to find a new route, as all traffic will then switch to this new route, without solving the problem. Rather, a QoS routing protocol must be used to carefully route different flows so as to efficiently use network bandwidth and to avoid oscillation.

In either case, whether congestion is transient or persistent, the default action of the routing protocol should be to do nothing when packets are lost due to congestion. With basic routing, the transport protocol will need to share available bandwidth among the flows on the route. Even with QoS routing, the transport protocol must be given a chance to converge before the system can determine whether an alternate path is needed.

Most ad hoc routing protocols have a major design flaw in that they are designed to react to all packet loss as a sign of mobility, without any regard to congestion. Both AODV [12] and DSR [15] assume that a single dropped frame in the MAC layer is a sign that all paths using the MAC layer destination as a next hop have failed. AODV also has the option of broadcasting periodic HELLO messages to inform its neighbors that it has not moved away; failure to receive a number of HELLO messages leads the node to believe that its neighbor has moved away. Similar behavior exists in ADMR [10], a multicast routing protocol for ad hoc networks. ADMR assumes that the sender transmits at a constant rate and infers that a route is broken if a number of consecutive packets are not received.

This design flaw is also present in routing protocols that use passive ACKs and HELLO messages to determine link availability. The DSR IETF draft [72] specifies a mechanism using passive acknowledgments; a node listens for the next hop to forward the packet and, when overhead, concludes that the link is still active. However, if congestion occurs, these passive ACKs are likely to get lost, causing DSR to mistakenly conclude that a link has failed. WRP [74] depends on HELLO messages being exchanged between nodes, or can use an ACK of a link update message, to conclude

that a link is still active. If WRP does not receive these messages, regardless of the cause, it triggers a route repair process.

The costs incurred due to this design flaw are significant because finding a new route involves flooding the network to some extent. Whenever it determines that a route has failed, AODV drops all packets queued for that next hop and uses flooding to try to repair the route. DSR uses any existing alternate routes, and once all paths to the destination are invalidated it notifies the sender that the route has failed. This in turn causes the sender to use flooding to try to find a new route. When ADMR determines that a branch of a multicast tree has failed, an affected node or group member uses flooding to try to find a new route to the rest of the tree.

We have observed two severe consequences of this design flaw in our work. First, we have shown that ADMR may suffer from congestion collapse due to excessive route discovery packets when operating in dense networks [9]. When many nodes send route discovery messages, they cause loss due to collisions, which in turn leads to ADMR sending more route discovery packets! The second consequence we have observed is very long route convergence times in ADMR and DSR, even in sparsely populated networks. When multiple nodes flood route discovery packets, this can cause congestion, which in turn may cause the routing protocol to believe that some discovered routes have already failed. We have observed in several simulations that it is possible for DSR or AODV to take several minutes to stabilize under these conditions, with many route-finding attempts occurring redundantly.

During more typical steady state conditions, the price of misinterpreting packet loss is increased routing protocol overhead, leading to reduced application throughput. In this paper we demonstrate that this is a significant concern, with routing overhead consuming as much as 700 Kbps in a 2 Mbps wireless network.

To solve the loss detection problem for ad hoc routing protocols, we design a Mobility Detection Algorithm, MDA, that uses MAC-layer statistics to distinguish

between mobility and congestion-based losses. Using MDA can significantly reduce routing overhead, leading to corresponding increases in throughput.

One of the advantages of MDA is that it uses a cross-layer design so that it is completely independent of the routing protocol. We are able to use MDA to significantly improve the performance of both AODV and DSR. Because of its independence, MDA is easily adapted to work with any kind of link availability mechanism. We report results showing that our performance improvements for AODV hold whether it uses packet loss, HELLO messages, passive acknowledgments, or network-layer acknowledgments to determine whether a route is alive.

To demonstrate the advantages of MDA, we conduct a simulation study that examines routing protocol performance in conjunction with a congestion-controlled transport protocol. This study is unique in that it is the first to show the impact of routing overhead on application throughput. Prior studies of routing protocols send constant bit rate (CBR) traffic, usually only a small number of packets per second, and measure just the percentage of packets delivered correctly. We use ATP for a transport protocol because it has been designed specifically to overcome some of the shortcomings of TCP in an ad hoc wireless network [19].

Our results show that MDA successfully differentiates between packet loss due to mobility versus loss due to congestion. As a result, the routing protocol only rebuilds routes when the current one has truly been broken, significantly lowering routing overhead. In a scenario dominated by congestion, MDA improves ATP throughput by 50 to 100%, with similar gains for CBR flows. In a scenario dominated by mobility, MDA properly notifies the routing protocol, with no negative impact on application throughput. In more mixed scenarios, MDA improves ATP throughput by 10-100%, depending on the routing protocol. We also evaluate appropriate settings for MDA parameters and show that its behavior is very close to that of an omniscient routing protocol.

6.2 Mobility Detection Algorithm

MDA is based on the observation that proving a node has moved is not possible, but it is easy to conclude that a node has *not* moved. Whenever the MAC layer reports a transmission failure, instead of reporting a failed link, MDA waits for a short time. If the destination of the lost frame transmits a packet during this time, then MDA knows it has not moved, so the loss must be due to congestion. If MDA does not hear a frame from the destination, then it assumes the destination has moved and notifies the routing protocol so that it can repair the route.

The complete MDA algorithm is shown in Algorithm 6.1. The main state MDA keeps is a global credibility value, initially set to *threshold*, typically 1 or 2. MDA interprets high credibility as indicating that frame loss is due to mobility, while low credibility (anything less than *threshold*) implies loss is due to congestion. The credibility value is kept globally, rather than separately for each neighbor, because this enhances shared learning across all neighbors. This is particularly effective for detecting congestion-induced losses; in a wireless medium, if any frame is dropped due to congestion then it is highly likely that subsequent lost frames are also due to congestion.

If the MAC layer drops a frame due to transmission failure, MDA first checks if any CTS signal was received from the destination during any transmission attempt. If a CTS was received, then that neighbor must still be within transmission range; this causes credibility to be immediately set to zero. Otherwise, MDA starts a credibility observation on this suspect neighbor by starting a timer. Finally, MDA checks the credibility value and sends a failure notification to the routing layer only if the current credibility equals *threshold*.

To determine whether a node has in fact moved, MDA has the MAC layer notify it of all frames that it overhears from any neighbor. If the timer for a node expires before MDA observes a frame from this neighbor, then the neighbor is assumed

Figure 6.1: Mobility Detection Algorithm

```
1: credibility = threshold;
2:
3:
4: if transmission failure to node X then
5:    if received CTS from X during attempt then
6:       credibility = 0;
7:    else
8:       mdaTimer.set(t, X);
9:       if credibility == threshold then
10:         notify routing protocol node X has moved;
11:      end if
12:   end if
13: end if
14:
15: if mdaTimer for node Y expires then
16:   credibility = min(threshold, credibility++);
17: end if
18:
19:
20: if hear from any node Z then
21:   if mdaTimer.isset(Z) then
22:      mdaTimer.cancel(Z);
23:      credibility = 0;
24:   end if
25: end if
26:
```

to have moved, and credibility is increased. If MDA hears from a neighbor with an outstanding timer, then it sets the credibility to zero cancels the timer.

Using this algorithm, MDA has a simple decision to make each time the MAC layer reports a dropped frame due to multiple retransmissions failing. If the credibility is at the *threshold*, then MDA reports a link failure to the routing protocol. If the credibility for the neighbor is below the *threshold*, then MDA assumes that the loss is due to congestion and does not give any signal to the routing protocol.

Note that MDA's decision is purely based on observation. Whenever a frame is dropped, a node examines all packet transmissions for a given time interval to

determine whether it can hear anything from that neighbor. MDA can conclusively determine when a neighbor has **not** moved, if it ever hears from that neighbor. However, when MDA does not hear from a neighbor for a certain time, it infers that the neighbor has moved, and this decision may be wrong. The neighbor may still be within range, but may not be transmitting or there may be enough congestion that it cannot be reached.

MDA works well in conditions of both high mobility and high congestion. If mobility is high and congestion is low, link failures (primarily due to mobility) are detected quickly because congestion will not have reduced the credibility value. During conditions of high congestion, losses will always be reported as due to congestion, even though a node may have moved. This is acceptable because the transport protocol should lower its sending rate first, to prevent congestion collapse. It will then be easier for MDA to determine which nodes have moved during this period, and the routing protocol can react to route changes when the network is not as heavily congested.

The performance of MDA depends on the setting of two parameters – the *threshold* and the duration of the timer. If the *threshold* is set too high, MDA will not react quickly enough to mobility based losses. However, a very small *threshold* may cause misinterpreted losses, and hence incorrect repair responses. The timer value should be set fairly high for two reasons. First, if congestion is low, a node may simply be sending at a slow rate. Second, during congestion it will be difficult to hear from a node to determine whether it has moved. However, a very large timer will make it difficult for MDA to react quickly to route failures, since the worst-case reaction time is equivalent to the threshold times the timer value. We evaluate a wide range of timer settings in our simulations to determine the appropriate value.

6.3 Simulation Methodology

Our main objective in evaluating MDA is to determine how much it can reduce routing overhead and increase application throughput. To properly evaluate throughput, we use ATP, a transport protocol with a congestion control algorithm designed especially for ad hoc wireless networks. Because ATP is new and still has some flaws, this has the effect of limiting the throughput gains that MDA can achieve. However, ATP is the best available choice at this time. Future transport protocols may take better advantage of the limited available bandwidth in wireless networks, and in this case MDA's performance enhancements should be even greater.

In our simulations, we use the $ns2$ simulator [71], version 2.28. For routing protocols we use the default implementations for DSR (with minor, bug-fixing changes) and AODV. Because no version of ATP was available, we implemented ATP in $ns2$ based on the original publication [19]. In our experiments all nodes communicate using IEEE 802.11 at the MAC layer, with a $250m$ radio range, free-space radio signal propagation, and a maximum data rate of 2 Mbps. We run each simulation for 600 seconds and take the average of 5 experiments for each data point.

Unless otherwise specified, MDA uses a credibility timer of 1 second. Most simulations use a threshold of 1 or 2, though in one case we use a threshold as high as 10.

In our simulations we collect the following metrics:

- **Route Failures**: The total number of route failures reported during the simulation. MDA should reduce the number of route failures since it suppresses sending notifications of congestion-induced packet loss to the routing protocol.
- **Routing Overhead**: All control messages originated or forwarded by any node, converted to Kbps. In most cases we report total routing overhead, summed over the entire simulation. We also use instantaneous overhead, calculated with

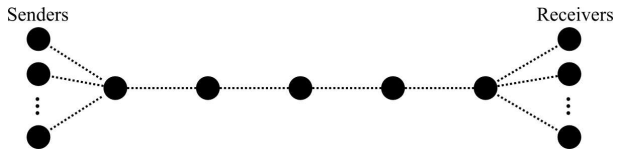


Figure 6.2: Topology for Congestion Scenario

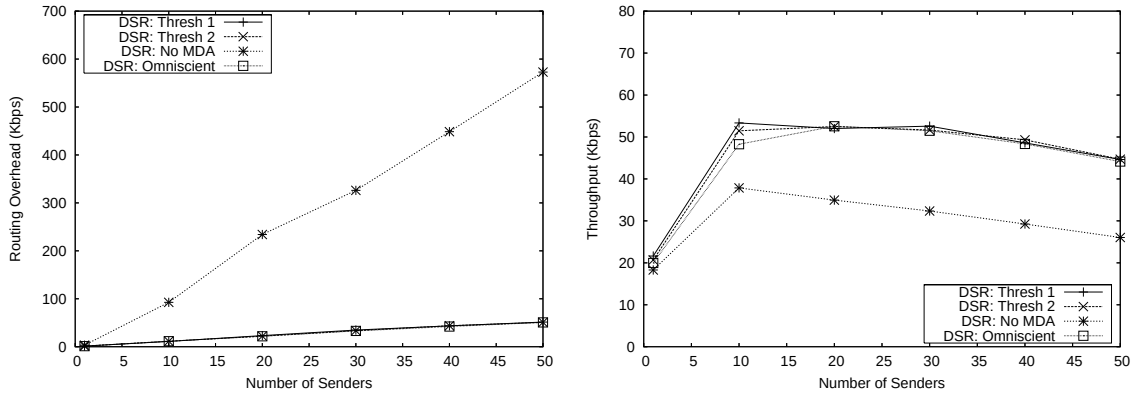
an Exponentially Weighted Moving Average that is updated every second. For DSR, control packets consist of ROUTE REQUEST, ROUTE REPLY, and ROUTE ERROR messages. For AODV, control packets consist of ROUTE REQUEST, ROUTE REPLY, and ROUTE ERROR messages.

- **Throughput**: For congestion-controlled flows we report the total throughput summed over all flows in Kbps. We calculate instantaneous throughput using an EWMA that is updated every second.
- **Packets Delivered:** For constant-rate flows we report the total number of packets received per second. We calculate instantaneous packets delivered using an EWMA that is updated every second.

6.4 Detecting Congestion-Induced Loss

To determine how well MDA detects congestion-induced loss, we simulate a scenario that isolates congestion from mobility. Shown in Figure 6.2, this scenario consists of a static *dumbbell* topology, where a group of senders and a group of receivers are located on either side of a linear connection of five nodes. The nodes are spaced far enough apart so that communication must occur over the shared path. We establish 50 senders on the left to transmit data to corresponding receivers on the right.

In this scenario, MDA works exactly as planned and never reports any route failures because all packet loss is caused solely by congestion. Without MDA, DSR reacts to as many as 2,000 route failures and AODV assumes as many as 10,000 route failures during the 10-minute simulation. As a result, MDA dramatically reduces



(a) Routing Overhead (b) Throughput

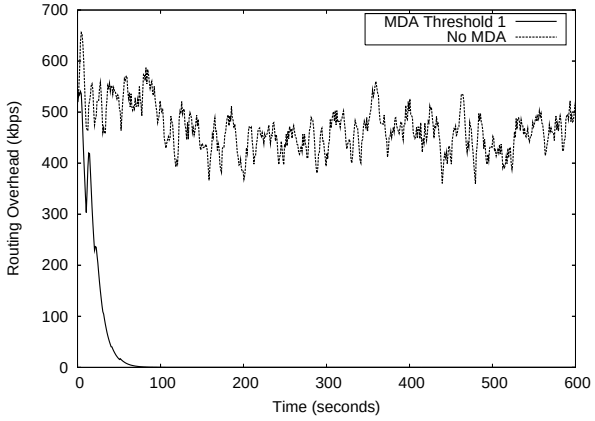
Figure 6.3: Congestion Scenario with ATP flows running over DSR

routing overhead, as shown in Figure 6.3. This in turn increases throughput for DSR by 50%. We see similar performance improvements for AODV, with throughput gains as high as 100%.

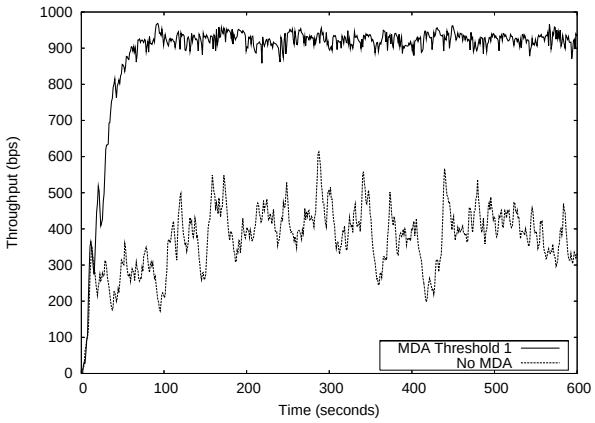
From these results we observe that a threshold of 1 is sufficient for MDA to properly detect congestion-induced loss. Higher thresholds do not significantly improve performance in this case because losses due to congestion can be reliably detected, and it takes only one such detection to reduce credibility and suppress notification of route failures.

To confirm that MDA provides consistent help to ATP, we plot the behavior of one of the 50 ATP flows over time in Figure 6.4. When used with MDA, DSR's routing overhead falls to zero, which causes ATP's instantaneous throughput to double and become more smooth. We observe similar results for CBR flows using several different transmission rates.

To illustrate that these performance improvements are not simply a product of a special case topology, we also simulate a static network of 50 randomly-placed nodes. This scenario again emphasizes congestion, since there is no mobility, but the congestion is spread out over the entire network. As shown in Figure 6.5, using MDA

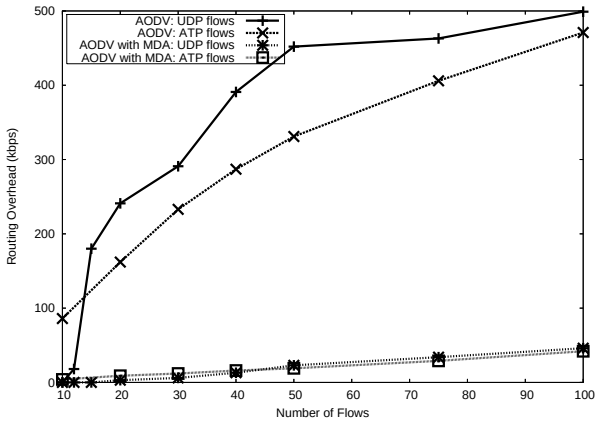


(a) Instantaneous Routing Overhead

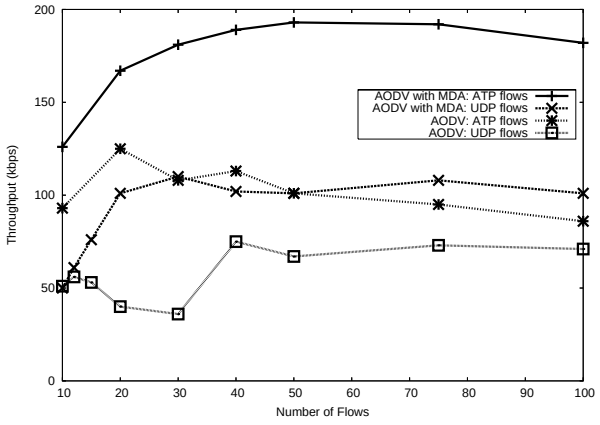


(b) Instantaneous Throughput for One Flow

Figure 6.4: Congestion Scenario with 50 ATP flows running over DSR



(a) Routing Overhead



(b) Throughput

Figure 6.5: Random Static Network with AODV flows

reduces AODV routing overhead by a factor of 10 and doubles ATP throughput. Similar gains are achieved for CBR flows.

6.5 Detecting Mobility-Induced Loss

To determine how well MDA detects mobility-induced loss, we simulate a scenario that emphasizes mobility over congestion. Shown in Figure 6.6, this scenario consists of a pair of a source and a destination; the pair move synchronously so that they periodically use a different route. The topology also consists of four parallel routes, *Route1-Route4*, each with four hops. The nodes are spaced far enough apart so that

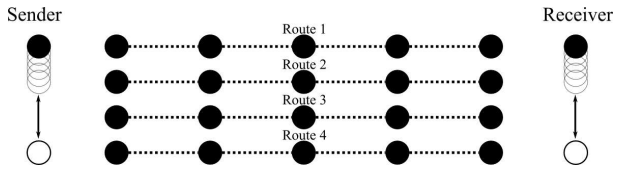


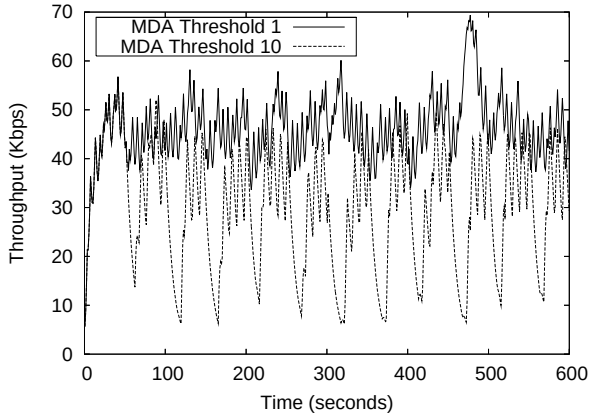
Figure 6.6: Topology for Mobility Scenario

communication must occur over these routes and so that the routes are completely isolated from each other. For experiments with this scenario, the source and destination cycle through each of the available routes every 50 seconds, starting at *Route1* and moving through *Route4* and back. In order to avoid congestion, the source transmits at a very low rate of 4 packets per second, with a packet size of 64 bytes.

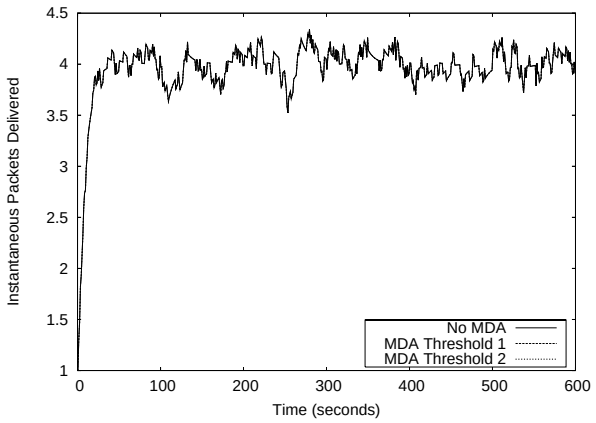
The goal of this scenario is to ensure that MDA does not adversely affect throughput by preventing the routing protocol from reacting to route failures. As shown in Figure 6.7(a), MDA is able to maintain high throughput for ATP, as long as the threshold is low. With a threshold of 1, MDA ensures that ATP gets the same throughput as without MDA (not shown). Results are similar for thresholds of 2 and 3. With a threshold of 10, however, MDA takes too long to identify mobility-induced losses.

The poor throughput with a threshold of 10 occurs because ATP tries to aggressively use all available throughput, causing congestion-based packet loss. MDA detects this loss and reduces the credibility to zero. The higher the threshold, the longer it takes for MDA to increase the credibility back to the point where it will begin notifying the routing protocol of a route failure. ATP throughput drops severely each time the route changes (every 50 seconds) because it takes it takes at least 10 seconds (1 second for each credibility level) to notify the routing protocol of the route failure.

The ability of MDA to properly detect mobility-based losses is better illustrated with a constant-rate flow sending at a low rate. Figure 6.7(b) shows the packet delivery ratio for a CBR flow sending 4 64-byte packets per second. The throughput



(a) ATP Throughput



(b) UDP Packet Delivery Ratio

Figure 6.7: Mobility Scenario with ATP or CBR flow running over DSR

is exactly the same, regardless of whether MDA is used or not, and regardless of the MDA threshold setting. Because the flow never causes congestion, MDA correctly interprets all packet loss as mobility-induced. Whenever the MAC layer loses a frame, MDA does not hear from the destination node within the allotted time interval, so credibility is never reduced below the threshold. This clearly illustrates that MDA does not interfere with the routing protocol's ability to react to route failures and discover a new route.

6.6 Differentiating Between Congestion and Mobility

To test MDA more completely, and to test its ability to differentiate between congestion and mobility, we generate a random topology of 100 nodes in a square field of $(1000m)^2$. We then vary both mobility (speeds from 0 to 30 m/s) and congestion (number of senders from 0 to 100). Our results that isolate mobility and congestion confirm those of our previous two scenarios, so we omit them. Here we report results for the general case where all nodes move using the random Waypoint mobility model, with a speed of 20 m/s and a pause time of 10 seconds. We then vary the number of ATP or CBR flows from 0 to 50. CBR flows send 10 packets per second with a packet size of 64 bytes.

Because this scenario includes both congestion and mobility, we introduce a new metric, correct route failure decisions. To calculate this metric, we use an *omniscient* version of MDA that is aware of the location of all nodes at every instant. The omniscient protocol can determine, for each packet loss, whether the destination node is in range of the source node, and thus accurately ascribe the loss to either mobility or congestion. We then report the percentage of correct decisions made by the standard version of MDA.

Figures 6.8 and 6.9 show that MDA again provides performance improvements for both DSR and AODV. Although MDA makes correct route failure decisions only

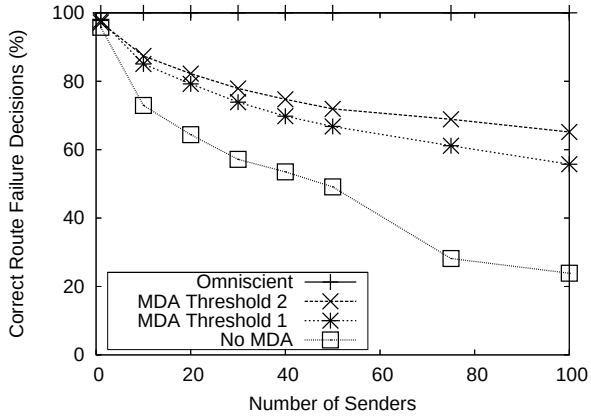
about 60% of the time, it still reduces routing overhead and improves throughput to levels that are nearly as good as the omniscient version. Throughput gains are more modest when mobility is the dominant factor, since the routing protocol is doing the right thing even without MDA. Improvements are more evident at higher loads, with throughput gains as high as 300%.

MDA's incorrect route failure decisions stem from how it handles loss due to mobility. When nodes are mobile, there will be many times when MDA's credibility metric is at the threshold, in order to react to mobility properly. At these times, MDA will make a mistake on the first frame that is lost due to congestion and assume it is instead due to mobility. The higher MDA's threshold, the better it will properly categorize congestion losses, but the longer it will take to recognize mobility. Likewise, there will be times when MDA's credibility is low due to congestion, causing it to make a mistake on packets lost due to mobility. As shown in these results, MDA's imperfection does not have a significant impact on system performance.

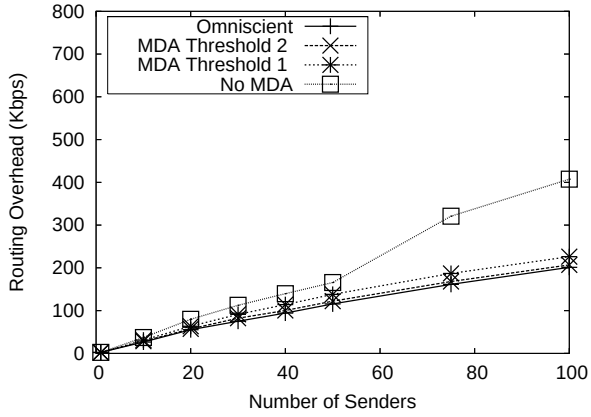
We achieve similar results for constant-rate flows in the random scenario. As congestion increases in the network, both DSR and AODV make correct route failure decisions only about 5 to 20% of the time. MDA makes the correct decision about 60% of the time. This significantly decreases routing overhead, but provides only a small gain in the packet delivery ratio. This is because a flow that is not using congestion control can artificially increase its packet delivery ratio by sending as fast as it can, while ignoring packet loss.

6.7 MDA Timer Setting

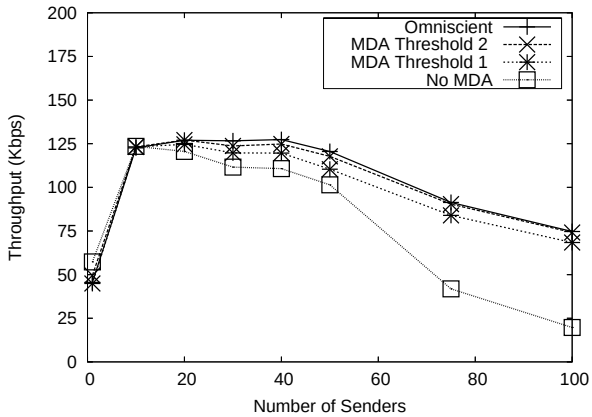
To determine the appropriate setting for MDA's timer, we repeated the random scenario with a range of timer settings. Figure 6.10 shows the routing overhead and throughput for 50 ATP flows in the random scenario, but with no mobility. For timer values below 1 second, MDA is not able to hear from the destination node in time,



(a) Correct Route Failure Decisions

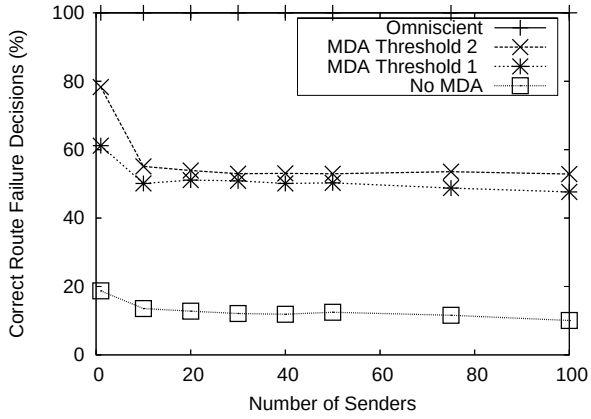


(b) Routing Overhead

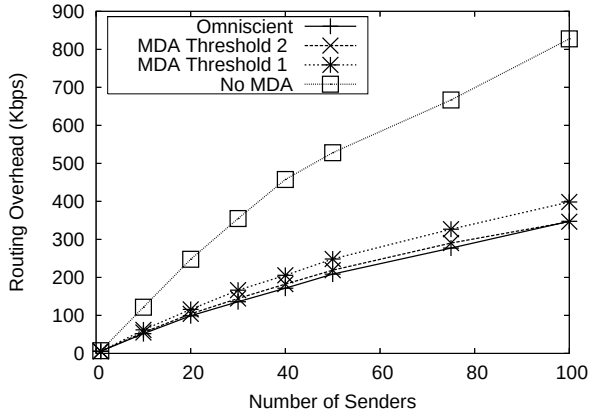


(c) Throughput

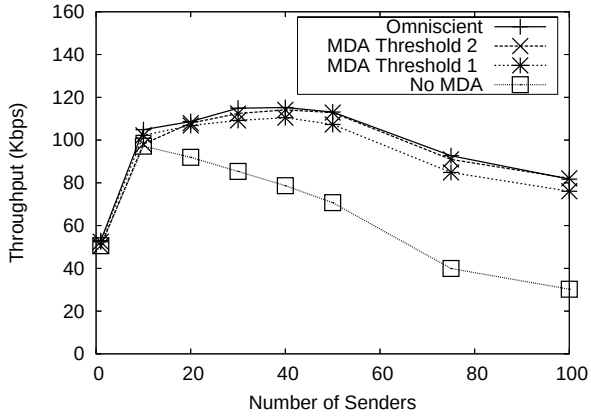
Figure 6.8: Random Scenario with ATP flows over DSR



(a) Correct Route Failure Decisions

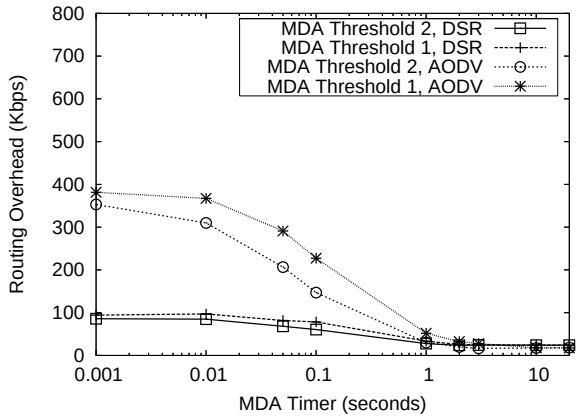


(b) Routing Overhead

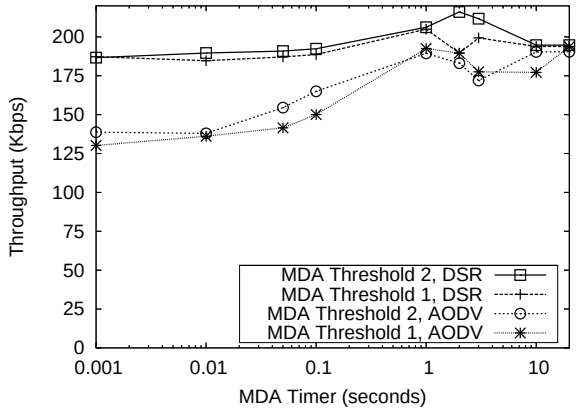


(c) Throughput

Figure 6.9: Random Scenario with ATP flows over AODV



(a) Routing Overhead

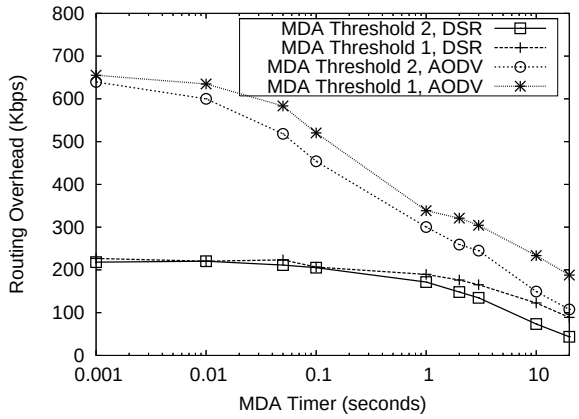


(b) Throughput

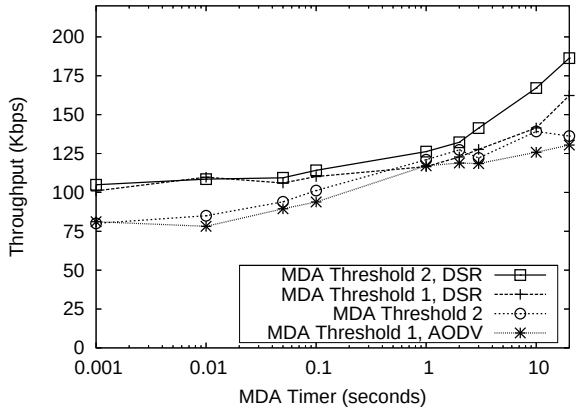
Figure 6.10: Static Random Scenario with ATP flows

so it frequently and incorrectly notifies the routing protocol of a route failure. This substantially routing overhead, particularly for AODV, and correspondingly reduces throughput. Both the routing overhead and the throughput level off for timer values larger than 1 second.

Figure 6.11 shows similar results for 50 ATP flows in the random scenario, where all nodes move as before with a speed of 20 m/s and a pause time of 10 seconds. The difference with this data is that, because the nodes are mobile, high MDA timers continue to reduce routing overhead and improve throughput. The problem here is that a high timer value prevents the routing protocol from reacting to legitimate



(a) Routing Overhead



(b) Throughput

Figure 6.11: Mobile Random Scenario with ATP flows

route failures, which of course reduces routing overhead. This also increases overall throughput because fewer flows are competing for the available bandwidth.

These results indicate that our timer setting of 1 second works well. If desired, a shorter timer of $\frac{1}{3}$ or $\frac{1}{2}$ second could reduce the latency required to react to route changes, while sacrificing some of the improvement in routing overhead and throughput.

6.8 Cross-Layer Architecture

To demonstrate the utility of MDA as part of a cross-layer architecture, we implemented several additional route failure detection mechanisms for AODV in *ns2*. The mechanisms we use include HELLO messages, passive acknowledgments, and explicit network-layer acknowledgments. We then repeated our above experiments with each of these mechanisms. While some mechanisms, such as HELLO messages, lead to substantially longer latency in detecting a failed route, in most cases MDA was able to reduce routing overhead and increase throughput, as in our previous results. To avoid repetition, we do not include the detailed results here.

6.9 Related Work

One alternative to MDA is to use signal strength measurements to determine whether a node has moved [75, 76]. In this work, a node may initiate a search for an alternate path when a next hop along the current path begins to move out of range, as determined by signal strength. One of the complications is that the mechanism must compensate for variations in signal strength due to fading, multi-path effects, or power conservation mechanisms. This approach is shown to significantly decrease the number of broken paths and to improve packet latency for TCP.

While our work focuses on routing protocols, there has also been a great deal of work on helping transport protocols react properly to lost frames. Most of the work in this area takes an end-to-end approach to solving this problem and does not interact directly with the MAC layer at each node. For example, ATP [19] ignores dropped frames as a sign of congestion, and the sender instead collects explicit rate information from each node along its path to the destination. One enhancement of TCP [18] uses out of order packets detected by the receiver as a sign of a route change caused by mobility, allowing TCP to respond differently to these losses. TCP Casablanca [77]

assigns priorities to packets, and assumes that lower priority packets are dropped first. A receiver can then use the dropping pattern that it observes to determine when loss is due to congestion and when it is due to mobility. Cen et al. explore a range of end-to-end mechanisms for distinguishing congestion from other types of packet loss [78]. As an exception to this approach, TCP-ELFN [79, 80] interacts directly with the routing protocol. When TCP-ELFN receives a route failure notification from the routing protocol, it freezes its congestion window and its state and then restarts once a new route has been installed.

6.10 Conclusion

In this paper we have demonstrated the benefits of using a cross-layer Mobility Detection Algorithm to determine whether a lost frame in a wireless network is due to mobility or congestion. By determining when a lost frame is truly a sign of a route failure, MDA significantly reduces routing overhead and can increase throughput by 10 to 100%, depending on the routing protocol and the mobility scenario. MDA can be used with any mobile ad hoc routing protocol, even those that use passive acknowledgments or HELLO messages to maintain routes.

We plan to work in several additional related areas. First, we plan to test MDA with other congestion control algorithms. Many of our results show a dramatic decrease in routing overhead, with significant but more modest gains in throughput. It is possible that a different congestion control algorithm may be able to take better advantage of the reduce routing overhead afforded by MDA. Second, MDA currently considers only mobility and congestion as sources of loss. Frames may also be lost due to interference from other technologies such as Bluetooth, microwave ovens, and cordless phones. A complete architecture should also detect interference from these sources and use alternative frequencies or other methods to avoid the interference when possible.

Chapter 7

Hop-by-Hop Congestion Protocol (HCP)

This chapter is an extended version of the paper published as "Hop-by-Hop Multicast Transport for Mobile Ad Hoc Wireless Networks", by Manoj Pandey and Daniel Zappala, The Fifth IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS), 2008, Atlanta.

Multicast transport is a challenging problem because the source must provide congestion control and reliability for a tree, rather than a single path. This problem is made even more difficult in mobile ad hoc networks due to problems caused by contention, spatial reuse, and mobility. To address these problems, we design a hop-by-hop multicast protocol, **H**op-by-Hop **C**ongestion **P**rotocol or HCP, which pushes transport functionality into the core of the network. Although this approach requires per-flow state, a hop-by-hop approach simplifies congestion control, enables local recovery of lost packets, and provides an efficient use of wireless capacity. We use a simulation study to demonstrate the effectiveness of this approach and compare its efficiency to application-layer multicast.

7.1 Introduction

In this chapter we consider the problem of multicast transport across mobile ad hoc wireless networks. It is well known that it is difficult to provide unicast transport over multiple wireless hops, due to contention, spatial reuse, mobility, and other

challenges of wireless networks [81, 82]. Multicast transport is further complicated because a source must provide reliability and congestion control over a multicast tree, and different branches of the tree may lose different packets and may have different available bandwidths over time. Of primary concern with multicast is scalability – it is infeasible for a source to track state and receive feedback from all of the group members simultaneously.

Current approaches for solving this problem mirror the solutions tried on the Internet: (1) an end-to-end multicast transport layer built on top of an unreliable, network-layer multicast service [24, 83], and (2) an application-layer multicast overlay built using unicast transport connections between participating hosts [4, 84].

With a multicast transport layer, packets are sent to group members unreliably using a multicast routing protocol that operates at the network layer [10, 8]. The transport protocol then uses feedback from receivers in the form of ACKs or NACKs to regulate the source's sending rate and to provide end-to-end reliability [2, 85, 3]. The benefit of this approach is that network layer multicast is very efficient, since a tree can be built that minimizes hop count or power consumption. The downside is that it is very difficult to provide an efficient and scalable transport protocol that can manage the differing loss rates and congestion on the tree.

Application-layer multicast avoids some of the complexities of multicast transport by building an overlay using TCP (or other unicast transport protocol) connections [62, 64, 86, 66, 65, 63]. The advantage of this approach is that reliability and congestion control only needs to be provided between pairs of hosts, rather than for a whole tree at once. However, the distribution tree usually imposes greater delay (called stretch) and higher bandwidth usage (called stress) as compared to a tree built at the network layer [4]. Because capacity and power are constrained in an ad hoc network, inefficiencies due to both stress and stretch should be avoided.

In this chapter, we take a different approach by designing a hop-by-hop multicast transport protocol. Rather than pushing multicast transport to the edges of the network, where it becomes either complex or inefficient, we push transport functionality *into* the network. With hop-by-hop multicast transport, every node in the multicast tree performs both congestion control and reliability. Whenever a node on the tree forwards a packet, it uses credit-based congestion control to ensure that it does not overflow the packet queues of its children. In addition, each node on the tree caches packets for the application, and can repair losses for any group members that are downstream.

The hop-by-hop approach has been previously studied for providing unicast transport on both the Internet and ad hoc networks [87, 6]. The end-to-end approach has long been preferred for transport protocols because it simplifies the core of the network. Hop-by-hop protocols typically require per-flow state in routers [88, 89], and this is difficult to scale to large numbers of flows. However, recent work shows that this may be feasible in core Internet routers [90], and this should certainly be less of a concern in ad hoc wireless networks, since they will typically have far fewer flows to handle.

While no variant of a hop-by-hop transport protocol has been proposed for multicasting in ad hoc wireless networks, this approach for multicasting has a lot of promise. This chapter examines and evaluates its advantages. We begin by outlining a few of these advantages:

- *Efficient use of network bandwidth*: End-to-end protocols typically send explicit ACKs or NACKs; both RALM[2] and ReACT[3] use one of these forms of end-to-end signaling. However, sending these packets consumes network bandwidth, which is usually small for ad hoc networks. In hop-by-hop approach, we use feedback from the MAC layer at each hop to determine whether a given packet was received by the next hop and eliminate the need for sending ACKs.

- *Local Recovery via buffering*: The ability of HCP to use buffering at each intermediate node, allows for a faster local recovery. When receivers request retransmission, they forward the request towards the source. On the way, each intermediate node examines this message; if an intermediate node has the requested packets in their buffer, then they retransmit it to the receiver. Thus the source does not need to retransmit every lost packet, which leads to lower load on the network.

- *Faster response time*: A hop-by-hop approach is attractive because it provides faster response to congestion than that of an end-to-end transport. This method reacts more quickly to congestion since all decisions are made locally at each hop. The primary goal of each node is to avoid sending extra data that the next node cannot buffer; this regulation greatly simplifies the design of the congestion control algorithm.

- *Exploiting receiver heterogeneity*: Most of the existing multicast protocols fail to address an important requirement – different receivers can have different available bandwidths; these existing protocols often select a rate suitable for the lowest bandwidth receiver [2, 3, 27], and others group receivers into rate sub-trees [28]. A hop-by-hop approach is uniquely able to send data at different rates to different receivers since it uses buffering to enable receivers to obtain different rates for the same multicast source, rather than requiring the source to send at the speed of the slowest receiver. If a node on the multicast tree receives data faster than it can send it, rather than telling its parent to slow down, it caches the packets until it can send them. The size of the cache determines how much of a mismatch there can be between different reception rates. If there is a large difference between the incoming rate and the outgoing rate, then the cache will eventually fill, causing the parent node to slow down.

This behavior is very relevant for ad hoc networks, since different regions can experience varying degrees of congestion. Varying degrees of congestion can be caused due to varying network traffic, node density, and interference. The assumption that all receivers have similar bandwidth and hence there should be one global sending rate, which is usually of the low-bandwidth receiver, can potentially be unfair to high-bandwidth receivers.

These reasons motivate us to design a Hop-by-Hop Congestion Protocol, HCP. In addition to the above advantages, HCP is also designed to have its own unique advantages. First, it uses a combination of hop-by-hop and end-to-end reliability; it uses MAC layer reliability at each hop and on top of that provides end-to-end reliability. Second, HCP uses feedback from the MAC layer to estimate the correct sending rate. Third, HCP uses fair queuing at each hop to ensure fairness among the flows passing through an HCP node. Lastly, HCP also integrates well with ASSM to provide state-setup at each hop in the multicast tree and to request lost packets.

Our simulations demonstrate that HCP is effective in providing both reliability and congestion control. HCP delivers high throughput and fairness if there are multiple co-existing multicast groups that compete for the same bandwidth. We also run simulations to understand the behavior of limited buffer space on the performance of HCP; our results indicate that a buffer as small as 100 kilobytes is sufficient for HCP. We then compare HCP to application-layer multicast using an overlay of reliable connections. Because HCP is built on top of shortest-path routing, it is able to provide greater efficiency and much higher throughput under varying conditions of traffic and mobility.

7.2 Related Work

While the hop-by-hop approach has not previously been used for multicast transport in ad hoc networks, recent research has designed hop-by-hop congestion control algorithms for unicast flows in wireless networks. One approach uses congestion pricing to regulate the rate of flows in a mesh network [91]. Other protocols have been developed for sensor networks, using back pressure [92, 93] or rate-based congestion control [94, 95] to control the sending rate of each sensor. The foundation of this work is rate-based and credit-based congestion control for high speed networks [88, 89].

In ad hoc networks, several multicast transport protocols have been designed to run on top of a best-effort multicast routing protocol. RALM [2] sends a window of packets and selects one receiver to provide feedback. The receiver returns an ACK if it has all of the packets, or a NACK if it is missing some. The source retransmits any missing packets to the entire group, one-by-one, then moves on to the next window of packets. ReACT [3] uses the same mechanism as RALM and adds local recovery, so that a receiver can get missing packets from any other receiver on the path between itself and the source. It also tries to distinguish between contention and congestion losses so that the the source only slows down due to congestion. In CALM [85] a source sends at an application-specified rate, and receivers send a NACK if they miss a certain number of consecutive packets. Upon receiving a NACK, the source retransmits a missing packet and asks a targeted receiver to reply with an ACK. It then moves on to the next receiver until all have responded with an ACK, then reverts to the initial rate. All of these protocols slow down severely when congestion occurs, by limiting the rate of the source to one packet at a time.

In contrast to this approach, a number of protocols use application-layer protocols to provide multicast transport, by building an overlay of unicast transport connections. ALMA uses receiver-oriented methods to build a tree of group members, reconfiguring the tree whenever mobility or congestion occurs [64]. Additional

mechanisms are used to prevent loops and to ensure data continuity when the tree changes. Both AMRoute [86] and PAST-DM [62] build a mesh of connections to connect the group members, then construct a shared tree using the mesh. Recent work has investigated the use of the Scribe structured DHT for multicast in ad hoc networks and makes some recommendations to improve its performance [63].

Other research has developed reliable multicast protocols that implement only reliability, without congestion control. RMA requires every receiver to ACK every packet sent by the source [96], which can lead to ACK implosion. An Anonymous Gossip provides local recovery by having receivers exchange pairs of lost messages with each other [97]. ReMHOC [98] is patterned after SRM [24]; a receiver sends NACKs that can be answered by any other group member. Recent research makes the case for using FEC to provide reliability for multicast in wireless mesh networks [99].

Tang and Gerla use a combination of a reliable broadcast MAC protocol and some modifications to ODMRP to provide multicast congestion control without reliability [100]. The reliable MAC ensures that it never overflows a downstream queue when it broadcasts a message. If queues become too large, ODMRP carries an ECN message to the source to tell it to slow down. This scheme does not guarantee reliability because the MAC protocol reverts to unreliable broadcast when contention is high, and there is no way to retransmit packets when they are lost due to contention or mobility.

7.3 HCP Architecture

To provide hop-by-hop multicast transport, we design HCP architecture as shown in Figure 7.1. As packets arrive from the application layer or from a parent in the multicast tree, they enter the core of HCP, which implements congestion control and reliability. This component provides reliability and local recovery by keeping a

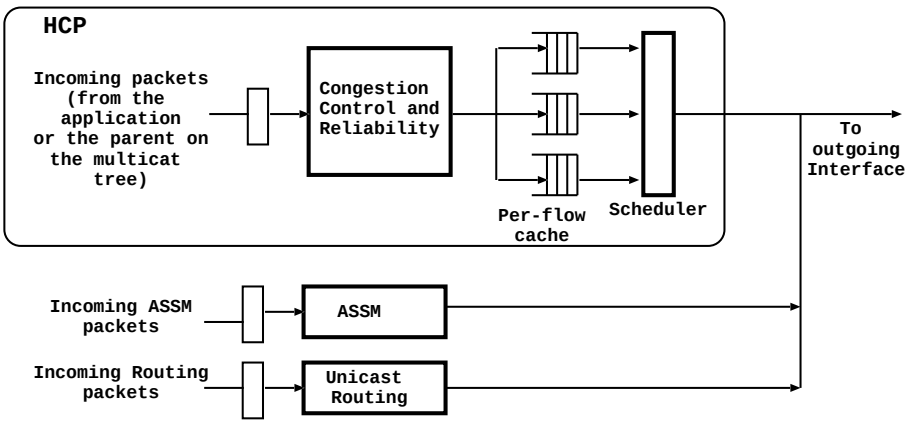


Figure 7.1: HCP Architecture

separate buffer of packets for each flow. This component also provides congestion control, using a module called, Scheduler, to ensure that it sends data at a correct rate. If any packets arrive for HCP that do not have associated per-flow state, they are discarded. Further, the Scheduler uses fair queuing [101] to determine the order in which packets are sent from each of these per-flow buffers. This ensures fairness among all of the multicast flows traversing this node. Fairness over wider areas is addressed by existing research [102] and is out of scope for this paper.

7.3.1 State Setup and Forwarding

HCP interacts with ASSM in several ways; two of these interactions assist HCP in state setup and forwarding. First, HCP uses ASSM to establish per-flow buffers and state at each node on a multicast tree. When a multicast receiver sends a Join message for the first time, it includes an *initialization* container in the Join message. As the Join message travels upstream, ASSM contacts HCP at each hop and delivers this container, instructing HCP to establish state for the flow if it doesn't already exist and also to allocate application-layer buffer. In Figure 7.2, we provide Psuedocode that outlines the initialization process at HCP forwarders; this process creates an entry in the Scheduler and provides APIs for buffering and reading data. When forwarders

1 function **initForwardingState**(S, G) :

```
1: /*
2: * When a new state is setup, HCP forwarder allocates a buffer, inserts a
3: * record in the Scheduler and adds the file descriptor in the entry
4: */
5: ...
6: fd = allocate(size);
7: Scheduler.addEntry(S, G, fd);
8: ...
```

function **forwardingStateGetGroupSpecificData**$(S, G, data, datasize)$:

```
1: /*
2: * Get the record in the Scheduler. From the Scheduler, get
3: * the file descriptor. Read from the file descriptor.
4: */
5: index = Scheduler.getEntry(S, G);
6: data = Scheduler[index].fd.read(start, datasize);
7: return (data);
```

function **forwardingStateAddGroupSpecificData**$(S, G, data, datasize)$:

```
1: /*
2: * Get the record in the Scheduler. From the Scheduler, get
3: * the file descriptor. Write to the file descriptor.
4: */
5: index = Scheduler.getEntry(S, G);
6: Scheduler[index].fd.write(start, datasize, data);
```

Figure 7.2: HCP Forwarder Node

need to buffer a packet or send a packet for a given group, they use this entry to do so.

The state that ASSM creates is "soft state" and it times out if HCP does not refresh it periodically by sending Join messages. If a receiver wants to leave the tree, it can stop refreshing the Join messages and the state on its branch eventually times out. To leave the group more quickly, a node can send a Prune message upstream to immediately delete the state on its branch.

The second interaction with ASSM appears in the form of forwarding packets on the multicast tree. When a new packet arrives, HCP determines how to forward the packet by getting the forwarding entry from ASSM. This entry contains a list of child nodes to which it should send the packet. Usually, multicast packets are forwarded using broadcast, but this causes multicast to suffer a high loss rate when it must contend with TCP traffic, which uses the RTS/CTS exchange. Instead, HCP uses a form of semi-reliable broadcast. Each time HCP forwards a packet, it chooses one of the children and forwards it the packet using the same RTS/CTS exchange as a unicast packet. This ensures that one child receives the packet reliably, and the other children attempt to receive the packet through promiscuous listening. As packets are forwarded, each child will get some packets reliably and other packets by snooping.

A fortunate consequence of using semi-reliable broadcast is that HCP can detect when a child moves away, since the RTS/CTS exchange will fail. In this case, HCP removes the forwarding state for that child. If this is the only child downstream from this node, HCP leaves the rest of its state in place and instead waits for a timeout before it deletes the state. This allows the route to be repaired without losing all the packets in the parent node's application-buffer.

7.3.2 Scheduling and Fairness

```
1 function sendNextPacket() :
 1: /*
 2: * If there are packets for retransmission, then decide whether the packet
 3: * needs to be new data or a retransmission packet.
 4: */
 5: if (retransmissionBuffer.getSize() > 0) then
 6:     /* Returns a number between 0 and 100 */
 7:     randomNumber = random() * 100
 8:
 9:     if (randomNumber < CONFIG − RETX − PERCENT) then
10:         /* Send Packet from Retransmission Buffer */
11:         forwardPacket(retransmissionBuffer.head());
12:         retransmissionBuffer.removeHead();
13:         retransmissionBuffer.size − −;
14:         return;
15:     end if
16: end if
17:
18: /* Forward the Packet */
19: forwardPacket(getNextNewPacketFromBuffer())
20: return;
```

Figure 7.3: Scheduler: Callback Function

HCP uses the Scheduler component to estimate the correct sending rate at every hop. The Scheduler depends upon the feedback from the MAC layer to deliver packets. The MAC layer continuously observes the IP queue-size and if the queue-size becomes less than a given threshold size, IP-QUEUE-THRESHOLD, then the MAC invokes a callback function provided by Scheduler. This callback triggers the Scheduler to send the next packet. However, the Scheduler delivers only one packet at a time to prevent the IP queue from overflowing. For our implementation, we set the value of IP-QUEUE-THRESHOLD to 2.

Figure 7.3 provides the Pseudocode for Scheduler's callback function *sendNextPacket*(). In this Psuedocode, CONFIG-RETX-PERCENT represents the percentage of forwarding packets that can be retransmission packets. In our implementa-

```
1 function assignTokenToNextFlow(passedCounter) :
 1: /*
 2: * Get the next record after the current record that has the token. Assign
 3: * token to the next record.
 4: */
 5:
 6: index = getNextEntryWithValidFlows(Scheduler, passedCounter);
 7: Scheduler[index].hasToken = TRUE;
 8: return;


  function getNextNewPacketFromBuffer() :
 1: /* Find the entry in the Scheduler that has the token */
 2: index = Scheduler.getEntryWithToken();
 3:
 4: if Scheduler[index].isActive then
 5:       assignTokenToNextFlow(index);
 6:       return (Scheduler[index].fd.read(start, datasize));
 7: else
 8:       /*
 9:       * If the flow is inactive, then assign token to the next
10:       * flow and call this procedure again
11:       */
12:       assignTokenToNextFlow(index);
13:       return(getNextNewPacketFromBuffer());
14: end if
15:
16: /* Here, for the first time. It appears that no flow has the token. */
17: assignTokenToNextFlow(0);
18: return(getNextNewPacketFromBuffer());
```

Figure 7.4: Scheduler: Fairness Algorithm

tion, we keep this parameter to 50; thus, providing equal weightage to both first-time packets and as well as retransmission packets.

The implementation for Scheduler's fairness mechanism is simple. As shown in Pseudocode 7.4, the Scheduler uses a token-based system to poll a flow for sending a packet. When the MAC layer invokes $sendNextPacket()$, the Scheduler either sends a retransmission packets or uses $getNextNewPacketFromBuffer()$ to get packet from the flow, which holds the token. Then it uses $assignTokenToNextFlow()$ to handover the token to the next eligible flow. In token-assignment, we only consider those flows that are active and have packets to send; we render a flow inactive, if its multicast state has not been refreshed recently by ASSM Join message.

7.3.3 Reliability

For data forwarding, HCP uses a form of semi-reliable broadcast. When HCP forwards a packet it randomly chooses one of the children and forwards the packet to the chosen child node using the same RTS/CTS exchange as a unicast packet. This provides MAC layer reliability for data forwarding to that node. Other children attempt to receive the packet through promiscuous listening. As HCP forwards packets, each child receives some packets reliably and others by snooping.

However, with a semi-reliable broadcast, it is likely that some packets will be lost. To provide end-to-end reliability, HCP receivers monitor the sequence numbers of incoming packets, detect sequence number gaps, and request retransmissions from upstream nodes using NACKs. Each node on the multicast tree buffers the most recent packets it has forwarded and can respond to these requests.

To send a NACK, a receiver creates a *request* container that contains a list of the missing packets and inserts this container into an ASSM Join message (Figure 7.5). The list of packets is a set of tuples of the form (l_1, l_2), where l_1 is the first sequence number and l_2 is the last sequence number of the gap.

```
1 function receiverUpdateContainerForASSMJoin(packet) :

 1: /*
 2: * The receiver copies retransmission information in ASSM Join container.
 3: */
 4: assmHeader = getAssmHeader(packet);
 5: groupOwner = assmHeader− > groupOwner;
 6: groupAddress = assmHeader− > groupAddress;
 7:
 8: retxInfo = (retxInfo∗)getASSMJoinContainer(packet);
 9: index = Scheduler.getEntry(groupOwner, groupAddress);
10: tupleList = getMissedTuples(Scheduler[index]);
11:
12: /*
13: * If there are non-zero tuples appended, then need to specify the role
14: * type field of the ASSM header
15: * /
16: if length(tupleList) > 0 then
17:     updateASSMJoinContainer(retxInfo, tupleList);
18:     assmHeader− > roleType =RETRANSMISSION;
19: end if
20:
```

Figure 7.5: End-to-end Reliability: Sending ASSM Join

```
1 function
  forwarderOrSourceHandlerAssmLayerReceivingASSMJoin(packet) :

 1: /*
 2: * Upstream members (forwarders and source) use this routine. The
 3: * forwarder retransmits packets that it has in its application buffer;
 4: * the source retransmits the remaining packets.
 5: */
 6: assmHeader = getAssmHeader(packet);
 7: groupOwner = assmHeader-> groupOwner;
 8: groupAddress = assmHeader-> groupAddress;
 9: retxInfo = (retxInfo*)getASSMJoinContainer(packet);
10: index = Scheduler.getEntry(groupOwner, groupAddress);
11: tupleList = getMissedTuples(retxInfo);
12: fd = getApplicationBufferHandler(Scheduler[index]);
13: /*
14: * Loop over all the missed sequence numbers tuples specified in the
15: * of ASSM Join container and check if we have the tuple.
16: */
17: for each tuple in tupleList do
18:     if (fd.checkIfYouHaveThisTupleInBuffer(tuple)); then
19:         /*
20:         * Add this tuple to the Retransmission Buffer. Then clear
21:         * this tuple information from the ASSM container.
22:         */
23:         retransmissionBuffer.addTuple(tuple);
24:         removeRetxTuple(retxInfo, tuple);
25:     else if (fd.checkIfYouHaveThisTuplePartiallyInBuffer(tuple)); then
26:         /*
27:         * If The Buffer has only partial sequence numbers, then create a new
28:         * tuple, tupleNew, with those sequence numbers, which we do not
29:         * have. From ASSM container, remove the old tuple and add
30:         * the new one. Also, add tupleNew to retransmissionBuffer.
31:         */
32:         tupleNew = adjustRetxTupleWithPartialInfo(retxInfo, tuple);
33:         retransmissionBuffer.addTuple(tupleNew);
34:         removeRetxTuple(retxInfo, tuple);
35:         addRetxTuple(retxInfo, tupleNew);
36:     end if
37: end for
38: return
```

Figure 7.6: End-to-end Reliability: Processing ASSM Join

As the Join message travels upstream, each HCP node on the tree examines the container and determines if its application-buffer contains any of the requested packets present in the container (Figure 7.6). If the node does have one or more of these packets, then it schedules these packets for retransmission and removes the appropriate tuples from the list. If there are still tuples remaining, then it signals ASSM to continue forwarding the Join message upstream. In the worst case, the source receives the NACK and re-sends any missing data.

When retransmitting packets, an HCP node follows several policies. First, it always re-sends data using an RTS/CTS exchange to the child that sent it the request. This ensures that the packet makes it at least one step closer to the receiver that is missing the packet. Second, by setting to CONFIG-RETX-PERCENT parameter to 50, each node gives equal priority to first-time packets and retransmission packets. This allows receivers experiencing loss to get the retransmission, while not penalizing receivers without loss too heavily.

7.3.4 Congestion Control

HCP uses congestion control to estimate the correct sending rate at each hop by ensuring that an upstream node does not overflow the per-flow buffers at its children directly downstream on the tree. Each forwarder node maintains a count of available buffers for each of its downstream nodes for a given flow. Each time it successfully transmits a packet for that flow, it decreases this count. It transmits only if there is some space available with each of the downstream nodes. To communicate its available buffer space, a downstream node relies on passive feedback. When a downstream node forwards a packet to its own downstream nodes, it includes the current buffer space for the flow in the protocol header. When the upstream node overhears the packet, it is able to synchronize the buffer count for that child with this amount.

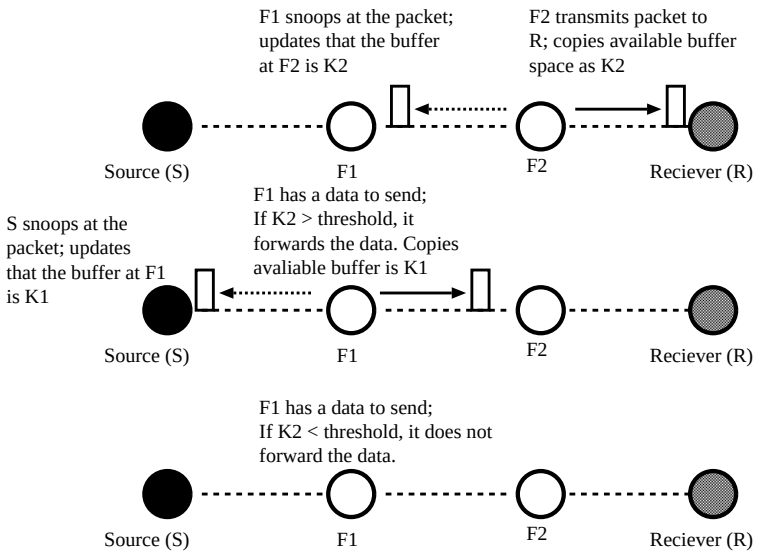


Figure 7.7: Buffer overflow avoidance scheme

Figure 7.7 illustrates the congestion control mechanism in detail. When an upstream node ($F1$) snoops the packet sent by its downstream node ($F2$), it reads the available buffer and updates this in its *Scheduler* table. Since by design, a majority of downstream nodes in HCP use snooping to listen to data packets, snooping is already configured with HCP. When the upstream node has data to send, it first verifies if the downstream node ($F2$) has available buffer space; if the available buffer with the downstream node is greater than a threshold, AVAILABLE-THRESH, it forwards the next packet. Otherwise, it waits and allows the downstream node to continue sending more packets and make additional room in its buffer. When the upstream node forwards a packet, it also decreases the available buffer space for its downstream nodes in the Scheduler by 1. HCP does this because if the downstream node successfully receives the packet, then its buffer space is reduced by 1.

For this mechanism, an important consideration is the definition of available buffer itself. HCP's buffer contains two types of packets: those that have been transmitted at least once and those that have never been transmitted. Due to limited space, a node may drop those packets that have been transmitted at least once since

there is a chance they may not be needed again. However, a node should never drop a packet it has not yet transmitted downstream. Hence, we refer to the number of empty buffers plus the number of packets that have already been transmitted, as the available buffer. We say that the buffer is full when it contains only those packets that have never been transmitted.

7.4 Simulation Methodology

We use simulations to evaluate HCP and to compare it to application-layer multicast in an ad hoc network. We have implemented all three modules of our architecture: MDA, HCP and ASSM in *ns*2, and use AODV [12] for the unicast routing protocol. Unless specified otherwise, all our simulations use a default data rate of 11 Mbps and a packet size of 1024 bytes. In addition, the default size for HCP's per-flow buffers is 100 KBytes, and ASSM uses a refresh period of 1 second. Lastly, except stated otherwise, we run all the simulations for 5 random seeds and then average the results.

To compare HCP to an application-layer overlay multicast, we build a simple overlay protocol using ATP [19] connections to connect group members. We use ATP rather than TCP, since ATP is designed to avoid many of the problems with unicast transport in ad hoc networks. ATP uses a rate-based congestion control algorithm that measures link characteristics on the forward path to set the proper rate. ATP also aggregates ACKs into epochs to reduce the overhead of per-packet feedback.

We implement two variants of our simple overlay: optimal and random. With an optimal overlay, we are interested in knowing the best possible performance of an overlay; however, this requires recalculating the overlay whenever any node moves, which is prohibitively expensive in real deployments. With a random overlay, the expected performance is the worst case, since nodes pick neighbors randomly rather than optimally, leading to poor performance. For real-life deployments of an overlay, performance should lie between these two extremes.

We build the optimal overlay in two steps. First, using the *Floyd-Warshall* algorithm, we build a logical mesh that connects all nodes in the multicast group, where the weight of each link in the mesh is equal to the shortest distance between those two nodes in the ad hoc network [103]. Next, using the *Kruskal* algorithm, we calculate a minimum spanning for the members from this mesh; the links in the spanning tree represent an ATP connection in the overlay. Lastly, we also reconstruct the overlay whenever a node moves, so that it maintains optimality over the lifetime of the multicast group.

We build the random overlay by choosing random pairs of group members and connecting them with ATP. This model is useful for two reasons. First, this approximates an overlay that has been perturbed over time by mobility. Second, in most of the today's peer-to-peer overlays, member have only a partial view of the other members of the group and so the overlay may not be optimal.

To evaluate HCP, we perform the following five sets of simulations.

- We begin by examining the unicast behavior of HCP using a multicast group with only one source and one receiver. This allows us to compare HCP to unicast transport protocols; we choose ATP as the unicast transport protocol.
- We next examine the multicast behavior of HCP. We study several multicast related properties of HCP: (a) the ability to retransmit missed packets when a given link breaks in the multicast tree, (b) the ability to estimate the correct sending rate when network bandwidth changes, and (c) the ability to share bandwidth fairly when multiple multicast groups co-exist.
- We then examine the impact of limited buffering available at intermediate nodes. If nodes have larger buffers, then when a group member moves and loses a packet, it is likely that an intermediate node will have the lost packets in its buffer. Thus, this intermediate node can handle retransmissions instead of the source. Even for a file transfer of $1GB$, it is likely that intermediate nodes have

enough disk space to buffer the entire file. However, for the case where nodes have limited buffer, HCP will be unable to buffer the entire file at intermediate nodes, and the source will have to handle some retransmissions. In this set of simulations, we examine the trade-off between buffering and retransmissions.

- We next evaluate HCP with both variants of the overlay: optimal and random. In this set, we provide higher traffic loads by varying both group size and the number of groups.
- Our final set of simulations evaluates HCP along with the above two variants of the overlay under conditions of mobility.

7.4.1 Metrics

We use a combination of several metrics along with packet and sequence traces to examine the simulation results. Our metrics are as follows:

- *Instantaneous throughput* is the current throughput rate for a receiver. For this estimation, we sum all the bytes received in a given second to calculate the instantaneous rate for that second. We repeat this calculation every tenth of a second. Instantaneous throughput is plotted only for one run; hence for simulations, where we measure instantaneous throughput, we run them only for one seed.
- *Combined throughput* is the total bytes received during the entire simulation by all receivers, divided by the duration of the simulation.
- *Delay* is the time taken by a packet to arrive at the receiver. To understand the behavior of the protocols better, we categorize delay into two types: (a) delay when the packet arrives for the first time and (b) delay when the packet arrives as a retransmission packet. For retransmission packets, delay is the time

difference when the packet arrives after retransmission at a receiver and the time when the packet was originally sent by the sender.

- *Packet Loss*: We categorize lost packets with four types: (a) mobility based losses, (b) passive losses, when downstream nodes fail to snoop, (c) contention losses, when the MAC layer tries the maximum number of times to transmit the frame and then drops the frame, and (d) congestion losses, when HCP receives a packet, but cannot keep it since its buffer is full. Unless stated otherwise, we plot loss as percent of total packets sent in the simulation.
- *Sequence Number Trace*: Besides these metrics, we also use sequence number traces. In a sequence number trace, we plot the sequence number received for a given node vs. time.

7.5 Unicast Results

We begin our simulations by comparing HCP with ATP. ATP is a transport protocol with a congestion control algorithm designed especially for ad hoc wireless networks. There are several reasons that motivate us to compare HCP with ATP. First, ATP uses a rate based congestion-control algorithm, which is different than that of HCP; HCP uses a credit based control, where it uses the buffer available at downstream node to limit the rate. Second, ATP uses an end-to-end feedback mechanism, sending a periodic ACK with information about the received rate. Thus, ATP serves as a good test for our hop-by-hop approach.

HCP is designed for multicast flows; however, as a boundary condition, any multicast flow can be reduced to a unicast flow if it has one source and one receiver. The advantage of using this unicast behavior is that it eliminates complexities introduced when there are multiple receivers or multiples sources. We explore this behavior in the next section.

The objective for this set of simulations is simple: to test if HCP can deliver better throughput than ATP and if it is fair to multiple co-existing flows. We do not include mobility for these simulations since we want to focus on congestion. Further, we run both HCP and ATP with MDA since this enables us to do a fair comparison.

7.5.1 Throughput for Unicast Scenario

For this test, we use a linear chain topology and vary the number of nodes in the chain. As the number of nodes in the chain is increased, nearby nodes experience increased collisions and increased contention. For the chain, we place 5 flows with the source at one end of the chain and the receiver at the other end of the chain; next, we measure the combined throughput for all these flows.

Figure 7.8 shows the output of this simulation where we plot the combined throughput for all 5 flows. Overall, both HCP and ATP deliver similar results. As the number of nodes in the chain increases, throughput for both ATP and HCP decreases. This behavior has been well-studied in literature [49, 81]. When the number of nodes in the chain increases, the degree of collision also increases. This increased rate of collision happens because there are more nodes in the neighborhood of each forwarding node. Hence, a node has to contend with a larger number of neighbors on each side of the chain. Due to increased contention, the throughput decreases substantially.

However, with fewer hops, HCP performs better than ATP. This poor performance of ATP can be explained due to its rate estimation algorithm. ATP assumes that the network is linear and so when the receiver reports an average delay of Δ seconds per packet, then ATP concludes that if it takes Δ seconds for one packet to go through, then for every 1 second epoch, it can send $1/\Delta$ packets. This is a good assumption when the rate is smaller, but when rate becomes higher, the assumption about the network's linearity does not hold. This is because at a lower rate, most of the times the MAC layer can send data by using only one transmission. However,

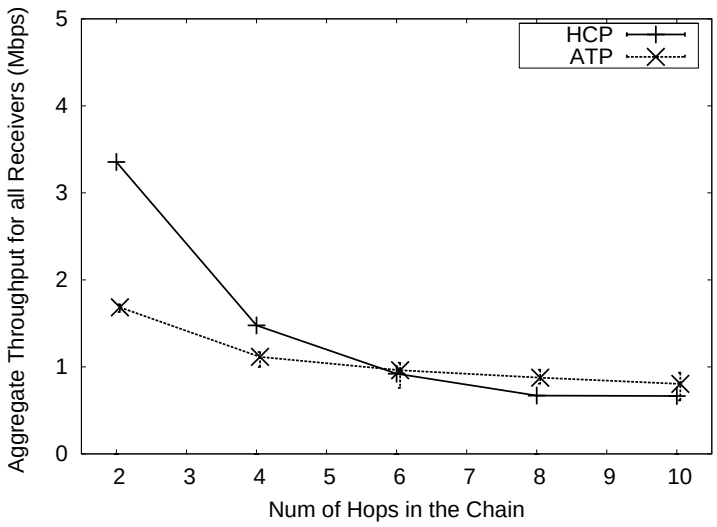


Figure 7.8: Throughput for ATP and HCP for Linear Topology Scenario

at a higher rate, the MAC layer might have to retry sending the same data several times, before it successfully transmits it. Further, the MAC layer also backs off using an exponential time whenever it has to retry. Hence, when ATP starts sending a large number of data, then the initial assumption that per packet delay is Δ, does not hold true. The ATP receiver has to wait for one full epoch to report this bad estimation for the current epoch. After one epoch passes, the ATP receiver reports a higher value of Δ (meaning a lower rate); thus, ATP's rate estimation is prone to fluctuation.

HCP's rate estimation is based on a simple feedback from the MAC layer; this feedback is of the order of milli-seconds unlike ATP's one second epochs. Whenever local contention increases, the MAC layer stops HCP from enqueuing any new data and thus eases the network load. When the network load reduces, the MAC successfully sends the data and only then it allows HCP to forward its next packet.

However, when the number of chain increases in the network, contention due to increased nodes begins to dominate over the rate-estimation algorithm. Hence, both ATP and HCP see a reduced throughput as we increase the number of nodes in the chain.

7.5.2 Fairness for Unicast Scenario

In this section, we examine another integral aspect of a congestion control protocol – fairness when there are multiple competing flows. We use two linear chain topologies, one with 3 nodes and another with 6 nodes. For both topologies, we measure instantaneous throughput and hence, we run these simulations for only one seed. As the size of the chain increases, network contention increases. By varying the size of the linear chain, we are able to show fairness with varying network contention.

Three nodes chain topology

The first linear chain topology consists of 3 nodes. We start three flows at 0 seconds, 10 seconds, and 30 seconds. The main premise is that as a new flow is introduced, the bandwidth should be shared evenly among all the flows. Further, when a flow is terminated, the protocol should release the bandwidth for other flows, which the protocol should redistribute fairly among remaining flows.

As shown in Figure 7.9, HCP is able to share the bandwidth smoothly when the number of flows decreases or increases. ATP, on the other hand struggles with fluctuations in throughput (Figure 7.10). As explained in the previous section, this fluctuating behavior is because of ATP's end-to-end approach for congestion control.

Six nodes chain topology

For the second linear chain topology, we place six nodes in the chain and start two flows. Adding nodes in the chain leads to an increased degree of collisions; using a longer chain allows us to observe the behavior of congestion control algorithm amidst increased contention.

HCP flows are able to share the bandwidth evenly (Figure 7.11). A point of interest in the HCP graph occurs when the throughput dips at around 30 seconds.

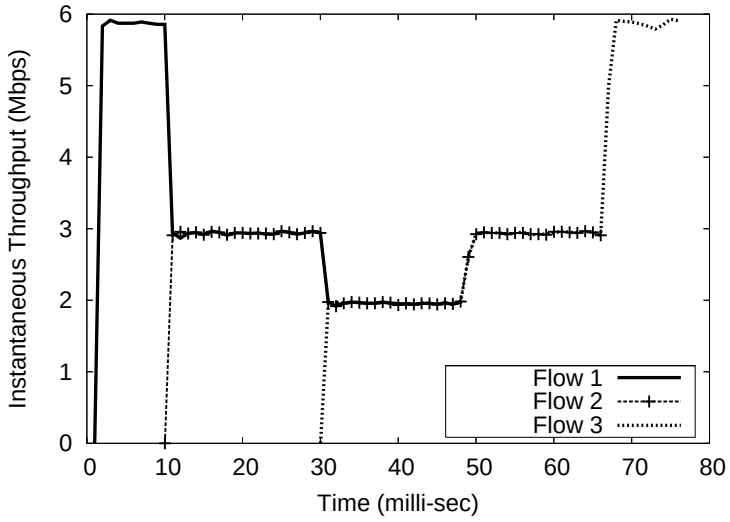


Figure 7.9: Instantaneous Throughput for HCP for Small Linear Chain

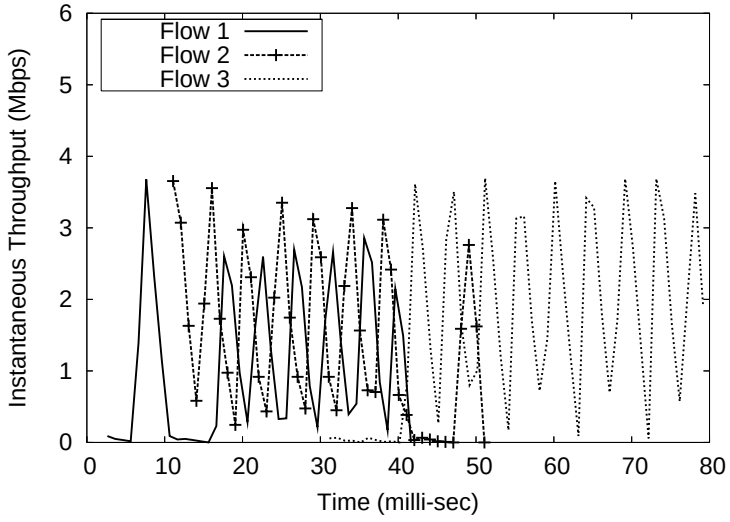


Figure 7.10: Instantaneous Throughput for ATP for Small Linear Chain

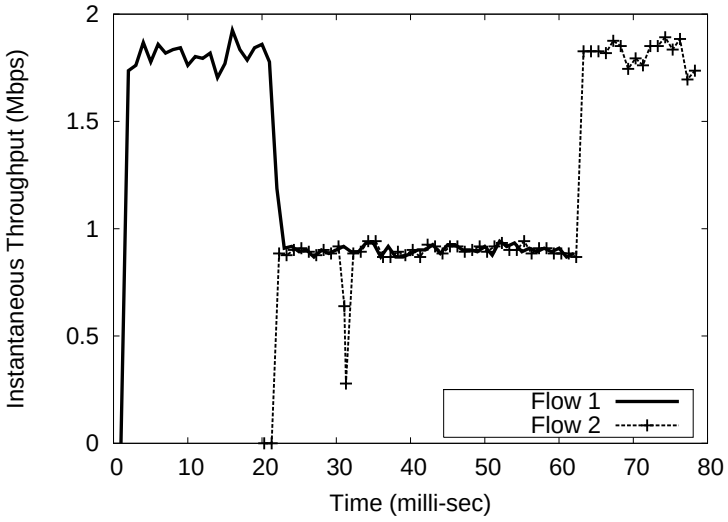


Figure 7.11: Instantaneous Throughput for HCP for Long Linear Chain

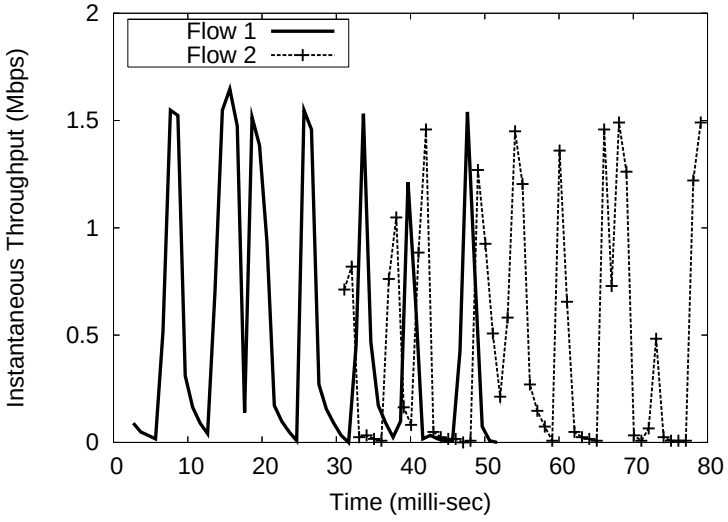


Figure 7.12: Instantaneous Throughput for ATP for Long Linear Chain

When we analyzed the packet trace, we found that this dip is due to a false link break. A false link break can occur because the routing protocol considers a packet loss as a sign of link break. However, this may not always be the case; a packet loss can occur due to congestion. MDA avoids false link breaks. It begins by assuming that the a packet loss is due to mobility and hence permits the first false link break to be treated as a regular link break. However, MDA switches on its monitoring mechanism and if it finds that the link is not broken, then it suppresses subsequent false link breaks. This suppression is apparent from the graph as there are no other dips in the throughput graph. On the other hand, ATP continues to see fluctuations in its throughput (Figure 7.12).

7.6 Multicast Results

In this section, we demonstrate the multicast behavior of HCP. For this, we run two types of simulations. First, we examine the reliability component of HCP and demonstrate that HCP is able to retransmit lost packets. Second, we examine the congestion control algorithm and demonstrate that when there are multiple co-existing groups, they all share the bandwidth equally. In this set of simulations, we also show that HCP is able to handle network bandwidth heterogeneity well – in other words, it provides different throughput to different receivers according to the available bandwidth at these receivers. Since we are interested in measuring instantaneous throughput, we run these simulations for only one seed.

7.6.1 Reliability

We first examine the reliability offered by HCP and demonstrate that it can retransmit packets that are lost due to either mobility or contention. For comparison, we also use the optimal overlay along with HCP. In this section we use the topology shown in Figure 7.13. The network consists of one source, S, and two receivers, $R1$ and $R2$.

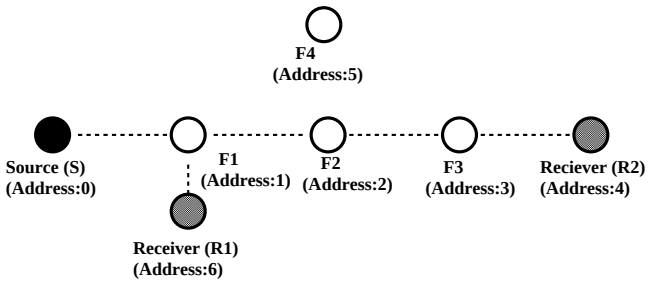


Figure 7.13: Reliability Topology

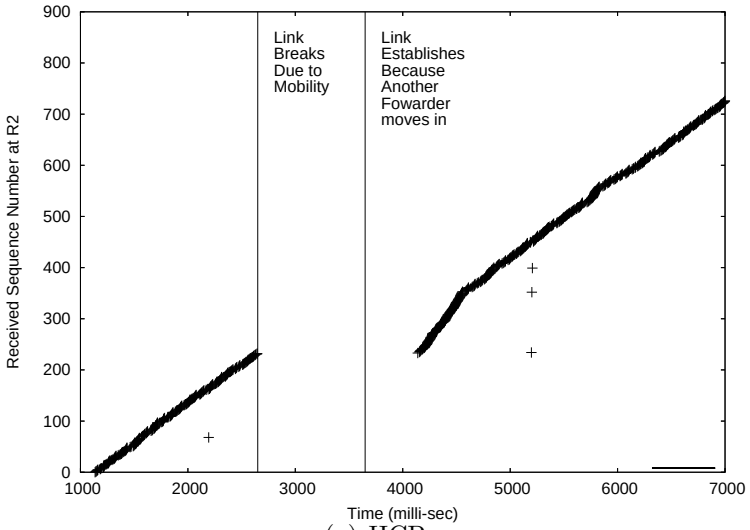
Four other nodes act as forwarders. The multicast tree initially uses forwarders $F1$, $F2$, and $F3$.

Loss Due to Mobility

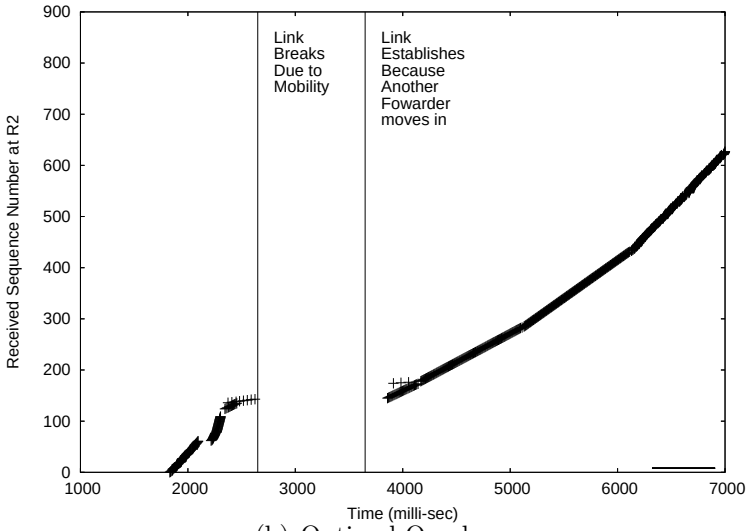
We consider two cases when packets are lost due to mobility. For the first case, we move forwarding node $F2$ far away, breaking the tree, and then one second later move $F4$ to its previous position, repairing the tree. For the second case, we move receiver $R2$ to a distant location, preventing it from receiving packets, and then move it back to its previous location one second later. In both cases $R1$ is not a member of the group; we make $R1$ join the group when we study loss due to contention in the next section. Since the topology is a chain topology, all forwarding on the tree is done with MAC-layer reliability. This allows us to isolate loss that occurs due to mobility from the case when it occurs due to semi-reliable broadcast.

Figure 7.14 shows packet traces at $R2$ for the case when a forwarder in the middle of the tree moves. Traces are shown for both HCP and for the optimal overlay. The vertical lines on the graphs show the events when $F2$ moves out of range of $F1$ and when $F4$ moves into its previous location.

This trace shows that HCP prevents most packet loss because $F1$ stops transmitting when the tree breaks and buffers the packets it has received so far. Once the receiver rebuilds the tree, creating new forwarding state, $F1$ resumes transmitting the packets from its buffer. $F2$ also has some packets it has buffered, but since it has



(a) HCP



(b) Optimal Overlay

Figure 7.14: Packet Trace When Forwarder Moves

been partitioned from the rest of the network and is no longer on the tree, this state eventually times out.

This trace also shows that while HCP achieves better throughput than the optimal overlay, the overlay does recover from the link failure more quickly. The relatively slow recovery for HCP is due to the fact that ASSM is configured to refresh the tree every second, so a new Join is not sent immediately when the route breaks. To fix this problem, a receiver could track the average inter-packet delay and then trigger a new Join if this delay ever grows too large.

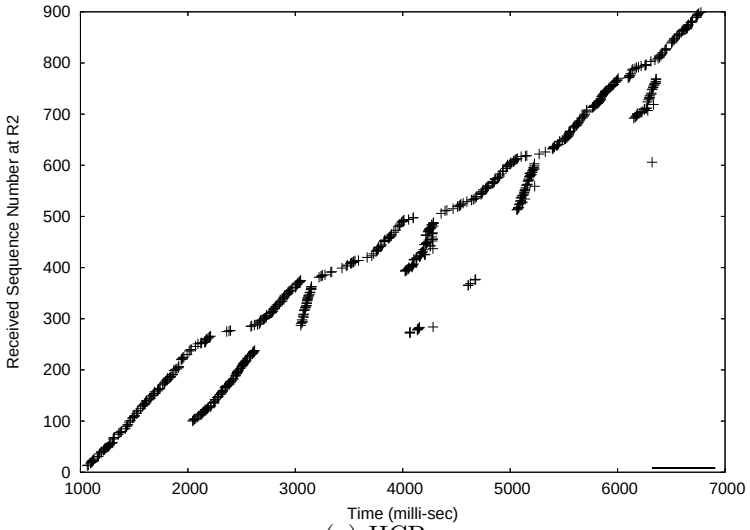
We see similar results for the case when the receiver moves.

Packet Loss due to Contention

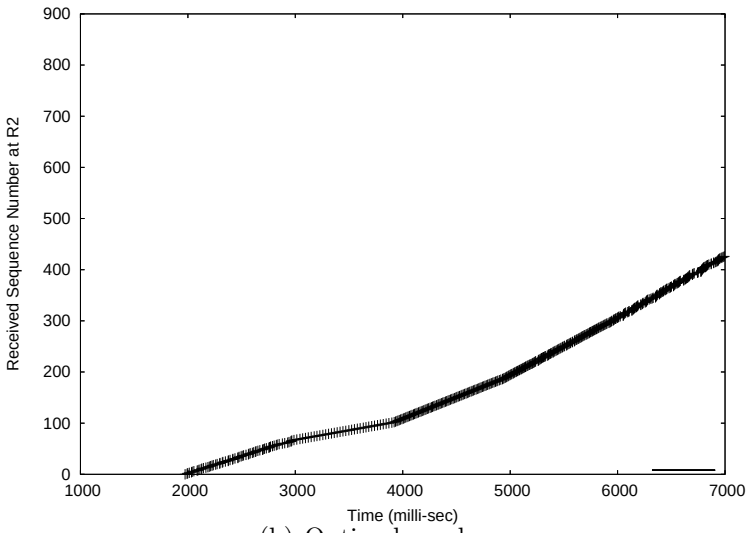
We next examine packet loss due to contention by adding receiver $R1$ to the multicast tree. In this scenario, all nodes ares static. Forwarder $F1$ now has two children and hence it uses semi-reliable broadcast, alternating the RTS/CTS exchange between $R1$ and $F2$. This causes loss for both receivers because promiscuous listening is not always reliable, particularly when surrounding nodes are sending at the same time.

Figure 7.15 shows packet traces at $R2$ when there is loss due to contention. For HCP, the trace shows that there is substantial loss even without mobility, because of the reliance on promiscuous listening. A tail of retransmitted packets occurs about once a second, when the receiver requests them using a NACK. The overlay avoids this loss since it uses two unicast connections, rather than a tree, and always uses MAC-layer reliability.

Despite the higher loss rate and the subsequent retransmissions, HCP is still able to achieve higher throughput. The main reason is that the overlay is not as efficient as the tree. Let us illustrate this using Figure 7.13. The overlay can consist of a connection from S to $R1$, and then another connection from $R1$ to $R2$, so every packet must be forwarded twice on hop $F1-R1$ as compared to HCP. When a packet



(a) HCP



(b) Optimal overlay

Figure 7.15: Packet Traces with Contention

arrives at $F1$, it forwards it to $R1$ on the first connection. However, $R1$ must then forward it back to $F1$ on the second connection, and then $F1$ forwards it on to $F2$. With HCP, the original packet transmitted by $F1$ is sometimes received by both $R1$ and $F2$. Even if one of them misses it, it is retransmitted at most once by $F1$, which still saves one packet transmission. We revisit this behavior later in Section 7.8.

7.6.2 Congestion Control

We next examine the congestion control component of HCP and demonstrate that it recalculates rates as conditions change, while also sharing bandwidth fairly among competing flows. We also show that HCP can accommodate receivers with different available bandwidths, delivering a high rate to a faster receiver, while buffering packets for the slower receiver.

Our first scenario uses a simple chain topology, consisting of a set of four nodes. The source sits at the one edge of the chain while the receiver sits at the oppose end of the chain. We increase the number of flows by having the same receiver subscribe to multiple groups, each using the same source. We use a total of 3 groups starting at 2, 10, and 25 seconds. In this scenario, we measure instantaneous throughput and hence we run simulations for only one seed.

As shown in Figure 7.16, HCP is able to converge quickly to a new rate, while also sharing bandwidth equally among all groups. HCP reduces its rate appropriately as more groups share the bandwidth, and also increases its rate as transmission for some groups becomes complete. Though we don't show results for the overlay in this case, we found that it did not converge to new rates as quickly as HCP because it uses end-to-end rate adjustment.

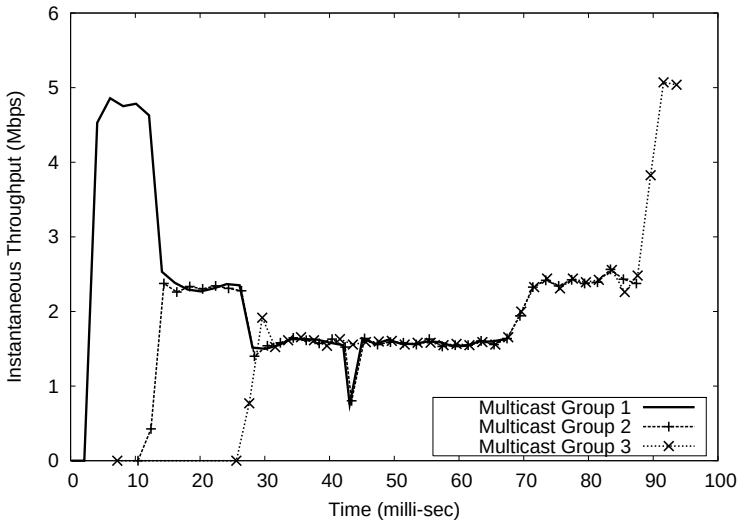


Figure 7.16: Congestion Control with Multiple Multicast Groups

Heterogeneity

We next consider the case where different receivers have different available bandwidths. We use the topology shown in Figure 7.17, which consists of two chains, one with a high data rate (11 Mbps) and one with a low data rate (1 Mbps). Since we measure instantaneous throughput, we run these simulations for only one seed.

In ad hoc networks, bandwidth heterogeneity can occur due to mobility, fading, and interference. HCP accommodates heterogeneity by buffering at intermediate nodes to handle different regions with varying bandwidths. For slower links, data is buffered and is sent at the (slow) speed of the link. Hence, the source does not need to send to every receiver at the rate of the slowest group member.

Figure 7.18 shows the instantaneous throughput received by both receivers, calculated over one second intervals. HCP is able to send at a higher rate for $R2$ while also providing a lower rate for $R1$. Packets are buffered on the slower path as needed.

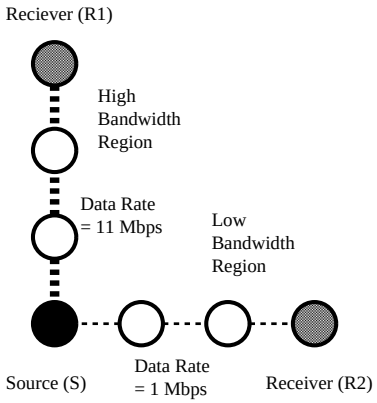


Figure 7.17: Topology for Heterogeneous Receivers

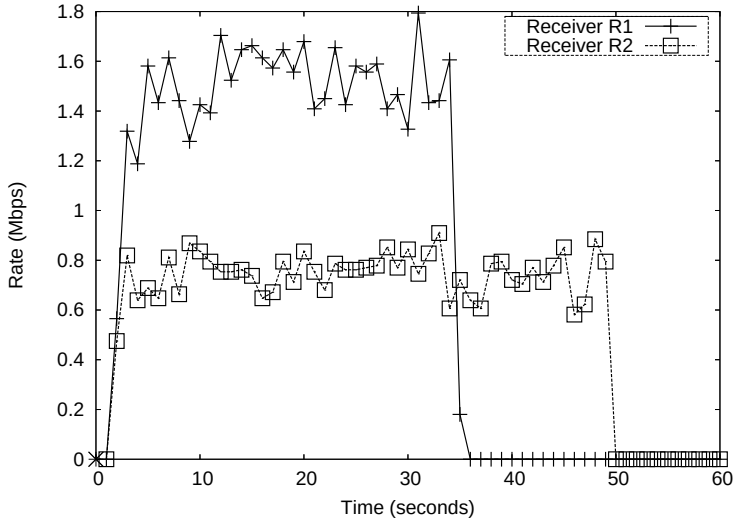


Figure 7.18: HCP: Multiple Rates in a Single Multicast Tree

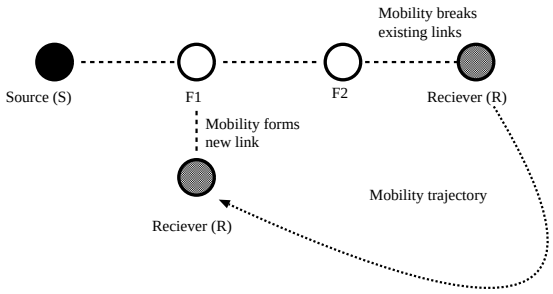


Figure 7.19: Buffer Size Topology

7.7 Limited Application Buffer

An important factor in the performance of HCP is its buffer size. In all previous experiments, each flow is allocated a buffer of of 100 KBytes. Since mobile devices may have buffer constraints, we explore the effectiveness of HCP as the buffer size changes. For this section, we use the topology shown in Figure 7.19, which consists of a chain of four nodes. We vary the buffer size at each node from 5 to 200 KBytes and transfer a 5 GB file from the source to the receiver using 1KB-sized packets.

We first examine the effects of buffer size when there is no mobility. Figure 7.20 plots the number of packets that must be dropped by $F1$ due to buffer overflow. We plot two lines – one for the case with credit-based flow control and one without. In both cases, 50 KBytes is enough buffering to prevent packet loss. As the buffer size grows smaller, packet loss increases, though it is extremely small compared to the total number of packets sent for the 5 GB file. Notice that packet loss occurs even when we use congestion control because of the unreliability of passive feedback. If a node is unable to receive feedback concerning the amount of space available downstream, it may send when the downstream buffer is full, causing loss.

We next examine the effects of buffer size when a receiver moves. In this simulation, the receiver moves from its first location to the second location indicated

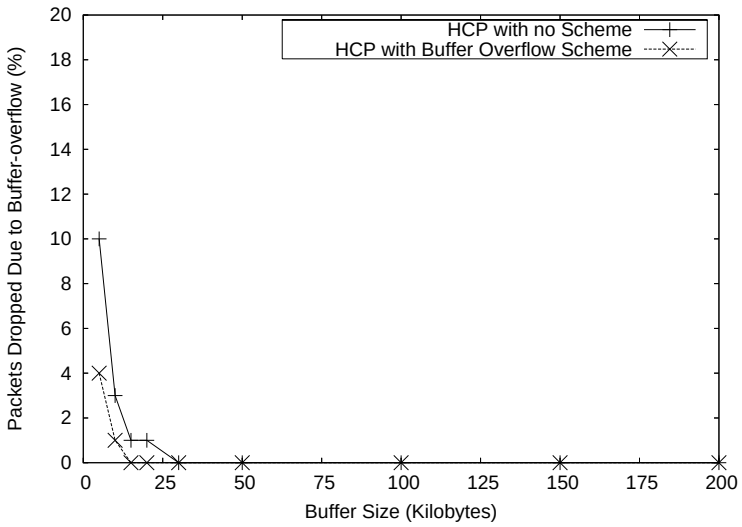


Figure 7.20: HCP: Packet Loss vs. Buffer Size

in the topology. In this case, the receiver will miss some packets, then request them when it joins the tree again. As the buffer size decreases, there is a greater chance the lost packets will need to be requested from the source.

Figure 7.21 plots the number of lost packets that must be retransmitted by the source for different movement speeds and buffer sizes. Two trends are evident: more buffering is needed when the receiver moves more slowly, and larger buffer sizes provide more local recovery.

7.8 Evaluation at High Traffic Load

In this section, we evaluate HCP during conditions of high load. We create load conditions by varying both group size and the number of groups.

7.8.1 Varying group size

For this simulation, we vary the number of group members for one multicast group; we keep one source and vary the number of receivers. We place 50 static nodes randomly in a field of size 1000mX1000m. Each HCP node has a buffer of 100 KB.

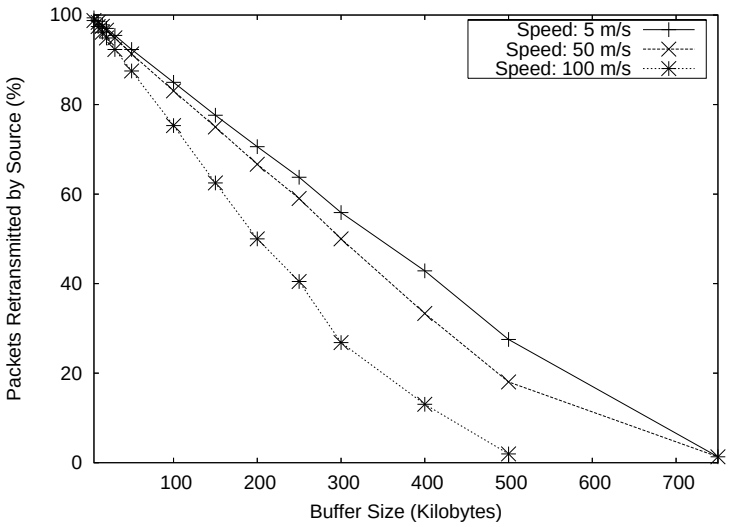


Figure 7.21: HCP: Source Retransmission versus Buffer Size

Since the overlay buffers packets in the application layer, we do not impose any buffer limitation for nodes in the overlay.

In Figure 7.22, we plot the combined throughput for all receivers for each run on Y-axis and the number of receivers on the X-axis. The result demonstrates that HCP scales well when the group size becomes large. HCP provides higher throughput than overlay because with overlay, all forwarding is done using unicast, leading to redundant transmissions. However, with HCP each forwarder unicasts the data only once to a randomly selected downstream node and other downstream nodes merely snoop and listen. Thus, multiple nodes can listen to a single transmission. It is because of this reason that even though the data rate of the medium is 11 Mbps, HCP is able to provide a throughput as high as 14 Mbps! With a small number of receivers, both HCP and optimal-overlay perform similarly. The reason for this behavior is because with a small number of receivers, the network load is less and the overlay is able to deliver packets reliably using its unicast links. It is only when the number becomes large, that the load becomes so high that overlay's unicast does not scale well. As expected, with the random variant of the overlay, we find that the throughput achieved is worst, which shows that the random variant forms inefficient

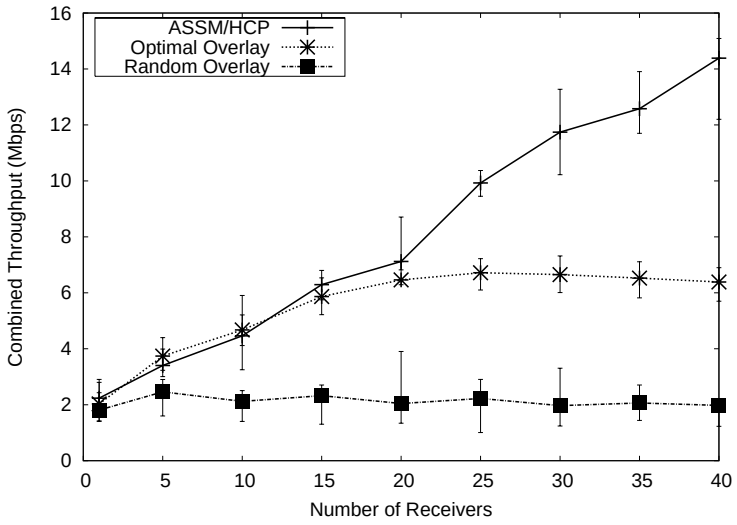


Figure 7.22: Throughput as a function of group size

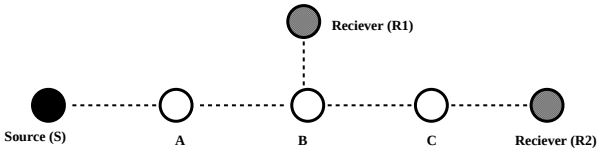


Figure 7.23: Understanding Peer-to-peer Overlay structure

trees even when the group size is small. The actual performance of an overlay will lie between the random and the optimal cases, depending upon how often the overlay is re-constructed.

For the overlay, the advantage of using ATP connections among peers comes at a cost. Since members setup connections among themselves, instead of letting the network build an efficient tree, the overlay often leads to redundant transmissions. We use Figure 7.23 to demonstrate this problem in more detail. This topology has three members for a given multicast group: source S and two receivers, $R1$ and $R2$. The optimal way of connecting all these members in an overlay is to connect S to $R1$ and then connect $R1$ to $R2$. The total number of hops is $3 + 3 = 6$.

However, if one were to do a native multicast, where the tree is built at the network layer, the total number of hops would be reduced. With native multicast, S can connect to A, which in turn can connect to B. A single broadcast at B would

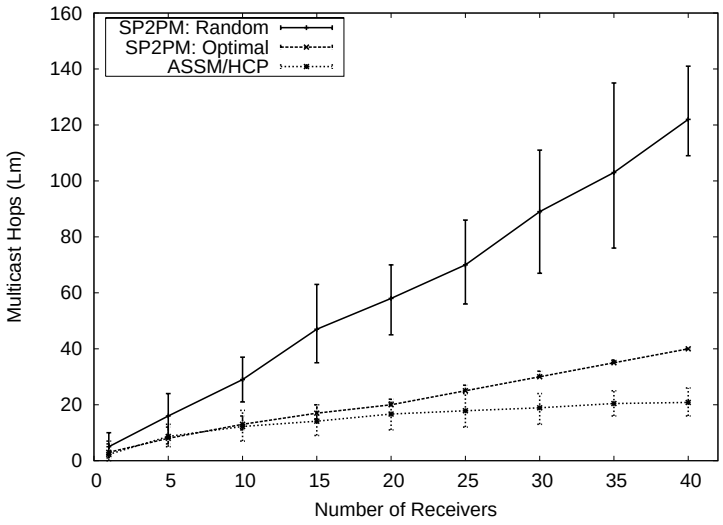


Figure 7.24: L_m as a function of group size

ensure that both nodes $R2$ and C receive the packet. Lastly, C can connect to $R2$. Hence, the total number of hops is $2 + 1 + 1 = 4$, which is smaller than the earlier cost of 6 hops.

To better explain the scalability of HCP, we vary group size and plot the expected number of forwarding hops, L_m for both HCP and the overlay, as outlined in Section 3.6. For a tree to do efficient forwarding, it should have a lower L_m. When L_m is high, more and more intermediate nodes spend their bandwidth in data forwarding. A protocol with lower L_m will have spare bandwidth for forwarding additional data which can lead to higher throughout.

To determine L_m for HCP, we run a simple experiment, where we send CBR traffic at a very low rate for one group. We count the total number of packets forwarded and total packets sent. L_m is the ratio of packets forwarded to the packets sent, and provides an estimate of the total number of hops in the tree. To determine L_m for the overlay, we obtain values from the logical mesh created using *Kruskal*'s Algorithm in Section 7.4. Likewise, for the random variant of overlay, once all edges are selected, we count the total forwarding hops in the tree.

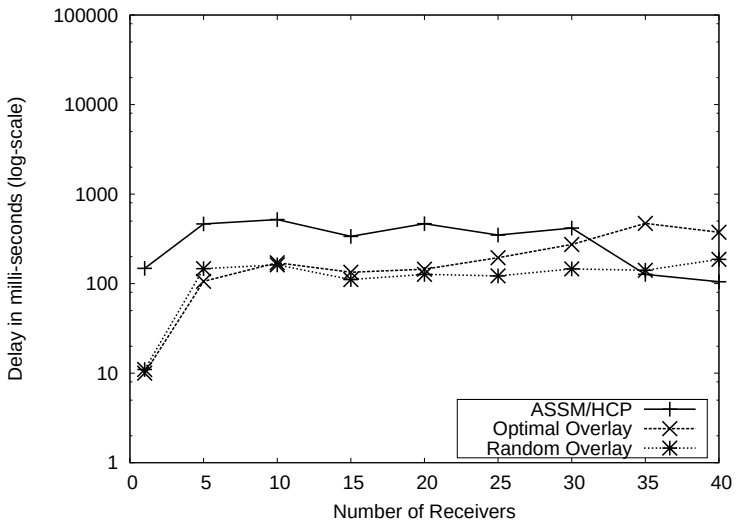


Figure 7.25: First-time Delay vs. group size

In Figure 7.24, we plot L_m for all of these protocols, along with the minimum and maximum values. HCP has a low L_m. This behavior is expected since it builds an efficient tree like ADMR (Figure 3.13). On the other hand, the overlay builds trees with a lot of stress. As expected, the random variant does worst among all the three protocols. When we look at the corresponding throughput for all of these protocols in (Figure 7.22), we find that it correlates well with L_m.

Next, we plot delay for first-time packets in Figure 7.25. HCP incurs higher delay than the overlay, because of per-hop buffering. HCP packets spends more time waiting in the buffer. On the other hand, overlays send data using rate-based congestion control, and hence they can keep buffering requirements fairly small at intermediate nodes. Furthermore, unlike HCP, in an overlay, it is only the peers that buffer packets, instead of every intermediate forwarder.

However, one of the ways to improve the delay is to enhance our current buffering mechanism. In our present design, we maintain a single buffer for both local recovery and for keeping packets, which have not been sent yet. Our current design allows each intermediate nodes to buffer a large number of unsent data packets. Allowing a large number of unsent data packets means adding more to the delay. If we

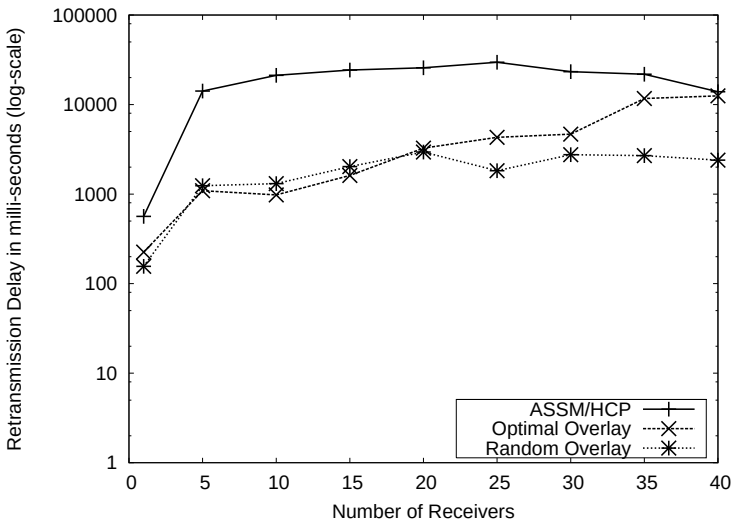


Figure 7.26: Retransmission Delay vs. group size

were to have a smaller cache for unsent data packets, then the delay can get reduced. In addition, we can keep a large buffer for caching the sent packets, so that local recovery can be achieved effectively. However, the trade-off with using a small cache for unsent data packets is that it reduces the ability of an upstream node to cater to heterogeneous children. If the upstream node has only a few packets in its unsent data, then the fast-bandwidth downstream nodes can receive only a small number of extra packets than that of the slow-bandwidth children – this way, the efficacy of sending different rates to different receivers can possibly become limited. We defer this study to a future work.

When we plot delay for retransmission packets in Figure 7.26, we find a similar trend. HCP once again incurs higher delay than overlay. Once again, the limited buffer increases the delay even for retransmission packets. When we studied several simulation traces, we found that besides limited buffering, two other reasons might be at work leading to higher retransmission delay. First, due to congestion and contention, some of the downstream intermediate nodes drop even the retransmission packets. Hence, the receiver is forced to request the retransmission of the same packet, which leads to additional delay. Second, for retransmission, if there are

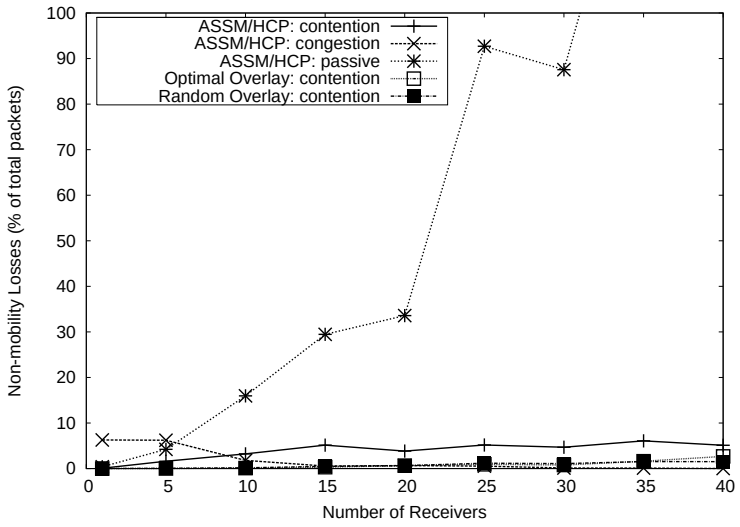


Figure 7.27: Packet Loss vs. Group Size

multiple receivers requesting for a given packet, then we send it using unicast to one of the receivers and let others snoop it. Since snooping is not reliable, this means that the retransmission packets can get lost and the receiver would need to request for retransmission of the same packet. Since the receiver sends information for lost packets once every second, when retransmission packets are dropped, it incurs delay of the order of seconds. As a future work, we should consider the case of sending retransmission packets reliably for all the receivers requesting a given packet.

Lastly, we plot different losses for this scenario in Figure 7.27. For HCP the passive loss increases with an increase in the group size. Essentially an increase in group size implies that the number of downstream nodes for a given forwarder increases; this leads to a higher passive losses. However, since the fanout (number of downstream nodes for a given upstream node) increases, it does not lead to a significant increase in the contention loss. Even though the number of receivers increases, the upstream nodes can still get away with just one unicast forwarding! Since the passive loss can occur at each forwarder (including the sender) and with more receivers the fan-out can become large, we find that the passive losses can equal or exceed the total number of packets being sent. For HCP, the passive loss exceeds

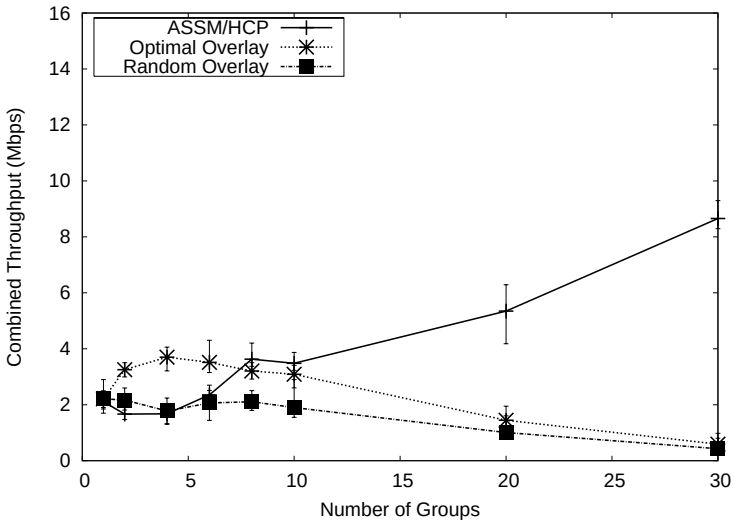


Figure 7.28: Throughput vs. Number of Groups

100% when the group size becomes large – this means that each packet is lost at least once on the tree. This strongly motivates us to explore mechanisms, like BMW that can provide reliable broadcasting and hence can reduce the passive loss [70]. Even though HCP is able to outperform the overlay in terms of throughput (Figure 7.22), reducing the passive loss can potentially provide a further increase in the throughput. In Figure 7.27, we find that the other types of losses appear to be relatively small.

7.8.2 Varying number of groups

Next, we evaluate both HCP and the overlay by varying the number of groups. For each group, we keep the composition of its members same: one source along with 4 receivers. The remaining simulation parameters are the same as those of the previous section.

As compared to the previous case of varying group size, varying the number of groups adds more stress. With more groups, data must be forwarded for each group; in general, this increased load leads to lesser throughput for protocols (Figure 7.28). For the overlay, throughput decreases when the number of groups becomes high. A study of simulation traces reveals that the underlying protocol, ATP, struggles with

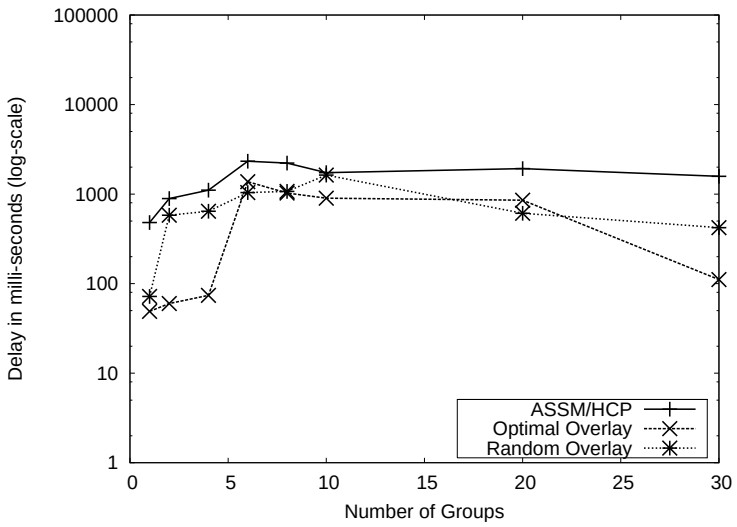


Figure 7.29: First-time Delay vs. Number of Groups

its rate mechanism when the number of groups becomes high. Several ATP ACK packets, which carry the rate information, are dropped due to increased load. The sender often does not have an accurate sending rate information and might continue to use an earlier slower sending rate. In fact, for some of the senders, several initial ATP ACKS are lost and hence they continue to use the very first initial, low rate proposed by ATP.

However, we find that HCP is able to deliver more throughput with increasing groups. This increase is seen because with increasing groups, the number of receivers increase. Additional receivers occupy different areas of the field and thus, HCP is able to harvest the bandwidth from a larger portion of the network. This behavior is similar to the case when we increase the group size.

The pattern for first-time delay is similar to the previous case of varying group size. However, due to increased contention, the delay values are higher (Figure 7.29). The pattern for retransmission packets is also similar except that at larger number of groups, we find that overlay's retransmission delay is reduced (Figure 7.30). This is because, even though the overlay can send a lot of data, due to inaccurate rate

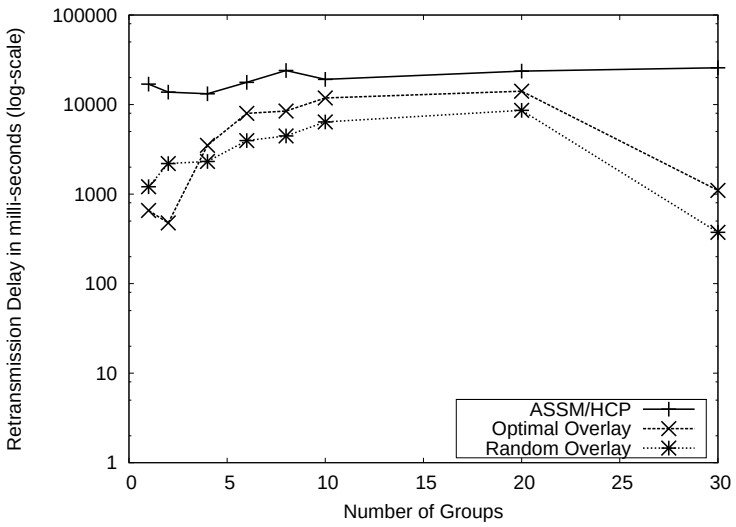


Figure 7.30: Retransmission Delay vs. Number of Groups

estimation, the network remains under-utilized and so retransmission packets are forwarded quickly.

When we study loss-pattern, we find that for HCP, the number of congestion losses increases as the number of groups is increased (Figures 7.31). The reason for this increased congestion loss can be attributed to the congestion-control mechanism. When a downstream node forwards a packet, it copies its available space into the header, and the upstream node depends upon snooping to receive this info. However, snooping is itself not reliable and hence it is possible that for some packets, the upstream node does not get the correct estimate of buffers available at its downstream nodes. Thus, we need to explore approaches to reduce snooping loss and hence congestion losses with HCP. Unlike the case of varying group size, passive loss appears to be small because the number of receivers per group and hence the fanout remains constant leading to a smaller passive loss. Other types of losses, like the case of varying group size, appear to be small.

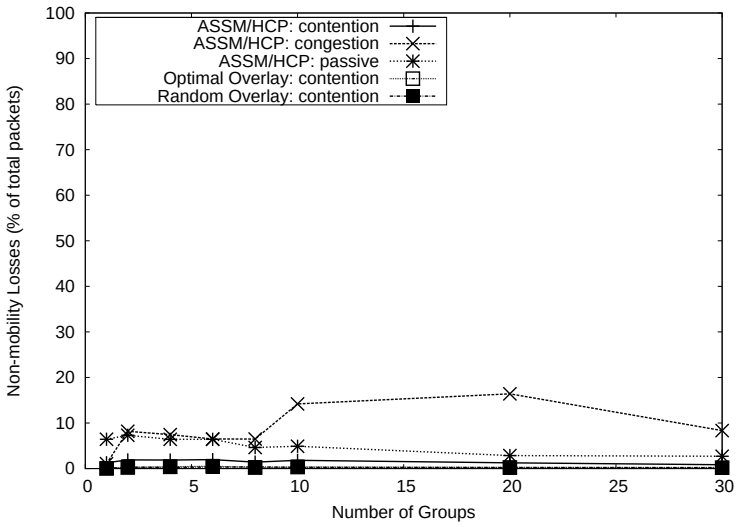


Figure 7.31: Packet Loss vs. Number of Groups

7.9 Evaluation with Mobility

Lastly, we examine the effects of mobility on the performance of HCP and the overlay. We place 50 mobile nodes in a field of 1000mX1000m; nodes move using the random waypoint mobility model, with a pause time of 30 seconds. We run two sets of simulations. In the first set, we vary mobility and keep the number of groups and the composition of each group constant; this allows us to isolate the effects of mobility from other traffic parameters. In the second type, we repeat one of our previous scenarios of varying group size, but we introduce an ambient mobility; this allows us to study the combined effects of two parameters, mobility and increasing group size, on protocol performance.

7.9.1 Vary Mobility

We begin by varying mobility and keep the group numbers and per-group composition same. We keep 3 groups and each group has 5 members; out of these 5 members, one of them is the source. Similar to Section 3.6, we increase the speed of these mobile

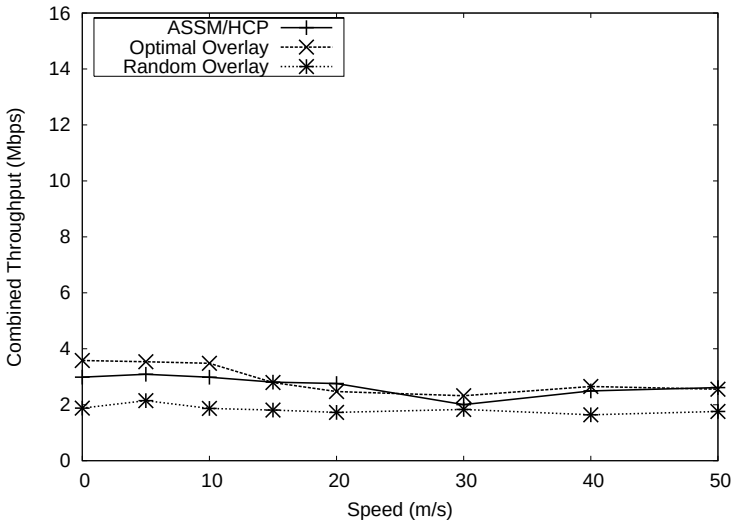


Figure 7.32: Throughput as a function of mobility (each group has 5 members)

nodes from 0 to 50m/s. To explore further, we also run an additional scenario, where we increase the number of group members to 8, in which one of them is the source.

When we plot throughput with varying mobility in Figure 7.32, we find that with increasing speed, the degradation in throughput does not occur significantly. This is a very welcome result. With CBR flows (Section 3.6), once a packet is dropped, UDP does not retransmit it and hence we see a lower packet delivery ratio with increasing mobility. However, HCP and the overlay do retransmit lost packets and hence throughput does not degrade significantly with mobility. Like all previous cases, the random variant of the overlay ends up providing a lower bound on achievable throughput. However, when we increase the number of members for each group to 8 instead of 5 and we find that HCP shows a clear advantage over both variants of the overlay (Figure 7.33).

The delay pattern is similar to earlier cases, of varying group size or varying the number of groups. For the sake of repetition, we omit these graphs. However, from our results we find that delay incurred is slightly higher. This increase in delay is seen because frequent link breaks cause packets to wait in the buffer until a route

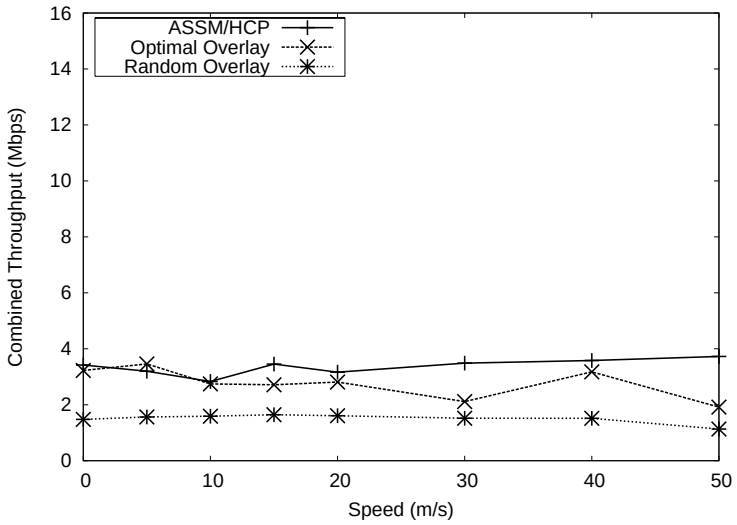


Figure 7.33: Throughput as a function of mobility (each group has 8 members)

is found; for HCP, packets sit at any node's buffer and for overlay, packets wait at one of the peer's buffer.

In general, packet loss is small with increasing mobility (Figures 7.34). We find that contention and congestion based losses decrease with increasing mobility. The reason for this noticeable decrease is that with increasing mobility, more and more links break leading to fewer active members. Hence, this leads to lesser contention losses. Further, due to link breaks, lesser packets are received; this leads to lesser congestion losses. However, an important difference is seen for HCP's passive loss, which increases with mobility. Upon studying the trace, we find that when a receiver moves away, then it leaves behind the forwarding state at its earlier upstream node in the tree. This is due to the soft-state nature of ASSM; the old forwarders continue to keep stale downstream receivers for the tree for a certain period. Hence, in effect,there are more downstream nodes attached to an upstream node, when mobility increases. The loss pattern for the case when there are 8 members per group exhibits a similar trend.

We also plot mobility losses in Figures 7.35. We find that HCP suffers slightly more mobility losses than the overlay. Overall, the losses are small when speed is

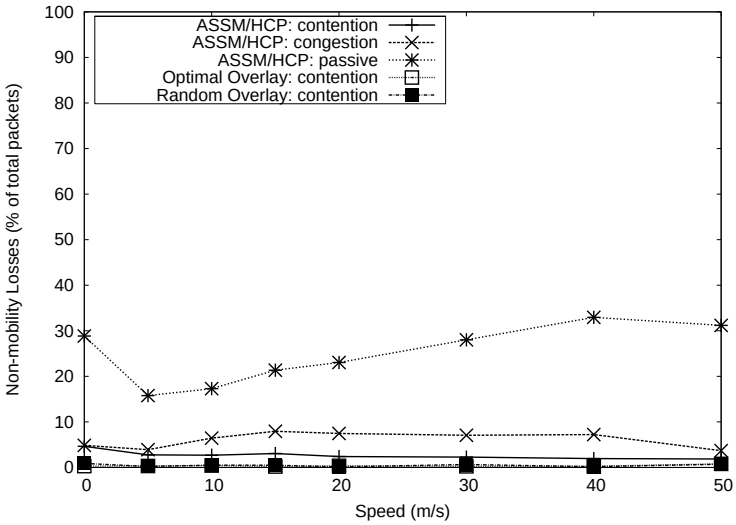


Figure 7.34: Packet Loss for HCP and overlay

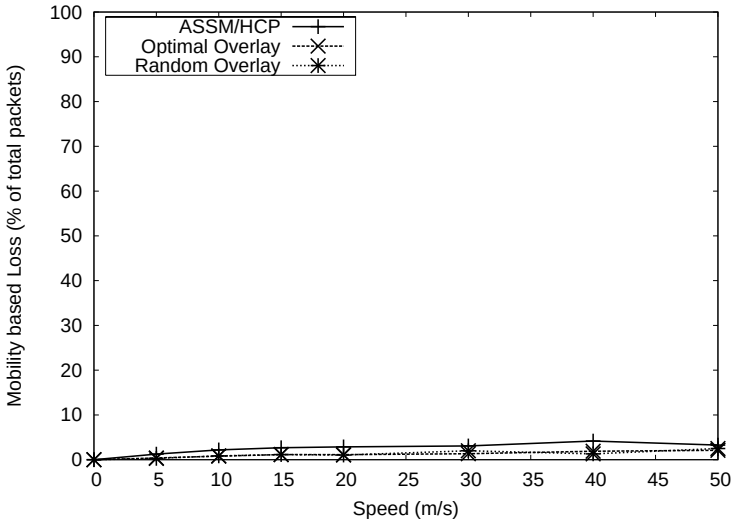


Figure 7.35: Mobility-based Packet Loss vs. Mobility

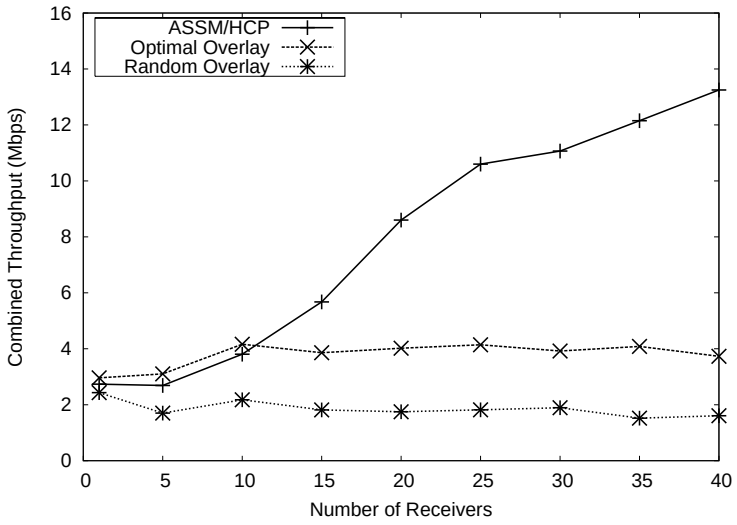


Figure 7.36: Throughput vs. Group Size with ambient mobility

increased. The loss pattern for the case when there are 8 members per group also exhibits a similar trend.

7.9.2 Varying Group Size with ambient mobility

Next, we repeat one of the previous scenarios, where we vary the group size, but we make nodes mobile with a constant ambient mobility. We choose an ambient speed of 20 m/s, since from the previous scenario, we find that a speed of 20 m/s affects these protocols moderately.

The throughput pattern is same as that of the earlier scenario, where we vary the group size (Figure 7.36). However, due to ambient mobility, both HCP and overlay provide lower throughput than that of the static case. We find that the pattern for various types of losses (mobility and non-mobility) and delay are similar to that of the static case. However, the delay values are slightly higher, which can be explained due to frequent link breaks. For the sake of repetition, we omit these graphs.

7.10 Conclusion

In this chapter, we design and evaluate the third and last module of our multicast transport architecture: a hop-by-hop congestion control transport protocol, HCP. There are several novel design elements in HCP. First, HCP uses a multicast state setup protocol, ASSM, for setting up transport state at each hop of the multicast tree. Second, it uses application layer buffering at each node. Third, instead of estimating the sending rate or the transmit window size, HCP depends upon the MAC layer's ability to send data at a fast rate.

For both multicast and unicast flows, our results show that as compared to ATP, HCP provides higher throughput, ensures better fairness among all the receivers, and estimates the correct sending rate when the number of co-existing flows increases or decreases. Further, HCP caters to receivers that have different available bandwidth; HCP delivers data to each receiver according to the bandwidth they can afford. Our results also show that HCP can perform satisfactorily for a buffer-size as low as 100KB.

Under realistic conditions, with varying traffic load and mobility, HCP outperforms an application-level overlay, running on top of ATP, in terms of throughput. HCP has a significant advantage in terms of scalability with load. As the group size or the number of groups increases, HCP can deliver much higher throughput than an application-level overlay.

Some additional work can be done to improve HCP. First, we need to explore mechanisms that can provide reliable broadcasting at the MAC layer and hence reduce the passive losses. We can use mechanisms like BMW [70]. A trade-off is present here because with each reliable broadcast, the message is sent reliably to all neighbors or at least those neighbors that are members of the multicast group. Typically, this is achieved using the sequence of RTS/CTS/DATA/ACK – using this sequence for each forwarding at each hop in the tree may be expensive.

Second, we need to explore designs to provide faster recovery for retransmission packets. This might be achieved by using a better caching strategy. The delay for all packets can be reduced by splitting the current single per-flow buffer into two buffers: (a) a smaller buffer for unsent packets and (b) a separate buffer for storing packets intended to aid retransmissions. The smaller buffer for unsent packets would mean lesser delay at each hop but would also limit the ability of HCP to cater to heterogeneous receivers.

Third, we need to explore cache replacement mechanisms, besides the current LRU scheme in our implementation, for replacing data from the per-flow cache. With LRU, all nodes delete cached data at approximately the same time. So large caches are needed to avoid having the same source repair all the loss. It might be better to have a randomized cache replacement, so if the node is missing the data, another intermediate node might have it.

Fourth, we need to provide faster repair with ASSM. An HCP receiver can measure the average inter-packet delay, and then trigger a repair if this delay grows too large, rather than waiting for the next timer expiration. This would allow ASSM to perform better when mobility is high.

Next, we should explore using a rate-based congestion control algorithm to reduce contention-based losses. Currently, HCP's congestion control algorithm depends upon the feedback from the MAC layer to deliver packets. However, it needs to be seen if replacing this algorithm with a rate-based algorithm can lead to lower contention losses.

Lastly, one could explore the retransmission behavior of HCP. In our current implementation, we use an equal mix of retransmission packets and new packets, however, additional policies can also be explored. As an example, one can keep track of downstream nodes that make frequent requests for retransmission; it might be more reliable to select that downstream node for unicast forwarding, which makes

the most frequent requests for retransmission. Thus, this can potentially reduce the retransmission efforts.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this dissertation, we build and evaluate a multi-modular multicast transport architecture that provides routing, reliability, and congestion control. The dissertation consists of four steps – evaluation study, building ASSM, building MDA, and building HCP. The last three steps build three different modules that work in tandem to provide the above services in an efficient manner. We provide a design of each of these modules along with a simulation-based evaluation of their performance.

In the first step, we conduct a rigorous evaluation of two existing multicast routing protocols. Our goal is to identify cases where ad hoc multicast routing protocols can improve their performance. Our evaluation provides us with several recommendations. Our first recommendation is that designers need to focus on optimizing for both mobility and density. Protocols should react to link breaks, but with a controlled routing overhead. Designers should also be aware of the pitfalls associated with high density situations, avoiding problems such as ACK implosion and broadcast storms. Our second recommendation is that routing protocols should not attempt to monitor packet loss and repair routes when loss is high; such packet loss may in fact be due to congestion and the increased repair traffic can lead to a state of congestion collapse. Our third recommendation is that for achieving high capacity, protocols

should use a bandwidth-efficient tree rather than a mesh and should also have low control overhead.

In the second step, we build and evaluate a novel multicast state setup protocol, ASSM. This state setup protocol uses existing routes provided by the unicast routing protocol instead of generating its own network-wide route discovery flood. ASSM interacts well with transport protocol and assists it in setting transport state at intermediate forwarder nodes in the multicast tree. ASSM also uses an SSM style multicast to conserve multicast address space and to provide simple source discovery. Our results demonstrate that it is possible to design a scalable multicast routing protocol for ad hoc networks without using hierarchy or an overlay. Using SSM semantics eliminates much of the overhead previously required for joining a multicast group. Our work also shows that localizing repair decisions at the group members enables the multicast routing protocol to provide both low repair latency and low repair overhead.

The third step builds the next module of our transport architecture – a cross-layer Mobility Detection Algorithm that determines whether a lost frame in a wireless network is due to mobility or congestion. As one of the lessons learned in the first step, it is important to enable the routing protocol to avoid unnecessary route repair that can lead to lower performance. MDA determines when a lost frame is truly a sign of a route failure before it reacts. Using simulations, we demonstrate that MDA is able to reduce routing overhead, which enables it to increase throughput significantly; MDA can increase throughput by 10 to 100%, depending on the routing protocol and the mobility scenario.

In the fourth step, we design and evaluate a hop-by-hop congestion control transport protocol, HCP, for multicast applications. HCP has several abilities that makes it provide high throughput but with a lower overhead. HCP supports local routing of lost packets by having each of the intermediate forwarders buffer packets

and retransmit them if the downstream nodes miss them. Furthermore, the buffering capability enables each forwarder node to store and transmit data at different rates – an ability that is useful when the network has different regions with different available bandwidths. Our simulation results demonstrate that HCP is able to adjust its sending rate as the network bandwidth changes, is fair to other co-existing flows, and can deliver data at different rates to different receivers as per their available bandwidth. Furthermore, when we compare HCP with an optimal peer-to-peer application layer multicast under varying conditions of traffic and mobility, our results indicate that HCP is able to deliver better throughput with a low overhead.

8.2 Future Work

However, much remains to be explored!

For ASSM, its current approach of using a periodic Join message to repair link break should be replaced by a more reactive approach; a group member should be able to measure the average inter-packet delay, and then trigger a repair if this delay grows too large. In the future, we plan to examine alternative forwarding methods, and methods for handling multiple sources in the same group. We also plan to study scalability in terms of the network size, group size, and the traffic rate. Finally, we plan to compare ASSM to multicast routing protocols that use a hierarchy or overlay to demonstrate that multicast scales better when it is build on top of scalable unicast routing.

For MDA, we need to explore several additional related areas. First, MDA currently considers only mobility and congestion as sources of loss. However, frames may also be lost due to interference from other technologies such as Bluetooth, microwave ovens, and cordless phones. A complete architecture should also be able to detect interference from these sources and determine the correct nature of loss. Second, we can test MDA with other congestion control algorithms. Many of our results show

a dramatic decrease in routing overhead, with significant but more modest gains in throughput. It is possible that a different congestion control algorithm may be able to take better advantage of the reduced routing overhead afforded by MDA.

For HCP, we need to address several additional mechanisms. First, it would be helpful to reduce the amount of packet loss due to contention. One possibility would be to use rate-based congestion control to slow down the forwarding nodes, so that promiscuous listening succeeds more often. Another possibility is to substitute a reliable MAC-layer multicast for our semi-reliable algorithm; BMW [70] is likely a good starting point. Finally, we would like to further study the tradeoffs of buffering versus local recovery. For example, perhaps a parent and a child should cache different packets, so there is a chance that if one of them is missing a packet, then the other one has it. In the absence of a cache coordination protocol, random cache replacement might provide this benefit without any overhead.

With our multicast transport architecture, we conduct several simulations for demonstrating that HCP is fair when there are multiple HCP co-existing flows. However, there are two cases when additional work needs to be done. First, in our current simulations, all competing flows use the same source, destination, and the intermediate routers. The next logical step would be to examine fairness, when multiple flows pass through the a set of common intermediate routers, but have different sources and destinations. Second, we also need to study fairness when an HCP flow contends for the medium with a non-HCP flow like ATP.

Bibliography

[1] I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges," in *Ad Hoc Networks 1 pp. 13-64*, 2003.

[2] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla, "A Reliable, Congestion-Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks," *5th International Symposium on Wireless Personal Multimedia Communications*, pp. 252–256, Oct, 2002.

[3] Venkatesh Rajendran, Katia Obraczka, Yunjung Yi, Sung-Ju Lee, Ken Tang, and Mario Gerla, "Combining Source and Localized Recovery to Achieve Reliable Multicast in Multi-Hop Ad Hoc Networks," in *IFIP-TC6 Networking Conference*, May, 2004, pp. 112–124.

[4] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang, "A case for End System Multicast," in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12.

[5] Chao Gui and Prasant Mohapatra, "Efficient Overlay Multicast for Mobile Ad Hoc Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, March 2003.

[6] Yung Yi and Sanjay Shakkottai, "Hop-by-hop Congestion Control over a Wireless Multi-hop Network," in *IEEE INFOCOM*, 2004.

[7] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol," RFC 1075, November 1988, http://www.rfc-editor.org/rfc/rfc1075.txt.

[8] S. Lee, W. Su, and M. Gerla, "On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks," in *ACM/Baltzer Mobile Networks and Applications, special issue on Multipoint Communication in Wireless Mobile Networks*, 2000.

[9] Manoj Pandey and Daniel Zappala, "A Scenario Based Evaluation of Ad Hoc Multicast Routing Protocols," in *IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), Italy*, June 2005.

[10] J. Jetcheva and D.B. Johnson, "Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks," in *ACM MOBIHOC*, October 2001.

[11] E.M. Royer and C.E. Perkins, "Multicast Operation of the Ad-Hoc On-Demand Distance Vector Routing Protocol," in *Mobile Computing and Networking*, 1999.

[12] C.E. Perkins and E.M. Royer, "Ad Hoc On Demand Distance Vector (AODV) Algorithm," in *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[13] A. Adams, Jonathan Nicholas, and William Siadak, "Protocol Independent Multicast-Dense Mode (PIM-DM): Protocol Specification (Revised)," Sep 2003, Internet Draft (Work in Progress), draft-ietf-pim-dm-new-v2-04.txt.

[14] S. Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Chinggung Liu, and Liming Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Motivation and Architecture," Oct 1996, Internet Draft (Work in Progress), draft-ietf-idmr-pim-arch-01.psDraft.

[15] D.B. Johnson and D.A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, Imielinski and Korth, Eds., vol. 353. Kluwer Academic Publishers, 1996.

[16] Manoj Pandey, Daniel Delorey, Lei Wang Qiuyi Duan, Charles Knutson, Daniel Zappala, and Ryan Woodings, "RIA: An RF Interference Avoidance Algorithm for Heterogeneous Wireless Networks," in *IEEE Wireless Communication and Networking Conference (WCNC), Hong Kong*, 2007.

[17] Ryan Woodings and Manoj Pandey, "WirelessUSB: A Low Power, Low Latency and Interference Immune Wireless Standard," in *IEEE Wireless Communication and Networking Conference (WCNC), Las Vegas*, April 2006.

[18] Feng Wang and Yongguang Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response," in *ACM MOBIHOC*, 2002.

[19] Karthikeyan Sundaresan, Vaidyanathan Anantharaman, Hung-Yun Hsieh, and Raghupathy Sivakumar, "ATP: a reliable transport protocol for ad-hoc networks ," in *ACM MOBIHOC*, 2003.

[20] Kartik Chandran, Sudarshan Raghunathan, S. Venkatesan, and Ravi Prakash, "A Feedback Based Scheme for Improving TCP Performance in Ad-Hoc Wireless Networks," in *International Conference on Distributed Computing Systems (ICDCS)*, 1997, pp. 472–479.

[21] Gavin Holland and N. Vaidya, "Analysis of TCP Performance Over Mobile Ad Hoc Networks," in *IEEE/ACM MOBICOM*, August 1999, pp. 219–230.

[22] R. Chandra, V. Ramasubramanian, and K. Birman, "Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks," in *21st International Conference on Distributed Computing Systems (ICDCS)*, 2001, pp. 275–283.

[23] J. Luo, P.Th. Eugster, and J.-P. Hubaux, "Route Driven Gossip: Probabilistic Reliable Multicast in Ad Hoc Networks," in *IEEE INFOCOM*, 2003.

[24] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 784–803, Dec. 1997.

[25] John C. Lin and Sanjoy Paul, "RMTP: A Reliable Multicast Transport Protocol," in *IEEE INFOCOM*, San Francisco, CA, Mar. 1996, pp. 1414–1424.

[26] R. Yavatkar and Manoj L, "Optimistic strategies for large-scale dissemination of multimedia information. ," in *ACM Multimedia* , Aug 1993.

[27] S. Bhattacharyya, D. Towsley, and J. Kurose, "The loss path multiplicity problem in multicast congestion control," in *IEEE INFOCOM*, March 1999.

[28] Injong Rhee, Nallathambi Ballaguru, and George N. Rouskas, "MTCP: Scalable TCP-like congestion control for reliable multicast," in *IEEE INFOCOM*, Mar. 1999.

[29] Lorenzo Vicisano1, Luigi Rizzo, and Jon Crowcroft, "TCP-like congestion control for layered multicast data transfer," in *IEEE INFOCOM*, 1998.

[30] Steven McCanne, Van Jacobson, and Martin Vetterli, "Receiver-driven Layered Multicast," in *ACM SIGCOMM*, New York, Aug. 1996, vol. 26,4, pp. 117–130, ACM Press.

[31] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *NOSSDAV*, June 2001.

[32] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D Joseph, and John D. Kubiatowicz, "Tapestry: A global-scale overlay for rapid service deployment," *IEEE Journal on Selected Areas in Communications*, 2003, Special Issue on Service Overlay Networks, to appear.

[33] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker, "A scalable content-addressable network," in *Conference on applications, technologies, architectures, and protocols for computer communications.* 2001, pp. 161–172, ACM Press.

[34] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker, "Application-level multicast using content-addressable networks," *Lecture Notes in Computer Science*, vol. 2233, 2001.

[35] Vijay Raisinghani and Sridhar Iyer, "Cross layer design optimizations in wireless protocol stacks," *Computer Communications*, 2003.

[36] I Akyildiz, Y Altunbasak, F Fekri, and R Sivakumar, "AdaptNet: An Adaptive Protocol Suite for the Next-Generation Wireless Internet," *IEEE Communications Magazine*, pp. 128–136, March 2004.

[37] W. Yuen, H. Lee, and T. Andersen, "A simple and effective cross layer networking system for mobile ad hoc networks," in *PIMRC*, 2002.

[38] Lusheng Ji and M. Scott Corson, "Differential Destination Multicast - A MANET Multicast Routing Protocol for Small Groups," in *IEEE INFOCOM*, 2001.

[39] Seungjoon Lee and Chongkwon Kim, "Neighbor Supporting Ad Hoc Multicast Routing Protocol," in *ACM MOBIHOC*, Aug 2000.

[40] Subir Kumar Das, B. S. Manoj, and C. Siva Ram Murthy, "A Dynamic Core Based Multicast Routing Protocol for Ad Hoc Wireless Networks," in *ACM MOBIHOC*, June 2002.

[41] C. Wu, Y. Tay, and C. Toh, "Ad hoc Multicast Routing Protocol Utilizing Increasing id-numberS (AMRIS) Functional Specification," Nov 1998, Internet Draft (Work in Progress), draft-ietf-manet-amris-spec-00.txt.

[42] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Communications & Mobile Computing (WCMC)*, 2002.

[43] S. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols," in *IEEE INFOCOM*, 2000.

[44] P. Johansson, T. Larsson, N. Hedman, B. Mielczarek, and M. Degermark, "Scenario Based Peformance Analysis of Routing Protcols for Mobile Ad Hoc Networks," in *IEEE MOBICOM*, August 1999.

[45] X. Hong, M. Gerla, G. Pei, and C. Chiang, "A Group Mobility Model for Ad Hoc Wireless Networks," in *ACM MSWiM*, August 1999.

[46] F. Bai, N. Sadagopan, and A. Helmy, "IMPORTANT: A framework to systematically analyze the Impact of Mobility on Performance of RouTing protocols for Adhoc Networks," in *IEEE INFOCOM*, 2003.

[47] T. Ozaki, J. B. Kim, and T. Suda, "Bandwidth-Efficient Multicast Routing for Multihop, Ad-Hoc Wireless Networks," in *IEEE INFOCOM*, 2001.

[48] P. Gupta and P. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. IT-46, no. 2, pp. 388–404, March 2000.

[49] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris, "Capacity of Ad Hoc wireless networks," in *Mobile Computing and Networking*, 2001, pp. 61–69.

[50] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks," in *Workshop on Parallel and Distributed Simulation*, May 1998.

[51] J. Jetcheva and D. B. Johnson, "Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks," July 2001, Internet Draft (Work in Progress), draft-ietf-manet-admr-00.txt.

[52] V.A. Davies, "Evaluating Mobility Models Within an Ad Hoc Network," M.S. thesis, Colorado School of Mines, Mathematical and Computer Sciences, 2000.

[53] C. Bettstetter, "Smooth is Better than Sharp: A Random Mobility Model for Simulation of Wireless Networks," in *ACM MSWiM*, July 2001.

[54] Manoj Pandey and Daniel Zappala, "The Effects of Mobility on Multicast Routing in Ad Hoc Networks," in *Technical Report, UO-CIS-TR-2004-2*, March 2004.

[55] A. Erramilli and R.P Singh, "A Reliable and Efficient Multicast Protocol for Broadband Broadcast Networks," in *ACM SIGCOMM*, August 1987.

[56] J. Chuang and M. Sirbu, "Pricing Multicast Communications: A Cost-Based Approach," *Telecommunication Systems*, vol. 17, no. 3, pp. 281–297, July 2001.

[57] G. Phillips, H. Tangmunarunkit, and S. Shenker, "Scaling of Multicast Trees: Comments on the Chuang-Sirbu Scaling Law," in *ACM SIGCOMM*, 1999.

[58] Yunjung Yi, Mario Gerla, and Katia Obraczka, "Scalable Team Multicast in Wireless Networks Exploiting Coordinated Motion," *Ad Hoc Networks Journal*, August 2003.

[59] D. R. Cheriton and H.W. Holbrook, "EXPRESS Multicast: Making Multicast Economically Viable," in *ACM SIGCOMM*, 1999.

[60] C. Gui and P. Mohapatra, "Scalable multicasting in mobile ad hoc networks," *IEEE INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 3, pp. 2119–2129 vol.3, 7-11 March 2004.

[61] P. Sinha, R. Sivakumar, and V. Bharghavan, "MCEDAR: multicast core-extraction distributed ad hoc routing," *IEEE Wireless Communications and Networking Conference, 1999*, pp. 1313–1317 vol.3, 1999.

[62] Chao Gui and Prasant Mohapatra, "Overlay multicast for manets using dynamic virtual mesh," *Wireless Networking*, vol. 13, no. 1, pp. 77–91, 2007.

[63] Andrea Passarella and Franca Delmastro, "Usability of legacy p2p multicast in multihop ad hoc networks: an experimental study," *EURASIP J. Wirel. Commun. Netw.*, vol. 2007, no. 1, pp. 38–38, 2007.

[64] M. Ge, S.V. Krishnamurhty, and M. Faloutsos, "Application versus network layer multicasting in ad hoc networks: the alma routing protocol," *Ad Hoc Networks Journal*, vol. 4, no. 2, pp. 283–300, 2006.

[65] Li Xiao, Abhishek P. Patil, Yunhao Liu, Lionel M. Ni, and Abdol-Hossein Esfahanian, "Prioritized Overlay Multicast in Mobile Ad Hoc Environments," *IEEE Computer 37(2): 67-74*, 2004.

[66] Kai Chen and Klara Nahrstedt, "Effective location-guided tree construction algorithms for small group multicast in MANET," in *IEEE INFOCOM*, June 2002.

[67] M. Ge, Krishnamurthy S.V., and Faloutsos M., "Overlay Multicasting in Ad Hoc Networks," in *Third Meditteranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2004.

[68] Manoj Pandey, Roger Pack, Lei Wang, Qiuyi Duan, and Daniel Zappala, "To Repair or Not to Repair: Helping Routing Protocols to Distinguish Mobility From Congestion," in *IEEE INFOCOM MiniSymposia, Anchorage*, 2007.

[69] C. Ho, K. Obraczka, G. Tsudik, and K. Viswanath, "Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks," in *International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999.

[70] Ken Tang, Katia Obraczka, Sung-Ju Lee, and Mario Gerla, "A Reliable, Congestion-Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks ," in *WPMC*, 2002.

[71] S. McCanne and S. Floyd, "NS Simulator, http://www.isi.edu/nsnam/ns/," .

[72] David B. Johnson, David A. Maltz, and Yih-Chun Hu, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)," July 2004, Internet Draft (Work in Progress), draft-ietf-manet-dsr-10.txt.

[73] B.P. Crow, I. Widjaja, J.G. Kim, and P.T. Sakai, "IEEE 802.11 Wireless Local Area Networks," in *IEEE Communications Magazine, pp. 116–126*, September 1997.

[74] S. Murthy and J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks," in *ACM Mobile Networks and App. J., Special Issue on Routing in Mobile Communication Networks*, Oct 1996.

[75] Dhananjay S. Phatak Tom Goff, Nael B. Abu-Ghazaleh and Ridvan Kahvecioglu, "Preemptive Routing in Ad Hoc Networks," in *ACM MOBICOM*, 2001.

[76] Fabius Klemm, Zhenqiang Ye, Srikanth V. Krishnamurthy, and Satish K. Tripathi, "Improving TCP Performance in Ad Hoc Networks Using Signal Strength Based Link Management," *Ad Hoc Networks*, vol. 3, no. 2, pp. 175–191, 2005.

[77] Saad Biaz and N. Vaidya, "Derandomizing Congestion Losses To Improve TCP Performance over Wired-Wireless Networks," *IEEE/ACM Transactions on Networking, August 2004*, Aug 2004.

[78] S. Cen, P. Cosman, and G. Voelker, "End-to-end Differentiation of Congestion and Wireless Losses," in *ACM Multimedia Computing and Networking*, 2002.

[79] Gavin Holland and Nitin H. Vaidya, "Analysis of TCP Performance Over Mobile Ad Hoc Networks," in *IEEE/ACM MOBICOM*, August 1999, pp. 219–230.

[80] J. Monks, P. Sinha, and V. Bharghavan, "Limitations of TCP-ELFN for Ad Hoc Networks," in *MOMUC 2000*, 2000.

[81] Zhenghua Fu, Petros Zerfos, Haiyun Luo, Songwu Lu, Lixia Zhang, and Mario Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," in *IEEE INFOCOM*, 2003.

[82] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE J–SAC*, vol. 19, no. 7, pp. 1300–1315, 2001.

[83] S. Jamaloddin Golestani and Krishan K. Sabnani, "Fundamental observations on multicast congestion control in the internet," in *IEEE INFOCOM*, 1999.

[84] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron, "Scribe: a large-scale and decentralized application-level multicast infrastructure," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1489–1499, Oct 2002.

[85] Su Wen, J. Griffioen, and K.L. Calvert, "Calm: congestion-aware layered multicast," *Open Architectures and Network Programming Proceedings, 2002 IEEE*, pp. 179–190, 2002.

[86] M. Liu, R. Talpade, and A. McAuley, "Amroute: Adhoc multicast routing protocol," Technical Report TR 99-1, CSHCN, 1999.

[87] P. P. Mishra and H. Kanakia, "A hop-by-hop rate-based congestion control scheme," *ACM SIGCOMM*, Aug. 1992.

[88] Partho P. Mishra and Hemant Kanakia, "A Hop by Hop Rate-Based Congestion Control Scheme," in *ACM SIGCOMM*, 1992.

[89] Cüneyt Özveren, Robert Simcoe, and George Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," in *ACM SIGCOMM*, 1994.

[90] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, "On the Scalability of Fair Queueing," in *ACM HotNets*, 2004.

[91] Yung Yi and Sanjay Shakkottai, "Hop-by-Hop Congestion Control over a Wireless Multi-Hop Network," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, 2007.

[92] Bret Hull, Kyle Jamieson, and Hari Balakrishnan, "Mitigating Congestion in Wireless Sensor Networks," in *ACM SenSys*, Baltimore, MD, November 2004.

[93] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell, "CODA: congestion detection and avoidance in sensor networks," in *ACM SenSys*, 2003.

[94] A. Woo and D.C. Culler, "A Transmission Control Scheme for Media Access in Sensor Networks," in *ACM MOBICOM*, 2004.

[95] C.-T. Ee and R. Bajcsy, "Congestion Control and Fairness for Many-to-One Routing in Sensor Networks," in *ACM SenSys*, 2004.

[96] T. Gopalsamy, M. Singhal, D. Panda, and P. sadayappan, "A Reliable Multicast Algorithm for Mobile Ad Hoc Networks," in *International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[97] R. Chandra, V. Ramasubramanian, and K. Birman, "Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks," in *21st International Conference on Distributed Computing Systems (ICDCS)*, 2001, pp. 275–283.

[98] A. Sobeih, H. Baraka, and A. Fahmy, "ReMHoc: a reliable multicast protocol for wireless mobile multihop ad hoc networks," in *IEEE Consumer Communications and Networking Conference, (CCNC)*, 2004.

[99] Dimitrios Koutsonikolas and Y. Charlie Hu, "The case for fec-based reliable multicast in wireless mesh networks," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 491–501.

[100] K. Tang and M. Gerla, "Congestion control multicast in wireless ad hoc networks," *Computer Communications*, vol. 26, no. 3, pp. 278–288, February 2003.

[101] Alan Demers, Srinivasan Keshav, and Scott Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *ACM SIGCOMM*, 1989.

[102] Kaixin Xu, Mario Gerla, Lantao Qi, and Yantai Shu, "Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED," in *ACM MOBICOM*, 2003.

[103] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, McGraw-Hill, 2001.