A Performance Comparison of Reliable Multicast Protocols over Mobile Ad Hoc Networks

by

LAN HUANG

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario Canada

September 2004

Your file Votre référence
ISBN: 0-612-99762-6
Our file Notre référence
ISBN: 0-612-99762-6

ABSTRACT

Reliable Multicast, which provides lossless delivery of a data stream from one sender to a group of receivers, has been intensively studied by the wired network research community in recent years. A number of reliable multicast protocols have been proposed and developed for the wired network environment. Wireless ad hoc networks, a class of wireless networks with no centralized entities, exhibit intrinsic characteristics that hinder wired reliable multicast protocols from being directly applicable. Nevertheless, there is an apparent demand for reliable multicast protocols that is amenable to wireless domains. More specifically, it is becoming increasingly important to determine how effectively such a protocol can improve communication reliability in wireless ad hoc networks.

In this thesis, we study the application of four known classes of wired reliable multicast protocols on ad hoc networks. These classes are: Sender-initiated, Receiver-initiated, Tree-based and Ring-based. Using several simulation scenarios, we observe the behavior of three representative protocols, namely SRM, TMTP, and RMP, in reacting to various testing stimuli in wireless ad hoc networks, and compare their performances. We demonstrate that wired reliable multicast protocols may not always be adequate for wireless ad hoc networks. Based on observations, we make recommendations on how the performance of such protocols can be improved to be amenable to the wireless ad hoc network environment.

ACKNOWLEDGMENTS

I have received enormous assistance and encouragement on this thesis, from people too numerous to name. I gratefully give special thanks to the following, however.

I would like to thank my supervisor, Dr. Hossam Hassanein, for advising me through my research. I also thank him for his advice, support and encouragement, and the freedom he gave me to make my own decisions.

I would like to thank Kenan Xu, Yong Xu, Mohammed Al-Riyami, Nidal Nasser, Tiantong You, and the rest of the Telecommunications Research group for their friendship, advice, encouragement and shining examples. Special thanks go to Abdelhamid Taha for his help in proofreading the thesis.

Financial support of Communications and Information Technology Ontario (CITO) is greatly appreciated.

Thank my husband Yan Li, my parents and family for their love and constant encouragement. This could not have been done without them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

λ	packet arrival rate
ACK	positive ACKnowledgement
AG	Anonymous Gossip
ALF	Application Level Framework
AMRoute	Adhoc Multicast Routing
AODV	Ad Hoc On-Demand Distance Vector protocol
B	Background traffic
CBR	Constant Bit Rate
CTS	Current Token Site
DCF	Distributed Coordination Function
DM	Domain Manager
DSR	Dynamic Source Routing protocol
DVMRP	Distance Vector Multicast Routing Protocol
G	Group size
IP	Internet Protocol
LAN	Local Area Networks
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
N	Number of nodes

NACK	Negative Acknowledgement
NS-2	Network Simulator 2
NTS	Next Token Site
ODMRP	On-Demand Multicast Routing Protocol
P	Pause time
PCF	Point Coordination Function
PDA	Personal Data Assistant
R	transmission Range
RALM	Reliable Adaptive Lightweight Multicast
RMP	Reliable Multicast protocol
RMTP	Reliable Multicast Transport Protocol
SRM	Scalable Reliable Multicast protocol
TCP	Transfer Control Protocol
TMTP	Tree-based Multicast Transport protocol
TRP	Token Ring Protocol
TTL	Time-To-Live

1 INTRODUCTION

Multicast is defined as a one-to-many, or many-to-many type of communication. That is, the transmission of the same information from one or multiple senders to several destinations [1]. Multicast provides an efficient way of disseminating data from a sender to a group of receivers. A sender's data stream is transmitted only once on links shared along the paths to a set of destinations [1]. Without multicasting, a source would have to send a separate copy of the same data to each individual receiver, and valuable bandwidth would be unnecessarily wasted. Examples of applications making extensive use of multicast include software distribution and video conferencing. Different multicasting applications have different requirements. For example, video conferencing can tolerate minor video data loss but cannot tolerate the delay of voice data associated with retransmissions. On the other hand, data dissemination applications, such as file distribution can afford large end-to-end delay, yet cannot sacrifice data loss, i.e., it considers reliable delivery to all receivers as the main requirement [2]. Reliable multicast provides a lossless multicast service that data dissemination applications require.

A Mobile Ad Hoc Network (MANET) is a communication network that is formed by a group of autonomous mobile nodes [3]. A mobile node can be any computing or communication device with a wireless interface, such as a laptop computer with a wireless LAN card, a mobile phone, or a Personal Data Assistant (PDA) handheld with infrared interface. These mobile nodes connect freely and dynamically together into arbitrary and temporary topologies via wireless links, allows people and devices to communicate with each other without using a physical network infrastructure [4]. The term *network infrastructure* refers to the facility of which the sole purpose is to carry the data generated by each mobile node to the respective destination nodes [5]. In an ad hoc network such as the one shown in Figure 1-1, each of the mobile nodes operates in a distributed peer-to-peer mode and there is no central administration. Due to the limited transmission range of wireless network interfaces, routes are often constructed in "multi-hops" [6], i.e. a source-to-destination path could pass through several intermediate nodes.

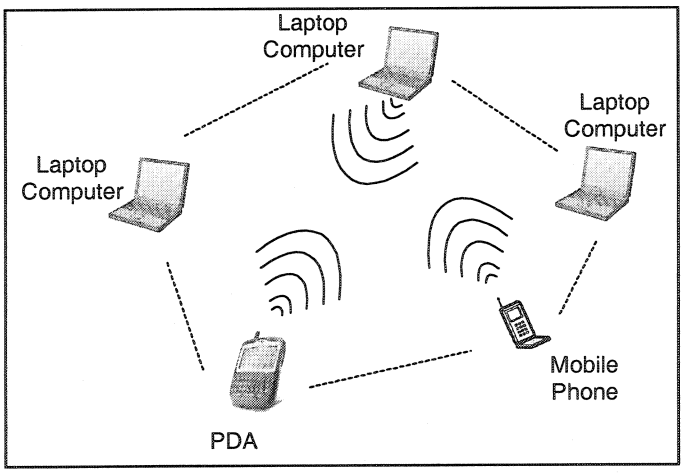


Figure 1-1 An example of mobile ad hoc networks

Due to its ease of deployment, a mobile ad hoc network is an attractive choice for scenarios where rapid deployment and dynamic reconfiguration are necessary and a wired network is non-existent or unavailable. Some examples of where ad hoc networks are deployed are military battlefields, emergency search and rescue sites after a hurricane or earthquake, and conference room where participants share information dynamically using their mobile devices. These applications lend themselves well to multicast operation since most of them require close collaboration of teams for audio and video message change. Multicasting is a very useful and efficient means of supporting such group-oriented applications. This is especially the case in ad hoc networks where bandwidth is scarce and mobile nodes have limited power. Moreover, applications in critical situations such as disaster recovery or battlefield scenarios require reliable multicast operation that provides errorless and timely data delivery services.

Reliable Multicast can guarantee reliable data multicast in ad hoc networks. However, providing reliable multicast service faces several key challenges in ad hoc networks. The network topology can change randomly and rapidly, at unpredictable times [3]. Wireless links generally have limited bandwidth [4]; congestion is typically the norm rather than the exception. Moreover, the majority of nodes rely on short-living batteries [3]. Therefore, we need an efficient algorithm that reduces the amount of control and retransmission traffic while multicasting is maintained reliable. Many applications in ad hoc networks are life-critical. Hence, the protocols should also minimize end-to-end delay in addition to reliable delivery to meet application requirements.

In recent years, reliable multicast protocols have been intensively studied by the wired network research community. A number of reliable multicast protocols have been proposed and developed for the wired network environment. Due to the nature and intrinsic characteristics of wireless ad hoc networks, the reliable multicast protocols for wired networks may not be able to be applied in ad hoc networks. The issues of which reliable multicast protocols are more amenable to wireless domain and how effectively they can improve communication reliability over wireless ad hoc networks are, therefore, becoming increasingly important to be overcome. We opt to take an approach of extending and enhancing the existing protocols to cover ad hoc networks instead of creating new protocols that are only tailored for ad hoc networks. This is to take advantage of the wide acceptance and stability of the current protocols. More importantly, we can eventually obtain a unified solution that works equally well in wired networks and wireless ad hoc networks, to achieve seamless integration of both networking platforms. We argue that the design choices underlying wired reliable multicast protocols are not adequate for ad hoc networks. Protocols for ad hoc networks must handle node mobility. In addition, ad hoc networks are extremely sensitive to network load and congestion. Generating additional control message overhead without performing adequate congestion control will considerably degrade the performance. While we acknowledge that wired protocols were not designed for ad hoc networks, studying their behavior in various scenarios will definitely yield insights that will be useful in designing new protocols for ad hoc networks.

This thesis presents an overview of well known classes of reliable multicast protocols proposed for wired networks, and evaluates the performance of three representative protocols over wireless ad hoc networks using the network simulator NS-2 ([7]).

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of mobile ad hoc networks and four known classes of reliable multicast protocols proposed for wired network, namely, Sender-initiated, Receiver-initiated, Tree-based, and Ring-based. Chapter 3 describes three representative reliable multicast protocols for wired network: Scalable Reliable Multicast protocol (SRM) [8], Tree-based Multicast Transport protocol (TMTP) [9], and Reliable Multicast protocol (RMP) [10]. In chapter 4 we develop a comprehensive simulation model and utilize it to observe the behavior of the three reliable multicast protocols under various simulation scenarios. We report on the strengths and weaknesses of each protocol and propose modifications to the protocols in order to achieve better performance over wireless ad hoc networks. Finally chapter 5 presents conclusions and future work.

2 RELATED WORK

In this chapter, we first present the characteristics of mobile ad hoc networks, in addition to describing unicast and multicast routing protocols of mobile ad hoc networks that were used in our simulation. We then review four classes of reliable multicast protocols for wired networks, and briefly discuss their strengths and weaknesses. We present two proposed reliable multicast protocols for ad hoc networks at 2.4.

2.1 Mobile Ad Hoc Networks

The concept, definition and utility of mobile ad hoc networks have been investigated for more than two decades. Initially, ad hoc networks were primarily aimed at tactical, military networks. In recent years, there is a growing interest in exploring the possibility of applying ad hoc networks for civilian purposes. In this section, we discuss the characteristics of ad hoc networks. We also discuss the unicast and multicast routing protocols that we chose to use in our simulation.

2.1.1 Characteristics of Mobile Ad Hoc Networks

Ad hoc networks have many distinguished characteristics that lend themselves to applications like home networks, sensor networks and emergency responses network [11]. In the following, we list the key characteristics of ad hoc network [3, 4, 6]:

(1) Mobility - Most or all nodes in ad hoc network are wireless, mobile devices. They can freely join and leave a network, and they can arbitrarily move during communication. Mobility brings convenience.

(2) No fixed network infrastructure - Ad hoc networks were initiated as a mean of communicating in battlefield where network infrastructures are not always available or infrastructure can be easily interfered or destroyed by an enemy. One important goal of ad hoc network is not to rely on any form of infrastructure for support in routing, network management and transmission. All nodes are equipped with packet forwarding capabilities, i.e. each mobile node acts not only as a host but also as a router [6]. Each node in ad hoc networks can communicate with its peers directly in single hop or via its peers' relays, i.e. in multi-hop, without using an existing infrastructure. Naturally, this fact significantly reduces the cost of deploying infrastructure and the cost of maintaining and administrating network. Also, without the physical location limitation of infrastructure, ad hoc networks can be wherever the mobile nodes are. They can also assemble and disassemble as demand requires.

(3) Dynamic network topology - The cost of mobility is network topology changing frequently and unpredictably. There are many reasons causing network topology to be changed. For instance, each node in ad hoc network can arbitrarily decide to join

in and leave the network. Also it is disconnected from the network when a node accidentally moves out of wireless transmission range, or runs out of power, or gets unexpected transmission inferences, etc.

(4) Bandwidth and energy constraints - Wireless links have limited bandwidth. Moreover, since wireless links have lower capacity than hardwired links, traffic congestion is typical rather than exceptional. Therefore, ad hoc networks' applications must minimize various network overheads. Most nodes in an ad hoc network are battery powered [11] in order to support mobility. Because battery life is limited, protocols and applications need to be efficient, so that the limited battery life can drive devices for longer period.

2.1.2 IEEE 802.11 MAC

In an ad hoc network, the medium is shared by all mobile nodes that are in the same radio communication range, and the radio frequency bandwidth is limited. Therefore, Medium Access Control (MAC) schemes are needed to coordinate access to the shared medium in the network [12].

The IEEE 802.11 protocol [13], which defines the MAC and Physical Layer standards for wireless connectivity for fixed, portable and moving stations within a local area, is the most popular MAC for wireless LANs. It consists of Distributed Coordination Function (DCF) and Point Coordination Function (PCF), where DCF is more commonly deployed function. The DCF is based on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) mechanism. A station that has information to transmit listens to the medium

before transmitting. If the medium is busy, the station will defer its transmission. This reduces collisions and enhances performance. Details of the IEEE 802.11 specifications can be found in [13].

2.1.3 Dynamic Source Routing Protocol

Numerous protocols have been proposed to solve the multi-hop routing problem in ad hoc networks. Several performance analysis [6, 14, 15] have shown that Dynamic Source Routing protocol (DSR) [16] and Ad Hoc On-Demand Distance Vector protocol (AODV) [17] are the most prominent unicast routing protocols to date. In this section, we briefly describe DSR, which we use as the unicast routing protocol in our simulation.

DSR is an on-demand routing protocol where routes are only created when desired by the source node. A key difference of DSR from other on-demand protocols is the use of source routing, where a source of a packet determines the complete route from itself to the destination, and includes the route in the packet header. All of the intermediate nodes along the route simply forward the packet to the next hop indicated in this predetermined route, also called source route. No routing decision is made at intermediate nodes. The advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since the packets themselves already contain all the routing decisions. The obvious disadvantage is that data packets must carry source routes.

DSR consists of two major operations: Route Discovery and Route Maintenance. Each node maintains a cache of source routes it has learned so far, called route cache. When a

node attempts to send a data packet to a destination, it first checks its route cache to determine whether it already has a route to the destination. If an unexpired route to the destination is found, the node uses this route to send the packet. Otherwise, the node initiates a Route Discovery operation to discover a route. Route Discovery works by broadcasting the network with ROUTE REQUEST packets. A route request contains the address of the destination as well as a ROUTE RECORD that records the nodes that the request has passed by. Each node receiving a route request checks whether it knows a route to the destination, i.e. the desired route is contained in its route cache, or itself is the destination. In both cases, the complete route from the initiator to the destination is found. A ROUTE REPLY packet with the route included is generated and then forwarded to the initiator. Otherwise, the node appends its own address to the route record of the route request and re-broadcasts the route request to its neighbors. Because of the broadcasting, a node may receive multiple copies of the same route request. To limit the number of route request propagated, each node maintains a list of the IDs of the recently seen requests. A node only forwards the route request if the node has not yet seen this route request and the node's address does not already appear in the route record of the packet. Therefore, the node can also detect that a route request has gone through a cycle.

Figure 2-1 illustrates the formation of the route record as the route request packet propagates through the network.

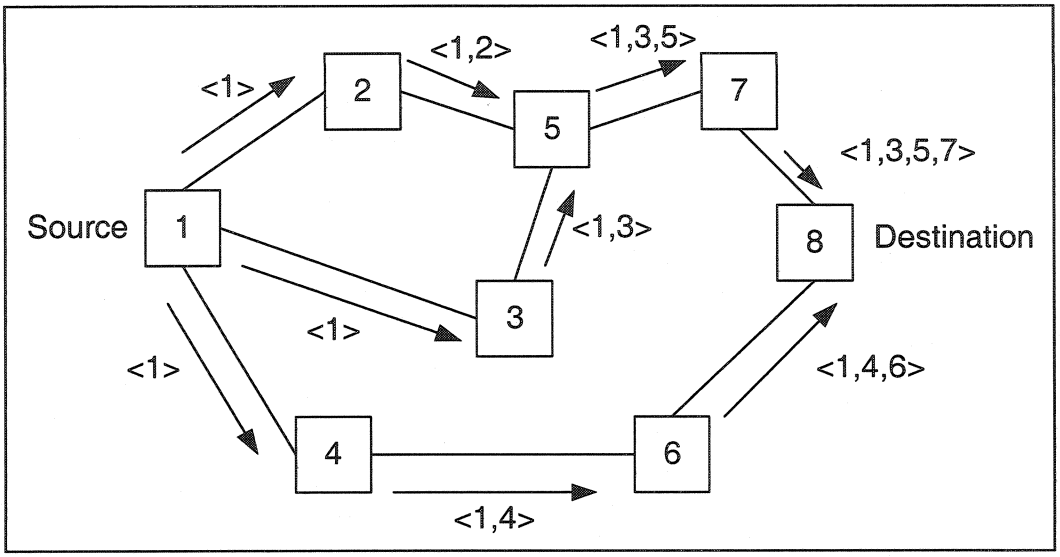


Figure 2-1 Building of the route record during route discovery [18]

Figure 2-2 shows the route reply packet being sent by the destination. If the destination generates the route reply packet, it places the route record from the route request packet into the route reply packet. On the other hand, if an intermediate node generates the route reply packet, then it appends its cached route to the route record of the route request packet and puts that into the route reply packet. The route request builds up the path traversed across the network. The route reply packet routes itself back to the source by traversing this path backward. The route carried back by the route reply packet is cached at the source for future use.

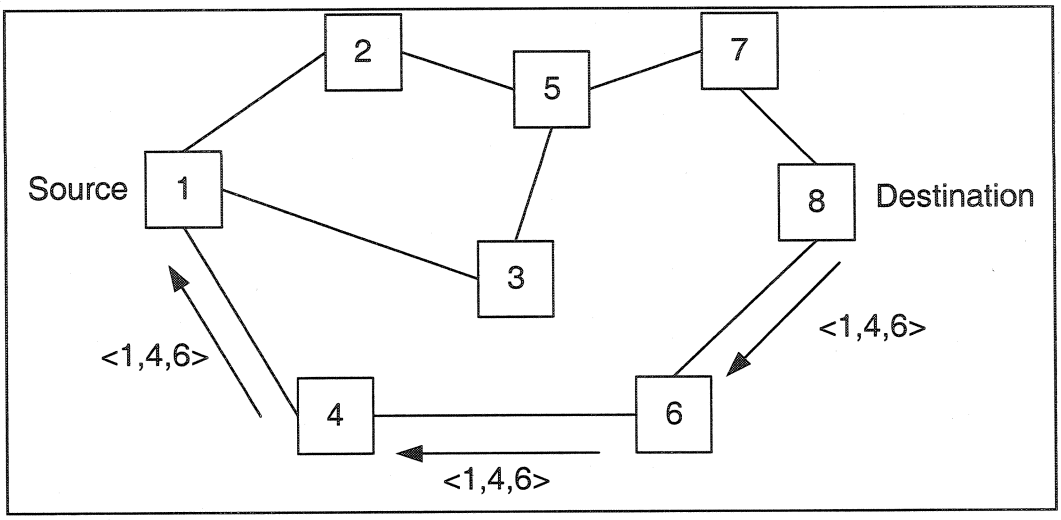


Figure 2-2 Propagation of the route reply with the route record [18]

Route maintenance is invoked when a route is broken. Routes may become invalid due to the node movement. To quickly adapt to this change, each node constantly monitors the links it uses to forward packets. If a node in a route finds out that it cannot forward packets to the next node in the route, it immediately sends a ROUTE ERROR packet to the source of the route. Therefore, the source is able to quickly detect an invalid route and stop using it any longer. The source removes any route using this link from its cache. A new route discovery process must be initiated by the source if this route is still needed.

An advantage of DSR over some of the other on-demand protocols is that DSR does not make use of periodic routing tables, thereby saving bandwidth and reducing power consumption. Additionally, DSR allows nodes to keep multiple routes to a destination in their cache. Hence, when a route is broken, the source can check its cache for another valid route [14]. On the other hand, as the network becomes larger, control packets

(route requests and route replies) and data packets also become larger, since they need to carry the addresses of every node in the route. This can be a problem, since ad hoc networks have limited available bandwidth.

2.1.4 On-Demand Multicast Routing Protocol

Various protocols have been proposed to perform multicast routing in wireless ad hoc networks. They can be categorized to multicast tree-based and mesh-based.

Like traditional multicast routing protocols in wired networks (e.g., Distance Vector Multicast Routing Protocol (DVMRP) [19]), in multicast tree-based routing protocols, a shared multicast tree is established to provide connections among multicast group members and to forward data. Ad hoc Multicast Routing (AMRoute) [20] is a multicast tree-based routing protocol. As pointed out in [21, 22, 23], tree-based routing protocols do not perform well in ad hoc networks because multicast tree are fragile and must be readjusted as connectivity changes. If a single tree link breaks because of node movement, packet collision, or congestion, destinations cannot receive packets.

To overcome these limitations, the On-Demand Multicast Routing Protocol (ODMRP) [21] has been developed by the researchers from University of California in 2000. ODMRP is a mesh-based, instead of a tree-based, multicast routing protocol that provides richer connectivity among multicast members.

The ODMRP protocol is mainly comprised of three parts, namely mesh creation, data forwarding and mesh maintenance. In the following, we briefly introduce these three parts.

Mesh Creation

In ODRMP, group membership and multicast routes are established and updated by the source in an on demand fashion. Similar to on demand unicast routing protocols (e.g. DSR), a request phase and a reply phase comprise the protocol. When a multicast source has packets to send, it periodically broadcasts to the entire network a member advertising packet, called JOIN QUERY. This periodic transmission refreshes the membership information and updates the routes as follows. When a node receives a non-duplicate JOIN QUERY, it stores the upstream node ID (for backward learning) and originating source into its routing table and rebroadcasts the packet. The routing table provides the next hop information when transmitting JOIN REPLIES. When the JOIN QUERY packet reaches a multicast receiver, the receiver broadcasts a JOIN REPLY to its neighbors. The JOIN REPLY contains all source and next node ID that the receiver gets from its routing table. When a node receives a JOIN REPLY, it checks if the next node ID of one of the entries matches its own ID. If it does, the node realizes that it is on the path to the source and thus is part of the forwarding group. It then sets the FG_FLAG (Forwarding Group Flag) and broadcasts its own JOIN REPLY built upon matched entries. The JOIN REPLY is thus propagated by each forwarding group member until it reaches the multicast source via the shortest path. This process constructs (or updates) the routes from sources to receivers and builds a mesh of nodes, the "forwarding group".

The forwarding group is a set of nodes responsible for forwarding multicast packets [21]. It supports shortest paths between any member pairs. As shown in Figure 2-3 all nodes inside the "bubble" forward multicast data packets. A multicast receiver also can be a forwarding group node if it is on the path between a multicast source and another receiver. By building a mesh and supplying multiple routes, multicast packets can be delivered to destinations in the case of node movements and topology changes [22]. Moreover, the drawbacks of multicast trees (frequent tree reconfiguration) are avoided.

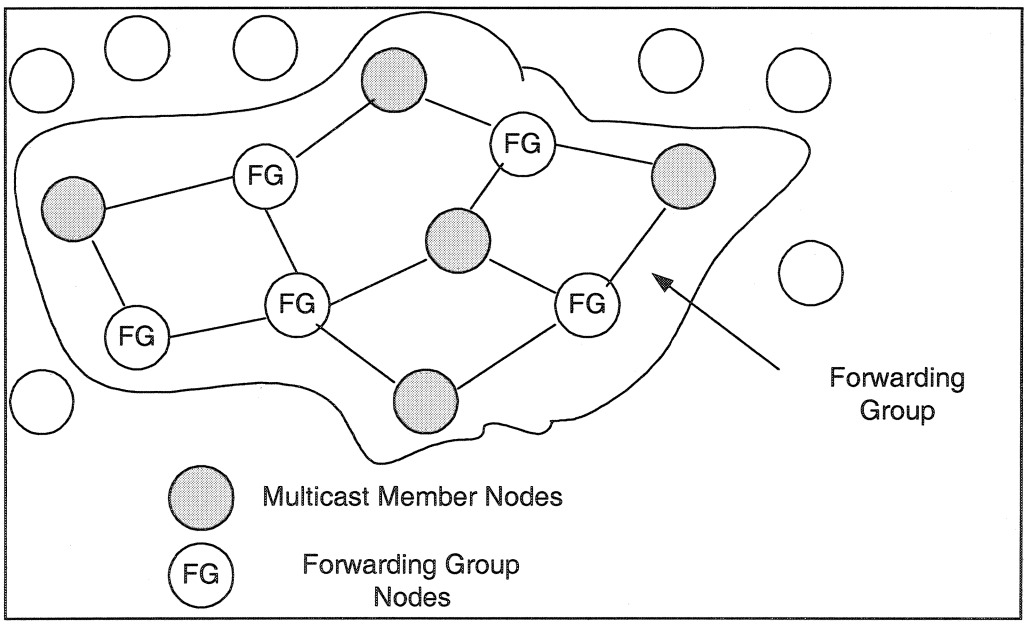


Figure 2-3 The forwarding group concept [21]

Figure 2-4 shows an example of a JOIN REPLY forwarding process. When receivers send their JOIN REPLY messages to next hop nodes, an intermediate node I_1 sets the FG_FLAG and builds its own JOIN REPLY since there is a next node ID entry in the

JOIN REPLY received from R_1 that matches its ID. The JOIN REPLY built by I_1 has an entry for sender S_1 but not for S_2 because the next node ID for S_2 in the received JOIN REPLY is not I_1.

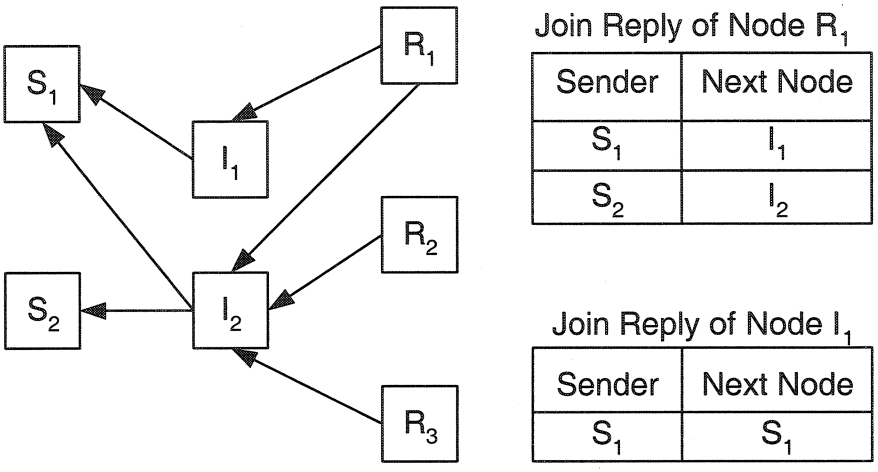


Figure 2-4 An example of a JOIN REPLY forwarding [21]

Data Forwarding

After the group establishment and route construction process, a multicast source can transmit packets to receivers via selected routes and forwarding groups. When a node receives a multicast data packet, it checks the setting of its FG_FLAG. If the flag is set and the data packet is not a duplicate, the node then forwards the data packet to its neighbors.

Mesh Maintenance

In ODMRP, no explicit control packets need to be sent to join in or leave the group. If a multicast source wants to leave the group, it simply stops sending JOIN QUERY packets

since it does not have any multicast data to send to the group. If a receiver no longer wants to receive from a particular multicast group, it does not send the JOIN REPLY for that group. Nodes in the forwarding group are demoted to non-forwarding nodes if not refreshed (no JOIN REPLIES received) before they timeout.

Existing studies ([21, 22, 23]) show that in a harsh environment, where the network topology change frequently, mesh-based protocols outperform tree-based protocols. The availability of alternate routes provides robustness to mobility. We chose ODMRP as underlying multicast routing protocol in our simulation.

2.2 Reliable Multicast

Multicast is defined as transmission of a message from a sender to a group of specific receivers. It is different from unicast, where a message is sent from one source to one destination; or broadcast, in which a message is sent from a source to the whole network. At internet layer of TCP/IP model, IP multicast provides a service of transmission of IP datagrams from a source to a set of zero or more hosts identified by a single IP destination address [24]. With IP multicast, datagrams are delivered to all members of its destination host group. IP multicast delivers datagrams with "best-effort" reliability, i.e., it attempts to send datagrams to the group, but does not guarantee that the datagrams will arrive at all members. Therefore, just as unicast applications require TCP ([25]) on top of IP unicast, a multicast application requires a reliable multicast transport layer protocol on top of IP multicast to guarantee lossless delivery of a data stream to a group of receivers. Reliable multicast is one prime requirement of many applications, such as file or data distribution, financial information, medical images, etc. Reliable multicast transport

layer protocols can employ sequence numbers, positive acknowledgements (ACKs) and/or negative acknowledgements (NACKs) to achieve reliability [26]. Throughout the rest of this thesis, we will simply use "reliable multicast protocols" in reference to reliable multicast transport layers protocols.

2.3 Classes of Reliable Multicast Protocols

Recently, research has been very active on reliable multicast in wired networks. Many reliable multicast protocols have been proposed [8, 9, 10, 27]. A summary of these protocols can be found in [28]. The reliable multicast protocols for wired networks can be classified into the following four types: Sender-initiated [27], where all receivers send ACKs for each packet that they receive; Receiver-initiated [8], where the receivers send NACKs on detection of transmission error or packet loss; Tree-based [9, 29], where receivers are organized into subgroups to relieve the sender from processing all control messages from all receivers; and Ring-based [10, 30], where receivers are organized in a logical ring and receivers take turns to acknowledge the packets received to ensure reliability. In the following sections, we will describe these four types of reliable multicast protocols.

2.3.1 Sender-Initiated Protocols

Sender-initiated protocols are an extension of reliable unicast protocols [31]. The most traditional reliable unicast protocols (e.g. TCP) and many earlier reliable multicast protocols (e.g. Negative Acknowledgements with Periodic Polling protocol [27]) are based on this approach. In sender-initiated protocols, each packet sent by a sender is

acknowledged by each receiver to ensure reliability. The sender takes the responsibility for reliable data delivery by maintaining the state information of each receiver, i.e., to whom it should send data packets and from whom it should receive feedback, and processing the feedback from receivers. The sender uses an ACK list to keep track of receivers' feedback on each packet multicast. Upon reception of an ACK from a receiver, the sender updates the ACK list for the corresponding packet and the corresponding receiver. The sender uses transmission timers to detect packet loss and transmission error. The sender starts a timer at the time of a packet being transmitted. If the timer expires prior to the sender having received either ACKs or NACKs for the packet from receivers, then the sender considers the packet is either lost or that a transmission error has occurred. The sender retransmits the packets and restarts the timer.

Figure 2-5 depicts a simple example of sender-initiated protocols. The basic mechanism of sender-initiated protocol is summarized as follows:

- The sender multicasts a data packet to all receivers and starts a transmission timer.
- Each receiver uses unicast to transmit ACK to the sender to confirm the data packet received correctly or NACKs when they detect transmission error or packet loss.
- When receiving a NACK or the transmission timer expires, the sender retransmits (by multicasting) the corresponding packet to all receivers and resets the timer.
- Upon receiving an ACK from a receiver, the sender updates the ACK list for the corresponding packet and the corresponding receiver.

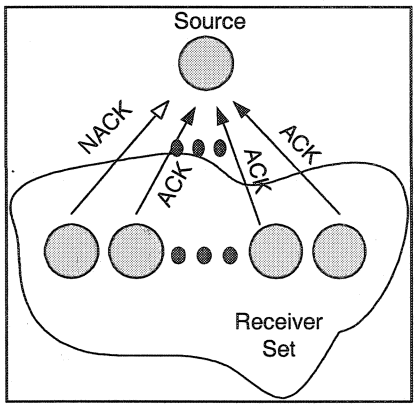


Figure 2-5 Sender-initiated protocols [28]

Sender-initiated protocols have the following drawbacks:

- Not scalable: A sender must process all ACKs or NACKs from receivers. In order to be capable of processing all ACKs and NACKs, the sender has to have sufficiently high processing capability. How much capability is sufficient depends on the number of the receivers the sender is to manage. The more the receivers, the higher processing capability the sender has to have. Also, the sender must know the receiver set [28]. The sender must continuously track the changing set of active receivers and the reception state of each. These processing overheads can overwhelm a sender with limited processing capability if the multicast group has large number of members.
- ACK-implosion problem: When there is a larger number of receivers returning ACKs to the sender over a short interval, and a significant overhead is incurred by the sender, an ACK-Implosion problem is said to have taken place [8]. If the sender is not capable of processing all ACKs quickly enough, it can lead to severe

bottlenecks at the sender and the performance of the network is significantly reduced.

Because sender-initiated type protocols are not scalable and have ACK-implosion problem, their implementation in mobile ad hoc networks would not be practical. We hence omitted such class from comparison with other types of reliable multicast protocols in this thesis.

2.3.2 Receiver-Initiated Protocols

To alleviate the poor scalability and ACK-implosion problems exhibited in sender-initiated protocols, receiver-initiated protocols reduce the processing burden for reliable delivery on the sender's part [8]. In receiver-initiated protocols, each receiver is responsible for detecting transmission error and packet loss. A receiver detects a packet loss when it finds a gap in sequence number of received packets. As seen in Figure 2-6, receivers inform the sender via NACKs whenever they request retransmission of a packet, and they do not send ACKs to confirm correct reception. Receivers use retransmission timers to detect either loss of NACKs or subsequent packet retransmission.

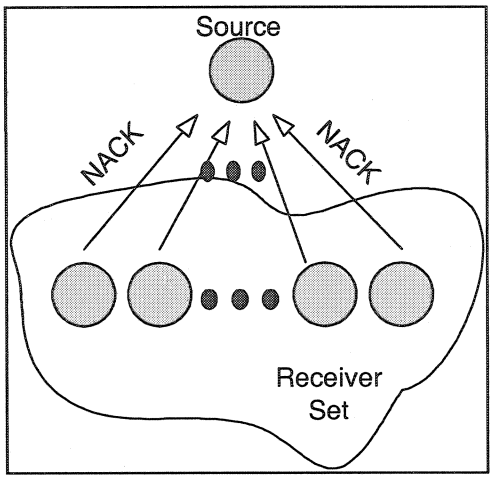


Figure 2-6 Receiver-initiated protocols [28]

There are three variations of receiver-initiated protocols [32].

(1) Unicast NACK

The basic mechanism of the receiver-initiated protocol with unicast NACK is summarized as follows:

- The sender multicasts data packets to all receivers
- Whenever a receiver detects a packet loss, it unicasts a NACK to the sender and starts a retransmission timer
- Only the sender is involved in issuing retransmission packets. Upon receiving a NACK, the sender retransmits (by multicasting) the corresponding packet to all receivers
- The expiration of a timer without receiving the corresponding retransmitted packet serves as a detection of loss of NACK or retransmitted packet

One problem caused by unicast NACK is that when many receivers detect transmission error or packet loss, the sender has to deal with flood of NACKs returned from the receivers within a very short period of time. In this case, the NACK-implosion problem occurs [28]. Similar to the abovementioned ACK-implosion problem, the NACK-implosion problem can seriously reduce the network performance, especially when transmission error occur frequently to many receivers. To alleviate NACK-implosion, receiver-initiated protocol with NACK suppression scheme is developed.

(2) Multicast NACK with NACK suppression scheme

The basic mechanism of the receiver-initiated protocol with NACK suppression scheme is summarized as follows:

- The sender multicasts data packets to all receivers
- NACK suppression scheme [31]: whenever a receiver detects a transmission error or packet loss, it first sets a NACK timer to wait for a random period of time then multicasts a NACK to the sender and other receivers when NACK timer expires. Upon hearing this first NACK message, other receivers that lost the same packet and intended to send the same NACK reset their timers (exponential backoff), and behave as if they had actually sent the NACK. This way, receivers suppress their NACKs and sending duplicate NACKs to the sender is avoided. It is claimed that, with the NACK suppression scheme engaged, ideally only one NACK arrives at the sender for any packet lost.

- Only the sender is involved in issuing retransmission packets. Upon receiving a NACK, the sender multicasts the corresponding packet to all receivers.
- The expiration of a NACK timer without receiving the corresponding retransmitted packet serves as a detection of a loss of NACK or retransmitted packet.

It was shown in [33] that the unicast NACK approach and multicast NACK with NACK-suppression approach have similar performances when dealing with a small number of receivers and the network has low packet loss rate. As the number of receivers and the packet loss rate increase, the multicast with NACK-suppression approach outperforms the unicast NACK. This is because in unicast NACK approach, the sender has to process possibly more than one NACK from each affected receiver, while in the multicast with NACK-suppression approach, the sender (ideally) only needs to deal with one NACK for each lost packet. Although the multicast with NACK-suppression approach creates more traffic and overhead at the receivers, it has the advantage of reduced processing at sender – thus, it has higher scalability.

(3) Multicast NACK with NACK / Repair suppression scheme

This approach is similar to (2), with the exception that any receiver having a copy of the requested data can issue a repair (retransmission) [8]. Upon receiving a NACK, receivers who have correctly received and cached the missing packet set their repair timers to wait for a random period of time. When one receiver's timer expires and a

repair has not been received from any other node, the receiver then multicasts the repair to the rest of group. Other receivers who have the same data packet and have scheduled to multicast the repair cancel their repair timers when they hear the retransmission. This way duplicated repairs are suppressed [8].

The multicast NACK with NACK / Repair suppression scheme approach can support a similar number of receivers as that of the multicast NACK with NACK suppression scheme approach. Fast error recovery is one advantage of multicast NACK with NACK / Repair suppression scheme approach. If a receiver is located far away from the sender, it does not have to wait for a long end-to-end delay for a retransmission from the distant sender [32]. Rather, it can quickly obtain a repair from a nearby node. On the other hand, the drawbacks of this approach are as follows. Firstly, receivers have to allocate a certain amount of memory for caching data. In ad hoc networks, this can greatly reduce the performance of mobile nodes with limited memory capabilities. Secondly, repair-suppression creates additional process burden for each node and consumes additional bandwidth of network.

In summary, receiver-initiated protocols reduce the processing burden at the sender, and free the sender from the responsibility of maintaining the state of each receiver. Since there are no ACK messages relayed, the ACK-implosion problem is completely avoided. Therefore, senders are able to deal with a larger number of receivers without reducing the network performance, i.e., receiver-initiated protocols have better scalability. It was

shown in [33] that receiver-initiated protocols have better throughput than sender-initiated protocols.

There are, however, several drawbacks that should be mentioned. First, a sender cannot ascertain when it can release the cache of a transmitted packet. This is because a sender receives only NACK messages and does not receive positive confirmation on reception of packets from all the receivers. A polling mechanism can be incorporated into receiver-initiated protocols, allowing a sender to periodically check each receiver for correct receptions. If confirmed, sender would release the packet from its buffer, making more efficient use of an already limited memory [31]. However, in addition to their implementation complexities, such polling mechanisms would increase the processing burden at the senders. Obviously, this makes readily applying received-initiated protocols to ad hoc networks an inefficient and impractical solution. The second drawback of receiver initiated protocols is that implementing NACK/Repair suppression consumes additional bandwidth and increases processing burdens to multicast NACKs and repairs. In certain cases, this can result in wasting valuable bandwidth on unnecessary multicast traffic over a large area, even if the packet loss and recovery are restricted to a small area.

2.3.3 Ring-Based Protocols

Ring-based reliable multicast protocols were originally developed to provide support for applications that require an atomic and total ordering of transmission to all receivers [28]. Total ordering means that all messages multicast to a group are guaranteed to be delivered to all members in the exact same order [30]. A reliable and total order

multicast is called an atomic multicast [34]. For example, if a node N1 multicasts a packet m1, and a node N2 multicasts a packet m2 later in time, then an atomic system would guaranty that all receivers receive m1 before m2.

Ring-based protocols are based on the token-ring technique. The Token Ring Protocol (TRP) [35] was the first protocol using the token-ring technique for reliable multicast. In addition to guaranteeing that all receivers can receive every message without error or loss, TRP also guarantees that every receiver places the received messages in the same sequence in that they are transmitted. TRP was originally used to build a distributed database on Ethernet [29]. In the early 1990's, this protocol was adapted to operate on a multicast network over the Internet and was later renamed Reliable Multicast Protocol (RMP) [10]. We will discuss the details of RMP in next chapter.

Figure 2-7 depicts an example of ring-based protocols. Receivers are organized as a logical token ring, and one receiver is selected as the token site. The token site is the only node that is responsible for sending ACKs to the source. Each ACK contains a sequence number, called a timestamp [10]. All receivers place packets in the order indicated by the sequence number. When a node detects a gap in the sequence number, it sends a NACK to the token site to request a selective packet retransmission. This way, all nodes have a global ordering of the packets for delivery to application layer [28]. Total ordering of packet transmission is a unique advantage that ring-based protocols have over other protocols.

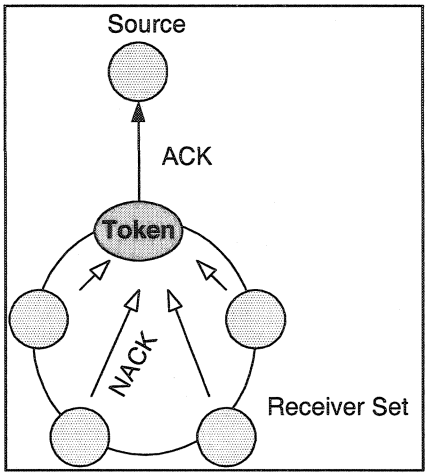


Figure 2-7 Ring-based protocols [28]

Receivers take turns to become the token site and acknowledge the packets sent from the source to ensure reliability. The ACK that is multicast to the source and other receivers also serves as a token passing mechanism [35]. A receiver is qualified to become a token site only if it has received all packets until the current one. When the next token site node receives the ACK, it will check if it has received all the packets until the current one. If true, the next token site will replace the current token site. Otherwise, the next token site node will request the missing packet from the current token site. In this manner, when a token site multicasts an ACK, it implicitly acknowledges that it has received all packets up to the current one. For example, when a token site multicasts an ACK for packet 3, it is implicitly acknowledging correct reception of packet 0, 1, 2 as well (assume sequence number of packets starts from 0).

The basic mechanism of ring-based protocols is summarized as follows [10]:

- The source multicasts packets to all receivers and starts a transmission timer.

- When the token site receives a packet correctly, it informs the source by multicasting an ACK to the source and other receivers.
- If the source does not receive an ACK for a transmitted packet from the token site before the timer expires, the source considers the packet is lost and retransmits it.
- Whenever a receiver along the token ring detects a packet loss or transmission error, the receiver unicasts a NACK to the token site to request packet retransmission.
- Upon receiving the NACK, the token site uses unicast to send the requested packet to the requestor.

The above summary shows that ring-based protocols are a combination of sender-initiated protocols and receiver-initiated protocols. Ring-based protocols operate as sender-initiated protocol between the source and the token site and as receiver-initiated protocols between the token site and receivers. This combines the throughput advantage of receiver-initiated protocols with the reliability of sender-initiated protocols [31].

Ring-based protocols have the following advantages: First of all, the source only needs to process the ACKs returned from the token site. When there is no transmission error, the sender only need to process one control message (i.e. ACK) per packet transmission, independent of the number of receivers. This significantly reduces the processing overhead at sender. Secondly, ring-based protocols guarantee consistent ordering of packet delivery. On the other hand, ring-based protocols have several drawbacks: Firstly, the process to setup and maintain a logical token ring is complicated. This translates to significant overhead when a ring is large and requires nodes to be equipped with high and

efficient processing powers. Therefore, it is not suitable for mobile ad hoc networks in that not every device has a high processing capability. Secondly, it has been shown that the throughput of ring-based protocol is just slightly better than that of sender-initiated protocols, but worse than that of receiver-initiated protocols and tree-based protocols [28]. It is also claimed that the low throughput is caused by unicast retransmissions in error recovery. Thirdly, ring-based protocols only guarantee that all receivers "eventually" receive a packet [36]. They do not guarantee end-to-end packet transmission delay, i.e., a packet might take long time to reach a receiver.

2.3.4 Tree-Based Protocols

One main characteristic of tree-based protocol is that it divides receivers into subgroups with each subgroup having a group leader, and distributes the responsibility of error recovery over group leaders [28].

Figure 2-8 depicts a simple example of tree-based protocols. Receivers and the source form a tree with the source as the root of the tree. Receivers are divided into groups where each group has a group leader and one or more children. A group leader may also be a child of another local group. Children without children are at the bottom of the tree and are called leaves.

As with other multicast protocols, the source multicasts every packet to all receivers. However, only the source's direct children, i.e. group leaders, can send feedback to the source indicating which packets are received either correctly or incorrectly. When a child of a subgroup detects a packet loss or transmission error, it sends a NACK to its group

leader, where the latter is in charge of retransmission within the local group. The ACKs or NACKs the children of a subgroup send to their group leader are referred to as local ACKs or local NACKS [28], and are meaningful only within their own group.

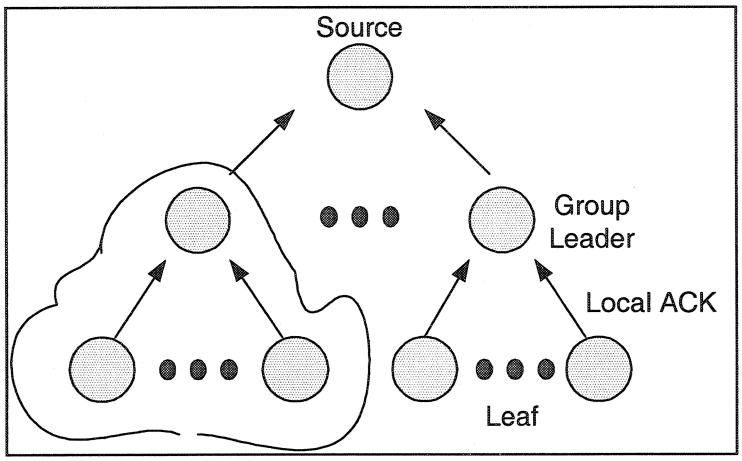


Figure 2-8 Tree-based protocols [28]

Tree-based protocols have three variations:

(1) ACK-based scheme

The ACK-based scheme is similar to sender-initiated protocols as receivers, including leaf nodes or the group leader other than the root, send ACKs to their group leaders to indicate correct reception of packets. If ACK timer expirations occur at group leaders or the root, multicast retransmissions are invoked. Reliable Multicast Transport Protocol (RMTP) is an example of this approach [37]. The basic mechanisms of RMTP are summarized as follows [37]:

- The sender multicasts data packets to all receivers.
- Each direct child of the source unicast its own ACKs/NACKs to the source at periodic intervals. Based on the feedback, the source determines which packets

have been received correctly and which need to be retransmitted. Notice that the source usually uses unicast to retransmit the request packets. If the number of NACKs for a packet exceeds certain threshold, the sender multicast the packet to the entire tree.

- Other receivers that are not the direct children of the source send their ACKs or NACKs to their group leaders at regular intervals. Group leaders usually use unicast to retransmit packets. If the number of NACKs for a packet exceeds certain threshold, the group leader locally multicasts the packet.

In this scheme, a group leader does not consolidate ACKs of the receivers in its group, but uses these messages to perform local retransmissions to the receivers. Two reasons make RMTP impractical to be implemented in mobile ad hoc networks. Firstly, RMTP assumes that a multicast tree, rooted at the source and spanning over all receivers, is set up at network layer. The receivers are then grouped based on this tree. However in ad hoc networks, it requires intensive processing from mobile nodes to maintain a tree network structure since the topology in ad hoc network is frequently changing. Secondly, RMTP assumes that there is information about the approximate location of receivers and based on that information, some receivers are chosen as group leader candidates, i.e. group leader candidates are pre-assigned. In mobile ad hoc networks, it creates too many processing burdens for each node to select group leader candidates based on nodes approximate locations since mobile nodes can move arbitrarily.

(2) NACK-based with NACK suppression scheme

The NACK-based with NACK suppression scheme approach is similar to the receiver-initiated protocols with NACK suppression scheme. Tree-based Multicast Transport Protocol (TMTP) [9] is an example of this scheme. To limit the scope of the NACK multicasts and packet retransmissions, TMTP uses the Time-To-Live (TTL) field to restrict packet transmission radius. When TTL value is small, error recovery is completely local. By increasing the TTL value, besides a child's parent, other group leaders or receivers near by who can hear a NACK can answer the retransmission request. In another words, each group leader not only provides error recovery to receivers in its local region but also for other receivers in its vicinity [9]. This ensures fast error recovery. Another advantage of TMTP is that a tree is self-organized and does not rely on any centralized coordinator [9]. The tree is built completely at the transport layer and, thus, does not require any explicit support from or modification to IP multicast routing protocols. The basic mechanisms of TMTP are summarized as follows [9]:

- The sender multicasts data packets to all receivers.
- When a child detects transmission error or packet loss, the child multicasts a limited scope NACK. NACK suppression scheme is also applied.
- When a node receives the NACK and has the requested packet, it multicasts the requested packet to the local domain.

(3) Aggregate ACK-based scheme

In this scheme, a group leader sends an aggregate ACK to its parent after all of its children have acknowledged correct reception. Aggregate ACKs start from the leaves of a tree, and propagate toward the source. The use of aggregate ACKs is to let group leaders decide when to safely delete packets from memory [28]. After a group leader or the source has received an aggregate ACK for a packet, it can safely remove the corresponding packet from its buffer since all members in this sub-hierarchy have already received it correctly. Lorax [38] is an example of this scheme.

Using aggregate ACKs can ensure the protocol operates correctly even if group leaders fail [28]. For example, without using aggregate ACKs, a group leader B fails after it has acknowledged correct reception of a packet to its group leader, the source A. If a member of B's local group needs a retransmission, neither A nor B can retransmit the data packet since B is not functioning and A has removed the packet from its memory after receiving ACKs from all its immediate children. With aggregate ACKs, the data in the memory are removed only when all hierarchical members under this group leader have received it correctly. Even if B fails, without receiving aggregate ACKs for the packet, B's leader A does not release this packet from its memory. On the other hand, comparing with local ACKs and local NACKs, aggregate ACK has a relatively lower throughput. In some cases, if the source is to schedule retransmission based on aggregate ACKs, it would have to pace based on the slowest path in the tree, whereas with local ACKs or NACK, retransmission can be scheduled within a local group, which is within relatively smaller range.

The basic mechanism of aggregate ACK-based tree scheme is summarized as follows [39]:

- The source multicasts packets to all receivers.
- Upon correctly receiving a packet, leaf nodes send aggregate ACKs to their group leaders.
- Upon receiving a data packet correctly, group leaders send an ACK to their parents.
- Group leaders wait for a certain period of time to receive ACKs from their children. If timeouts occur, packets are retransmitted.
- Group leaders wait to receive aggregate ACKs from their children. Upon reception of all aggregate ACKs, the group leaders release the corresponding packet from the memory and send an aggregate ACK to its parent.

In tree-based protocols, the source interacts only with its immediate children and does not deal with all its hierarchical children directly. Hence, the source does not have to maintain the state information of all children, and is not responsible for error recovery for all of them. The relative processing overheads are distributed among the entire tree instead of only over the source, which makes tree-based protocols very scalable. In [28], it is shown that a tree-based reliable multicast protocol is the most scalable way of achieving reliability. Another advantage of tree-based protocols is the parallel processing in sub-trees. Packet retransmission can be carried out in different sub-trees simultaneously. This significantly reduces network congestion and improves network throughputs. For example, in a wide area network, if we organize the tree by geographic

domain (e.g., the receivers in Ontario and Alberta are in different sub-trees), the error recovery in a region can proceed independently without causing additional traffic in other regions [9]. The third advantage is that end-to-end delay is significantly reduced since lost packets are recovered by local retransmissions as opposed to retransmissions from the original sender, i.e., the source [9, 37].

2.4 Reliable Multicast for Ad Hoc Networks

Reliable multicast is vital to the success of mission critical applications in ad hoc networks. Some reliable multicast protocols have been proposed for ad hoc networks in recent years, such as Anonymous Gossip (AG) and Reliable Adaptive Lightweight Multicast (RALM) protocols.

AG protocol [40] is a reliable multicast protocol that does not require a group member to have any knowledge of the other group members. AG proceeds in two phases. In the first phase, packets are multicast to the group using any unreliable multicast protocol. In the second phase, periodical anonymous gossip takes place in the background as each group member recovers any lost data packet from other members of the group that might have received it. The second phase consists of periodical rounds of the following steps:

- Node A randomly chooses another member of the group, say B.
- A sends B the information about messages it has received or not received.
- B checks to see if it has received any of the messages listed by A.
- Then A and B could exchange messages which are not part of each other's message history.

RALM protocol [41] is a reliable, congestion-controlled protocol designed for the ad hoc network. The basic mechanism of RALM is as follows:

- Each multicast source maintains a Receiver List. The source selects a feedback receiver from the Receiver List and multicasts data packets.
- The feedback receiver replies the source with either ACK to indicate successful reception or NACK to request retransmissions. ACK or NACK are sent by unicast. Lost packets are requested one at a time until the feedback receiver has all up-to-date packets. Retransmitting one packet at a time slows down the transmission of the source when congestion is detected.
- Upon reception of a NACK, the source retransmits packet by multicast. If the sender of NACK is not in the Receiver List, that node is added to the list. Once the feedback receiver obtains all the packets, it unicasts an ACK to the source.
- Upon reception of an ACK, the source removes the feedback receiver from the Receiver List, chooses a new feedback receiver in a round robin fashion from the Receiver List, and repeats this process until the list is empty.
- When the Receiver List is empty, the source reverts to the application sending rate.

The main contribution of RALM is the introduction of TCP-like window-based congestion control mechanism used to reduce overhead and increase efficiency. When losses are detected, the congestion window is halved; otherwise, it increases linearly.

2.5 Summary

This chapter describes the related work on reliable multicast and wireless ad hoc networks. First, we discussed the characteristics of mobile ad hoc networks, and presented the unicast and multicast routing protocols that we chose to use in our simulation. Second, we described four known classes of reliable multicast protocols for wired networks and discussed their advantages and limitations. Sender-initiated protocols are simple but suffer from the ACK implosion problem. Receiver-initiated protocols with NACK suppression scheme overcome the problem of ACK/NACK implosion. However, they require the sender to have large memory buffers. Ring-based protocols provide total ordering reliable multicast service. Nevertheless, they are complicated to be implemented and packets end-to-end delay can be long. Tree-based protocols are most scalable compared to other protocols. Finally, we discussed reliable multicast protocols proposed for ad hoc networks.

3 REPRESENTATIVE RELIABLE MULTICAST PROTOCOLS

In this chapter we describe the three representative reliable multicast protocols that we select to evaluate their performance in mobile ad hoc networks. The three protocols are SRM, TMTP, and RMP. They respectively represent receiver-initiated type of protocols with NACK / Repair suppression scheme, tree-based type protocols, and ring-based type protocols. We implement simulation models for these protocols according to their basic mechanisms with certain limitations, which are elaborated on after analyzing the results of early stage debugging simulation. We modify certain portions of the algorithms in order to improve performance and eliminate certain features to cope with lack of support from the underlying routing protocols.

3.1 Scalable Reliable Multicast - SRM

SRM [8] is a receiver-initiated type protocol with NACK / Repair suppression. SRM is based on Application Level Framework (ALF) protocol model [42] that has been proposed to help adapt transport-level services, such as reliability, to the needs of specific

applications. To meet diverse application requirements, ALF-based protocols normally leave as much functionality and flexibility as possible to the application. Therefore, SRM is designed to meet only the minimal definition of reliable multicast, i.e. eventual delivery of data to all the group members, without enforcing any particular delivery order. Shifting the difficulty of coping with unordered packets to the applications also relieves the protocol from the overhead of reordering packets. The design of SRM design aims at quick packet loss recovery.

3.1.1 Data Transmission

A sender divides the message to be transmitted into fixed-size packets, and assigns each data packet a sequence number, starting from 0. The sender multicasts packets at a regular interval to an IP multicast address, with an associated port number, representing a multicast group. SRM does not enforce any particular delivery order. The packet sequence number is used only for the purpose of detecting packet loss. SRM shifts the packet reordering responsibility to the upper layer applications for the purpose of minimizing constraints and maximizing flexibility of implementation.

3.1.2 Packet Loss Recovery

Packet loss is detected when a receiver finds a gap in packet sequence numbers. Receivers only send NACKs when detecting packet loss and do not return ACKs to confirm on correct receptions. To perform loss recovery, SRM relies on NACK with NACK / Repair suppression scheme, of which the basic mechanism has been discussed in Section 2.2.2. For illustration, we give an example of SRM's loss recovery mechanism. In particular, we show how SRM configures its NACK timers and Repair timers, i.e., the

manner in which SRM makes a decision on how long to delay a request packet or repair packet before eventually sending it.

Assume receiver A is one participant of a SRM based multicast group. When A detects a packet loss by finding a gap in packet sequence numbers, it sets a NACK timer, called a repair request timer in SRM. When the repair request timer expires and A has not heard the same repair request (i.e., NACK) from any other receiver, it then multicasts the repair request to the whole group, and doubles the request timer to wait for the repair packet. The repair request timer is a function of the estimated delay from A to the source of the packet. The request timer value is randomly selected from the uniform distribution on

$$[C_1 \cdot d_{s,A}, (C_1 + C_2) \cdot d_{s,A}],$$

where C_1 and C_2 are repair request timer parameters (C_1 and C_2 are 2 in our simulation). $d_{s,A}$ is the estimate of one-way delay between A and the source of packet.

If A receives the same repair request before its own repair request timer expires, then A does a random exponential backoff, and resets its repair request timer. This is called NACK or request suppression. In another word, if a receiver sees a repair request with the same information as its scheduled request, it then reschedules the request with a new delay. On average, the new delay is double the current delay. If the current request timer value is chosen from interval

$$2^i \cdot [C_1 \cdot d_{s,A}, (C_1 + C_2) \cdot d_{s,A}],$$

where i indicates the number of backoffs, starting with 0, then the backoff timer value is randomly selected from interval

$$2^{i+1} \cdot [C_1 \cdot d_{s,A}, (C_1 + C_2) \cdot d_{s,A}].$$

For every backoff, the parameter i is increased. This means that the node will be waiting longer periods of time before sending a new request. In this thesis, the upper limit for the i parameter is denoted as requestBackoffLimit_. In our simulation, the value of requestBackoffLimit_ is 5.

Any receiver that has a copy of the requested data, i.e. a responder, can issue retransmitted packets, called repairs in SRM. When responder B receives a NACK from A, B sets a repair timer. The repair timer is set to a value randomly selected from the interval

$$[D_1 \cdot d_{A,B}, (D_1 + D_2) \cdot d_{A,B}],$$

where D_1 and D_2 are repair timer parameters (D_1 and D_2 are 1 in our simulation). The symbol $d_{A,B}$ denotes the estimated one-way delay between responder B and A. When B's repair timer expires and B has not heard the same repair from any other receiver, B multicasts the repair to the whole group. If B hears the repair from a receiver before its own repair timer expires, it avoids sending the same repair by canceling its own repair timer (Repair suppression).

There are cases in which multiple receivers may detect the same packet loss, or multiple receivers may attempt to respond to the same repair request. This can cause duplicate requests and repairs to be sent and may very well reduce the overall network throughput. SRM Request / Repair timer algorithm effectively addresses this issue by randomly

selecting timer values to de-synchronize node action. The number of duplicate requests or repairs is thus maintained at a low value. Moreover, the algorithm lets a node close to the request node to timeout first and multicast the repair. This latter feature ensures fast loss recovery.

3.1.3 Session Message

In SRM, lost packets are detected when gaps in the sequence number space occur. However, if the last packet or the last train of packets are lost, receivers can never detect the packet loss, because no more packets with higher sequence number are received. To solve this problem, SRM has all members in a group periodically multicast low-rate session messages. Each session message contains the sender's own highest sequence number sent so far and the highest sequence number received from every receiver. Receivers can use session messages to find sequence number gap if they miss the last one or more packets.

Session messages are also used to estimate one-way delays between nodes, which are needed in setting request and repair timers. All packets, including session packets, carry a source-ID and a timestamp. The session packet timestamps are used to estimate the node-to-node delays.

3.1.4 Implementation Decisions

We strictly followed the specification of SRM and did not make any major modifications in our implementation.

3.2 Tree-based Multicast Transport Protocol - TMTP

TMTP [9] is a tree-based receiver-initiated protocol with NACK suppression scheme. It provides reliable multicast communication for one-to-many bulk data dissemination applications. The following are the essential features of TMTP.

3.2.1 Control Tree Management

In TMTP, nodes use the expanding ring search method to dynamically self-organize into a hierarchical control tree as nodes join in and leave a group. Control tree management does not rely on any centralized coordinator. A control tree is built solely at the transport layer. It does not require any assistance from, or modification to, the underlying IP multicast infrastructure. It is easier for model implementation and offers flexible controls than RMTP in that tree is structured at the IP layer. This is the main reason why we chose the TMTP type model in our simulation.

TMTP organizes receivers into hierarchy of domains where a domain manager (DM) is the representative of each domain and is responsible for handling retransmissions within its domain.

Figure 3-1 shows a TMTP control tree with the source as the root of the entire tree. In TMTP, each domain manager can have a maximum of K children. That is, the source can have maximum K (K = 3 in our simulation) immediate children. Each of these children themselves can have maximum K receivers as their immediate children.

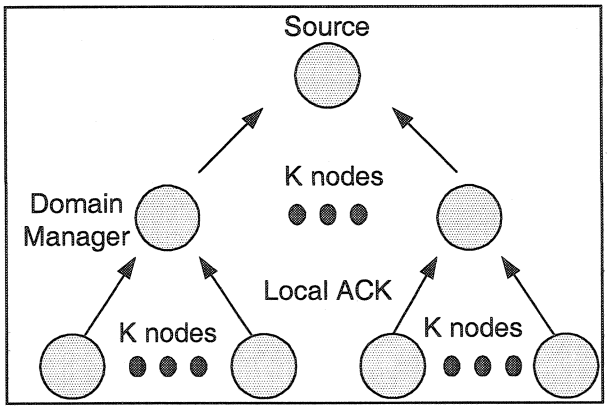


Figure 3-1 TMTP control tree [9]

Over the lifetime of a group, the control tree grows or shrinks dynamically in response to additions to and deletions from group membership. There are two operations associated with control tree management: JoinTree and LeaveTree. When a new node attempts to join in a group, it executes JoinTree operation to become a member of a control tree. When a node leaves a group, it executes LeaveTree operation.

Joining in a control tree starts with using the expanding ring search method to locate the nearest domain manager. Figure 3-2 presents a process of a new node joining a control tree. A new node begins the expanding ring search by multicasting a SEARCH_FOR_PARENT request packet with a small time-to-live (TTL) value in its IP packet header, and starts a search request timer. The small TTL value restricts the scope of searching for nearby nodes by limiting the transmission radius of the search request packet. Using a small TTL value to restrict the scope of a multicast message is called limited scope multicast. If the new node does not receive a response within the timeout period of its search request timer, it re-multicasts a SEARCH_FOR_PARENT request packet with a larger TTL value. Any existing node in the control tree that receives the

SEARCH_FOR_PARENT request packet can respond with a WILLING_TO_BE_PARENT packet if it does not already have the maximum K children. The new node selects the node that responds first as its domain manager and sends it a JOIN_REQUEST packet. Upon reception of JOIN_REQUEST, the domain manager can accept the request and return ACCEPT_JOIN packet to the new node for confirmation of acceptance. With this confirmation, the new node joins the control tree. Please see Appendix B for the details of JoinTree and LeaveTree algorithms.

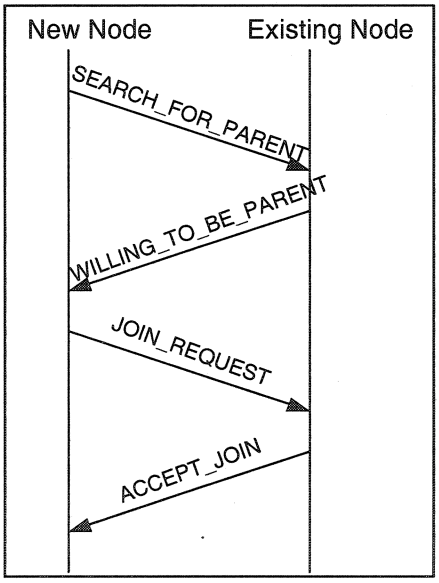


Figure 3-2 Joining in a tree

3.2.2 Data Transmission

TMTP exploits the efficient delivery mechanism of IP multicast for packet routing and delivery. When a sender wishes to send data, it multicasts data packets to an IP multicast

address (with port number) representing an entire group at a regular interval defined in configuration. The data packets arrive at all group members via standard IP multicast.

3.2.3 Packet Loss Recovery

The control tree takes the error control responsibility normally placed at the sender by distributing it across multiple nodes. This distribution of error control also allows error recovery to proceed independently and concurrently in different portions of the network. Each node in control tree is not only responsible for handling the errors that arise in its immediate children, but also provide error recovery for other nodes in its vicinity. TMTP uses restricted NACKs with NACK suppression scheme to respond to packet losses. When a child detects a missing packet, it multicasts a NACK to its parent and siblings with a small TTL value in IP packet header. Like the process of joining a tree, this small TTL value is used to restrict the scope of multicast. As in SRM, in order to avoid multiple receivers multicasting a NACK for the same packet, each receiver delays a random amount of time before transmitting its NACK. If the receiver hears a NACK from another member during the delay period, it suppresses its own NACK (NACK suppression). When a member receivers a NACK and has the corresponding data packet, it immediately responds by multicasting the missing packet to the local domain using a limited scope multicast message.

3.2.4 Implementation Decisions

In our simulation the underlying IP multicast routing protocol does not support limited scope multicast using small TTL value. Therefore, when a node joins a group and searches a parent, it has to use global multicast, i.e. multicast to all group members, to

locate a parent. For the same reason, NACKs and repairs are unicast between a member and its domain manager in our implementation. The modified loss recovery mechanism of TMTP is as follows:

- Whenever a child detects a packet loss, it unicasts a NACK to its parent and starts a request timer.
- On reception of the NACK, the parent unicasts the corresponding packet to the child.
- The expiration of the request timer without reception of the corresponding packet serves as a detection of loss of NACK or retransmitted packet. The child resends NACK to its parent and starts a new request timer. The value of the new request timer is double the previous request timer, i.e. exponential backoff.
- The child keeps on requesting until it receives the repair correctly or has repeated the request up to certain times, i.e. reached a predefined request threshold. If the request threshold is reached, the child gives up its current parent. It then uses multicast NACK with NACK Suppression scheme to request retransmission from the entire group and restarts the join procedure to find a new parent.

In the modified loss recovery mechanism, correlated losses are repaired individually, generating more traffic than a multicast repair if many children have lost the same packet. This may also cause NACK implosion if all children send NACK at the same time. We alleviate this problem by limiting the number of children per parent to 3.

TMTP cannot handle domain manager failure or node disconnection. Although we assume no node failure in our implementation, node disconnection must be sustained, e.g.

when a child cannot reach its parent since nodes can move arbitrarily in ad hoc networks. To solve this problem, we provide a new function that allows a child to find a new parent after it has failed to get a repair from its parent for a certain number times (5 times in our simulation).

3.3 Reliable Multicast Protocol - RMP

RMP [10] is a ring-based protocol. RMP has two distinguished characteristics compared to SRM and TMTP. These characteristics are:

- RMP provides a totally ordered reliable multicast service, i.e. packets from either one or more sources arrive at receivers in exactly the same sequence they are issued in.
- If no packets are lost, the only control messages a sender needs to manage is one ACK per packet sent, regardless of the number of receivers [35, 43].

Prior to discussing the operation of RMP, we need to clarify the terminology that will be used below:

- Ring: A ring is a logical structure that consists of members belonging to a group. It corresponds to a circular linked list in data structure.
- Token: A token is a flag that can be passed around among the members along the ring. Once a member accepts the token, it becomes the current token site.
- Current Token Site: A Current Token Site is the current token holder in a given ring. It is responsible for acknowledging packets on behalf of the ring and servicing retransmission requests from other members of the ring.
- Token List: A Token List is a list of operational members along a ring. Each group member maintains a token list.

RMP operation relies on three algorithms. The basic delivery algorithm operates on multicast messages during normal operation. It handles the delivery of packets to a group of members. The membership change algorithm creates a new token list and updates it at each member of the ring whenever a node joins in or leaves a ring. The reformation algorithm re-organizes a multicast group when the current token site fails to pass a token to its next token site. In what follows, we discuss these three algorithms.

3.3.1 Basic Delivery Algorithm

In RMP, each packet that is multicast from source S to its group contains a label (S, P_s) that uniquely identifies the packet as the P_s^{th} message from source S. When the current token site of the ring receives a packet correctly, it multicasts an ACK to the entire group. The ACK contains the same (S, P_s) pair that the corresponding packet has, along with a global sequence number assigned by the current token site. The global sequence number in an ACK is also called a timestamp that serializes the packets sent from all sources. Since we consider there is only one source in a group, in our simulation we did not implement the serialization part, i.e. the part that re-orders packets from different sources to provide the complete ordering service.

Source S transmits packets at regular interval. S starts a transmission timer for each packet it transmits. If S does not receive an ACK for a packet after the transmission timer expires, S considers the packet is lost, then retransmits it and resets the timer in exponential backoff of the last timer value, i.e. exponentially increase the period of the

timer. S keeps on retransmitting until an ACK is received or the number of times of retransmission reaches a threshold (We set threshold = 5 in our study).

Each receiver takes turn to become a current token site and acknowledge reception of packets as a token being passing around. At any given time, there is only one current token site in a ring. Token passing is one of the consequences when an ACK is multicast. The current token site passes a token to the next token site by using an ACK. Each ACK has a field to name the next token site. When this site receives the ACK, it checks to see if it has received all packets up to the current one corresponding to the ACK. If not, it requests the missing packets from the node that it currently believes is the current token site. Once it has all packets, it declares itself as the current token site. It announces this to other nodes either by sending an ACK for the next packet received or by unicasting a confirmation message to the previous token site if no packet is received within a predefined period of time. Since a receiver can not become a current token site until it acquires all packets that current token site has, when the token site sends an explicit acknowledgement for a packet it implicitly acknowledges that it has received all packets up to the current packet [31].

In summary, the ACK performs a number of functions:

- It lets the sender know that the token site has received the packet. In this way, it functions as a traditional positive acknowledgement.
- The timestamps in the ACKs provide a total ordering on messages.
- It transfers the token to next token site.

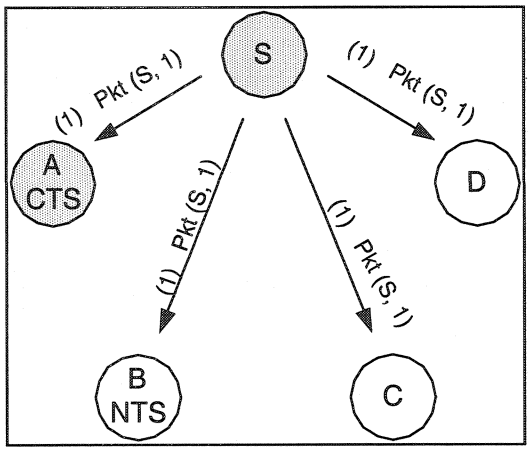
When the current token site sends an ACK to name the next token site, it sets an ACK timer. If the current token site has not heard from the next token site a confirming announcement prior to the timer's expiration, it resends the ACK. This process is repeated until the current token site receives an ACK or a separate confirmation message from the next token site. If the current token site has repeated sending the same ACK for more than a predefined number of times, it decides that its successor is unreachable and accordingly initiates the reformation algorithm to reconstruct a token ring.

The current token site is responsible for servicing retransmission requests until the next token site takes over the retransmission responsibility. This guarantees that there is always at least one site that has all packets and can respond to retransmission requests. The error recovery of RMP is summarized as follows:

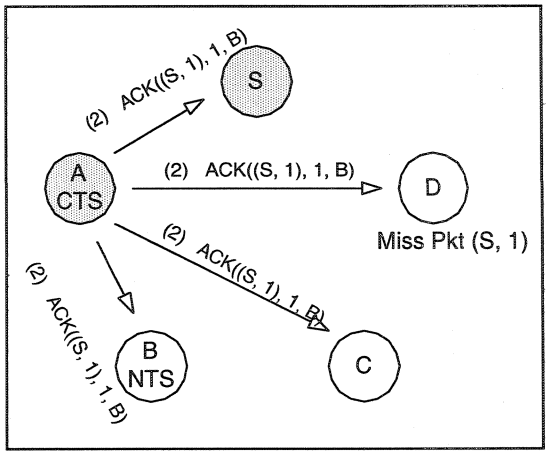
- Whenever a receiver detects a packet loss or transmission error, it unicasts a NACK to the node it currently believes to be the current token site and starts a request timer.
- Upon receiving the NACK, the current token site unicasts the missing packets to the receiver that requests it.
- The expiration of the request timer without receiving the corresponding packet serves as a detection of loss of NACK or retransmitted packet. The receiver resends a NACK to the current token site and resets the request timer as exponentially backoff of last timer value.
- The receiver repeats requesting until it receives the correct repair or the times of retry reaches a preset threshold. When threshold is reached, the receiver uses NACK multicast with NACK Suppression scheme to request retransmission from entire group.

Figure 3-3 depicts a RMP operation in absence of node failures. Frame 1 and 2 in Figure 3-3 shows the Source S sending packet 1 and receiving its ACK. Source S multicasts a packet with a label of (source, sequence number) pair, Pkt (S, 1) to its group. Node A, the current token site (CTS) multicasts ACK ((S, 1), 1, B) to the rest of group. Node D misses the packet 1 due to transmission error. The ACK for packet 1 is time-stamped as the global sequence number 1. The value in last field of the ACK, B, indicates that B is the next token site (NTS). When B receives this ACK, it checks that if it has all packets up to the one corresponding to the ACK. If so, it declares itself as NTS. Node A does not hand over the CTS responsibility to B until it receives either a confirmation message or an ACK from B. If A does not receive anything from B within a specified timeout period, A re-sends ACK ((S, 1), 1, B) to B by multicast. Node A repeats this process until it receives an appropriate response from B. If no such response is received, A considers B unreachable and initiates the reformation algorithm to reconstruct the ring.

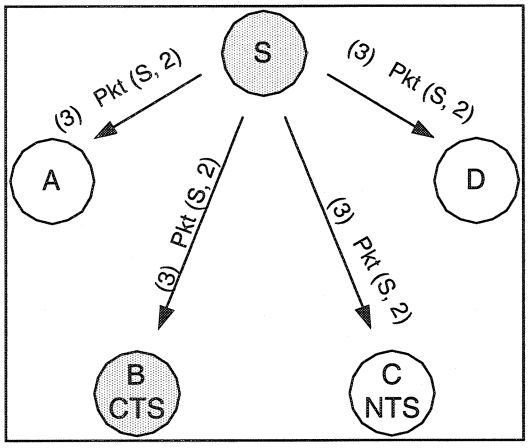
Frame 3 in Figure 3-3 is very similar to Frame 1. It shows S multicasts Pkt (S, 2). Now the CTS, Node B multicasts the corresponding ACK, ACK ((S, 2), 2, C), to the whole group in Frame 4. Node A knows that it has successfully passed the token to B upon receiving ACK ((S, 2), 2, C) from B. Similarly, B is now waiting for C to accept the token. Other nodes also know that B is now CTS when they receive ACK ((S, 2), 2, C) from B. In addition Node D, which missed Pkt (S, 1) detects the packet loss because it has received packet 2, but not packet 1. All other nodes have received both Pkt (S, 1) and Pkt (S, 2).



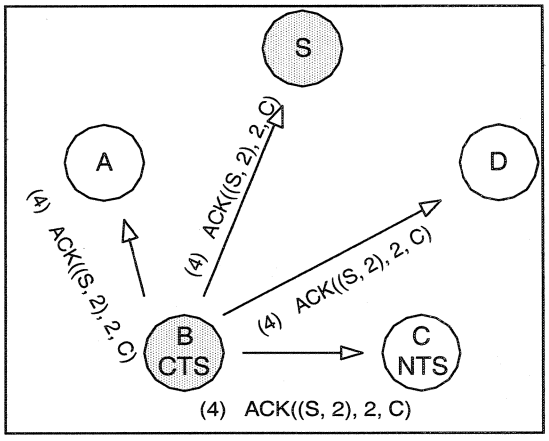
(1) Source sends packet 1



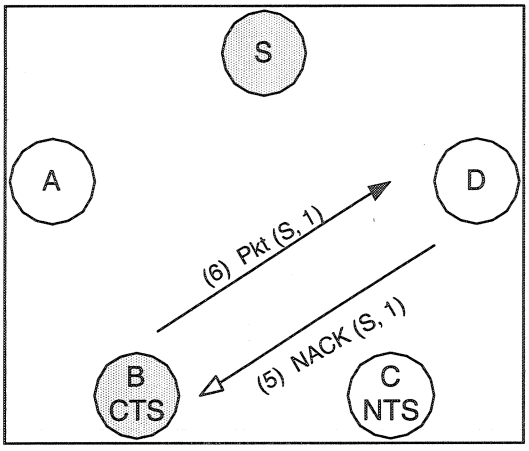
(2) Current token site sends ACK 1



(3) Source sends packet 2



(4) Current token site sends ACK 2



(5) D sends NACK to current token site

Figure 3-3 An example of RMP operation

Frame 5 in Figure 3-3 shows Node D using unicast to send NACK for Pkt (S, 1) to the node that it believes as CTS, which is B. Node B then responds to the request with a copy of Pkt (S, 1). This retransmission request is guarded by a timer as well. If the retransmission fails, D repeats the request.

3.3.2 Membership Change Algorithm

Group membership changes over time. Each member has a token list that consists of all operational members along a ring. The token list is updated whenever a node joins in or leaves a ring.

RMP allows nodes to join in a ring arbitrarily, one at a time. If a node is the only member in a group it creates a new ring that consists of itself only. If not, the node sends a Join Request message to an existing ring and informs the ring members that it wishes to join in. When the current token site hears the Join Request message, it adds the node to its token list, and multicasts an AcceptJoin message to the entire ring. This dialogue is illustrated in Figure 3-4. Other receivers then know that the new node has been accepted and update their token lists. RMP places the newly added member immediately after the current token site in token list. This forces the new node to take an immediate role in the ring. In Figure 3-5, node F wants to join a ring in which node A is the current token site. Node F is inserted right after node A.

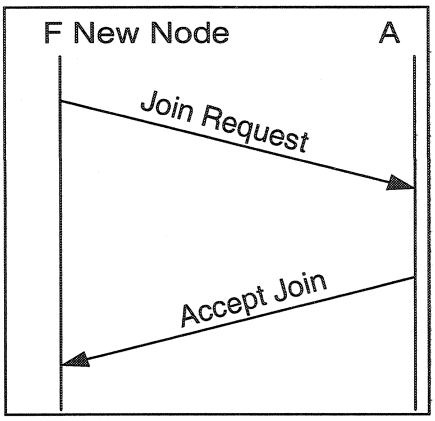


Figure 3-4 Join request dialogue in RMP

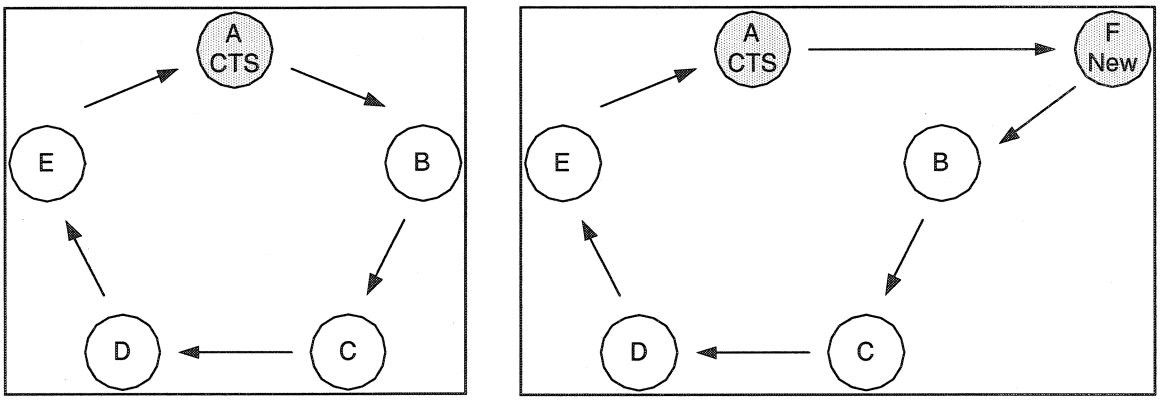


Figure 3-5 An example of joining in a ring

On the other hand, when a member wants to leave a ring, it multicasts a Leave Request message to the ring. A member can be removed only when it is about to receive a token, which guarantees that the resiliency requirements are not violated. The member that requests to leave stops processing the token ring when it receives a message from the current token site confirming its removal from the list. As shown in Figure 3-6, suppose node A is the current token site. When node A accepts the token, it checks if its

neighboring node, B, wants to leave the ring. If this is the case, node A removes node B from token list and multicasts a message to let other nodes know that node B is leaving. Node A's next node now becomes node C. The detailed algorithms of joining and leaving a ring are given in Appendix C.

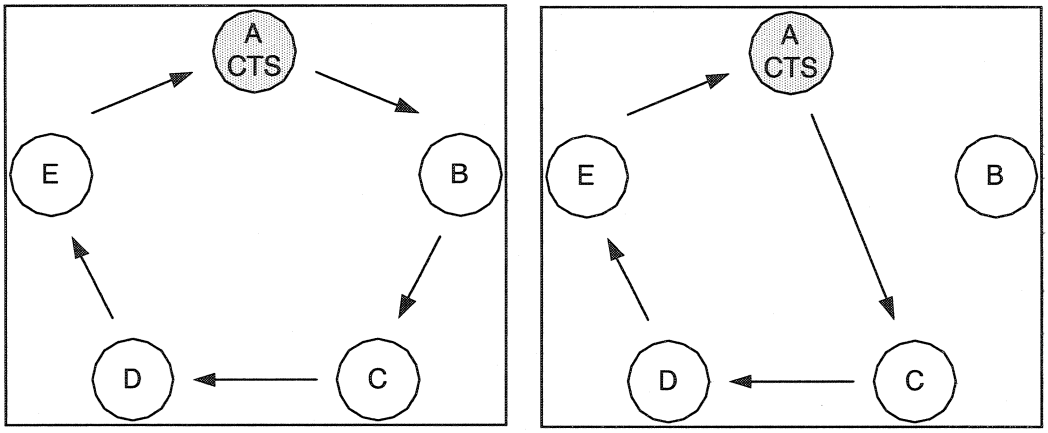


Figure 3-6 An example of leaving a ring in RMP

3.3.3 Reformation Algorithm

According to the specification of RMP protocol, multicasting a data packet, passing the token to the next token site, and requesting retransmission of a packet require positive acknowledgements. The sender sets a timer each time one of these operations is started, and retries the given operation if the timer expires. It keeps on doing this until it receives the correct response. If an operation is tried more than a set limit of tries, it decides that a failure has occurred and a reformation algorithm is run to reconstruct the list of nodes in the token ring.

In the reformation algorithm, the node that first detects the failure uses multicast packets to repeatedly poll all of the nodes. Every node then responds if it is still active and

reachable. From these responses, the initiator constructs a new token list and multicasts this list to the whole ring. The initiator of a reformation algorithm will be the new token site and responsible for acknowledging packets during the reformation process. The detailed reformation algorithm is in Appendix C.

3.3.4 Implementation Decisions

As mentioned in Section 3.3.1, since we consider there is only one source in a group, we did not implement the serialization part, i.e. the part that re-order packets from different sources to provide total order service.

As previously mentioned in Section 3.3.3, if a node repeatedly fails to receive the proper response to one of the actions that requires a positive acknowledgement it runs the reformation algorithm to reconstruct the token ring. Initial simulation results showed that even under light traffic load some nodes do not always respond in time. Therefore, the token ring was reconstructed very frequently, resulting in massive traffic in the network and network congestion. The performance of RMP hence degraded sharply. To solve this problem, we changed the protocol such that the reformation algorithm is invoked only when the current token site considers that its successor is unreachable. In this case, the current token site tries to recover from the failure by reforming the ring. The current token site that detects the failure uses multicast to repeatedly poll all of the nodes. Each node will then respond if it is still active and reachable. From these responses, the token site constructs a new token list and multicasts this list to the whole ring.

3.4 Summary

This chapter describes in detail the operations of the three representative reliable multicast protocols SRM, TMTP, and RMP, and how we implement each of them. Next chapter will show the simulation model and results of these three protocols in ad hoc networks.

4 PERFORMANCE ANALYSIS

We evaluate the performance of SRM, TMTP and RMP using simulation. The software simulator chosen is Berkeley's Network Simulator version 2 (NS-2) [7]. We build simulation models for these three protocols in the NS-2 environment. Based on their characteristics that were discussed in previous chapters, we develop multiple scenarios and run simulations in order to observe the behavior of these protocols in reacting to various testing stimuli in mobile ad hoc networks, in addition to comparing their performances.

4.1 NS-2

NS-2 is an object-oriented, discrete event-driven network simulator developed by the University of California at Berkeley. It provides support for simulating TCP, routing, and other protocols over wired and wireless networks. It is written in C++, with an OTcl interpreter as a front end [44]. OTcl, short for Object Tcl, is a Tcl script language with object-oriented extensions developed at MIT [45]. NS-2 separates detailed protocol implementation from simulation configuration. For fast packet and event processing, the

detailed protocol implementation requires a systems programming language (NS-2 uses C++). On the other hand, a simulation usually involves varying parameters or configurations, which requires efficient and easy configuration management. OTcl is an ideal choice for simulation configuration since it accommodates frequent changes [44].

To setup and run a network simulation, a user writes an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the functions in the library, and decides when traffic sources are to start and stop transmitting packets through the event scheduler.

We build simulation models of SRM, TMTP and RMP in the NS-2 simulator environment. Based on their basic mechanism and characteristics, we design a number of simulation scenarios and run simulations against the models of SRM, TMTP and RMP. We observe the behaviors of these three protocols in reacting to various stimuli of mobile ad hoc networks, and compare their performances. In all simulations 90% confidence levels are maintained with 10% confidence intervals (see Appendix D).

4.2 Simulation Model

This section describes the simulation model developed to evaluate the performance of three wired reliable multicast protocols over ad hoc networks.

4.2.1 Simulation Assumptions

The major assumptions made in our simulation are as follows:

- One sender per group – For simplicity, we only simulate one multicast group that consists of one source.
- All nodes are active – We assume each node has enough power to be alive all the time during simulation. This assumption guarantees that a node is always ready to relay packets if it is within transmission range of another node.

4.2.2 Experimental Setting

The NS-2 simulation environment we use is a four-layer network environment. From top to bottom they are the application, transport, network and Medium Access Control (MAC) layers. The application layer provides Constant Bit Rate (CBR) applications. The reliable multicast protocol models are implemented at transport layer. At the network layer, ODMRP is used for multicast routing and DSR is used for unicast routing. The Distributed Coordination Function (DCF) of IEEE 802.11 for wireless LANs is used as the MAC layer protocol. The Monarch extension package [46] developed by the researchers at Carnegie Mellon University provides an implementation of ODMRP in NS-2.

We configure our NS-2 simulation environment as a 50-node mobile ad hoc network. These mobile nodes can move around over a 1400 meters x 1400 meters flat space for 450 seconds of simulation time. There is one multicast group in this mobile ad hoc network. This group consists of one traffic source and nine receivers. The normal transmission range of each node is 250 meters. The channel capacity is 2 Mb/s.

4.2.3 Mobility Model

The input to each simulation run is a node movement scenario file that describes the motion of each node. We pre-generate multiple different node movement scenario files with various movement patterns according to the "random waypoint" model [6]. In "random waypoint", each node begins the simulation by remaining stationary at its current position for a certain pause time. It then selects a random destination from the 1400 meters x 1400 meters space and moves to the destination at a speed distributed uniformly between 0 and a maximum speed (we set maximum speed at 1 meter / second in our simulations). When reaching the destination, the node stays there for a second pause, chooses a destination, and so on till the end of the simulation.

4.2.4 Traffic Model

The traffic source of the multicast group in our simulation is a CBR application at the application layer. The sending rate is configurable in NS-2 environment. Each packet sent by the source has a fixed size payload of 512 bytes. In order to simulate a real network environment, we add CBR background traffic among the mobile nodes that do not belong to the multicast group in additional to the interaction traffic within the multicast group. To inject the background traffic, several source-destination pairs are

positioned randomly in the network, with unicast traffic maintained among them. The background sources and destinations are not members of the multicast group.

4.2.5 Selection of Timer Values

SRM uses dynamic timers for transmission of NACKs or repairs. The value of each timer changes during a given simulation run, depending on the delay between two nodes engaged. TMTP and RMP do not specify whether to use fixed or dynamic timer. In our implementation, we use fixed timers for TMTP and RMP. In TMTP and RMP, the expiration of the request (NACK) timer without reception of the corresponding packet serves as a detection of loss of NACK or retransmitted packet. RMP has three timers, namely the request timer, the transmission timer and the ACK timer. A source starts a transmission timer for each packet it transmits. If the source does not receive an ACK for a packet after the transmission timer expires, it retransmits the packet and resets the timer. When the current token site sends an ACK by multicast, it sets an ACK timer to ensure that the success of passing on the token. Recall that the ACK also serves as token. If the current token site has not heard from the next token site an announcement of becoming a current token site after the ACK timer expires, it resends the ACK. It is important to note that the transmission timer value must be larger than the value of ACK timer to prevent the source from unnecessary retransmission.

Table 4-1 through Table 4-4 list the parameters that are constant throughout our simulation. Parameters that vary are detailed in the "simulation results" section.

Parameter	Value
Simulation grid size	1400 m x 1400 m
Node placement	Random
Number of nodes	50 (except in scenario of "effect of number of nodes)
Multicast group	1
Number of sources in multicast group	1
Multicast routing protocol	ODMRP
Unicast routing protocol	DSR
MAC protocol	IEEE 802.11 DCF
Bandwidth	2 Mb/sec
Size of CBR packet	512 bytes
Maximum moving speed of nodes	1 meter / second
Simulation time	450 sec

Table 4-1 Simulation environment parameters

Parameter	Value
C_1, C_2	2
D_1, D_2	1
Maximum number of times a NACK is backoff	5
Session message interval	1 second

Table 4-2 SRM parameters

Parameter	Value
Initial value of request timer	1 second
Maximum number of times a NACK is resent	5

Table 4-3 TMTP parameters

Parameter	Value
Initial value of request timer	1 second
Maximum number of times a NACK is resent	5
Initial value of ACK timer	1 second
Maximum number of times an ACK is resent	5
Initial value of transmission timer at sender	3 second
Maximum number of times a packet retransmission is resent	5

Table 4-4 RMP parameters

4.2.6 Performance Metrics

We collect the following statistics from each simulation run:

- Average packet delivery ratio – the ratio of the number of data packets received by the multicast receivers to the number of data packets sent by a sender. We use this ratio to determine the reliability of a protocol. Ideally this ratio should approach 1, as each receiver should eventually receive all packets.

- Average end-to-end delay – the average time a packet takes to travel from source to destination. This delay includes the delay caused by retransmission. We use average end-to-end delay to evaluate a protocol's timeliness.

- Packet retransmission overhead – the ratio of the number of retransmitted packets to the number of packets sent by a sender. Retransmission overhead of data packet is more significant than that of other control packets, such as ACKs or NACKs, since ACK or NACK packets are usually smaller than data packets. Packet retransmission overhead is an important index for comparing protocols, as it measures the scalability of a protocol, the degree to which it can still function in congested or low-bandwidth environment, and its efficiency in terms of consuming mobile node battery power. Protocols that send large numbers of control packets can increase the probability of packet collisions and may increase data packet delay.

4.3 Simulation Results

We conduct multiple simulations in order to measure the performance of SRM, TMTP, and RMP in mobile ad hoc networks. We subject these three protocols to a range of ad hoc network characteristics:

- Node mobility (pause time) – P: how frequent a mobile node moves,
- Transmission range – R: the area over which transmission can be heard,
- Packet arrival rate – λ: how many packets a sender sends per second,
- Number of nodes – N: how many nodes within an ad hoc network,
- Group size – G: how many nodes within a multicast group,
- Background traffic – B: the traffic caused by mobile nodes other than the multicast group members.

We also conduct two special-case simulations that demonstrate the performance of protocols under various conditions:

- A cluster of nodes: a number of nodes belonging to a multicast group that stay within a small vicinity and the nodes stay stationary all the time,
- Different timer settings: set different value for the timers in TMTP and RMP.

The results are shown in Figures 4-1 – 4-9. In these figures, the three charts in each column show one simulation scenario result. Simulation results are obtained from multiple runs and the results are averaged over the runs (with 90% confidence level and 10% confidence intervals).

4.3.1 The Effect of Pause Time

Node mobility is controlled by setting various pause times – P. The simulation results show how node mobility, or pause time – P affects the performance of protocols in ad hoc networks.

4.3.1.1 Basic Mobility Experiment

We randomly choose one source and nine receivers from the 50 mobile nodes to form a multicast group. We run simulations with movement patterns generated for five different pause times: 70, 100, 130, 160, and 200 seconds. Table 4-5 lists the parameters we use for this experiment.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	4 Packets / Second
R	250 Meters
B	0.5 Mbit / Second
P	70, 100, 130, 160, 200 Seconds

Table 4-5 Parameters for basic mobility experiment

Column I – chart A in Figure 4-1 shows that the average packet delivery ratio increases as pause time increases (i.e. node mobility decreases) in all three protocols. The ratios of TMTP and RMP converge to 100% and the ratio of SRM has a trend towards 100% as the pause time increases to 200 seconds. For TMTP, high node mobility can increase the possibility of packet loss. A child has to frequently execute "Join" procedure to find a new parent because node movements frequently change the structure of the control tree. For the same reason, the child has to repeat requesting packet retransmission more often. These operations generate many additional network overheads that easily congest the

readily bandwidth constrained links. Accordingly, more packets will be dropped and more transmissions will eventually fail. This is why we see the ratio less than 100% at pause times between 70 and 100 seconds. In our TMTP model, we set NACK retry limit to five times. When reaching five NACKs, the node multicasts NACK to the entire group to request repair, and search for a new parent. In a similar fashion, ring reformation in RMP is frequently invoked when the next token site is unreachable due to node movements. We use a maximum of five times to limit the number of retries on reaching the next token site. When the limit is reached, the ring reformation is initiated. If the source does not receive an ACK message from the current token site due to node movements, it repeats retransmission. Again, all these operations cause further network overhead and lead to congestion. Even in the extreme case where the current token site continuously initiates ring reformation, it will still be responsible for serving packet retransmissions until the ring is reconstructed. This violates the design principles of ring-based protocols, where each node takes turn to be the current token site. The current token site can be overwhelmed by the operation of retrying token passing and responding to retransmission requests that may start dropping packets. This explains why we see the ratio lower than 100% at pause times between 70 and 100 seconds.

From Column I in Figure 4-1 we can observe that the ratio of SRM is generally lower than that of TMTP and RMP before it eventually converges to 100%. This is because SRM uses multicast for both NACKs and Repairs. When transmission error occurs frequently, there are many NACKs and Repairs multicast, while in TMTP and RMP receivers only send NACKs via unicast. The SRM network overheads are more

significant than that of TMTP and RMP. Therefore, congestion can be more serious and more packets can be dropped. Another reason is that SRM does not use ACK to make sure packets have successfully arrived at the receivers. In the case of all receivers missing a packet, the packet can not be recovered since the source does not perform retransmissions.

As node mobility decreases, the possibility of packet loss drops. TMTP and RMP less frequently execute the operations of tree restructure and ring reformation. SRM has less NACKs and Repairs to multicast. Thus, the network congestion is alleviated and the network has a better throughput. The chart shows TMTP and RMP have relatively better reliability than SRM when node mobility is lower.

Column I – chart B in Figure 4-1 shows that the average end-to-end delay for transmitting a packet drops as node mobility decreases. Notice that the delay includes retransmission time. When node mobility is high, there are more retransmissions than when node mobility is low. Therefore, the delay is longer around pause times between 70 and 100 seconds. We can also see from the chart that when node mobility is high, RMP has the longest delay and SRM has shortest delay with TMTP in between. This particularly demonstrates SRM's strength of fast error recovery, in which NACKs are multicast to the entire group. Any neighbor who hears the NACKs can respond to the NACKs with a repair. This shortens the end-to-end delay. On the other hand, if the current token site in RMP misses a packet due to node movements or other reasons, the source is the node that retransmits the packet. As aforementioned, high mobility may

cause frequent ring reconstruction. The source's continuous retransmission and frequent ring reconstruction will cause network congestion, which results in long delays. In TMTP, the parent uses unicast instead of multicast to request and retransmit a missing packet. Multiple retransmission requests can happen if the parent is unreachable or it does not have the requested packet, which further increases delay. In addition to the delay caused by ring or tree restructure discussed above, when the current token site or the parent is far away from the requestor node, the long transmission distance can result in larger delays.

We observe an undesired behavior of the TMTP model during simulation, which affects the performance of TMTP as well. TMTP does not prevent a node from finding one of its hierarchical children as its new parent, i.e., a sub-tree is disconnected from the whole tree structure and becomes an isolated ring. Consider a tree structure where node A has several children, one of which is B, which in turns has multiple children, one of which is node C. When the tree structure is changed due to node movements, A gives up its parent and looks for a new parent. Since TMTP is self-organizing, A may chose C as it new parent unless a certain algorithm is implemented to specifically avoid this problem. Such loops will render the entire tree structure broken, and it may take a considerably long time for a node to return to the main control tree. Naturally, the network performance will be seriously reduced. Note that this problem does not occur in wired networks since the control tree is organized based on geographic arrangements, i.e. all members in the same subnet belong to a definite sub-tree. In our implementation, we

specify that a node can not find its children as its parent as a temporary solution, but it is possible that a node chooses its grandchild as its new parent, as described above.

Column I – chart C in Figure 4-1 depicts that the packet retransmission overhead decreases as node mobility drops. When nodes move less frequently, the possibility of packet loss is low and it takes a lower number of retransmissions to deliver a packet. We can also see from the chart, for the same node mobility, RMP has the highest overhead and SRM has the lowest overhead with TMTP in between. This again proves SRM's fast error recovery. In SRM, retransmission requests and repairs are always multicast to the whole group. Any member that has a copy of the requested data can answer a request. Thus, in the case that a number of nodes all miss the same packet, one multicast repair is sent to the whole group. Also, with the Repair suppression scheme the number of retransmissions can be significantly reduced. On the other hand, a source in RMP repeats packet transmission either because a packet or an ACK message is lost due to the movement of the current token site or network congestion. In TMTP, in addition to the retransmissions caused by tree structural change, the retransmission via unicast between parent and receivers can generate many added packet retransmissions if many children loss the same packet.

4.3.1.2 Mobility Experiment with Lower Arrival Rate

In this experiment we simulate the models with packet arrival rate (λ) equal to 2 Packets / Second using the same movement scenario files used in section 4.3.1.1.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	**2 Packets / Second**
R	250 Meters
B	0.5 Mbit / Second
P	70, 100, 130, 160, 200 Seconds

Table 4-6 Parameters for mobility experiement with lower arrival rate

From Column II in Figure 4-1 we can see that all three models behave in a manner similar to their behavior with $\lambda = 4$ Packets / Second. The packet delivery ratio, average end-to-end delay and packet retransmissions all show the same trend. At each pause time point, the performance shown gives the same relative results that were discussed above. By comparing results for $\lambda = 2$ and $\lambda = 4$, we can see that the performance in the former is generally better than in the latter. This is expected since as the CBR application reduces its data rate, there is less traffic for the network to handle. There will be less contention for the wireless medium and less back off and waiting time at the underlying 802.11 protocol. In other words, network congestion is alleviated and any problem caused by network congestion with $\lambda = 4$ is therefore relieved. Hence, packet delivery ratio increases, and end-to-end delay and retransmission packet overhead drop.

4.3.1.3 Mobility Experiment with Lighter Background Traffic

The third mobility experiment tests how various pause times with lighter background traffic (B = 0.2 Mb/sec, 1/10 of the wireless channel capacity) affect performance.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	4 Packets / Second
R	250 Meters
B	**0.2 Mbit / Second**
P	70, 100, 130, 160, 200 Seconds

Table 4-7 Parameters for mobility experiment with lighter background traffic

We can see from Column III of Figure 4-1 that the general trend of B = 0.2 is similar to that of B = 0.5. The background traffic does not directly affect the performance of each node in the multicast group. However, it can be seen that background traffic affects the performance of nodes utilized by multicast groups for multi-hop routing. When these nodes are busy in handling the background traffic, there are less processing capabilities available for routing multicast packets. This causes longer transmission and retransmission delay, in addition to increasing packet loss. When background traffic is lighter, these phenomena are alleviated, the packet delivery ratio, average end-to-end delay and packet retransmission overhead all show better performance.

4.3.1.4 Mobility Experiment with Larger Transmission Range

The fourth mobility experiment tests how various pause time with larger transmission range (R = 350 Meters) affect the network performance.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	4 Packets / Second
R	**350 Meters**
B	0.5 Mbit / Second
P	70, 100, 130, 160, 200 Seconds

Table 4-8 Parameters for mobility experiment with larger transmission range

Column IV in Figure 4-1 shows that the general trends in performance with R = 350 Meters are similar to those with R = 250 Meters. However, it can be seen that a larger transmission range enhances the general performance. This is because when the transmission range is larger, number of hops that a packet travels to destination is decreased, and the packet can be delivered to destination more quickly. The control packet overhead will also be decreased. Therefore, with a lower traffic load and a high traffic throughput, a higher packet delivery ratio is achieved.

4.3.2 The Effect of Transmission range

In this section we conduct experiments to test how the node transmission range (R) affects protocols performance. We run simulations with node movement scenario files generated with pause time = 130 seconds. We set the node transmission range at 230, 250, 270, 290, 350 and 500 Meters.

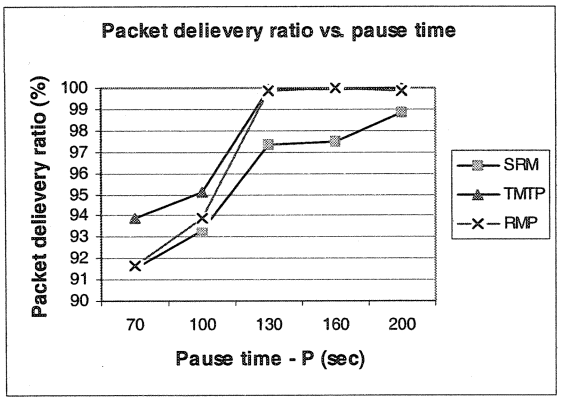
4.3.2.1 Basic Transmission Range Experiment

We use this experiment as the base case, which we can use to compare with later transmission range experiments. Table 4-9 lists the parameters we use in this experiment.

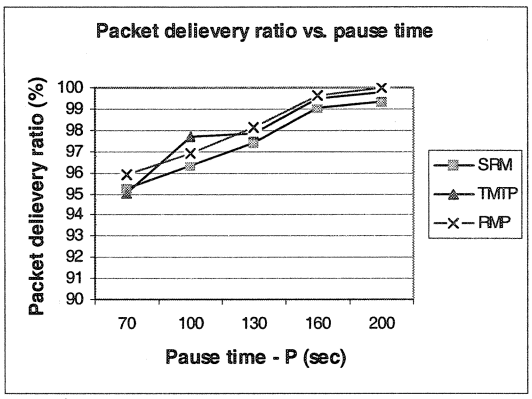
Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	4 Packets / Second
R	**230, 250, 270, 290, 350, 500 Meters**
B	0.5 Mbit / Second
P	130 Seconds

Table 4-9 Parameters for basic transmission range experiment

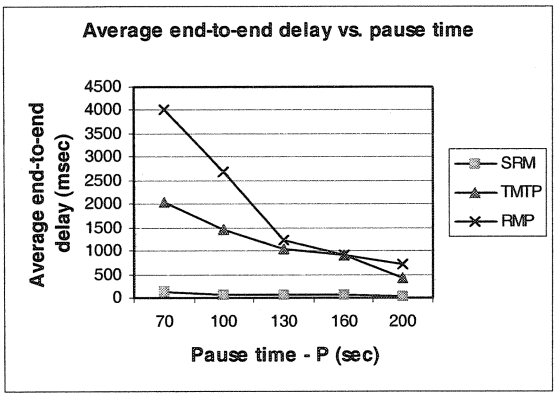
Column (I) - λ=4, R=250, B=0.5

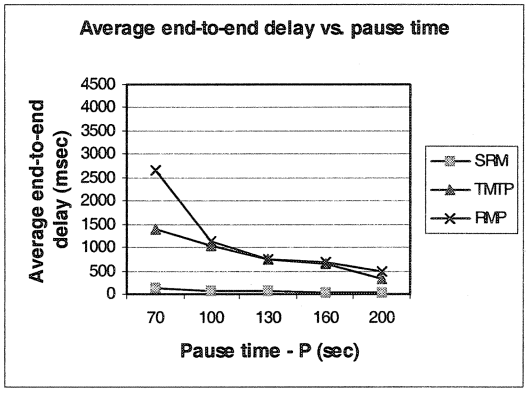


Column (II) - λ=2, R=250, B=0.5

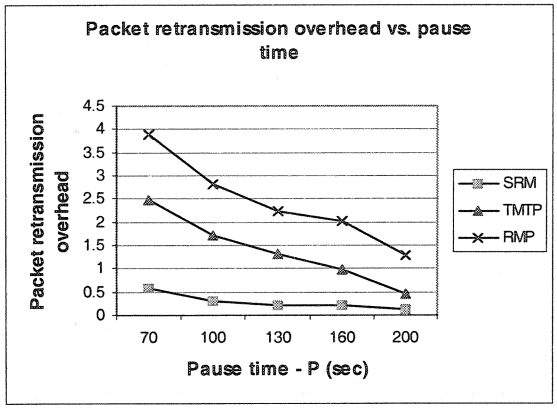


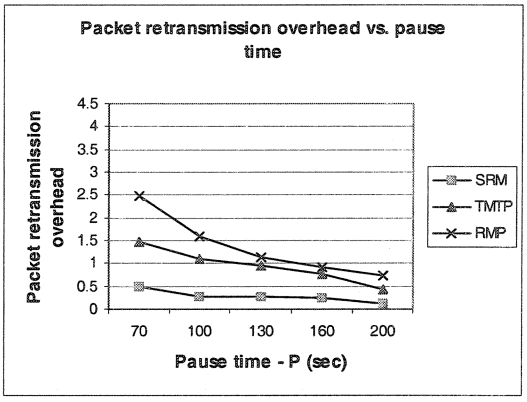
(A) Packet delivery ratio





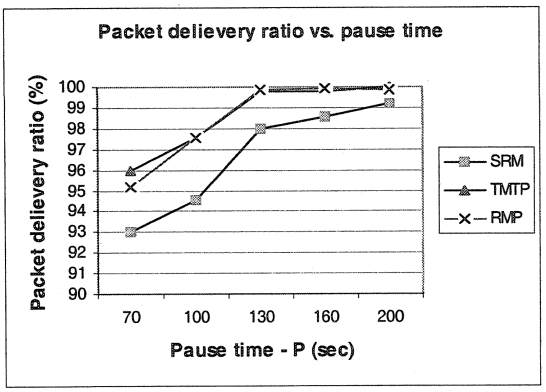
(B) Average end-to-end delay



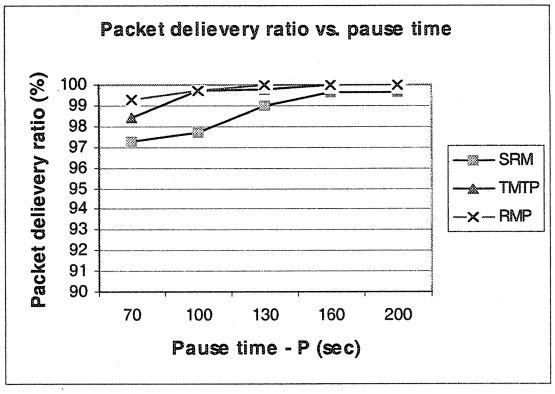


(C) Packet retransmission overhead

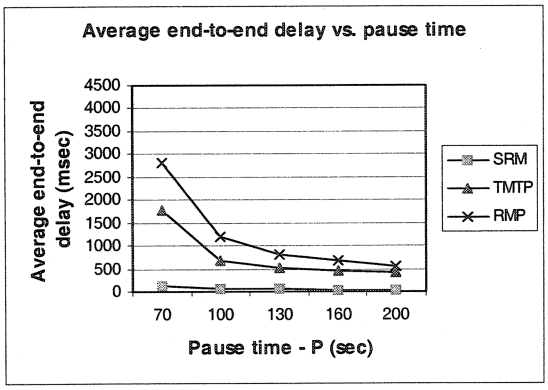
Column (III) – λ=4, R=250, **B = 0.2**

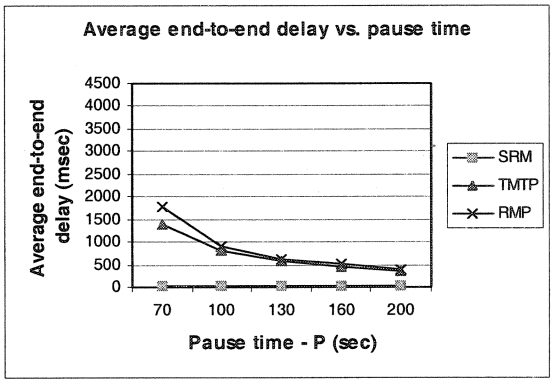


Column (IV) – λ=4, **R=350,** B = 0.5

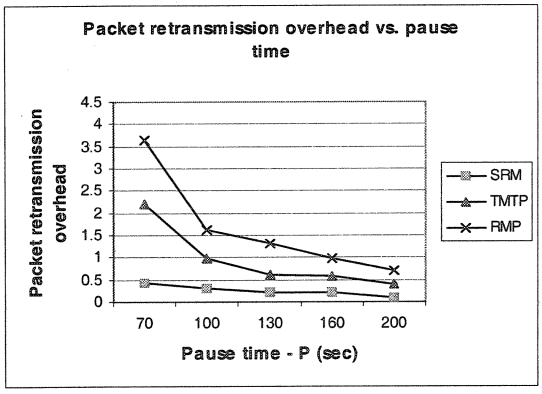


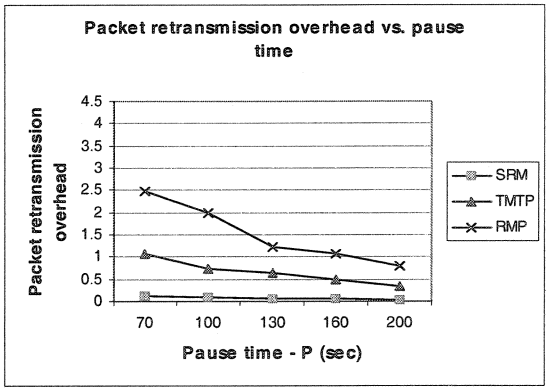
(A) Packet delivery ratio





(B) Average end-to-end delay





(C) Packet retransmission overhead

Figure 4-1 Effect of pause time

Column I – chart A in Figure 4-2 shows that the average packet delivery ratio of the three protocols increases as the transmission range increases. At R = 500 meters, all three ratios converge to 100%. As a node's transmission range increases, the range a single hop can reach is increasing. Therefore, for the same distance the underlying routing protocol needs less intermediate nodes to relay packets, or less hops in between source and destination. As in previous cases, background traffic can consume a significant portion of the processing capability of the intermediate nodes, resulting in insufficient processing capacity to work for multicast groups. Hence, multicast packets will be lost, and packet retransmissions in addition to propagation delays will be incurred. With less intermediate nodes, there will be a lower possibility of packet loss and retransmission and long propagation delays. This explains why at transmissions range less than 270 meters, packet delivery ratio is not 100%, average end-to-end delay is very high, and packet retransmission overhead is high.

Column I – chart A in Figure 4-2 shows that SRM has slightly lower packet delivery ratio than that of TMTP and RMP. This is because SRM's multicast NACK and Repair generate more traffic, which reduces the performance of the intermediate nodes. Together with the background traffic, multicast NACK can congest some nodes and have them fail to relay packets, introducing packet loss. Relatively, TMTP and RMP do not use such multicasting and bring less traffic burden to each intermediate node, creating lower packet losses.

From Column I – chart B in Figure 4-2, we can observe that RMP has a longer average end-to-end delay than TMTP at R = 230 meters. However after R > 270 meters, RMP has a shorter delay than TMTP. This is because the retransmission between source and token site, and token site to other receives contributes largely to end-to-end delay due to multi-hop routes. When R > 270 meters, the ring and tree have relatively stable control structure due to less intermediate nodes. In a stable ring, a receiver can always obtain the retransmitted packet from the current token site, which always has the requested packet. Also, a source in RMP always retransmits a packet if no ACK message is received from the current token site. This also helps receivers get requested packet more quickly. Thus, transmission delay is less in RMP. In TMTP, however, receivers unicast NACK messages to their parents. If the parent does not have the packet, the receiver has to execute a "Join" procedure to find a new parent after failing for a certain number of retries in a sending NACK message. This can cause longer transmission delay.

Column I – chart C in Figure 4-2 shows that RMP has higher packet retransmission overheads than TMTP, and SRM has the lowest overheads. As previously discussed, a short transmission range introduces more packet loss. In RMP, if the source does not receive an ACK message for a packet from the current token site due to a loss of a packet or an ACK message, the source keeps retransmitting a packet for five times before it gives up. This introduces many packet retransmissions. Relatively, RMP performs retransmissions in between the source and the token site and token site to other receivers, which means more retransmission than in TMTP. As SRM has a fast error recovery due

to enabling the NACK / Repair suppression scheme, packet retransmission is significantly limited.

4.3.2.2 Transmission Range Experiment with Lower Arrival Rate

In this experiment we test how a lower packet arrival rate with various transmission ranges affects network performance.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	**2 Packets / Second**
R	230, 250, 270, 290, 350, 500 Meters
B	0.5 Mbit / Second
P	130 Seconds

Table 4-10 Parameters for transmission range experiment with lower arrival rate

Column II of Figure 4-2 shows that as λ is brought down to 2 Packets / Second, packet delivery ratio, end-to-end delay and retransmission packet overhead all exhibit the same trends as that with λ = 4 Packets / Second. However, at lower transmission ranges, the lower packet arrival rate apparently improves network performance. We attribute this trend to an uncrowded network. Since less traffic is generated in the network, there will be less contention for the wireless medium and less back off and waiting time at the 802.11 level.

4.3.2.3 Transmission Range Experiment with Lighter Background Traffic

The third experiment set tests how lighter background traffic (1/10 of the wireless channel bandwidth) with various transmission ranges affects network performance.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	4 Packets / Second
R	230, 250, 270, 290, 350, 500 Meters
B	**0.2 Mbit / Second**
P	130 Seconds

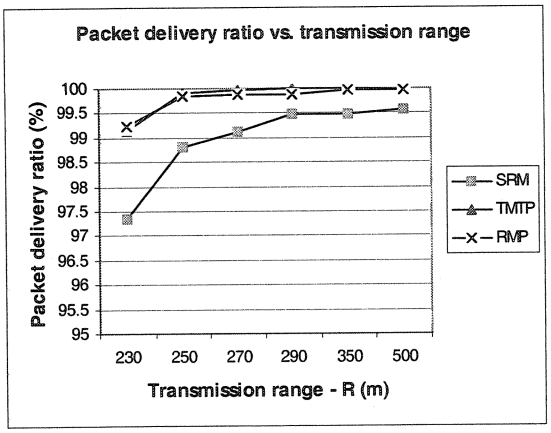
Table 4-11 Parameters for transmission range experiment with lighter background traffic

From Column III in Figure 4-2 we can observe that the performance of short transmission range is sensitive to background traffic. As background traffic becomes lighter, the intermediate nodes are less interfered. The network shows relatively better performance. Thus we see that the packet delivery ratios for the three protocols are all above 98%. The end-to-end delay is shorter by 1 second and overhead is dropped from 2.5 to 2 for RMP. Although the performance of long transmission ranges is not as sensitive as that of short ranges, the performance does improve slightly with the former.

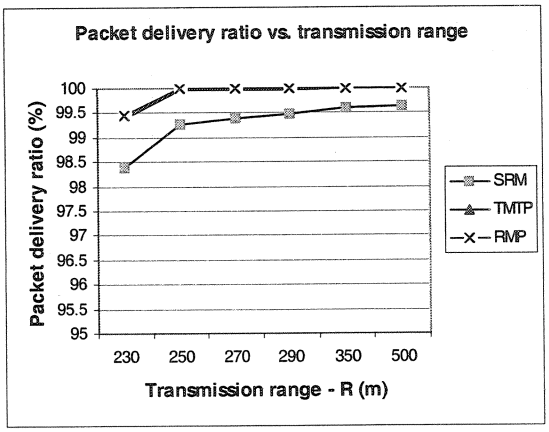
4.3.3 The Effect of Packet Arrival Rate

The experiments in this section test how different packet arrival rates (λ) affect network performance. We randomly choose one source and nine receivers among 50 mobile nodes to form a multicast group. The packet arrival rates vary from 1 to 5 Packets / Second.

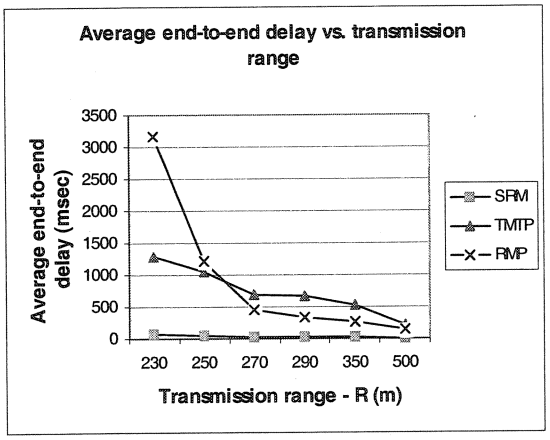
Column (I) - λ=4, B=0.5

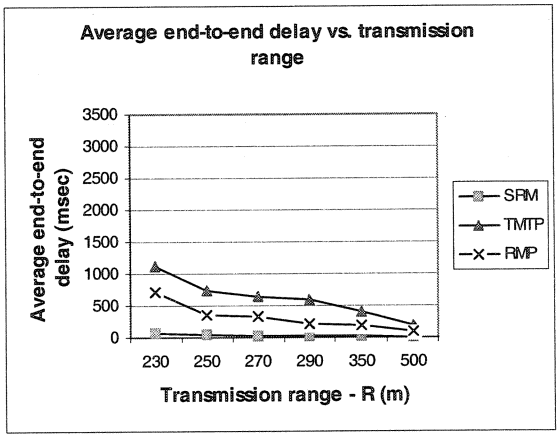


Column (II) - λ=2, B=0.5

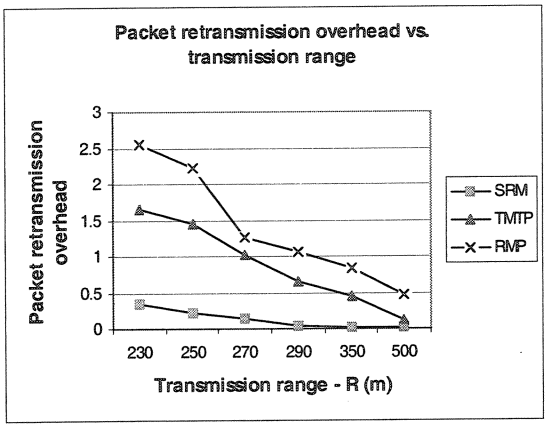


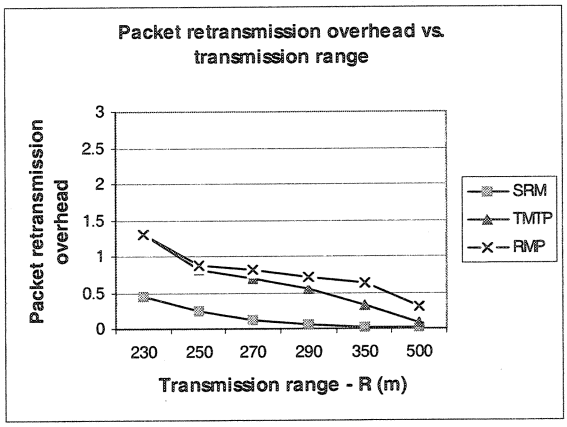
(A) Packet delivery ratio





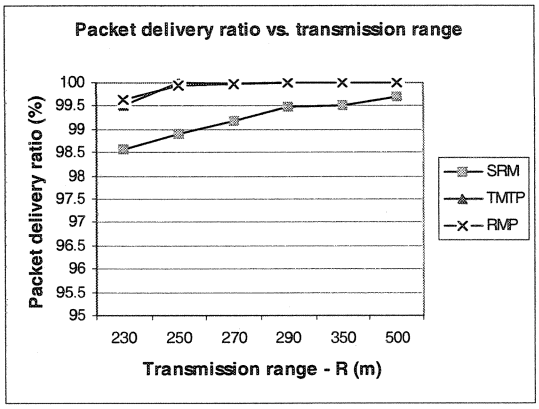
(B) Average end-to-end delay



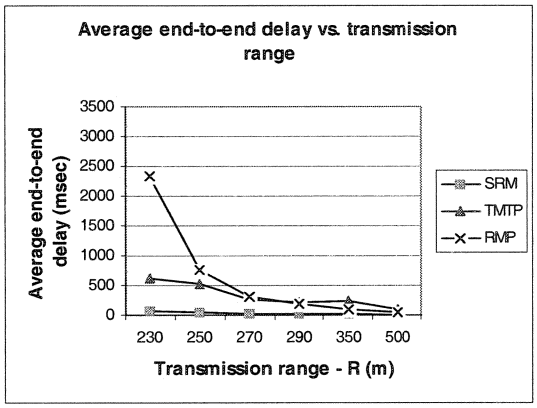


(C) Packet retransmission overhead

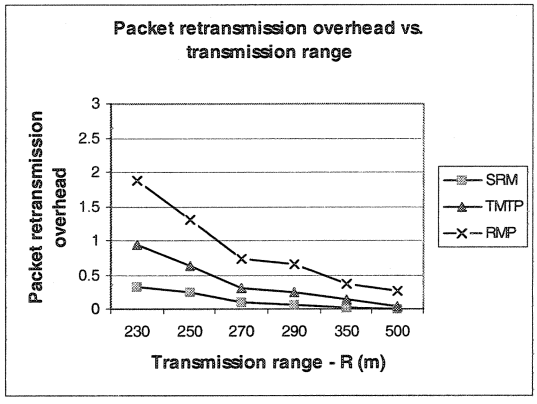
Column (III) - λ=4, **B=0.2**



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-2 Effect of transmission range

4.3.3.1 Basic Packet Arrival Rate Experiment

Figure 4-3 shows the performance of three protocols as packet arrival rate changes. Table 4-12 lists the parameters used in this experiment.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	1, 2, 3, 4, 5 Packets / Second
R	250 Meters
B	0.5 Mbit / Second
P	130 Seconds

Table 4-12 Parameters for basic packet arrival rate experiment

From Column I – chart A in Figure 4-3 we can see that both TMTP and RMP achieve almost 100% packet delivery ratio when $\lambda < 4$. SRM has a lower ratio, especially when $\lambda > 3$, where the ratio drops largely. SRM's poor performance stems from its attempts to recover lost packets by using multicast of NACKs and Repair. When packet arrival rate is high, these multicast traffics cause high network resource contention, which results in increasing packet losses. More packet loss in turn triggers more error recovery by multicasting NACKs and Repair, which further dampens the situation. On the other hand, Column – I chart B and C shows that obtaining repair from nears neighbors other than only from the source or the current token site helps SRM achieve better end-to-end delay and packet retransmission overheads than TMTP and RMP.

4.3.3.2 Packet Arrival Rate Experiment with Lower Background Traffic

The second packet arrival rate experiment shows that lighter background traffic (1/10 of the wireless channel bandwidth) with various packet arrival rates apparently improves network performance. The results, shown in Column II of Figure 4-3, are expected since

less traffic reduces network resource contention and leads to lower packet loss and better performance.

Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	1, 2, 3, 4, 5 Packets / Second
R	250 Meters
B	**0.2 Mbit / Second**
P	130 Seconds

Table 4-13 Parameters for packet arrival rate experiment with lower background traffic

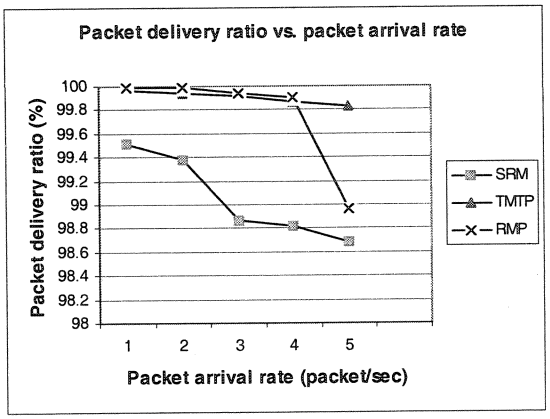
4.3.3.3 Packet Arrival Rate Experiment with Larger Transmission Range

The third experiment proves that a larger node transmission range (R = 350 Meters) can improve network performance. Nodes with a larger transmission range reduce the number of hops in between the source and the destination and are less interfered by the background traffic. Therefore, a better performance is achieved. From Column III – chart A in Figure 4-3, we observe that the longer transmission range apparently improves SRM's performance. This is because lower hop routes lead to less packet loss, and avoid more error recovery triggered by error recovery procedures, as we described in 4.3.1.1.

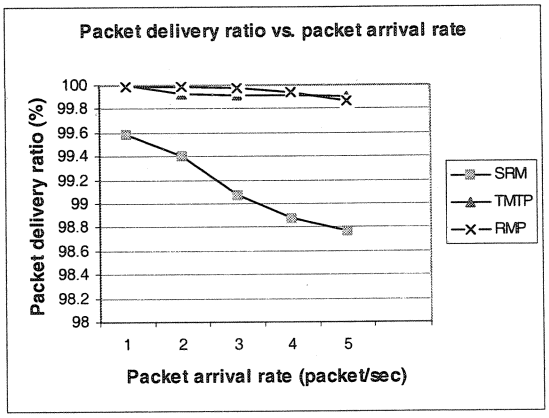
Parameter	Value
N	50 Node ad hoc network
G	10 Node multicast group
λ	1, 2, 3, 4, 5 Packets / Second
R	**350 Meters**
B	0.5 Mbit / Second
P	130 Seconds

Table 4-14 Parameters for packet arrival rate experiment with larger transmission range

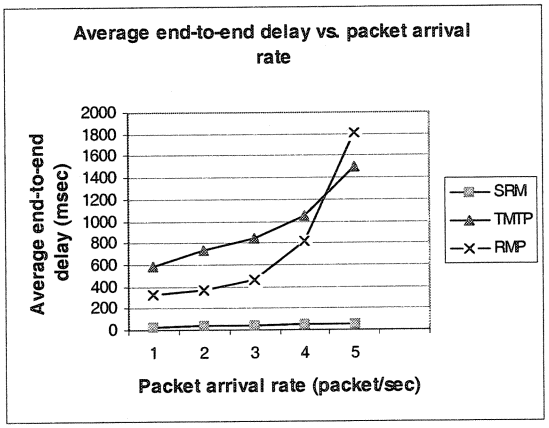
Column (I) – R=250, B=0.5

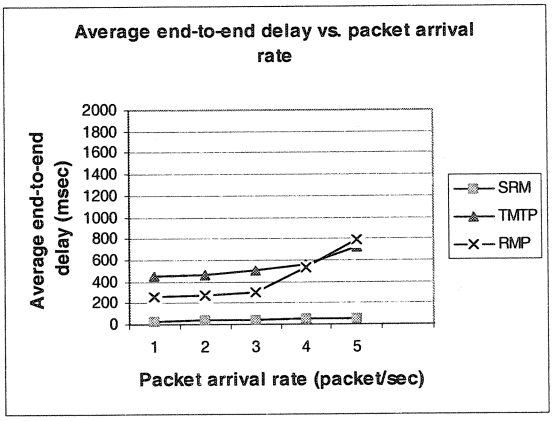


Column (II) – R=250, **B=0.2**

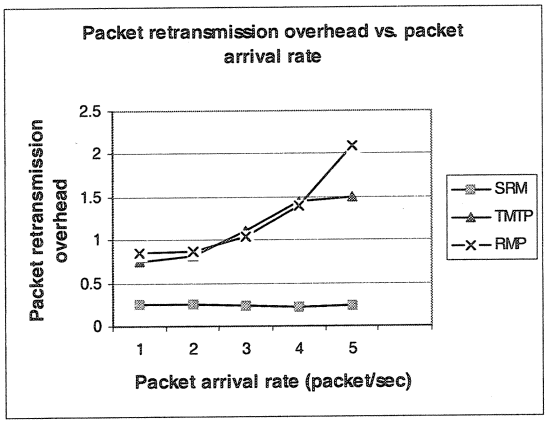


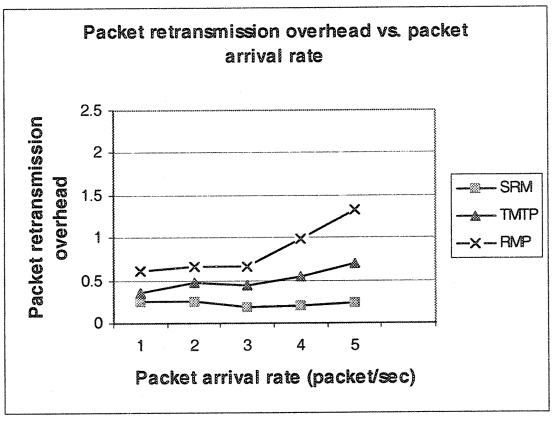
(A) Packet delivery ratio





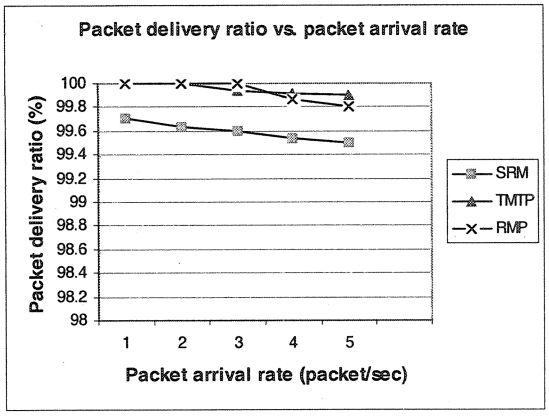
(B) Average end-to-end delay



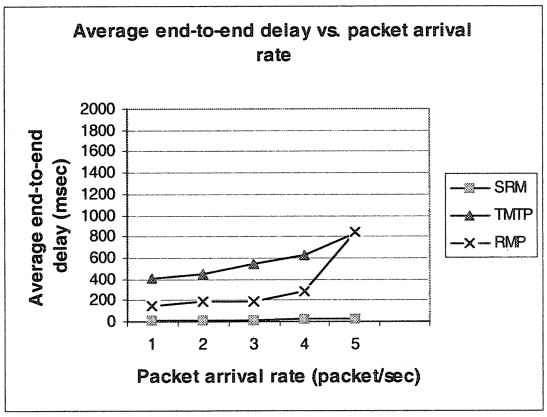


(C) Packet retransmission overhead

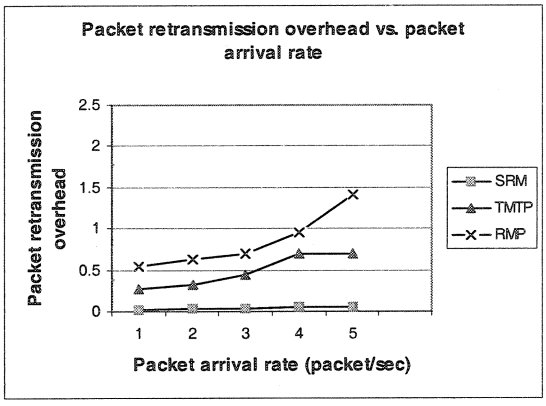
Column (III) – **R=350**, B=0.5



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-3 Effect of packet arrival rate

4.3.4 The Effect of Number of Nodes

The results of the experiments in this section show the effect of changing the number of mobile nodes in the network on performance. The number of nodes (N) is varied from 20 to 70. We randomly choose one source and nine receivers to form a multicast group.

4.3.4.1 Basic Number of Nodes Experiment

Figure 4-4 shows the performance of the three protocols with different number of nodes.

Parameter	Value
N	20, 30, 40, 50, 60, 70
G	10
λ	4 Packets / Sec
R	250 Meters
B	0.5 Mbit / Second
P	130 Seconds

Table 4-15 Parameters for basic number of nodes experiment

We observe from Column I – chart A of Figure 4-4 that with the increase of N, packet delivery ratio is increases up to some point (N = 40), then decreases. When N is small (e.g. N = 20), packet delivery ratio is low. Two factors explain this result. First, with small number of nodes, the network can not be fully connected, i.e. a node may not be able to transmit packets to other nodes since there are not enough nodes to relay packets. Packets have to be dropped if no connection can be found. Second, we inject the same amount of background traffic, independent of the number of nodes. Naturally, the background traffic will have more effect on a small network than in a larger one. It is due to these two factors that some packets may not arrive at their destinations, i.e. dropped, and will accordingly not be counted when we calculate average end-to-end

delay. This results in the low end-to-end delay when N is small as shown in Column I – chart B of Figure 4-4.

When the number of nodes is raised to some point (here is N = 40), the packet delivery ratio decreases. As the number of nodes further increases, more routing decisions are made and the number of routing packets exchanged increases, which contributes to network load. Therefore, as shown in Column I – B and C of Figure 4-4, more packets are dropped and more packets have to be retransmitted, resulting in longer delays and larger packet retransmission overheads.

Column I – chart C of Figure 4-4 shows that RMP's retransmission packet overhead is high with a small number of nodes. The reason is that the source keeps retransmitting a packet many times until it receives an ACK message for that packet. Two factors cause the source's retransmission. First, because connectivity is low with a small number of nodes, current token site may not be able to pass the token to next token site, resulting in frequently reconstructing the token ring. This reconstruction adds traffic to the network and may cause loss of an ACK message moving towards the source. The source will then keep retransmitting a packet. The second factor that can invoke resource's retransmission is when the current token site can not reach the source because of low connectivity.

4.3.4.2 Number of Nodes Experiment with Larger Transmission Range

The second experiment is to test the effect of number of nodes with larger transmission range (R = 350m) using the same node movement scenario files.

Parameter	Value
N	20, 30, 40, 50, 60, 70
G	10
λ	4 Packets / Sec
R	**350 Meters**
B	0.5 Mbit / Second
P	130 Seconds

Table 4-16 Parameters for number of nodes experiment with larger transmission range

As shown in Column – II in Figure 4-4, with a larger transmission range, the number of hops that a packet makes to a destination is decreased, resulting in a better performance.

4.3.5 The Effect of Group Size

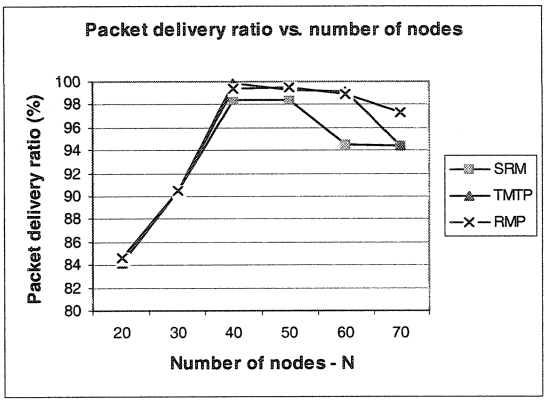
In this section, we test how varying the size of the multicast group affects the network performance, i.e., the scalability of each protocol. The group size varies from 5 to 25. We randomly choose one sender and various numbers of receivers to form a multicast group. We conducted experiments with two different pause times.

Figure 4-5 shows the performance of the three protocols with different group sizes. Parameter setting for this experiment is listed in the following table.

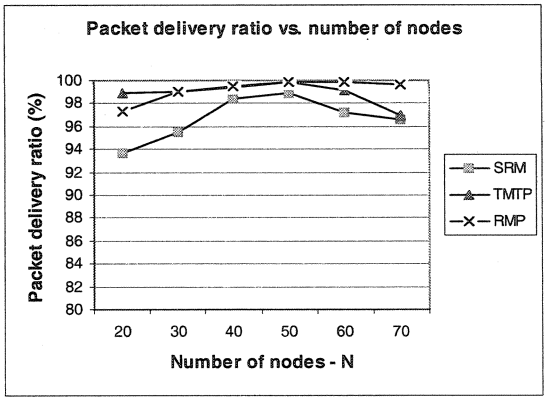
Parameter	Value
N	50
G	5, 10, 15, 20, 25
λ	1 Packets / Sec
R	350 Meters
B	0 Mbit / Second
P	70, 130 Seconds

Table 4-17 Parameters for group size experiment

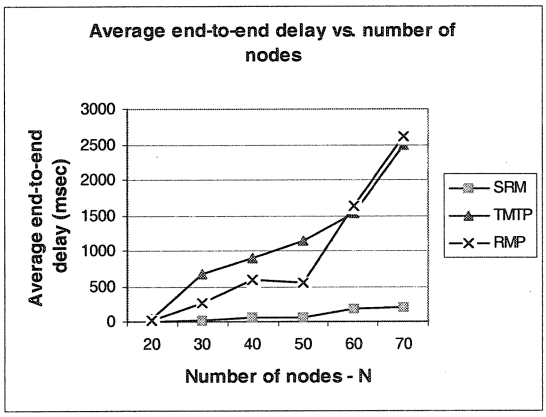
Column (I) - λ=4, R=250, B=0.5

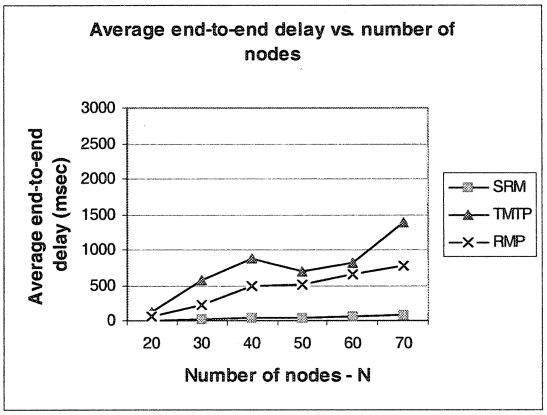


Column (II) - λ=4, **R=350**, B=0.5

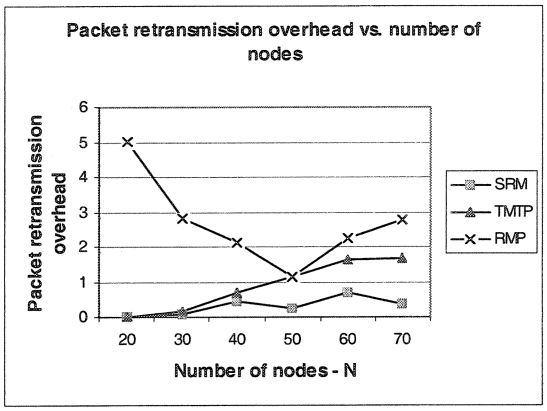


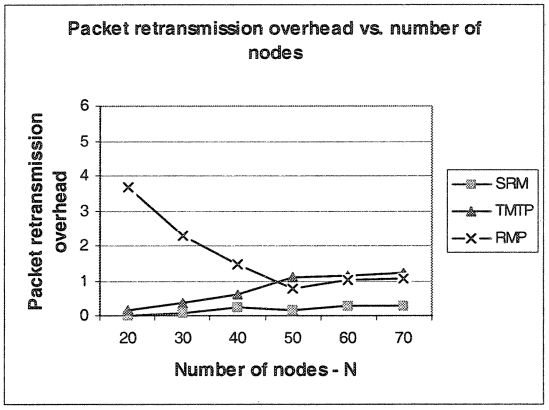
(A) Packet delivery ratio





(B) Average end-to-end delay





(C) Packet retransmission overhead

Figure 4-4 Effect of number of nodes

Packet delivery ratio charts show that TMTP and RMP are more scalable than SRM. As shown in Column I – chart A of Figure 4-5, SRM's packet delivery ratio drops quickly from 99% to 93% as node number increases from 5 to 25. This is again because of SRM's error recovery using multicasting of NACKs and Repairs. Larger size groups have a relatively higher rate of packet, NACK and Repair losses. The loss of NACK or Repair causes more control packet transmissions, and makes a readily busy network more congested. The ratios of TMTP and RMP do drop but within a very limited extent. With more receivers, there is a high probability that receivers are close to each other. Therefore, a child in TMTP can more quickly find a new closer parent, and a current token site in RMP can easily pass the token to next token site or reform a new ring. Hence, the tree or ring structure are quickly stabilized, greatly reducing packet loss. Also, since TMTP and RMP use unicast error recovery, they avoid generating further global multicast traffic, which aids error recovery. However, if there are too many receivers in a multicast group (e.g. G = 25), the traffic caused by multicast sender and receivers is very high, causing network congestion and degrading the performance of TMTP and RMP.

Column I – charts B and C in Figure 4-5 show that as the number of receivers increases, both the average end-to-end delay and packet retransmission overhead increase. This is caused by the network congestion due to having more receivers.

In Comparing Column II with Column I, we can see that the network performance with low mobility (P = 130) is much better than that with high mobility (P = 70). As we discussed in section 4.3.1, the possibility of packet loss under low mobility is minimized, and reconstruction of the trees and rings are less frequent. Therefore, the chance for network congestion is lowered and performance is improved.

4.3.6 The Effect of Background Traffic

The results of the experiments in this section show the effect of changing the background traffic on performance. The background traffic is varied from 0 to 2/5 of the wireless channel bandwidth.

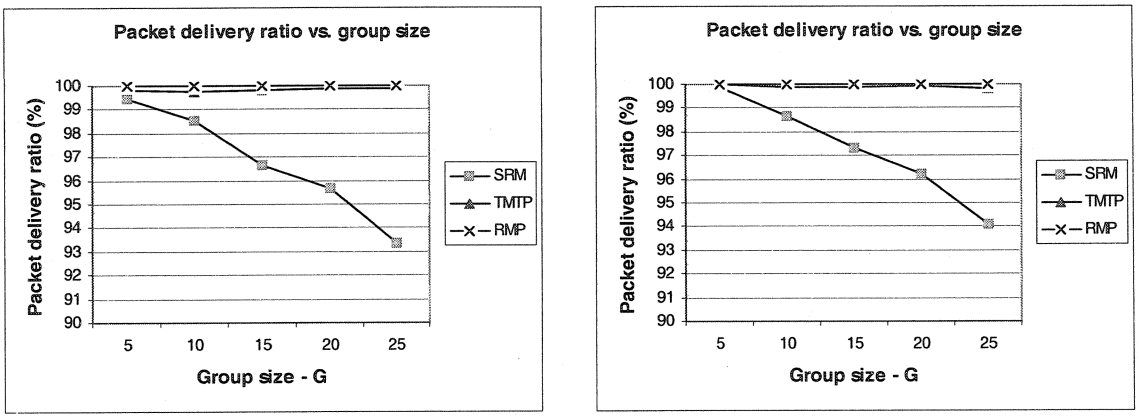
4.3.6.1 Basic Background Traffic Experiment

Figure 4-6 shows the performance of the three protocols with varying background traffic. Parameter setting of this experiment is shown in Table 4-18.

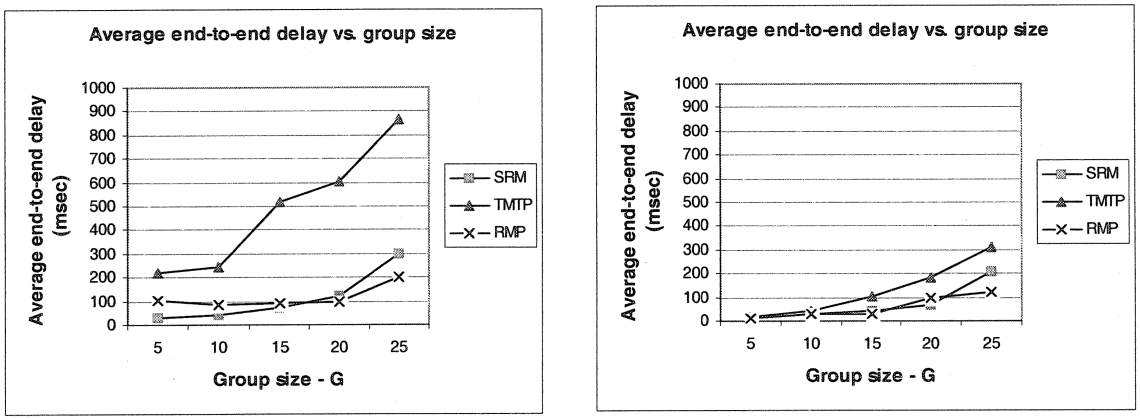
Parameter	Value
N	50
G	10
λ	4 Packets / Sec
R	250 Meters
B	0, 0.2, 0.4, 0.6, 0.8 Mbit / Second
P	130 Seconds

Table 4-18 Parameters for basic background traffic experiment

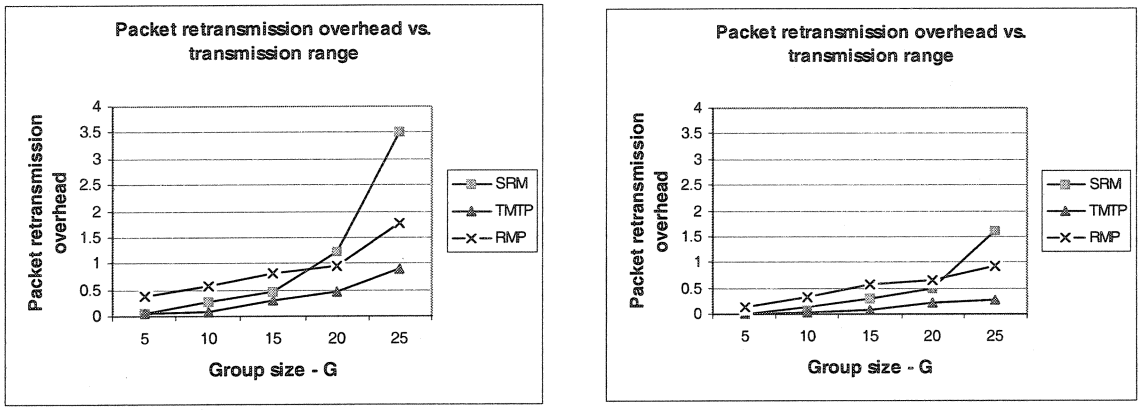
Column (I) – **P=70** Column (II) – **P=130**



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-5 Effect of group size

We notice that for all three protocols, as the background traffic increases, the packet delivery ratio decreases while the average end-to-end delay and packet retransmission overhead dramatically increase (Column I of Figure 4-6). The low packet delivery ratio results from the fact that heavy background traffic causes network congestion and packet loss increases. This congestion, in turn, causes frequent reconstruction of tree and ring in TMTP and RMP, respectively, which causes steady increase of end-to-end delay and packet retransmission overhead.

We also notice that background traffic has little impact on end-to-end delay in SRM. This is mainly due to that any member who has requested a packet can issue a repair. With repair suppression scheme, a node can quickly get the missing packet. On the other hand, TMTP and RMP are quite sensitive to the background traffic. TMTP and RMP produce heavy traffic themselves because of tree or ring reconstruction, in addition to unicast NACK and retransmission. More background traffic will make an already busy network congested in TMTP and RMP. Even though TMTP and RMP have better performance in terms of packet delivery ratio, the price they have to pay is to have longer end-to-end delay and more control packet overhead.

4.3.6.2 Varying Background Traffic with Lower Packet Arrival Rate

The second experiment was to test the effect of background traffic with lower packet arrival rate (2 packets/sec).

Parameter	Value
N	50
G	10
λ	**2 Packets / Sec**
R	250 Meters
B	0, 0.2, 0.4, 0.6, 0.8 Mbit / Second
P	130 Seconds

Table 4-19 Parameters for background traffic experiment with lower packet arrival rate

As shown in Column II of Figure 4-6, the results show trends similar to the ones in Column I. A better performance is achieved in this experiment due to the light multicast traffic on part of the sender and the receivers.

4.3.6.3 Varying Background Traffic with Larger Transmission Range

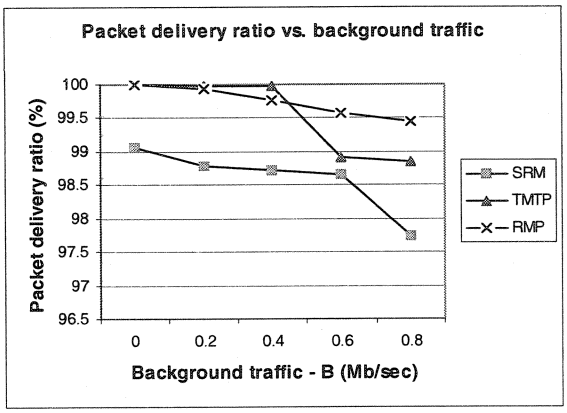
The third experiment was to test the effect of background traffic with larger transmission range (R = 350m).

Parameter	Value
N	50
G	10
λ	4 Packets / Sec
R	**350 Meters**
B	0, 0.2, 0.4, 0.6, 0.8 Mbit / Second
P	130 Seconds

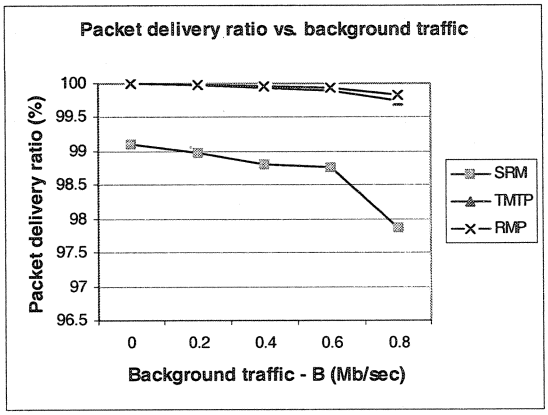
Table 4-20 Parameters for background traffic experiment with larger transmission range

As shown in Column III of Figure 4-6, the result shows trends similar to the ones that are in Column I. A better performance is achieved in this experiment because the number of hops decreases as the transmission range is increased, which lowers the possibility of packet loss. Hence, less tree and ring reconstructions are needed.

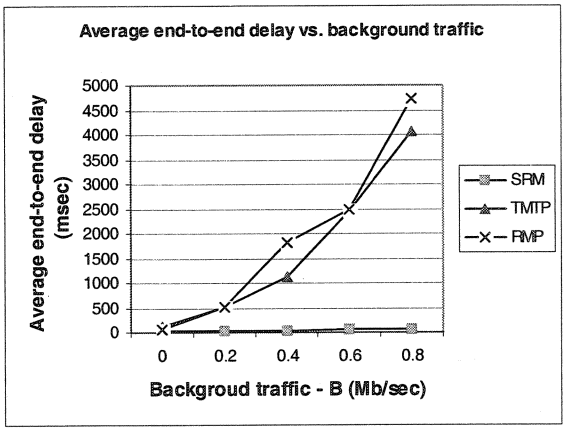
Column (I) - λ=4, R=250

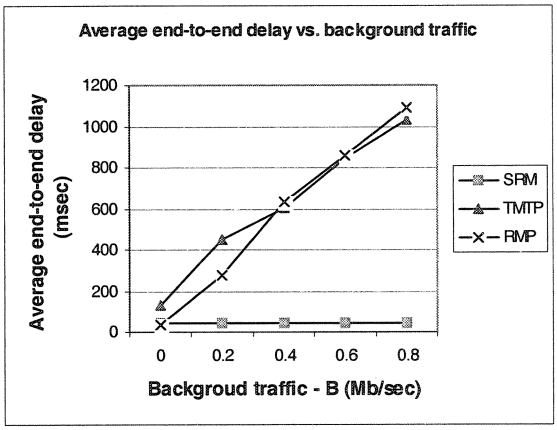


Column (II) - λ=2, R=250

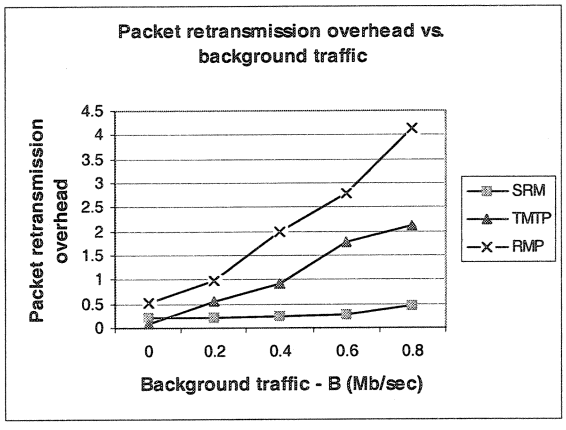


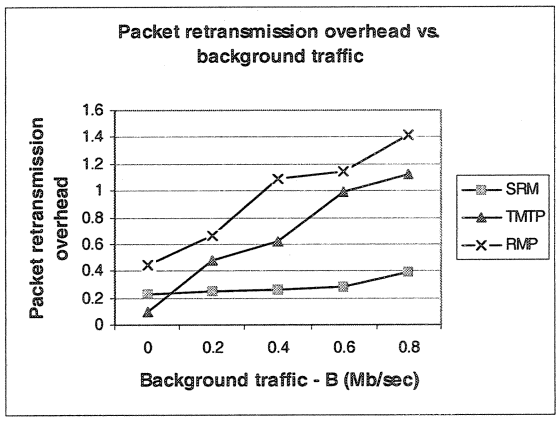
(A) Packet delivery ratio





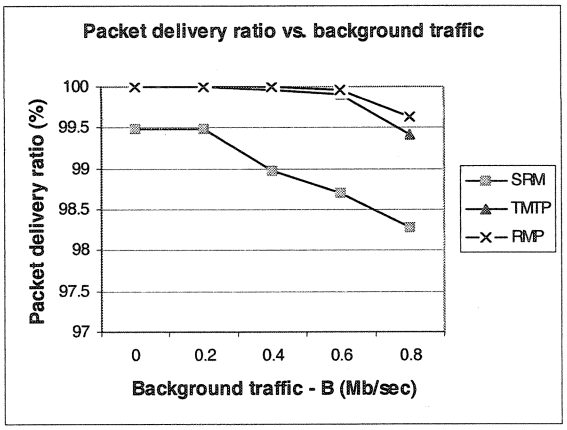
(B) Average end-to-end delay



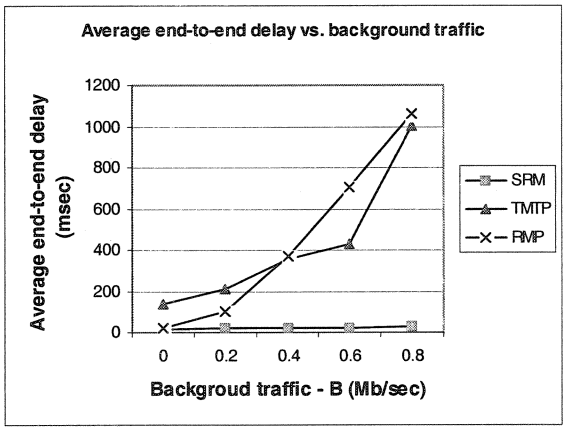


(C) Packet retransmission overhead

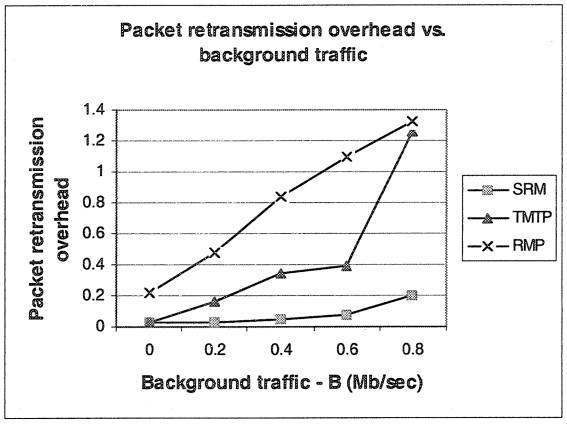
Column (III) - λ=4, **R=350**



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-6 Effect of background traffic

4.3.7 The Effect of Cluster of Nodes

The results of the experiments in this section show how the three protocols perform when receiver group is within close proximity (with no consideration for mobility).

4.3.7.1 Basic Cluster of Nodes Experiment

Figure 4-7 shows the performance of the three protocols with varying packet arrival rate when receivers are close to each other.

Parameter	Value
N	50
G	10
λ	1, 2, 3, 4, 5 Packets / Sec
R	250 Meters
B	0.5 Mb / Second
P	450 second

Table 4-21 Parameters for basic cluster of nodes experiment

We can see from Column I – chart A in Figure 4-7 that RMP and TMTP outperform SRM in terms of reliability. When a group of receivers are close to each other and packet arrival rate is not too high, current token site in RMP can successfully pass the token to the next token site and does not require reconstructing the token ring. Similarly, a child in TMTP has little difficulty reaching its parent and does not require frequent reconstructing the tree. Therefore, the amount of control packets is dramatically decreased and the network is not congested. This results in more packets successfully arriving at the destinations.

Column I – chart B of Figure 4-7 shows that the average end-to-end delay in TMTP is higher than that in RMP. The main reason for this behavior is that in RMP, a node that misses a packet first asks for retransmission from the current token site, which is

guaranteed to have a copy of the requested data packet. Another reason is that the ring is less frequently reconstructed, which results in less traffic in network, and a generally lower delay. On the other hand, a node in TMTP asks for retransmission from its parent, which may not have the requested data packet. Such a node would then have to make several requests until either the parent receives the packet or the node gives up the current parent after trying a number of times and ask all receivers for retransmission. This process is definitely time-consuming. This also explains why RMP and TMTP have close packet retransmission overheads as shown in Column I – chart C, which is different from pervious results in which RMP has more packet retransmission overhead than TMTP.

4.3.7.2 Cluster of Nodes Experiment with Lighter Background Traffic

Column II of Figure 4-7 shows the performance of three protocols with varying packet arrival rate with lighter background traffic.

Parameter	Value
N	50
G	10
λ	1, 2, 3, 4, 5 Packets / Sec
R	250 Meters
B	**0.2 Mb / Second**
P	450 second

Table 4-22 Parameters for cluster of nodes experiment with lighter background traffic

Charts A and B in Column II show similar trend as those in Column I except that all protocols have better performance due to lighter background traffic. Column II – chart C shows that RMP has less packet retransmission overhead than TMTP. We already discussed in previous section that RMP and TMTP have similar packet retransmission

overheads when B = 0.5. When B = 0.2, less ring reconstruction requests are issued and less traffic is generated. Hence, more packets can go through the network and less packet retransmissions are issued.

4.3.8 The Effect of Timer Settings

It was mentioned in section 4.2.5 that we implemented fixed timers for transmission of NACKs and repairs in TMTP and RMP. Timer values can have impact on TMTP and RMP performance. The results of the experiments in this section show how TMTP and RMP perform when applying different timer settings. Parameter setting of this experiment is shown in Table 4-23.

Parameter	Value
N	50
G	10
λ	1, 2, 3, 4, 5 Packets / Sec
R	250 Meters
B	0.2 Mb / Second
P	130 second

Table 4-23 Parameters for timer experiment

The setting of timers is listed in Table 4-24 and 4-25.

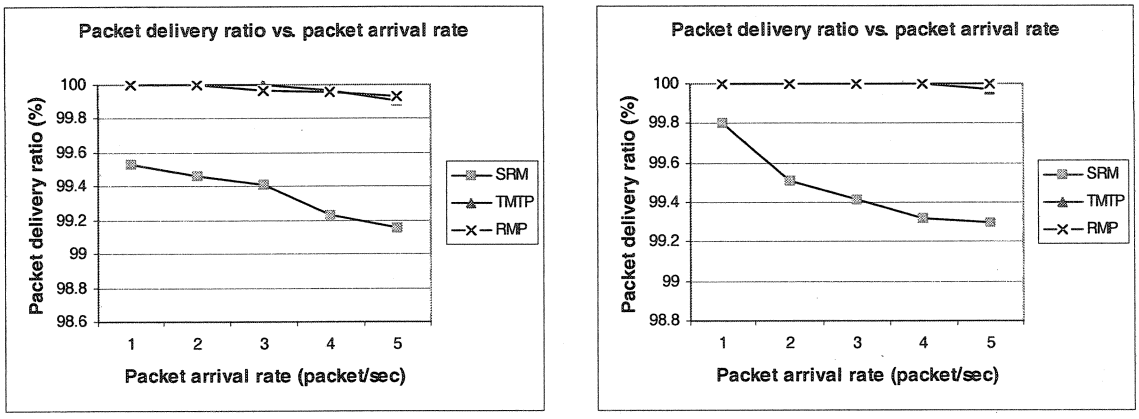
	Default Timer	Half Timer	Double Timer
Request Timer	1 second	0.5 second	2 seconds

Table 4-24 Timers for TMTP

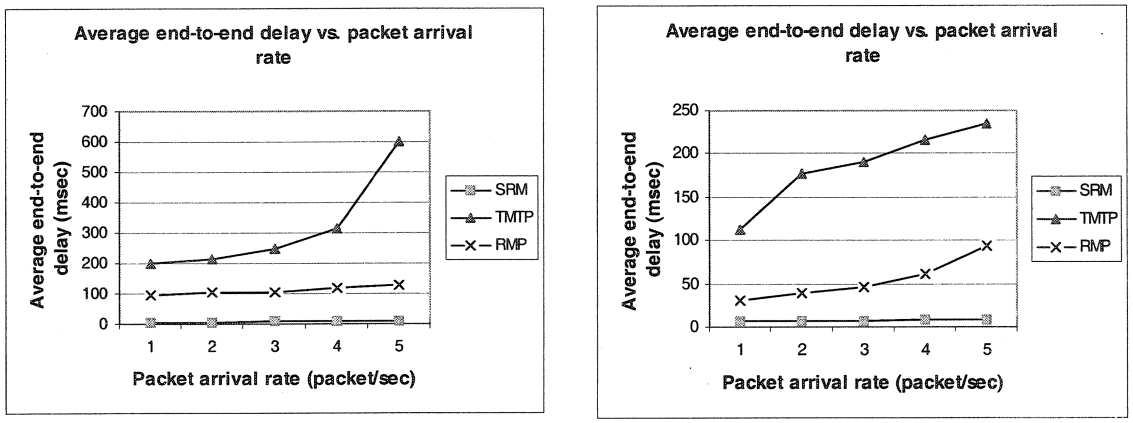
	Default Timer	Half Timer	Double Timer
Request Timer	1 second	0.5 second	2 seconds
ACK Timer	1 second	0.5 second	2 seconds
Transmission Timer at Sender	3 second	1.5 second	6 seconds

Table 4-25 Timers for RMP

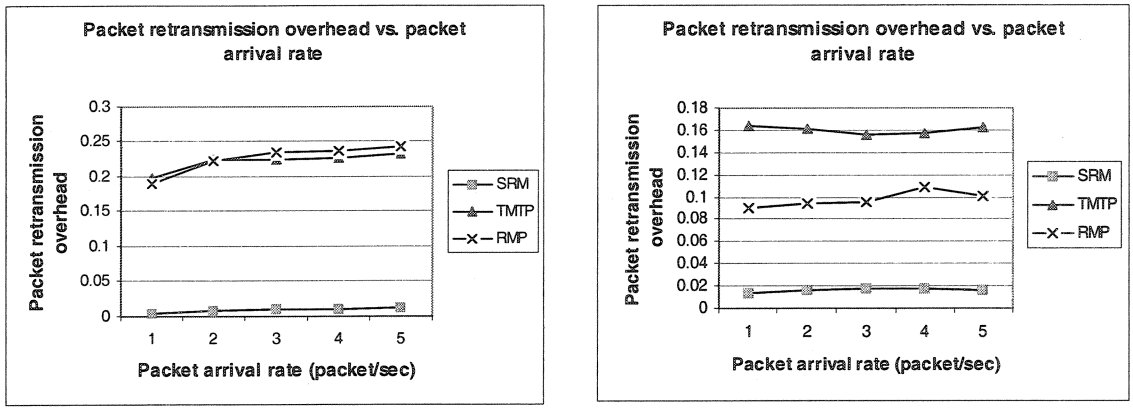
Column (I) – B=0.5 Column (II) – **B=0.2**



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-7 Effect of cluster of nodes

Figure 4-8 shows the performance of TMTP and RMP with varying packet arrival rate and different timer settings. When small request timer values are used in TMTP, retransmitted packets can be obtained quickly (as shown in Column I – chart B) at the expense of producing more retransmitted packets (as shown in Column I – chart C) and causing network congestion. Especially when λ is high, the frequent retransmission makes the already busy network more congested, which results in packet loss and packet delivery ratio being dramatically decreased, as shown in Column I – chart A. On the other hand, when large request timer values are selected, even though less control traffic is generated, nodes may not be able to quickly get the requested packet because the nodes have to wait a longer time to resend NACK, which translates into longer delay.

The setting of RMP request timer has the same impact as that of TMTP request timer. Here we only discuss ACK timer and transmission timer of RMP. Choosing small ACK timer values would lead to frequent ACK sent by current token site, which may cause network congestion. This leads to frequent reconstruction of ring when λ is high since the current token site will initiate ring reconstruction algorithm every time after it tries to send an ACK for five times. Therefore, compared to other timer settings, more packets have to be dropped and packet delivery ratio is much lower when λ is high as shown in Column II – chart A. On the other hand, when large ACK timer values are selected, even though less ACK will be generated, the current token site has to wait a longer time to reconstruct the ring if its next token site moves away. The current token site will then be the only node responsible for serving retransmission request during a considerably long

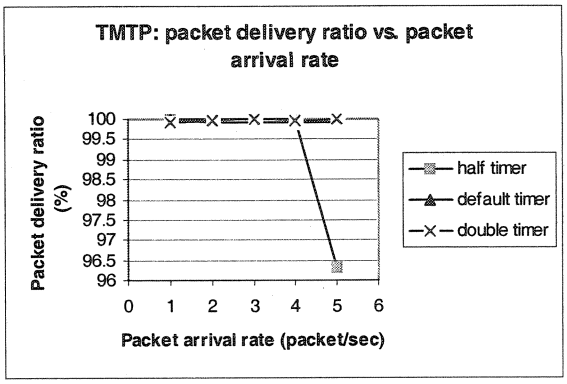
period of time, violating the design principles of RMP, i.e. each node taking turns to be current token site, and causing the current token site overwhelmed by NACKs.

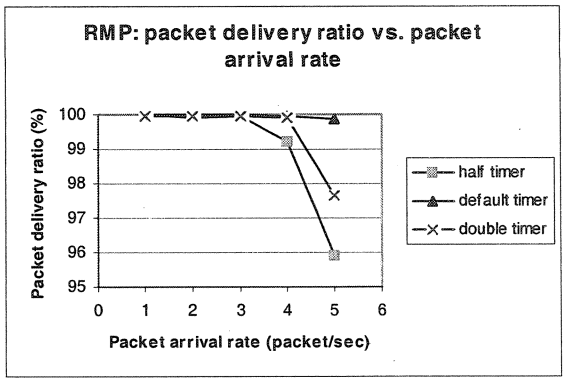
A source retransmits a packet if it does not hear the corresponding ACK with the transmission timer timeout. Choosing small transmission timer values would lead to frequent and unnecessary data packet retransmission sent by the source, which wastes network capacity and may cause network congestion. When λ is high, the frequent retransmission makes the already busy network more congested, which results in packet loss and packet delivery ratio sharp decrease as shown in Column II – chart A ($\lambda = 5$). On the other hand, when large transmission timer values are selected, even though less control traffic will be generated, nodes may not be able to quickly get the missed packet if the current token site also missed that packet.

Figure 4-9 shows the comparison of SRM, TMTP and RMP when half timer and double timer are applied in TMTP and RMP. Please refer to Column II of Figure 4-3 to see performance of SRM, TMTP and RMP when default timer is applied.

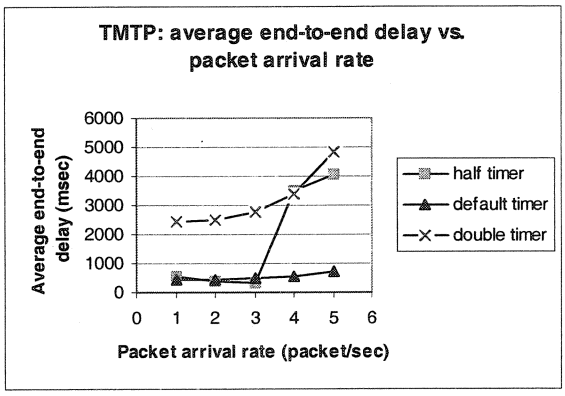
Column (I) - TMTP

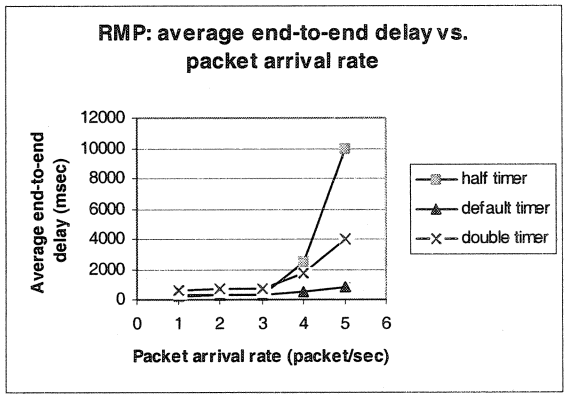
Column (II) - RMP



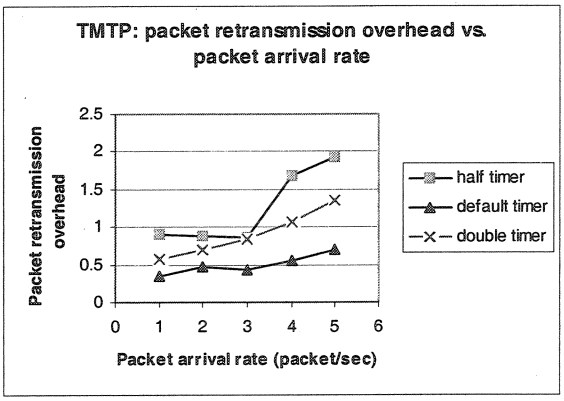


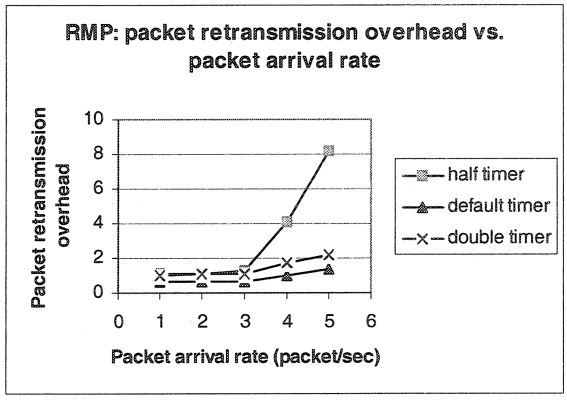
(A) Packet delivery ratio





(B) Average end-to-end delay



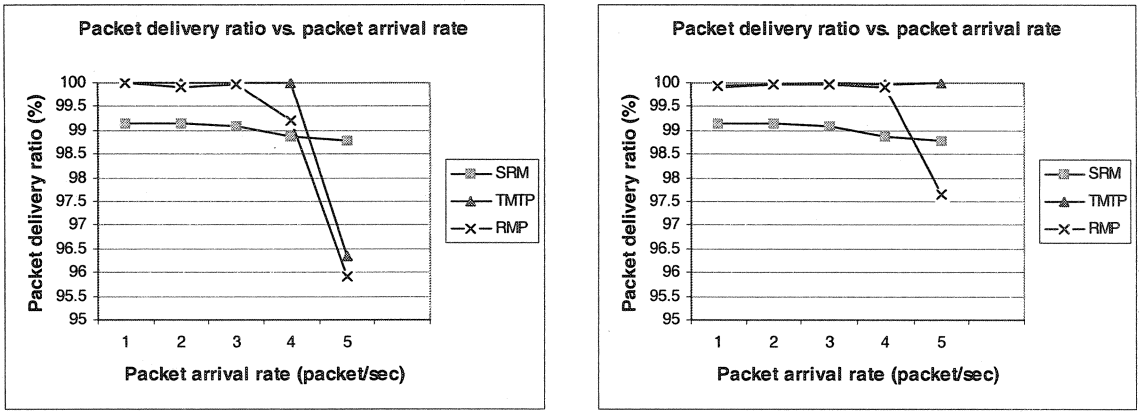


(C) Packet retransmission overhead

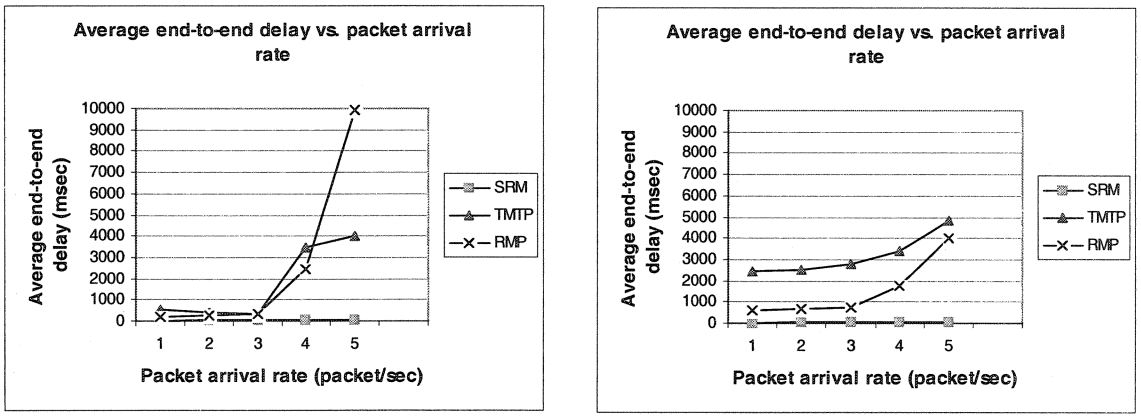
Figure 4-8 Effect of different timers on TMTP and RMP

Column (I) – Half timer

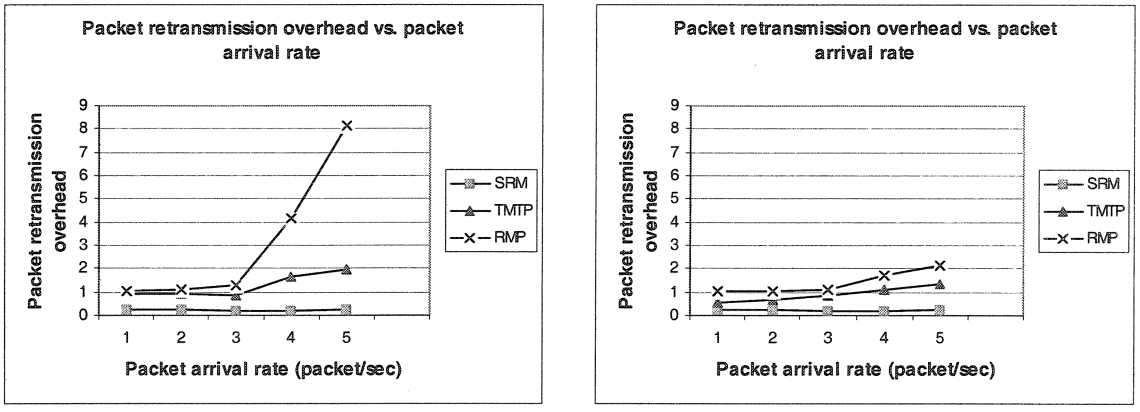
Column (II) – Double timer



(A) Packet delivery ratio



(B) Average end-to-end delay



(C) Packet retransmission overhead

Figure 4-9 Effect of half and double timers

4.4 Summary

In this chapter, we presented NS-2 simulator and our simulation model. We conducted a series of simulations to investigate how each parameter of the simulation model affects the performance of the reliable multicast protocols (SRM, TMTP and RMP). We then discussed the performance of SRM, TMTP and RMP over ad hoc networks based on simulation results. Table 4 - 26 shows a summary of the comparison of these three protocols.

	SRM (Receiver-initiated)	**TMTP (Tree-based)**	**RMP (Ring-based)**
Reliability	Low	Medium	High
End-to-end delay	Low	Medium	High
Retransmission overhead	Low	Medium	High
Scalability	High	High	Low
Power consumption	Low	Medium	High
Memory requirement	High	Medium	Low
Implementation Complexity	Low	Medium	High

Table 4 – 26 Comparison of SRM, TMTP and RMP

The conducted simulations showed that RMP outperformed TMTP and SRM in terms of reliability. This is because RMP uses ACK to at least ensure that the current token site actually receives the data packets, and that any receiver that misses a packet always requests retransmission from the current token site, which in turn is always guaranteed to have the requested data packets. If the source does not receive ACK for a packet from

the current token site due to packet loss or ACK loss, the source keeps retransmitting that packet for five times before it gives up. This retransmission increases the probability of receiving the missed packet and, therefore, the packet delivery ratio is high. However, the source retransmission also causes RMP to have the highest packet retransmission overhead in most cases (except in Column II – chart C of Figure 4-7), compared to SRM and TMTP. Another tradeoff in RMP to have high reliability is large end-to-end delay. In a harsh environment, e.g. high mobility, heavy traffic, etc, the ring reformation algorithm will have to be initiated several times to maintain the ring, which produces more traffic, causes network congestion and results in large packet end-to-end delay.

Results also showed that SRM has the smallest average end-to-end delay and the lowest packet retransmission overhead. Nevertheless, SRM has the lowest packet delivery ratio compared to TMTP and RMP. In SRM, any retransmission request is multicast to the entire group, and any group member who hears the retransmission request can respond with a repair. This shortens the end-to-end delay. Moreover, in the case where a number of nodes miss the same packet, only one repair is required to be multicast to the whole group. With Repair suppression scheme, the number of retransmission is significantly reduced. However, multicasting retransmission requests and repairs introduces significant overhead, which may cause network congestion and higher packet drop rate. Also, since ACK is not used in SRM, a packet may be permanently lost if no one received that packet. Therefore, the packet delivery ratio is low in SRM. Another drawback of multicasting retransmission requests and repairs is the additional overhead needed by the underlying multicast routing protocol, of which each transport layer

receiver must become a routing layer source and establish its own multicast source mesh. What is worse is the fact that if the retransmission requests / repairs do not occur frequently, the multicast routing protocol will time out the multicast mesh structure and, consequently, the mesh will have to be rediscovered – a process with high bandwidth consumption.

Simulation results also showed that TMTP did not achieve fast delivery and low packet retransmission overhead as it does in wired networks. Three main reasons contribute to this behavior:

- First, we used unicast NACKs and repairs in our simulation because the underlying ODRMP does not support restricted TTL multicast. A node that misses a packet, first requests packet retransmission from its parent. It may issue multiple NACKs if its parent is unreachable or the parent does not have the requested packet, thus delay can be large. Also, the parent has to retransmit the lost packet by using unicast for many times if many of its children lose the same packet. This causes added overheads and large delays.
- Second, the frequent reconstruction of tree structure causes a large amount of overheads in ad hoc network, and results in network congestion.
- Third, TMTP does not prevent a node from finding one of its hierarchical children as its new parent. One solution to this problem is that each node keeps a structure of the entire control tree and refers to that structure when searching for a new parent. However, this would require a more complicated tree maintenance algorithm that would require further traffic.

The fixed timer value scenario showed that it is significant to choose an optimal or near-optimal value for a timer in order for the protocols to achieve a high performance. For instance, in RMP, if the value for the transmission timer is too small, there would be many unnecessary retransmissions. If the value is too large, the protocol would be sluggish in responding to packet loss. In general, it is not suitable for ad hoc networks to use a fixed timer value to respond to dynamically changing network conditions.

5 CONCLUSIONS & FUTURE WORK

In recent years, reliable multicast in wired networks has been a very active area of research. Many reliable multicast protocols have been proposed for wired networks. Meanwhile, equal interest has been made in providing reliable multicast in mobile ad hoc networks. In order to obtain a unified solution that provides reliable multicast in a seamlessly integrated wired network and wireless ad hoc network, we need to investigate the efficiency of existing wired reliable multicast protocols in ad hoc networks. Hence we need to extend and enhance the current protocols to address the challenges in ad hoc networks, such as high nodal mobility and network congestion so that they can work well in integrated networks. In this thesis, we described four classes of wired reliable multicast protocols and evaluated the performance of three representative protocols, namely SRM, TMTP and RMP, in ad hoc network environment. We also discussed the simulation models and compared the performance of the three protocols under a variety of effects such as mobility, traffic load and multicast group size.

The performance metrics that we used were average packet delivery ratio, average end-to-end delay, and packet retransmission overhead. Each of the protocols studied performs well in some cases yet has certain drawbacks in others. RMP achieved highest

packet delivery ratio (close to 100%) but largest end-to-end delay and heaviest packet retransmission overhead due to sender continuous retransmission and ring maintenance procedures. In a harsh environment, e.g. a high node mobility rate or a high packet arrival rate, network will be congested by tremendous overhead caused by retransmission and ring maintenance. Consequently, additional packets are lost and performance of RMP is degraded. On the other hand, SRM performed the worst in terms of packet delivery ratio. It has the lowest average end-to-end delay and packet retransmission overhead due to multicasting retransmission requests and repairs with NACK / repair suppression scheme. Finally, simulations showed that some of the features of TMTP that makes it attractive in the wired domain, such as hierarchical structures for reduced end-to-end delay and scalability, were not as advantageous in the wireless domain.

Mobile ad hoc networks have highly dynamic topology and are very limited in bandwidth, power and processing capability. These characteristics pose several challenges in tailoring reliable multicast protocols for mobile ad hoc networks. The simulation results showed that, as we expected, the wired reliable multicast protocols are not adequate for ad hoc networks. In designing reliable multicast protocols for ad hoc network, the outcome protocols should produce minimum network overhead, thus preserving the limited network bandwidth and nodal processing capability. The outcome protocols should also maintain end-to-end delay within a satisfactory and guaranteed bound, in addition to providing reliable delivery.

This thesis identifies the relative merits of different wired reliable multicast protocols. The in-depth understanding of the relative merits and detailed performance comparison between the protocols over ad hoc networks serve as a cornerstone for development of more effective reliable multicast protocols for mobile ad hoc networks. To improve the performance of the three wired reliable multicast protocols in ad hoc networks, we recommend following enhancements and describe the open problems that should be addressed:

- **RMP**: Three timers (ACK timer, transmission timer, and request timer) are used in RMP. Choosing appropriate values for timers is significant in achieving high performance. The exact manner in which these values are chosen is viable to further research.

  As we mentioned in chapter 2 that even in wired networks, RMP only guarantees that all receivers "eventually" receive packets, but does not guarantee end-to-end delay. The work in [23] proposed a modified RMP that has lower delay than RMP. We can experiment the new scheme in ad hoc networks.

- **SRM**: Global multicasting retransmission requests and repairs may not be the best approach when a small neighborhood is affected by packet loss. The bandwidth cost incurred by loss recovery procedures is reduced if requests and repairs are multicast within a limited area. Therefore, in the future we need to develop a mechanism that can limit the scope of retransmission requests and of repairs. TMTP proposed using restricted TTL to limit the reach of retransmission requests and repairs. As we

pointed out earlier, since the underlying multicast routing protocol (ODMRP) does not support this restricted TTL, we need to investigate how to modify ODMRP to provide this feature. Another possible approach to restrict the multicast scope is to use separate local multicast groups for error recovery. A local multicast group consists of a subset of members who are close to each other. If a packet loss is detected, a member sends its request to its local group. If there is no reply after trying for several times, the request is multicast to all members.

- **TMTP**: Similar to SRM, we should use restricted multicast NACK / repair in TMTP as it was originally designed for, with ODRMP supporting local multicast. By multicasting NACK messages, a node losing a packet can get a repair not only from its parent, but also from any node near by. Ideally, receivers that have the same loss can get repairs by a single multicast repair. In this manner end-to-end delay and packet retransmission overhead can be significantly reduced.

Another problem to be solved in TMTP is to prevent a node from finding one of its hierarchical children as its new parent during the tree reconstruction procedure. One solution to this problem is to require each node to keep a structure of the entire control tree that would be referred to when searching for a new parent. However, this is not a practical solution since the significant overhead caused by refreshing the control tree structure can overwhelm mobile devices equipped with limited memory and processing capability. Hence, a more efficient solution is required.

- **Fixed or dynamic timers**: There are two ways to set timer values. First, a fixed timer value can be used depending on the network's typical behavior. We experimented with fixed timer values for TMTP and RMP. One limitation of fixed timer values is that they suffer from an inability to adequately respond to quickly changing network conditions, especially for ad hoc networks that are highly dynamic and unpredictable.

  Timer values can also be dynamic, which is how timers are set in SRM. A dynamic timer solves the fixed timer's problem – it responds to varying network conditions. Nevertheless, it introduces another set of drawbacks. To illustrate this, suppose that in RMP the source keeps track of the time taken to receive ACK packets and sets its transmission timer based on the average of the observed delays. This value may not reflect the true network condition for two reasons: (1) The current token site may not be able to acknowledge a packet immediately. (2) Network conditions may suddenly change. The consequence of using this timer value is that it can cause the system incorrectly respond to the network condition change, leading to more unpredicted performance drop. Therefore, a better mechanism is desired to provide more accurate timer value.

- **Congestion control**: A congestion control mechanism reacts to network traffic load and avoids having nodes overwhelmed by traffic. The performance of Ad hoc networks is sensitive to traffic load, and reliable delivery can be failed by network congestion. It is imperative for ad hoc network that reliable multicast protocols

incorporate congestion control mechanism for achieving reliability. However, among the currently proposed reliable multicast protocols for wired networks, only few have considered congestion control due to the complexity. It is hence desired to propose a congestion control mechanism that is tailored for reliable multicast protocols.

- **Combination of different protocols**: Receiver-initiated, tree-based and ring-based reliable multicast protocols have their advantages and limitations. We can combine these protocols into a new protocol so that the new protocol can absorb all the advantages. For instance, we can combine a ring-based protocol with a tree-based one. By using a clustering technique [47], we organize multicast group members into subgroups (i.e. cluster). Clustering is a known technique in the area of distributed network computing. Each cluster is represented by a leader, which acts as a token site. Among cluster leaders the RMP-like protocol is used. Inside of clusters, we can apply a tree-based protocol. The purpose of this combination is to combine the scalability of tree-based protocol and the reliability of ring-based protocol.

REFERENCES

[1] C. Jelger and T. Noel, "Multicast for Mobile Hosts in IP Networks: Progress and Challenges", *IEEE Wireless Communications*, Vol. 9, No. 5, October 2002, pp. 58 - 64.

[2] C. D. M. Cordeiro, D. H. Sadok, and J. Kelner, "Establishing a Trade-off Between Unicast and Multicast Retransmission Modes for Reliable Multicast Protocols", *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Proceedings of 8th International Symposium*, September 2000, pp. 85 – 91.

[3] S. Corson and J. Macker, "Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations", *Mobile Ad-hoc Networks Working Group,* http://www.ietf.org/rfc/rfc2501.txt, January 1999.

[4] I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges", *Ad Hoc Network Journal*, Vol. 1, No. 1, January 2003, pp. 13 – 64.

[5] F. Y. Loo, "Ad Hoc Networks: Prospects and Challenges", *Rinkou Paper*, http://www.mlab.t.u-tokyo.ac.jp/~ylfoo/Research/MANET_Rinkou.pdf, January 2004.

[6] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu, and J. Jetcheva, "A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols", *Proceedings of the ACM MOBICOM'98*, October 1998, pp. 85 - 97.

[7] ns-2 network simulator: http://www.isi.edu/nsnam/ns

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking,* Vol. 5, No. 6, December 1997, pp. 784 – 803.

[9] B. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications", *Proceedings of the third ACM international conference on Multimedia,* November 1995, pp. 333 – 344.

[10] B. Whetten, T. Montgomery, and S. Kaplan, "A High Performance Totally Ordered Multicast Protocol", *Workshop on Theory and Practice in Distributed Systems,* September 1994, pp. 33-57.

[11] L. D. Fife and L. Gruenwald, "Research Issues for Data Communication in Mobiel Ad-Hoc Network Database Systems", *ACM SIGMOD RECORD,* Vol. 32, No. 2, June 2003, pp. 42 - 47.

[12] T. You and H. Hassanein, "Controllable Fair QoS-based MAC Protocols for Ad Hoc Wireless Networks", *Proceedings of the 2004 International Conference on Parallel Processing Workshops (ICPPW'04), IEEE Computer Society*, August 2004, pp. 21 – 28.

[13] *IEEE 802.11 WG*, "ISO/IEC 8802-11:1999 (E) IEEE STD 802.11, 1999 Edition. International Standard [for] Information Technology-Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", 1999

[14] E. M. Royer, S. Barbara and C. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", *IEEE Personal Communications,* April 1999, pp. 46 - 55.

[15] C. E. Perkins, E. M. Royer, S. R. Das, and M. K. Marina, "Performance Comparison of Two On-demand Routing Protocols for Ad Hoc Networks", *IEEE Personal Communications*, Vol. 8, No. 1, February 2001, pp. 16 - 28.

[16] D. B. Johnson and D. A. Maltz, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks" *IETF Draft*, October 1999.

[17] C. E. Perkins and E. M. Royer, "Ad Hoc On-demand Distance Vector Routing", *proceedings of 2nd IEEE Workshop, Mobile Computing Systems and Applications*, February 1999, pp. 90 - 100.

[18] P. Misra, "Routing Protocols for Ad Hoc Mobile Wireless Networks", http://www.cis.ohio-state.edu/~jain/cis788-99/adhoc_routing/index.html, 1999

[19] S. E. Deering and D. R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs", *ACM Transactions on Computer Systems,* Vol. 8, No. 2, May 1990, pp. 85 - 110.

[20] E. Bommaiah, M. Liu, A. McAuley, and R. Talpade, "AMRoute: Ad-hoc Multicast Routing Protocol", Internet Draft, draft-talpade-manet-amroute-00.txt, August 1998.

[21] S. Lee, W. Su, and M. Geria, "On-Demand Multicast Routing Protocol in Multihop Wireless Mobile Networks", *Mobile Networks and Application 7*, 2002, pp. 441-453.

[22] S. Lee, W. Su, J. Hsu, M. Geria, and R. Bagrodia, "A performance Comparison Study of Ad Hoc Wireless Multicast Protocols", *IEEE INFOCOM 2000*, Vol. 2, March 2000, pp. 565 – 574.

[23] C. Cordeiro, H. Gossain, and D. Agrawal, "Multicast over Wireless Mobile Ad Hoc Networks: Present and Future Directions", *IEEE Network,* Vol. 17, No. 1, January 2003, pp. 52 – 59.

[24] S. Deering, "Host Extensions for IP Multicasting", RFC 1112, http://www.faqs.org/rfcs/rfc1112.html, August 1989.

[25] M. D. Rey, "Transmission Control Protocol Darpa Internet Program Protocol Specification", RFC 793, http://www.faqs.org/rfcs/rfc793.html, September 1981.

[26] R. Talpade amd M. H. Ammar, "Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service", *Proceedings of the 15th IEEE International Conference on Distributed Computing Systems*, June 1995, pp. 144 – 151.

[27] S. Ramakrishnan and B. Jain, "A Negative Acknowledgement Protocol with Periodic Polling Protocol for Multicast over LANs", *IEEE INFOCOM 87*, March/April 1987, pp. 502-511.

[28] B. N. Levine and J. J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols", *Multimedia Systems 6,* 1998, pp. 334–348.

[29] J. M. Chang, "Simplifying Distributed Database Systems Design by Using a Broadcast Network", *Proceedings of SIGMOD'84*, June 1984, pp. 223 – 233.

[30] M. Liu, "Chapter 6. Group Communication", *Distributed Computing*, http://www.aw-bc.com/info/liu/CH06.pdf, pp. 1 - 18.

[31] R. G. Lane, S. Daniels, and X. Yuan, "An Empirical Study of Reliable Multicast Protocols over Ethernet-Connected Networks", *IEEE International Conference on Parallel Processing (ICPP'01)*, September 2001, pp. 553 – 560.

[32] J. P. Macker, J. E. Klinker, and M. Scott Corson, "Reliable Multicast Data Delivery for Military Networking", *IEEE Military Communications Conference*, Vol. 2, October 1996, pp. 399 – 403.

[33] D. Towsley, J. Kurose, and S. Pingali, "A Comparison of Send-Initiated and Receiver-Initiated Reliable Multicast Protocols", *IEEE Journal on Selected Areas in Communications,* Vol. 15, No. 3, April 1997, pp. 398 – 406.

[34] R. Chow and T. Johnson, "Distributed Operating Systems & Algorithms", *Addison Wesley,* October 1998, pp.108 – 109.

[35] J. M. Chang and N. F. Maxemchuk, "Reliable Broadcast Protocols", *ACM Transactions on Computer Systems,* vol. 2, No. 3, August 1984, pp. 251—273.

[36] N. F. Maxemchuk, "Reliable Multicast with Delay Guarantees", *IEEE Communications Magazine*, vol. 40, No. 9, September 2002, pp. 96 – 102.

[37] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya, "Reliable Multicast Transport Protocol (RMTP)", *IEEE Journal on Selected Areas in Communications,* Vol. 15, No. 3, April 1997, pp. 407 – 421.

[38] B. Levine, D. Lavo, and J. Garcia-Luna-Aceves, "The Case for Reliable Concurrent Multicasting Using Shared Ack Trees", *Proceedings of the Fourth ACM Multimedia Conference (MULTIMEDIA'96)*, November 1996, pp. 365-376.

[39] C. Maihofer, K. Rothermel, and N. Mantei, "A Throughput Analysis of Reliable Multicast Transport Protocols", http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/2000-maihoefer-03.pdf, 2000

[40] R. Chandra, V. Ramasubramanian, and K. P. Birman, "Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks", *Proceedings of IEEE ICDCS 2001,* April 2001, pp. 275-283.

[41] K. Tang, K. Obraczka, S-J Lee, and M. Gerla, "A Reliable, Congestion Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks", *Proceedings of IEEE WPMC 2002,* October 2002.

[42] D. Clark and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proceedings of ACM SIGCOMM*, September 1990, pp. 201 – 208.

[43] N. F. Maxemchuk and J. M. Chang, "Analysis of the Messages Transmitted in a Broadcast Protocol", *Proceedings of ICC'84*, May 1984, pp. 1263 - 1267.

[44] K. Fall and K. Varadhan, "The ns Manual", December 2003

[45] MIT Object Tcl web site: ftp://ftp.tns.lcs.mit.edu/pub/otcl/README.html

[46] CMU Monarch Project Extensions to NS-2. http://www.monarch.cs.rice.edu/multicast_extensions.html, 2000

[47] S. Basagni, "Distributed Clustering for Ad Hoc Networks", *Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN'99), IEEE Computer Society*, June 1999, pp. 310 – 315.

APPENDIX A – SRM ALGORITHM

This appendix lists algorithms in protocol SRM.

SendMsg()

The algorithm for sending data packets is shown in Figure A-1. The procedure is as follows: For each packet, append header of transport layer protocol (i.e. SRM in this case) to network layer header (line 2). The SRM packet header includes type_ (types of the packet), sender_ (sender of the packet), and seqnum_ (sequence number of the packet). The destination of packet is set as the multicast group (line 3). Each message is then multicast to the entire group (line 4). Source sets a timer each time when it starts sending a message (line 5), and transmits next message if timer is expired (line 7).

```
SendMsg()

SOURCE:
Start {
    1.  For each data packet P {
    2.          Add transport layer protocol's header;
    3.          Set destination;
    4.          Multicast P to all members;
    5.          Start timer;
    6.          If (timeout) {
    7.                  Multicast next message;
    8.          }
    9.  }
} End
```

Figure A-1 Algorithm for source sending data packets

Recv()

The Recv() procedure, shown in Figure A-2, is called when a node receives a packet. Recv() can receive four types of packets: data, control, NACK, and repair packets.

Depending on the packet types, different methods are invoked (line 2, 4, 6, 8). Packets will be freed after it is finished processing (line 10).

```
Recv(packet)

Start {
    1.  If (packet type is "DATA") {
    2.          recv_data (packet);
    3.  } Else if (packet type is "NACK") {
    4.          recv_nack (packet);
    5.  } Else if (packet type is "REPAIR") {
    6.          recv_repair (packet);
    7.  } Else if (packet type is "CONTROL") {
    8.          recv_ctrl (packet);
    9.  }
    10.  Free (packet)
} End
```

Figure A-2 Algorithm for receiving packets

Recv_data()

On receiving a data type packet, Recv_data (packet), shown in Figure A-3, marks the arrival of this packet (line 1), and triggers retransmission requests if it detects packet loss. If the sequence number of the packet is higher than the last known sequence number of data from this source (line 2), the node then checks if it receives all packets whose sequence number is up to the current one (line 3). The node multicasts a NACK to request retransmission if it finds packet losses (line 5).

```
Recv_data (packet)

Start {
    1.  Mark the arrival of this packet;
    2.  If (receiving packet's sequence number > last known sequence number of data) {
    3.          For (each packet between last and current receiving) {
    4.                  If (I didn't receive packet) {
    5.                          Send NACK to request retransmission;
    6.                          Set last known sequence number to the receiving packet's
                                sequence number;
    7.                  }
    8.          }
    9.  }
} End
```

Figure A-3 Algorithm for receiving data packet

Send_nack()

A node invokes Send_nack(), shown in Figure A-4, to set a timer and schedule sending NACK. Timer value is randomly selected from interval $[C_1 \cdot d_s, (C_1 + C_2) \cdot d_s]$ (line 1), where C_1 and C_2 are repair request timer parameters (C_1 and C_2 are 2 in our simulation). d_s is the estimated one-way delay between the node who sends NACK and the source of packet. Backoff_ is computed by calling Backoff() shown in Figure A-5. The time to send NACK is determined by product of Delay and backoff_ (line 3). The node multicasts NACK to all group members after the repair request timer expires (line 4).

```
Send_nack(packet)

Start {
    1.  Delay = [C1ds, (C1 + C2)ds];
    2.  backoff_ = Backoff();
    3.  timer to send NACK = Delay * backoff;
    4.  Construct NACK and Multicast it to all group members after timer expires;
} End
```

Figure A-4 Algorithm for sending NACK

Backoff()

Figure A-5 shows the algorithm of exponential backoff. A node needs to exponentially backoff if it receives a NACK for the same missing data packet from before its own repair request timer for that data packet expires. Initially, number of backoffs is set to 0, backoff_ is set to 1, and backoff limit is set to 5. When Backoff() is invoked, the number of backoffs is incremented by 1 (line 1). If the node hasn't backoff for maximum number of times (line 2), increment the value of backoff_ by backoff_, i.e. the value of backoff_ is like 1, 2, 4, 8, 16, etc.

```
Backoff()

Start {
    1.  Increment number of backoffs by 1;
    2.  If (backoff counter <= backoff limit) {
    3.          Increment backoff_ by backoff_;
    4.  }
End
```

Figure A-5 Algorithm for exponential backoff

Recv_nack()

When receiving a retransmission request (NACK) for a particular packet (Figure A-6), a node can operate in one of three states. (1) The node has the data, and has seen an earlier NACK, upon which it has scheduled to send a repair for it. (2) The node has the data and has not received the same NACK before. (3) The node lost the same packet and has a send_nack event scheduled. In case (1), the node considers the NACK as a duplicate and ignores it (line 3). In case (2), the node instantiated a repair and schedules to send it (line

5). In case (3), in order to avoid sending duplicated NACKs, the node suppresses its own NACKS by canceling send_nack event and re-schedules, after having backed off its timer (line 8).

```
Recv_nack (packet)

Start {
    1.  If (I have the requested data packet) {
    2.          If (I have received same NACK within some period of time) {
    3.                  Duplicate NACK, ignore this NACK;
    4.          } Else {
    5.                  Schedule to send a repair;
    6.          }
    7.  } Else {
    8.          Cancel my own send NACK event and re-schedule another one;
    9.  }
} End
```

Figure A-6 Algorithm for receiving NACK

Send_repair()

A node invokes Send_repair() to schedule a timer for sending a repair (Figure A-7). The node sets a repair timer to a value that is randomly selected from interval $[D_1 \cdot d_A, (D_1 + D_2) \cdot d_A]$ (line 1). D_1 and D_2 are repair timer parameters (D_1 and D_2 are 1 in our simulation). d_A is the estimated one-way delay between the node who has the requested data packet and the requestor of missing data. If the node receives the same repair for the missing data before its repair timer expires, it cancels its repair timer (line 3). Otherwise, the node multicasts the repair to all group members after its repair timer expires (line 5).

```
Send_repair(packet)

Start {
    1.  Repair timer = [D1dA, (D1 + D2)dA];
    2.  If (I receive the same repair from other nodes before repair timer expires) {
    3.          Cancel repair timer;
    4.  } Else {
    5.          Multicast repair after the repair timer expires;
    6.  }
} End
```

Figure A-7 Algorithm for sending repair

recv_repair()

recv_repair() is invoked when a node receives a repair (Figure A-8). Like recv_data(), recv_repair () marks the arrival of the retransmitted packet (line 1). When the requesting node receives the repair (line 5), it can be operating in one of two states. (1) The node can be waiting for a repair. (2) The node has already received a repair. In case (1), the node cancels the timer for re-sending NACKs since it has received the repair (line 3). In case (2), the node will ignore the duplicated repair (line 7). To avoid repair-implosion, other nodes that have the request data and schedule sending repairs cancel their repair timers when they see the repair (line 5).

Send_ctrl()

When a member sends a session message, it schedules to send the next in a fixed interval (1 second in our simulation). Send_ctrl is shown in Figure A-9.

```
Recv_repair(packet)

Start {
    1.  Mark the arrival of this message;
    2.  If (I have sent NACK and waiting for this repair) {
    3.          Stop sending NACK;
    4.  } Else if (I have the data and scheduled to send repairs) {
    5.          Cancel my repair timers;
    6.  } Else {
    7.          Duplicate repair, ignore it;
    8.  }
} End
```

Figure A-8 Algorithm for receiving repair

```
Send_ctrl()

Start {
    1.  Every 1 second {
    2.          Multicast session packet to all group members;
    3.  }
} End
```

Figure A-9 Algorithm for sending control message

Recv_ctrl()

As shown in Figure A-10, when receiving a session control message a node updates its last known sequence number for the source (line 2), and computes its instantaneous distance to the sender of session message (line 4).

```
Recv_ctrl(Packet)

Start {
    1.  If (last known sequence number < total number of packets that source sent) {
    2.          Last known sequence number = total number of packets that source sent;
    3.  }
    4.  Distance to the sender of session message = round trip delay of the session message
        / 2;
```

Figure A-10 Algorithm for receiving control message

APPENDIX B – TMTP ALOGRITHM

This appendix lists algorithms in protocol TMTP.

JoinTree()

Figure B-1 outlines the process of joining a control tree. A new node begins searching for a parent by multicasting a SEARCH_FOR_PARENT message (line 3). If the node does not receive a response within timeout, it resends the SEARCH_FOR_PARENT message. The node repeats this process until it receives a WILLING_TO_BE_PARENT message from one or more existing nodes in the control tree (line 6). All existing nodes will respond with a WILLING_TO_BE_PARENT message if they have not already had the maximum number of children (line 17 - 18). Upon receiving WILLING_TO_BE_PARENT message, the node sends a JOIN_REQUEST message to its potential parent, i.e. the sender of WILLING_TO_BE_PARENT (line 7). The potential parent then sends ACCEPT_JOIN message back to the node to confirm the joining success (line 21). When the node receives ACCEPT_JOIN, it sets the sender of ACCEPT_JOIN as its parent and stop looking for a parent (line 9 – 11).

LeaveTree()

Figure B-2 shows the algorithm for leaving a tree. The operation for leaf nodes in the tree to leave is straightforward (line 2 – 3). On the other hand, the operation for internal nodes is complicated by the fact that internal nodes are a crucial link in a control tree and breaking those links cause all the relative nodes to re-join in a tree. When an internal node departs a control tree, it first notifies its children of its departure and asks them to find new parents (line 5). The children then find their new parents by using the process

of joining a tree (line 15). When all the children find their new parents successfully (line 16), the internal node now can leave the tree (line 6 – 8).

```
JoinTree()

NEW NODE:
Start {
    1.  FoundParent = False;
    2.  While (FoundParent == False) {
    3.          Multicast a SEARCH_FOR_PARENT message;
    4.          Start timer;
    5.          Wait for WILLING_TO_BE_PARENT responses or timer expires;
    6.          If (receive WILLING_TO_BE_PARENT within timeout) {
    7.            Send JOIN_REQUEST to parent;
    8.            Wait for ACCEPT_JOIN reply;
    9.            If (ACCEPT_JOIN received) {
    10.                 Set sender of ACCEPT_JOIN as my parent;
    11.                 FoundParent = True;
    12.           } Else      /* try again */
    13.         } Else   /* try again */
    14. }
} End

EXISTING NODES:
Start {
    15. Receive request message;
    16. If (request is SEARCH_FOR_PARENT) {
    17.         If (MAX_CHILDREN not exceeded) {
    18.           Send WILLING_TO_BE_PARENT message;
    19.         } Else    /* Do not respond */
    20. } Else if (request is JOIN_REQUEST) {
    21.         Send ACCEPT_JOIN message;
    22. }
} End
```

Figure B-1 Algorithm for joining a control tree

LeaveTree()

```
NODE TO BE LEAVING
Start {
    1.  If (I am a leaf node) {
    2.          Send LEAVE_TREE request to parent;
    3.          Receive LEAVE_CONFIRM, and terminate;
    4.  } Else {   /* I am an internal manager */
    5.          Send FIND_NEW_PARENT message to children;
    6.          Receive NEW_PARENT_FOUND reply from all children;
    7.          Send LEAVE_TREE request to parent;
    8.          Receive LEAVE_CONFIRM, and terminate;
    9.  }
} End

LEAVING NODE'S PARENT
Start {
    10. If (receive LEAVE_TREE) {
    11.         Remove node from control tree;
    12.         Send LEAVE_CONFIRM;
    13. }
} End

INTERNAL NODE'S CHILDREN
Start {
    14. If (receive FIND_NEW_PARENT) {
    15.         Restart the JoinTree procedure;
    16.         When finding new parent, send NEW_PARENT_FOUND;
    17. }
} End
```

Figure B-2 Algorithm for leaving tree

SendMsg()

The algorithm for sending data packets of TMTP is very similar to that of SRM as shown in Figure A-1, except appending TMTP header to network layer packet header on line 2.

Recv()

The algorithm for receiving packets of TMTP is same as that of SRM as shown in Figure A-2.

Recv_data()

The algorithm for receiving data packets of TMTP is same as that of SRM as shown in Figure A-3.

Send_nack()

When a receiver detects a packet loss, it repeats requesting for the packet until it receives it or it has tried a number of times. The receiver sets a timer each time it sends a NACK (line 2 and line 11), and retries the operation if the timer expires (line 13). The node needs to exponentially backoff like SRM does (Figure A-5) each time when it retries the operation (line 11). It keeps on doing this until it receives the correct repair (line 8). If the receiver has tried to send a NACK more than a set limit of tries but still could not get repair (line 16), it will multicast NACK to entire group (line 17) and find another node as parent (line 18), i.e. restart the JoinTree procedure as shown in Figure B-1.

```
Send_nack(packet, parent)

Start {
    1.  Construct and send a unicast NACK to parent;
    2.  Start timer;
    3.  Counter = 0;
    4.  Stop_sending = False;
    5.  Do {
    6.          Wait until a repair is received or timer expires;
    7.          If (receive repair within timeout) {
    8.                  Mark the arrival of this repair packet;
    9.                  Stop_sending = True;
    10.         } Else if (timeout) {
    11.                 Exponentially backoff and reset timer;
    12.                 Counter = Counter + 1;
    13.                 Resend NACK to parent;
    14.         }
    15. } While (Counter < threshold and stop_sending == False)
    16. If (Counter >= threshold) {
    17.         Multicast NACK to entire group;
    18.         Find another parent;
    19. }
} End
```

Figure B-3 Algorithm for sending NACK

Recv_nack()

The algorithm for receiving NACK packets of TMTP is same as that of SRM as shown in Figure A-6.

Send_repair()

When a node sends a retransmission request (NACK) using unicast (line 1), the parent then sends a repair back by unicast (line 2). If the parent is unreachable, then any node that hears the NACK can respond with the repair like SRM does (line 4).

```
Send_repair(packet, unicast_nack, requestor)

Start {
    1.  If (unicast_nack == True) {
    2.          Unicast repair to requestor;
    3.  } Else {
    4.          Multicast repair to requestor;
    5.  }
} End
```

Figure B-4 Algorithm for sending repair

recv_repair()

The algorithm for receiving retransmission packets of TMTP is same as that of SRM as shown in Figure A-8.

Send_ctrl(), Recv_ctrl()

Control packets that are used in JoinTree() and LeaveTree() are handled by routines Send_ctrl() and Recv_ctrl(). These two algorithms have been discussed in Figure B-1 and Figure B-2.

APPENDIX C – RMP ALOGRITHM

This appendix lists algorithms in protocol RMP.

Join_ring()

A new node calls join_ring() to add itself in a ring (Figure C-1). If the new node is the only member in the group it creates a new ring (line 2). Otherwise, it sends a multicast "JoinRequest" message to the entire ring informing the existing ring members that it wishes to join in (line 4). When the current token site gets this message, it adds the new node to the token list at the spot immediately after itself, and sends an "AcceptJoin" message to the entire ring (line 25). Two important fields in message "AcceptJoin" are "newTokenList" and "newNextTokenSite". "newTokenList" is the latest token list that consists of all ring members, including the new node. "newNextTokenSite" is the next token site named by the current token site prior to joining in of the new node. When the new node receives "AcceptJoin" from the current token site, it sets its next token site as "newNextTokenSite", i.e. new node's next token site is current token site's next token site (line 15). The new node also sets the sender of "AcceptJoin" as the current token site (line 16), and updates its token list with "newTokenList" (line 17). When an existing member of ring receives "AcceptJoin", it updates its own token list with "newTokenList" and make sure that its next token site is correct by checking the latest token list (line 26). The new node repeatedly sends the "JoinRequest" message (line 13) until it receives an "AcceptJoin" message or number of tries is greater than a threshold (line 20). If no AcceptJoin message is received after a certain number of tries, the new node is failed to join in (line 22).

Join_ring()

```
NEW MEMBER:
Start {
    1.  If (I am the only node in the group) {
    2.          Create new token ring;
    3.  } Else {
    4.          Multicast "JoinRequest" to token ring;
    5.          Set timer;
    6.          Counter = 0;
    7.          Stop_join = False;
    8.          Do {
    9.                  Wait until timer expires or AcceptJoin is received naming me as next
                        token site;
    10.                 If (timeout) {
    11.                         Reset timer;
    12.                         Counter = Counter + 1;
    13.                         Resend JoinRequest multicast message;
    14.                 } Else if (receive AcceptJoin from current token site) {
    15.                         Set my next token site as current token site's old next token
                                site;
    16.                         Set sender of AcceptJoin as current token site;
    17.                         Update token list;
    18.                         Stop_join = True;
    19.                 }
    20.         } While (Counter < Threshold and Stop_join == False);
    21.         If (Counter >= Threshold) {
    22.                 Return failure;
    23.         }
    24. }
} End

CURRENT TOKEN SITE:
Start {
    25. Upon receipt of "JoinRequest", adds the new member to the token list in the spot
        immediately after me, and multicasts "AcceptJoin" to token ring;
} End

REST OF RECEIVERS
Start {
    26. Upon receipt of "AcceptJoin", updates its own token list;
} End
```

Figure C-1 Algorithm for adding a new node to a token ring

Leave_ring()

To remove itself from a ring, a member multicasts a "LeaveRequest" message to the entire group (line 1). Each member of the group stores the request in a queue called "LeavingNodes" (line 3). It checks this queue each time it accepts the token (line 4). If the queue is not empty at that time (line 5), the token site dequeues it until it finds one from the next site in the ring or until the queue is empty (line 6). If a request from the next site in the ring is found, then the token site removes that site from the current token list and multicasts a "LeaveConfirm" message around (line 8 – 9). One field in "LeaveConfirm" message specifies which node has been removed from ring. When other nodes receive "LeaveFonfirm" message, they remove that node's LeaveRequest from leavingNodes queue, and remove the node from token list as well (line 13).

```
Leave_ring()

MEMBER to be leaving
Start {
    1. Multicast "LeaveRequest" message
    2. Continue processing until "LeaveConfirm" message is received
} End

OTHER MEMBERS
Start {
    3. Upon receipt of "LeaveRequest" message, store it in the LeavingNodes queue
    4. Upon accepting the token, {
    5.         If (LeavingNodes queue is not empty) {
    6.                 Dequeue LeavingNodes until finds LeaveRequest from next site or
                       queue is empty
    7.                 If (LeaveRequest from next site exists in LeavingNodes queue) {
    8.                         Remove next site from token list
    9.                         Multicast "LeaveConfirm" message to the ring
    10.                }
    11.        }
    12. }
    13. Upon receipt of "LeaveConfirm" message, remove "LeaveRequest" of the
        confirmed leaving node from LeavingNodes queue, and update token list
} End
```

Figure C-2 Algorithm for removing a node from token ring

SendMsg()

Figure C-3 shows the algorithm for sending data packets. Each packet is multicast to the entire group (line 4). Source sets a transmission timer each time when it starts sending a packet (line 5), and retransmits it if it does not receive ACKs from the current token site within a timeout period (line 13 - 14). The source keeps on retransmitting a packet until it receives ACK for it (lien 10 – 11) or it has re-sent the packet for more than a number of times (line 17).

```
SendMsg()

SOURCE:
Start {
    1.  For (each data packet P) {
    2.          Add RMP's header to P;
    3.          Set destination;
    4.          Multicast P to all members;
    5.          Start retransmission timer;
    6.          Counter = 0;
    7.          Stop_transmit  = False;
    8.          Do {
    9.                  Wait until an ACK is received or the retransmission timer expires;
    10.                 If (receive the ACK) {
    11.                         stop_transmit = True;
    12.                 } Else if (timeout) {
    13.                         Exponentially backoff and reset timer;
    14.                         Retransmit P;
    15.                         Counter = Counter + 1;
    16.                 }
    17.         } While ( (Stop_transmite == False) and (Counter < threshold))
    18. }
} End
```

Figure C-3 Algorithm for source sending data packet

Recv()

Recv() is very similar to that of SRM and TMTP as shown in Figure A-2, except that RMP handles one more packet type – ACK.

Recv_data()

Each time a receiver receives a data packet (Figure C-4), it marks the reception of the data packet (line 1). If the receiver detects a packet loss, it requests a retransmission by sending a NACK (line 5). If the receiver is the current token site, it multicasts an ACK to the whole group (line 11). This ACK passes the token to next token site and acknowledge to the source of receiving the receiving packet.

```
Recv_data(packet)

Start {
    1.  Mark the arrival of this packet;
    2.  If (receiving packet's sequence number > last known sequence number of data) {
    3.          For (each packet between last and current receiving) {
    4.                  If (I didn't receive it) {
    5.                          Send NACK to request retransmission;
    6.                          Set last known sequence number to the receiving packet's
                                sequence number;
    7.                  }
    8.          }
    9.  }
    10. If (I am the current token site) {
    11.         Multicast ACK to the group;
    12. }
} End
```

Figure C-4 Algorithm for receiving data packet

Send_ack()

Send_ack() is called when the current token site receives a data packet or retransmission of a data packet. The token site multicasts an ACK to entire group (line 1). If there is no response to the ACK within the specified timeout, the current token site will send ACK again until it receives acknowledgement from next token site (line 9 - 12). If the acknowledgement is not received after a specified number of attempts, the current token

site assumes that next token site is unreachable and initiates a reformation action to reconstruct the ring (line 15 - 16).

```
send_ack(packet)

Start {
    1.  Multicast ACK for this packet to group;
    2.  Start timer;
    3.  Counter = 0;
    4.  Stop_ack = False;
    5.  Do {
    6.          Wait until an ACK from next token site is received or timer expires;
    7.          If (receive ACK from next token site) {
    8.                  Stop_ack = True;
    9.          } Else if (timeout) {
    10.                 Exponentially backoff and reset timer;
    11.                 Counter = counter + 1;
    12.                 Retransmit ACK;
    13.         }
    14. } While (Counter < threshold and Stop_ack == False)
    15. If (counter > = threshold) {
    16.         Initiate the reformation protocol;
    17. }
} End
```

Figure C-5 Algorithm for sending an ACK

Recv_ack()

Upon reception of an ACK, Recv_ack() is called (Figure C-6). If the source receives an ACK for a packet, it will stop waiting for that ACK and previous ACKs, i.e. stop retransmitting corresponding packets since it knows that the current token site has successfully received all packets up to the one just acknowledged (line 2). A field in each ACK names the next token site. If the receiver is the node that is specified as the "next token site", it starts checking if it is qualified to become the next token site (line 3). If the receiver has received all of the packets whose sequence number is smaller than and equal to this ACK (line 4), it declares itself to be the token site (line 5). The next token

site notices the old token site of its role by multicasting an ACK for the next packet received, or by sending a unicast confirm message to the old token site if no message is received within a set period of time. When the old token site receives the notice, it sets the current token site as the sender of ACK (line 11) and stop sending ACKs (line 12). When all other receivers get an ACK, they know that the current token site is the sender of ACK (line 14).

```
recv_ack(packet)
Start {
    1.  If (I am the source) {
    2.          Stop waiting for all ACKs up to the one just received;
    3.  } Else if (I am the new token site) {
    4.          If (I have received all messages whose sequence number <= sequence number
                of this ACK) {
    5.                  Set myself as the current token site;
    6.          } Else {
    7.                  Send NACK to request retransmission of missing packets;
    8.                  Wait for retransmissions for missing packets;
    9.          }
    10. } Else (I am the current token site) {
    11.         Current token site = sender of ACK
    12.         Stop sending ACKs
    13. } Else {
    14.         Current token site = sender of ACK
    15. }
} End
```

Figure C-6 Algorithm for receiving ACK

Send_nack()

The algorithm for sending NACK in RMP is very similar to that of TMTP as shown in Figure B-3, except that a requestor sends NACK to the current token site instead of parent (line 1 and 13), and does not find another parent in RMP (line 18).

Recv_nack()

The algorithm for receiving NACK in RMP is same as that of SRM as shown in Figure A-6.

Send_repair()

The algorithm for sending repair in RMP is same as that of TMTP as shown in Figure B-4.

Recv_repair()

There are two types of repairs – repairs from group members and retransmission from source due to lost ACK. RMP deals with repairs from group members just like SRM as shown in Figure A-8 (line 1 and 10 – 16 in Figure C-7). For retransmission from the source (line 2 - 8), there are three possible reasons causing this retransmission. (1) The current token site does not receive original packet. (2) The current token site does receive the original packet and multicast ACK, next token site receives the ACK and becomes the current token site, but the source misses the ACK. (3) Both next token site and source miss the ACK. For case (1) and (3), the current token site multicasts ACK to the group, i.e. treat the retransmission packet as a new data packet (line 4 – 5). For case (2), we do nothing here since the ACK of next packet from the new token site implicitly acknowledges this retransmission packet.

```
Recv_repair(packet)

Start {
    1.  Mark the arrival of this packet;
    2.  If (the repair comes from source) {
    3.          If (I am the current token site) {
    4.                  If (packet is what I am responsible for acknowledge) {
    5.                          Multicast ACK to the group;
    6.                  }
    7.          }
    8.  }
    9.
    10. If (I have sent NACK and waiting for this repair) {
    11.         Stop sending NACK;
    12. } Else if (I have the data and scheduled to send repairs) {
    13.         Cancel my repair timers;
    14. } Else {
    15.         Duplicate repair, ignore it;
    16. }
} End
```

Figure C-7 Algorithm for receiving a repair

Send_ctrl(), Recv_ctrl()

Control messages (such as JoinRequest, AcceptJoin, etc.) that are used for ring membership and reconstruction of ring are handled by routines Send_ctrl() and Recv_ctrl(). The algorithms for these two routines are discussed in JoinRing(), LeaveRing() and Reformation() as shown in Figure C-1, Figure C-2 and Figure C-8.

Reformation()

Reformation() is invoked when the current token site considers that its successor is unreachable (Figure C-8). The current token site starts reformation by multicasting a "ReformationRequest" message to the ring (line 1). During reformation process the initiator is the current token site and responsible for acknowledging data packets from the

source. When other nodes receive "ReformationRequest", they send "RejoinConfirm" message to the initiator to confirm that they are alive and reachable (line 3). Once the initiator receives "RejoinConfirm" from a member, it adds the member to the token list immediately after itself, and sends an "AcceptRejoin" message with latest token list to the sender of "RejoinConfirm" (line 2). When a member who has sent "RejoinConfirm" receives "AcceptRejoin" from the initiator (line 4), the member sets its next token site as initiator's old next token site (line 5). It also sets the sender of "AcceptRejoin" (line 6) as the current token site and updates its token list (line 7).

Reformation()

INITIATOR
Start {
1. Create a ring, ***multicast*** "ReformationRequest" message
2. Once receiving "RejoinConfirm", add sender of "RejoinConfirm" to the token list in the spot immediately after me, send "AcceptRejoin" to the sender of "RejoinConfirm"

}

OTHER NODES
Start {
3. Upon receiving "ReformationRequest", send "RejoinConfirm" to initiator
4. If (receiving "AcceptRejoin") {
5. Set my next token site as initiator's old next token site
6. Set sender of "AcceptRejoin" as current token site
7. Update token list
8. }

} End

Figure C-8 Algorithm for ring reformation

APPENDIX D – CONFIDENCE INTERVALS

Normally, confidence intervals placed on the mean values of simulation results can be used to describe the accuracy of the simulation results. Consider the results of N statistically independent simulation runs for the same experiment: X_1, X_2, ..., X_N. The sample mean, $\overline{X}$ is given as:

$$\overline{X} = \frac{\sum_{i=1}^{N} X_i}{N}$$

The variance of the distribution of the sample values, S_x^2 is:

$$S_x^2 = \frac{\sum_{i=1}^{N} (X_i - \overline{X})^2}{N-1}$$

The standard derivation of the sample mean is given by: $\frac{S_x}{\sqrt{N}}$.

Under the assumption of independence and normality, the sample mean is distributed in accordance to the T-Distribution, which means the sample mean of the simulation runs fall in the interval $\pm\varepsilon$ within the actual mean with a certain probability drawn from the T-Distribution.

$$\varepsilon = \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$

where $t_{\alpha/2,N-1}$ is the value of the T-distribution with N-1 degrees of freedom with probability $\alpha/2$.

The upper and lower limits of the confidence interval regarding the simulation results are:

$$\text{Lower Limit} = \overline{X} - \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$

$$\text{Upper Limit} = \overline{X} + \frac{S_x t_{\alpha/2,N-1}}{\sqrt{N}}$$