A HIGH PERFORMANCE TOTALLY ORDERED MULTICAST PROTOCOL

Brian Whetten
University of California at Berkeley
whetten@cs.berkeley.edu

Simon Kaplan
University of Illinois at Urbana-Champaign
kaplan@cs.uiuc.edu

ABSTRACT

This paper presents the Reliable Multicast Protocol (RMP). RMP provides a totally ordered, reliable, atomic multicast service on top of an unreliable multicast datagram service such as IP Multicasting. RMP is fully and symmetrically distributed so that no site bears an undue portion of the communication load. RMP optionally provides K-resilient fault-tolerance, which guarantees message delivery so long as no more than K (a system configurable parameter) sites fail or partition away within a short period of time. RMP is scaleable to hundreds of simultaneous users.

It has commonly been held that a large performance penalty must be paid in order to implement these features; RMP discounts this. On SparcStation10's on a 1250 KB/sec Ethernet, RMP provides 842 KB/sec throughput and 3.1 ms packet latency to one destination. The performance stays roughly constant independent of the number of destinations. For two or more destinations on a LAN, RMP provides a higher aggregate throughput than any protocol that does not use multicast or broadcast.

1.0 INTRODUCTION

Totally ordered, reliable broadcast and multicast protocols have existed for quite some time [ChMa84], and provide a powerful tool for programming distributed systems and distributed databases [Chang84]. New applications such as Computer Supported Cooperative Work (CSCW) programs, groupware systems and shared tools can also benefit from this service. In the past, these protocols have had problems with performance, efficiency, and/or scalability. It has become a widespread belief that these are inherent problems with a totally ordered reliable multicast protocol [RaLi93]. In part, this concept resulted from the fact that in the past multicasts had to be implemented as a series of unicasts to each destination. Recent developments such as the IP Multicasting standard [Deering89] now allow a multicast datagram to be sent to multiple destinations over an internetwork. In the case where all destinations are on the same LAN, one multicast packet to all of them costs the same as a unicast packet to just one.

This paper presents the Reliable Multicast Protocol (RMP) which provides a reliable multicast service on top of unreliable datagram services such as IP and IP Multicasting. RMP is based on the token ring protocol first proposed by J. M. Chang and N. F. Maxemchuk in [ChMa83] and [ChMa84]. This protocol, which we name TRP, provides what can be thought of as a single N-way virtual circuit between a single group of processes connected by a broadcast medium. TRP is fully distributed, so that all processes play the same role in communication. The protocol provides atomic, reliable message delivery that is both totally and causally ordered, using causal ordering as defined in [Lamp78] and [BiJo87]. Finally, the protocol provides K-resilient fault tolerance, so that up to K processes can crash or partition away simultaneously without disrupting the message delivery between the rest of the processes. K is a user-configurable parameter that must be the same across all communicating processes. Higher values of K provide higher fault tolerance, but at the cost of higher latency of message delivery.

The Reliable Multicast Protocol (RMP) improves on the limited scalability of TRP by using multicast instead of broadcast, by allowing multiple simultaneous token rings, by providing a fully distributed group naming service, by improving the reformation algorithm for group membership changes that are not due to site crashes, and by implementing more efficient and robust flow and congestion control. The new reformation algorithm allows RMP to provide *virtual synchrony*, as defined by the ISIS project [BSS91]. This means that when new members join or leave a group these operations appear to be atomic, so that the sets of messages delivered before and after each membership change are consistent across all sites. These algorithms should allow RMP to scale to hundreds of users across the Internet.

A common belief in the research community is that totally ordered reliable multicast protocols are inherently slow. This belief has come about in large part due to the experiences researchers have had with the early versions of ISIS, which for a long time was the only system of this type available. ISIS has since become much faster [BSS91], but the misconception remains. Experience with RMP belies this concept. RMP was tested on 8 SparcStation10's on a 10 Mb/sec (1250 KB/sec) Ethernet. In this environment, the throughput to a single destination is 842 KB/sec, or 67.4% of the network capacity. For group communication to any group of two or more destinations on a LAN, RMP exceeds not only the maximum throughput of TCP/IP, but any other possible non-multicast and non-broadcast algorithm. This is because both the packet latency and throughput of RMP stay roughly constant as the number of destinations increase, whereas the performance of other algorithms decreases linearly. For a group with 8 destinations, RMP has a 6.8 MB/sec aggregate

throughput, which is 5.4 times the bandwidth of the supporting Ethernet. RMP pays a moderate latency penalty for its total ordering, but this is only a factor in short messages to a single destination. In groups of two or more, RMP latency is lower than most other protocols. Therefore RMP demonstrates that a fault tolerant, reliable, atomic, totally ordered multicast protocol can actually achieve much better performance in group communication than systems that don't provide these features. In addition, if the application does not need total ordering and atomicity of messages, RMP can provide roughly the same latency as other protocols by delivering packets immediately at the destination.

Section 2 describes the system model of RMP. Section 3 describes the token ring protocol RMP is based on. Section 4 describes the modifications RMP makes to the token ring protocol to allow it to scale to an internetwork. Section 5 analyzes the performance of RMP, and section 6 compares RMP to previous work. Finally, section 7 draws our conclusions and outlines future work.

2.0 SYSTEM USAGE

RMP is a transport level protocol that provides reliable delivery on top of a unicast or multicast unreliable datagram service. It is currently implemented as a C library that can be linked directly in to applications or can provide the transport layer for message servers that provide a high level communication interface to clients. The latter situation is more common; an example of this is the Message Bus II (MBusII), a publisher-subscriber model communication tool that has been implemented on top of RMP. The publisher-subscriber model allows groups of cleints to communicate without explicit knowledge of each other, which makes many applications easier to program.

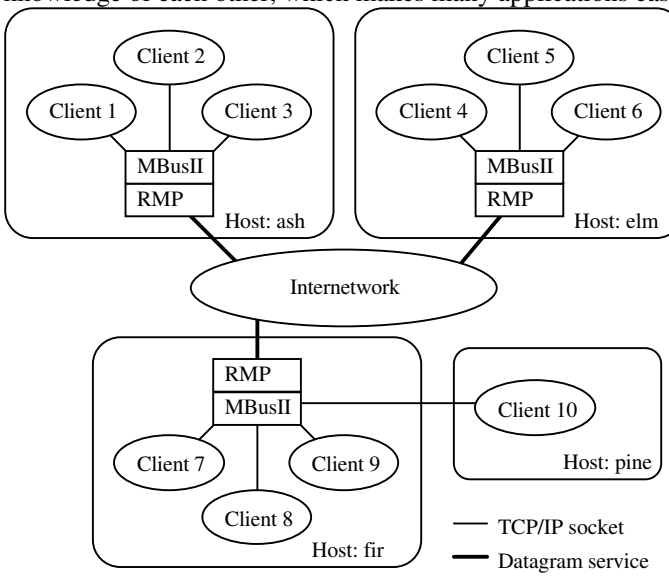

Figure 3.1: Interconnection of Clients and Servers

One MBusII server is run on each host, and multiple clients connect to it using TCP/IP streams. Since most clients are on the same host as a server, these streams become IPC calls that do not affect the network. This approach provides an easy to program interface and allows one message to be sent efficiently to multiple clients on the same host. Figure 3.1 shows an example of this client-server architecture.

3.0 TOKEN RING PROTOCOL DESCRIPTION

The biggest decision in building a reliable multicast protocol is how to guarantee reliability. Traditional protocols use positive acknowledgments (ACKs) from the destination(s) to acknowledge successful receipt of a packet. This approach does not scale well to a multicast system, because each destination has to send an ACK for each packet. This largely defeats the advantage of using multicast packets, because it decreases both the efficiency and the performance of the protocol. Even though these acknowledgments are small, because they all are sent at the same time they can cause network congestion. In addition, having to process an ACK from each destination increases the load on the sender and decreases the performance of the protocol. To get around this, many systems use negative acknowledgments (NACKs). Negative acknowledgments shift the burden of error detection from the source to the destinations. Packets are stamped with sequential sequence numbers which destinations use to provide reliable delivery by detecting gaps in the sequence numbers and requesting retransmission of the packets corresponding to the gaps. Because the information that a packet has been received is never propagated back to the sender, the senders in these protocols do not ever know for certain that a destination has received a packet. Because of this, senders have to indefinitely keep a copy of each packet sent if the protocol is to be considered truly reliable. In addition, a lost packet that is not followed by another packet may not be detected for quite some time.

Because of these problems, the token ring protocol (which we have named TRP) uses a combination of these two approaches. This section describes the basic algorithm of TRP as originally proposed by Chang and Maxemchuk. For a more rigorous analysis, as well as a complete description and proof of the correctness of the reformation protocol, please see [ChMa84] and [ChMa83]. [MaCh84] gives a complete analysis of the number of packets sent by the protocol, and [Chang84] describes the advantages of using this protocol to design a distributed database.

In TRP, all messages are stamped with a pair (host number, sequence number for that host) which uniquely identifies each message. Each message is then broadcast to the entire group, and all senders to a group must be members of

it. Messages are all handled by a primary receiver called the token site. When the token site receives a broadcast message, it broadcasts a positive ACK out to the group. Along with the (host, sequence) pair for that message, this ACK contains a sequence number, called a timestamp, which serializes the messages from all of the senders. This 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 acknowledgment.
- The timestamps in the ACKs provide a total and causal ordering on messages.
- The timestamps also provide a global basis for detection of dropped packets. The receivers can detect any missed packets, both ACKs and messages, through these global sequence numbers. With multiple simultaneous senders this provides more timely detection of missed packets than sequence numbers from each sender.

This does not solve the problem of guaranteeing the detection of dropped packets and the corresponding problem of unlimited buffer space. To solve this problem, the token site is passed among all sites in the token ring. The token is rotated (at least) once per each message sent, and each new token site is required to have all messages up to the current one before it accepts the token. Not only does rotating the token balance the load of the ACKs between the sites, it also solves the buffer problem. Given N members of a ring, once the token has been rotated N times, the token site knows that all messages with a timestamp at least N smaller than the current timestamp have been received at all destinations, and so the token site no longer needs to store these messages for retransmission.

Token passing is done as another consequence of the ACK. A field in each ACK names the new token site. When this site receives the ACK, it checks to see if it has received all of the messages and ACKs with timestamps smaller than this ACK. If not, it requests them from the previous token site. Once it has all the messages, it declares itself to be the new token site. It lets the other sites know that it is the new token site either by sending an ACK for the next message received, or if no message is received within a set period of time, by sending a unicast Confirm message to the old token site.

Each time a destination receives an ACK, it uses the ACK's timestamp to order the corresponding message. The destination can deliver a message after all of the preceeding messages have been received and delivered. If the destination is missing any ACKs or messages, it requests these by sending a unicast NACK to the site it currently believes to be the token site. This site is guaranteed to have a copy of the missing message or ACK, even if it is no longer the token site, and resends

the missing packet as a unicast to the site that requested it.

All three of these actions--broadcasting a message to the token site, passing the token to the next token site, and requesting retransmission of a packet--are reliable operations that use positive acknowledgments. 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, the site is presumed to have failed, and a reformation protocol is run to reconstruct the list of sites in the token ring.

Passing of the token also allows the protocol to provide K-resilient fault tolerance. K is a system wide parameter corresponding to the amount of time a packet is delayed by a destination before it is delivered. By delaying the delivery of a message until the token has been passed K times, it is ensured that both the current token site and the K previous token sites have that message. This allows up to K sites to crash or partition away and still guarantee reliability of message delivery to the sites that are still active. Along with this, passing the token guarantees that site failures and dropped messages are detected within N messages. In order to bound the amount of time before a lost packet or a failed site is detected, TRP sends null ACKs if the token site does not receive a message within a given period of time. With RMP, this is currently on the order of 1 second. If the ring goes quiescent for a period of time, the token is passed all the way around the ring once and then stops. At this point, all of the sites are guaranteed to have all of the messages.

As previously mentioned, if a site repeatedly fails to receive the proper response to one of the actions that requires a positive acknowledgment, it declares the site dead and runs a reformation protocol. In this protocol, the site that first detects the failure uses multicast packets to repeatedly poll the sites listening to a given broadcast address. From the responses to the polls it constructs a new token list, and this list is atomically updated at all of the sites in the new list with a two phase protocol. If more than one site starts the reformation at the same time, this will be detected and the reformations aborted, whereupon each of these sites delays for a random period of time and then starts the reformation process again. We omit this algorithm and its proof of correctness for brevity.

In normal situations with low error rates and moderate to high traffic, TRP requires very close to 2 broadcasts per message. As the traffic rate decreases, this increases to 2 broadcasts plus a unicast due to the messages required to confirm the transfer of the token. As the error rate increases, the number of packets sent increases, but it is always lower than that required with positive acknowledgments for groups of three or more sites including the sender [ChMa84].

4.0 EXTENSIONS TO THE TOKEN RING PROTOCOL

In order to provide better performance and to scale to multiple groups over an internetwork, RMP uses multicast instead of broadcast, allows for multiple token rings, provides a fully distributed name service, improves the reformation algorithm for membership changes that are not due to failures, and implements a more robust and efficient flow and congestion control algorithm.

4.1 Multiple Token Rings and Groups

Communication in RMP is organized around groups and token rings. Groups correspond to the set of processes that a message is to be delivered to while token rings correspond to the set of hosts that a message is to be delivered to. Both constructs are needed because multiple processes on one host may be in the same group.

A group name is a hierarchical text string, roughly modeled after the textual representation of IP addresses. There are two main differences between a group name and an IP address. First, IP addresses can only consist of up to four levels of text, whereas group names can consist of an arbitrary number of levels. Second, IP addresses are statically assigned and based roughly on the actual network topology, whereas the group names can be dynamically created or deleted at any time and can link an arbitrary group of sites together.

For every group name, there is a group number which is the binary representation of it. Group numbers are 64 bit addresses that are guaranteed to be unique across the network. They are formed by concatenating the 32 bit IP address for the first host to join a group along with a counter maintained by that host. Each host keeps the counter unique by incrementing it each time it is used and by initializing it to the host's current time (in seconds) every time that a RMP server is started at that host.

Token rings are the actual virtual circuit maintained between RMP servers, and it is the unit over which total ordering of messages is provided. RMP allows a single token ring to correspond to multiple groups, which allows ordering guarantees to be provided across groups and allows groups to be used at a finer level of classification than token rings. For example, if a set of hosts are engaged in a conference involving both a shared white board and a shared editor, a token ring could be used to denote the set of hosts and provide delivery to all of them, and separate groups could be set up for each of the active pages in the white board and for each of the active documents in the editor to make programming the effects of actions on these pages easier.

Groups are hierarchical in nature, and logically make up a forest of names. Each token ring corresponds to a subtree of this forest. When joining a group, a client requests whether or not it would like this group to correspond to a new token ring. If the client is the first member of this group, a new token ring is created for this group if possible. Token rings can only be created if they don't fragment a previously existing token ring. For example, if a token ring corresponds to the groups *uiuc.edu* and *wb.cb.uiuc.edu*, a new token ring could be created for a group *database.uiuc.edu* or *editor.cb.uiuc.edu,* but not for *cb.uiuc.edu.* If a ring for *cb.uiuc.edu* was created the members of the domain *wb.cb.uiuc.edu* would have to be moved from the old ring to the new ring. This would greatly complicate the membership algorithm, and so is not allowed.

Figure 4.1 shows an example with 6 RMP servers at two sites, all connected through the Internet. There are four token rings: the name server ring between one site at each location, one ring between the three Illinois servers, one ring between the three Berkeley servers, and a ring between two physically separated servers from the two locations. Each of these token rings has one or more groups associated with it, and for the purposes of this picture is named by the top-most group in this association.

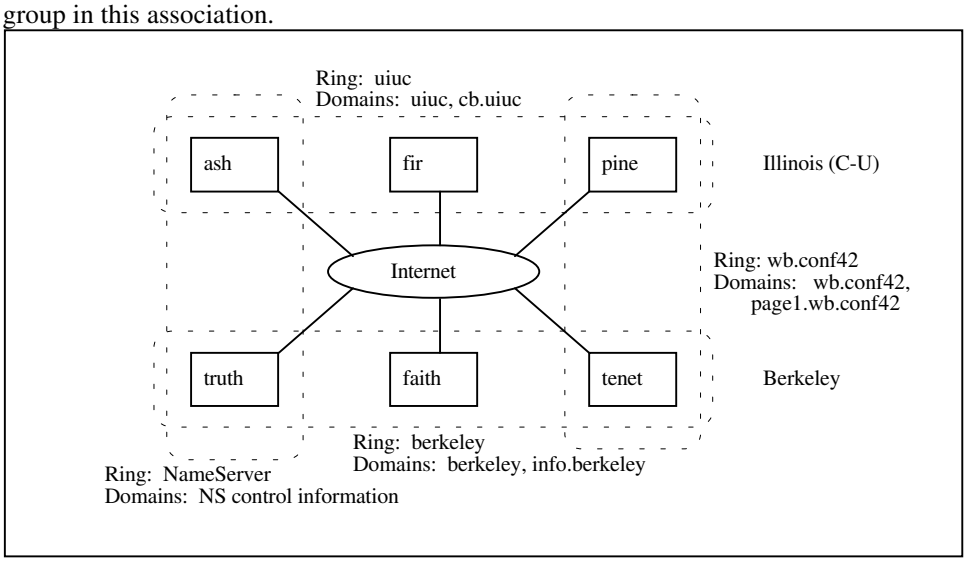

Figure 4.1: Use of Token Rings

4.2 Multicast Communication

Because the broadcast communication of TRP won't scale to an Internetwork, RMP is based on IP Multicasting. This service sends unreliable datagrams to multiple destinations. In the case where multiple destinations exist on a link that supports hardware multicasting, this packet can be delivered for the same price as a single IP packet to one of these destinations. The routing protocol sends each packet over an approximation of a minimum spanning tree between the

destinations, only splitting up the packet as necessary.

In order to provide multicast service over multiple token rings, multiple IP Multicast sockets have to be opened, one per token ring. A pool of available {address, port} pairs is maintained by the Name Server (see below). Because the TRP protocol also sends some unicasts, and in order to support sites which do not run IP Multicast, the address for a token ring also contains the IP addresses and associated UDP port numbers for all the members of the ring, along with a flag as to whether or not each site supports IP Multicast. Together, we call this information the extended multicast address for a token ring. In order to send a multicast packet to a ring, a machine in the ring first sends a packet to the IP Multicast address and port associated with the ring, then sends a UDP packet to each of the {IP address, port} pairs that are flagged as not supporting multicast. Part of the header for each RMP packet contains the IP Multicast address for the token ring the packet is destined for. This allows the UDP packets from all token rings to be sent to the same UDP port.

4.3 Distributed Name Server

There are three levels of mappings kept in the Name Server (NS): global mappings, token ring mappings, and local mappings. A list of the extended multicast addresses for all of the current token rings and the top group name that each ring corresponds to must be maintained across some single global set of name servers. This information is sufficient for RMP servers to map a group name into the appropriate token ring and join that ring. Because the token ring membership operations (described in section 4.4) are atomic, it is not necessary that all of this information be kept strongly consistent. Creations and deletions of a ring must be kept consistent across the name servers in order to prevent multiple instances of the same token ring, but other ring membership changes do not. The only consequence of the membership of the extended multicast addresses being inconsistent is that new members may fail when trying to join a ring which has non-multicast members in it. If this occurs, the operation can be retried until it succeeds. The global set of name servers all subscribe to the NameServer token ring and use it for all communication about the global mappings. These Global Name Servers (Global NSs) in turn can act as proxies for other Name Servers that want to make changes to the global token ring mappings. In the example shown in figure 4.1 there are two Global NSs, one at each location. Each of these name servers handles the global mappings for the three sites at their group. Forwarding of Global NS changes is done through the uiuc and berkeley rings. This topology is vulnerable to crashes of either NS, but this could be remedied by including multiple Global NSs in each of the forwarding rings.

The second level consists of the mappings from token ring addresses to the token list, group names and group numbers for each ring. This information allows membership changes to a group name to be mapped to the appropriate token ring and group number, allows messages to a given group number to be mapped to and from the appropriate token ring, and provides the information that TRP needs to function. These mappings can be maintained across just each token ring instead of across the global NSs because RMP requires that a client be a member of any group (and its token ring) that it sends to. All of the information at this level has to be kept strongly consistent, which is done by the membership change protocol described in section 4.4 with the help of the NS algorithms below.

The third level of information maps local clients to and from group numbers. These mappings allow messages sent to a group number to be delivered to the appropriate set of clients, and are only maintained locally. This means that the total membership of a group is not maintained in any single spot in RMP unless there is only a single server in that ring. The implementation of this level is straightforward.

Level	Information in Level	Mapping	Maintained Across
Global Level (top)	Existence of token rings Extended multicast addresses for rings	None Group name to ext. address	Global NSs (strong) Global NSs (weak)
Token Ring Level (middle)	Token list for TRP Group numbers for each ring Group numbers for each name in ring	Token ring to token list Group number to token ring Group names to numbers	NSs in each ring (strong consistency)
Local Level (bottom)	Clients subscribed to each group	Group number to client list	Individual NS

Figure 4.2: Classification of NS Information

The global extended multicast address mappings are the essence of the Name Server, along with the interaction of these mappings with the new membership algorithm described in the next section. In order to provide strongly consistent global information, any time that one of the Global Name Servers creates or deletes one of the global mappings, it attains a lock on these mappings at all of the Global Name Servers. A site obtains a lock by sending a "lock ring" message to the global token ring and waiting until this message is delivered back to it. The total order of these "lock ring" messages provides the guarantee that only one site holds the lock at a time. The algorithm for the AcquireLock() and ReleaseLock() functions is shown in figure 4.3.

The Global NSs need to stay consistent even if one of them crashes. As in the original TRP protocol, any time that a site crashes the reformation protocol will be called on each of the rings it was a member of. A new consequence of this reformation protocol is that the site that performs the reformation calls (through a proxy NS if necessary) ReformedRing() at

a Global NS. This routine frees up the lock if it was held by a crashed site, and updates the extended multicast address for that token ring with the new one. This algorithm is also shown in figure 4.3

AcquireLock()
1. Send "lock ring" message to token ring
2. Wait until we receive our message, keeping track of lock status of ring
3. If ring is currently locked by another site
    a. Wait until "unlock ring" message is received from site holding lock
    b. Goto 1.
4. Else
    a. We now hold lock on token ring

ReleaseLock()
1. Send "unlock ring" message to token ring
2. Wait until we receive our own "unlock ring" message
3. Lock on token ring is now freed

ReformedRing(GroupName, ExtendedAddress)
1. Reformation process notifies all sites in ring that ring is locked
2. If token ring for GroupName is locked
    a. Query all members of reformed ring for lock
    b. If lock is not held by any site in ring
        i. We now hold lock
3. If we do not hold lock
    a. AcquireLock()
4. Send new ExtendedAdress to ring
5. ReleaseLock()

Figure 4.3: Locking and Reformation Algorithms for Global NSs

The main operations that a Global NS performs are InitializeNS(), DeleteNS(), AddSiteToAddress(), and RemoveSiteFromAddress(). InitializeNS() is used when starting up a new Global NS. It is responsible for getting a copy of the current mappings from another Global NS, if one exists. The DeleteNS() algorithm is run when a RMP server is about to shut down, and simply removes all hosts from all extended addresses, removing addresses as necessary. AddSiteToAddress() adds a host to an extended address. RemoveSiteFromAddress() is the reverse of AddSiteToAddress(); it removes a site from an address, deleting it if necessary. These algorithms are presented in figure 4.4. All waiting done in these algorithms is done in a non-blocking way.

The AddSiteToAddress() and RemoveSiteFromAddress() functions interact with the AddMemberToRing() and RemoveMemberFromRing() routines below. AddSiteToAddress() is called to add a member to a ring, and it calls AddMemberToRIng(). This guarantees that if a ring is being created by AddMemberToRing(), then no other site can

simultaneously create that ring. RemoveSiteFromAddress() is called from RemoveMemberFromRing() in order to guarantee that if a ring is being deleted then the address for it is simultaneously removed.

InitializeNS()
1. Link NS into global token ring
2. AcquireLock() on all Global Name Servers
2. Request a copy of global data from one of the NSs
3. Wait for copy of data, store it
4. ReleaseLock() for Global Name Servers

DeleteNS()
1. For each local GroupNumber and each Client which is a member of that GroupNumber
    a. RemoveSiteFromAddress(GroupName, Site)

AddSiteToAddress(GroupName, Site)
1. If Address for GroupName doesn't exist
    a. AcquireLock() on all Global NSs
    b. If Address for GroupName doesn't exist
        i. Get IP Multicast address from free pool
        ii. Create a new Address, send to other Global NSs
    c. Else
        i. Goto 2a
    d. AddMemberToRing(Address, Site) (see figure 4.5)
    e. ReleaseLock()
2. Else
    a. Send new Address info to all Name Servers
    b. AddMemberToRing(Address, Site)
    c. If AddMemberToRing fails, goto 1.

RemoveSiteFromAddress(GroupName, Site)
1. If Address for GroupName only contains Site
    a. AcquireLock() on all Global NSs
    b. If Address for GroupName only contains Site
        i.. Put IP Multicast address for token ring back in free pool
    c. Send new Address or notification of deleted Address to all Global NSs
    d. ReleaseLock()
2. Else
    a. Send new Address to all Global NSs

Figure 4.4: Global NS Algorithms

There is some flexibility in the implementation of the Global NSs. Because there is a clean API to the rest of RMP, a different name service could be substituted, so long as it maintains enough consistent information on the existence of token rings to keep the creation and deletion of rings atomic. In the current implementation, each NS (one per RMP server) is considered a Global NS, and the second level information mappings for group numbers are kept globally as well. These are

artifacts of implementation, and can be changed in a future version.

4.4 Membership Change Protocol

As explained above, the name server takes care of keeping the extended addresses for token rings current, but this does not take care of ordering the sites in each token ring or keeping track of the group names and numbers for that token ring. The first problem was solved in the TRP protocol with a general reformation protocol. This protocol uses multicast messages to repeatedly poll all of the sites, which respond if they are still a member of the token ring. RMP originally used this protocol to handle all token ring membership changes. Experience showed that under heavy load some sites do not always respond in time to these poll commands. Also, the protocol is too slow for normal use. Because of these reasons, a new algorithm has been designed for updating the token ring in response to membership changes that are not due to site failures.

To add a new site to the ring, the new site calls AddSiteToAddress() at a Global NS(), through a proxy if necessary. This checks to see if the address exists, and atomically creates it at all Global NSs if not. That procedure then calls AddMemberToRing() with the address, which starts listening to the IP Multicast address for this ring. If it is the only member in the new address it creates a new ring. Otherwise, it sends a multicast message to the ring informing the ring members that it wishes to join. When the current token site gets this message, it adds the member to the token list in the spot immediately after it, and sends a "new token list" message to the ring using the normal protocol. The token site sends the ACK for it as well, naming the new site as the next token site. Passing the token requires confirmation to the old token site that the token has been received, and this functions as a positive acknowledgment that the join operation has been completed successfully. The new site must request and receive the last N packets and ACKs before it can acknowledge the token. This maintains the resiliency requirements for the TRP protocol and allows the new member to handle requests for retransmissions. The algorithm for this is shown in figure 4.5.

AddMemberToRing

NEW MEMBER (called from AddSiteToAddress() with new address)
1. Open up socket to IP Multicast address
2. If we are only site in address
    a. Create new token ring
3. Else
    a. Send unreliable "new member" multicast to token ring
    b. Set timer

c. Wait until timer expires or ACK is received naming us as new token site
d. If timer has expired more than X times
    i. Return failure
a. Else if timer expired
    i. Goto 3a.
b. Else
    i.. Request past N messages and ACKs from old token site
    ii. Store token list that accompanied ACK
    iii. Return success to Global NS and then proceed as normal, confirming token pass by sending an ACK or a Confirm

CURRENT TOKEN SITE
1. Upon receipt of "new member" message
    a. Create new token list with new member in spot after current token site
    b. Reliably send "new token list" message to token ring, along with ACK for it

ALL SITES
1. Upon delivery of "new token list" message
    a. Store and start using new token list

Figure 4.5: Algorithm for Adding a Site to a Token Ring

To remove itself from a ring, a member reliably multicasts a "delete member" message to the group. Each member of the group stores these requests in a queue that it checks each time it accepts the token. If the queue is not empty at that time, the token site dequeues requests until it finds one from the next site in the ring or until the queue is empty. 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 reliably multicasts the new list around. The current token site then sends out the ACK for this reliable multicast, naming the site after the removed site as the next token site. The effect of this algorithm is that a site is only removed when it is about to receive a token, which guarantees that the resiliency requirements are not violated. The removed site stops processing this token ring when it receives the message removing it from the list. When this occurs it has all of the messages up to the point where it is removed, and it knows that it will no longer be needed in the functioning of the protocol. After the "new token list" message is delivered, the site calls RemoveSiteFromList(). If the site is the last one in the token list, RemoveSiteFromList will delete the address for it.

RemoveMemberFromRing()
OLD MEMBER
1. Reliably multicast "delete member" message
2. Continue processing until "new token list" message is delivered
3. RemoveSiteFromList(Address, OldSite)

ALL SITES
1. Upon receipt of "delete member" message, store request in a queue

2. Upon receiving the token, scan queue for any requests from the next site in the ring.
3. If request exists, send "new token list message" removing site from list, along with ACK for it
4. Upon receipt of a "new token list" message, store it, start using it, and remove "delete member" requests from all sites in queue that aren't in the new token list

Figure 4.6: Algorithm for Removing a Site From a Ring

Group name lists for a token ring and the corresponding group numbers are also updated by sending the changes as totally ordered messages. These mappings are also sent to the token ring whenever the membership of a token ring is changed, so that new sites get the current group mappings.

This algorithm provides virtual synchrony of group membership changes when considered together with the NS algorithms in figure 4.4. We offer only an informal argument for this. To show that the protocol is correct and provides virtual synchrony we must show that a membership change always terminates, that two rings with the same address can never be created, and that all membership changes to a group (including creation and deletion of a ring) are totally ordered with respect to each other and with respect to the other messages in a ring. A membership change always terminates because the global lock can never be held indefinitely by one site and because a failed add is retried with the latest version of the address. If necessary, the reformation protocol will be called to guarantee this. Two rings with the same address can never be created because the creations and deletions of a ring are globally serialized. Membership changes to a ring are serialized over that ring by using a totally ordered message to perform the change among the token ring members. Group changes are also done with totally ordered messages, and so are also serialized with respect to each other, with respect to the changes in the token ring membership, and with respect to the messages to the token ring. Together, these things provide strongly consistent, totally ordered membership changes and message deliveries at the token ring level, while only requiring the Global NSs to keep track of a minimum of strongly consistent information.

4.5 Flow Control and Congestion Control

Two of the most common schemes used today for flow control are sliding windows and leaky buckets. With modifications, these schemes are also used for congestion control. Some reliable multicast schemes have adopted a leaky bucket scheme, which enforces explicit rate controls on each sender, because it is impossible to use a sliding window with a protocol based on NACKs. Calculating the values for these rates is difficult, and they must divide up the bandwidth between the senders on a relatively static basis. This decreases the flexibility and throughput attainable by these schemes.

Because of these problems, RMP uses a modified sliding window protocol for both flow and congestion control.

This algorithm is based on the algorithms proposed by Van Jacobson for TCP in [Jacobson88] but modifies them to take into account the fact that the ACKs in RMP are more expensive than those in TCP due to the rotation of the token site. A sliding window regulated by congestion and the ACKs from the token site is maintained by each sender. This provides congestion control. Because the Van Jacobson algorithms provide such good dynamic congestion control, RMP is also able to provide flow control with the same algorithms by treating NACKs as another form of congestion. The Van Jacobson algorithms for congestion control that are used by RMP include:

(1) round-trip-time variance estimation
(2) slow start
(3) dynamic window sizing on congestion
(4) exponential retransmit timer backoff

Round-trip-time variance estimation comes from the observation that when a network path becomes congested, the variance on packet latency becomes very high compared with the average. "If the network is running at 75% of capacity...one should expect the round-trip-time to vary by a factor of 16."[Jacob88] The proposed algorithm continually estimates this variance, and eliminates most of the spurious retransmissions while still maintaining timeouts small enough to detect dropped packets quickly.

The slow start algorithm (2) is used to linearly increase the window size from 1 packet to the maximum window size that the receiver allows that does not cause congestion, as calculated by algorithm (3). This is done by incrementing the window size by one packet each time that an ACK is received. Because the window size is constantly growing, slow-start actually increases the window fairly quickly. It will increase from 1 to W on a network with latency L in $L\log_2 W$ time.

With the assertion that the timer algorithm almost completely avoids retransmissions that are not due to lost packets and with the observation that most lost packets are due to congestion instead of errors, it follows that most expired timers signal congestion. Algorithm (3) uses this to respond aggressively to this congestion by exponentially reducing the window size by a constant number (currently 50%) each time that a timer expires. The original protocol actually uses a two-level bound on this window in the face of congestion. It reduces the window size to one packet any time an error occurs, uses slow start to quickly build up to 50% of the level before the error, and then uses a slower linear increase to build up from there. RMP only reduces the current window size by 50% because RMP modifies the packet lengths according to the window size. The algorithm used for this causes slow start to reach a window size W (assuming W is less than 16K) in O(W) time instead of O(logW) time, and this makes the cost of reducing the window all the way to 1 packet too high.

Finally, the exponential retransmit timer backoff is used to double the timer each time it expires, resetting it to the value calculated by (1) when an ACK is finally received. Both this and algorithm (1) are applied to the timers for all three classes of positive acknowledgments. Along with further decreasing congestion, in RMP this allows an efficient detection method for failed sites. The maximum value for a timer is clamped at a certain value (currently 2 seconds). Then up to N retransmissions (currently 10) are allowed before a site is declared dead. This takes into account the policy that sites over long latency links should be given more time to respond than those over short paths, because there is less probability of sustained congestion over short paths. As an example, with the current values this policy detects nearby sites within 5 seconds, and distant sites within 15-20 seconds.

These algorithms together allow RMP senders to quickly adapt their offered load to the available bandwidth. With modification, these algorithms also provide effective flow control. If a destination gets overrun by the senders, it will drop one or more packets. This will usually be detected by the destination very rapidly. At the most, it will be detected within N sent packets and hence N token passes. When this occurs, the TRP protocol has been changed so that the destination multicasts a NACK back to the group instead of unicasting it to the token site. In addition to requesting a retransmission of the packet from the current token site, it also informs the original sender (who is named in the NACK) that a destination has lost a packet from that sender. This is treated the same as an expired timer due to a lost packet, and causes the sender to decrease its window size by 50%. In order to make sure that multiple NACKs on the same packet do not each decrease the window size, a cache of the sequence numbers of the last three messages sent by this site that were dropped by another site is maintained by the sender. Incoming NACKs are compared against this cache, and the window size is only modified if a NACK for this message isn't in the cache.

```
if (A < MIN_PACKET && A < W)          // Avoid "Silly Window" effect
        Delay sending packet until an ACK is received
S = min(P, W/2);                      // Send up to 1/2 of the window at a time
S = max(S, MIN_PACKET);               // Send at least MIN_PACKET bytes
S = min(S, A);                        // Can only send up to A bytes
S = min(S, MAX_PACKET);               // Reduce effect of lost packets
```

Figure 4.7: Packet Size Algorithm

A problem that RMP faces with flow and congestion control is that the rotating token site introduces a higher overhead per acknowledgment than traditional protocols such as TCP. This is compounded by the protocol being more

complicated than TCP and thus requiring more processing per packet. To solve this problem, RMP uses larger packet sizes than does TCP. In an error free environment, having the IP or IP Multicasting layer do the fragmentation and reassembly is more efficient than having it done by RMP. If errors occur, the window size quickly drops to a single packet. Pseudo code to determine the size of the packet to be sent out (S), given the current window size (W), the available space in the window (A), and the offered packet size (P), is: shown in figure 4.7. The critical step in this algorithm is that up to half of the available window is sent at a time until the maximum packet size has been reached. This trades off a small amount of network utilization in some cases for higher efficiency of handling the packets.

5.0 PERFORMANCE

(Note to reviewers: The numbers in this section were measured with a previous version of RMP with a less efficient flow/congestion control algorithm. The new numbers should be even better, and we hope to rerun them by the time a camera ready copy is due. Additionally, we also hope to include comparisons with RPC and other RM protocols by this time.)

Performance testing was done on 8 SparcStation10's (Sun4m machines) on a very lightly loaded 10 Mb/sec Ethernet. All of the machines were running SunOS version 4.1.3. Throughput was measured by timing the transfer of a 5 MB data file, and so does not include message headers. Latency was measured by timing the sending of 10,000 minimum length messages (of 64 bytes, including all headers) with buffering and windowed flow control disabled. This causes the protocol to revert to a stop-and-wait acknowledgment system, in which case latency is equal to one over the number of messages sent per second. To eliminate the start up effects of the adaptive time-outs, an initial run was made in each test and discarded. Three valid runs were made in each test and the results were averaged together. While we do not show the standard deviations, they were quite small.

Figure 5.1 shows the aggregate throughput of the network as a function of the number of sources and destinations. Aggregate throughput is the throughput the user sees. It is computed by taking the amount of data sent from all of the senders and multiplying it by the number of destinations. In these tests, the sender is also a destination. This is done so that it can see its own messages totally ordered with the others. This is the way that most groups use totally ordered multicast, and is actually the worst case for the throughput of the protocol because of the increased CPU load of the senders. A single-server, TCP/IP based system, the MBusI, is shown for comparison. The maximum bandwidth of the Ethernet used in this

study is also shown. For the case of 2 sources and 8 sinks, RMP achieved an aggregate throughput of 6810 KB/sec. This is 5.45 times the bandwidth of the Ethernet. It is impossible for any solution that does not use either multicast or broadcast to achieve any result that breaks the Ethernet throughput boundary this way.

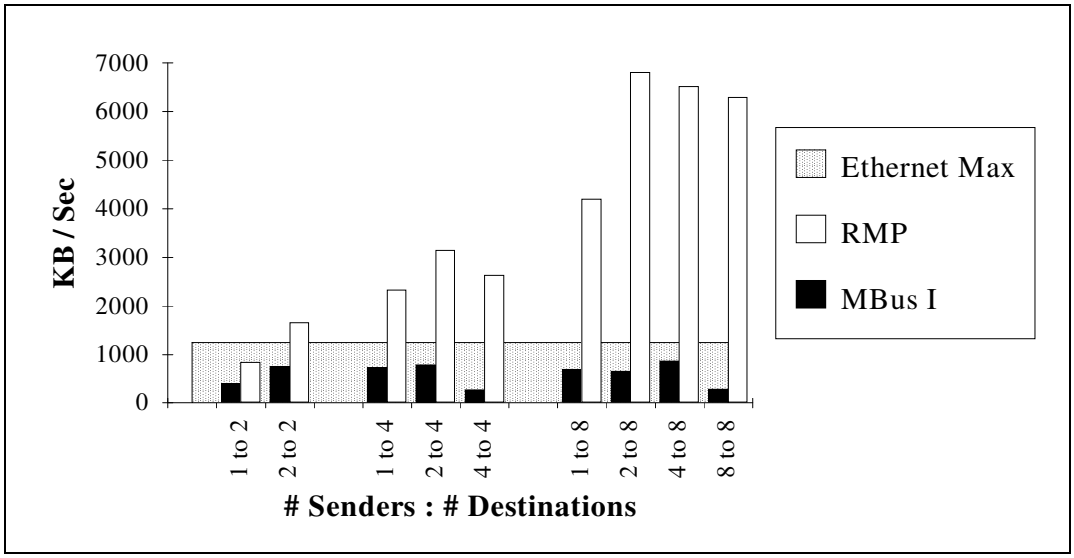

Figure 5.1: Aggregate Throughput (KB/sec)

In graph 2, we see the single sender throughput plotted against the number of destinations. The single sender throughput is equal to the aggregate throughput divided by the number of destinations. To be compatible with other published figures, in these tests the source was separate from the destinations. Data from the MBusI is included for comparison. The performance of all applications (such as UDP ISIS, Sun ToolTalk, the MBusI, and RPC) that do not use hardware broadcast or multicast drops off as a factor of 1/N. The graph is plotted with a logarithmic axis for throughput to better show this limitation, which is a fundamental limit of the network. In contrast, RMP stays roughly constant regardless of the number of destinations. RMP breaks the fundamental unicast limit for two destinations, so has fundamentally higher throughput in this environment for all groups of more than one destination. We have not included any numbers from other reliable multicast protocols because no fair comparisons on the same platforms have yet been made.

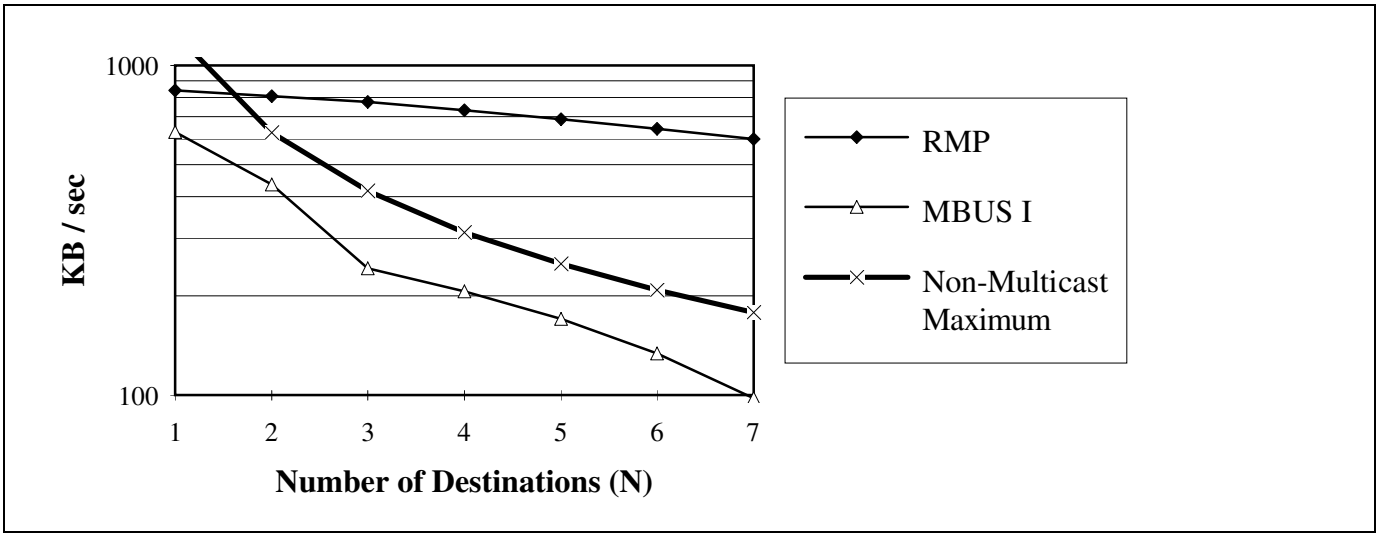

Figure 5.2: Single Sender Throughput (Logarithmic Scale, KB/sec)

In figure 5.3, we see the same factors, but with latency as the metric instead of throughput. Here again, the performance of RMP stays almost constant. While we have not yet been able to make fair comparisons to other protocols, the data that we have shows that the latency of protocols that do not take advantage of hardware multicast also scales linearly as a function of N [Clark94]. Note that this graph shows the case when fault tolerance is disabled and K=0. K does not affect throughput, but does increase latency. As K increases, the latency under heavy load will increase by a factor of roughly (K+2)/2. As the load decreases, the latency increases due to the increased time between packets and thus token passes.

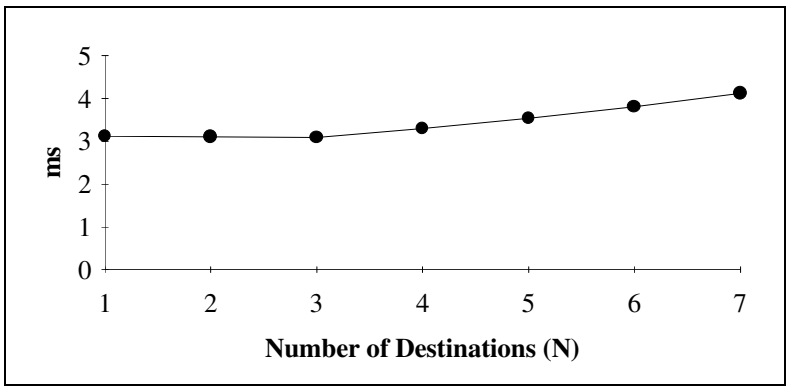

Figure 5.3: Single Sender Latency with K=0 (ms)

The worst case for RMP latency as compared to RPC latency is when the application needs to send a short message to a single destination. In this case, the round trip RPC should take about the same amount of time as RMP, but the message can be delivered twice as fast to the destination as the RMP message because RMP must wait for the ACK in order to provide

total ordering. It is a minor change to allow users to select some messages to be delivered immediately. This would not guarantee ordering or atomicity in the face of site failures, but would allow RMP to deliver messages immediately, making its latency to a single destination roughly equal to that of RPC and other low latency protocols. In this case, we expect RMP to have lower latency to multiple destinations than these protocols which scale linearly with N.

6.0 COMPARISON TO PREVIOUS WORK

Work as early as the V system [ChZw85] implements multicast communication between groups. The V system only implements "best effort" delivery semantics, and does not provide any ordering guarantees on messages.

The MBusI [Carroll93] was the original motivation for RMP. It provides a central server through which clients connect with TCP/IP streams, and an easy to use interface designed to ease the implementation of CSCW applications. It provides both total ordering of messages and reliable multicast, but has very limited scalability.

The Totem protocol [AMSM92] is perhaps closest to RMP in its approach, and has reported similar throughput levels as RMP under heavy load. It also uses a token ring approach, but only provides for a single ring. Totem avoids using any ACKs by only allowing the token holder to send a message and by having each message automatically pass the token. This provides high throughput under high load over a low latency network, but provides longer latency under low and asymmetrical loads. In addition, because it only allows a single sender to transmit at a time it will provide lower throughput over longer latency networks.

The ISIS system [BiJo87], [BSS91] is one of the pioneering protocols in this field. It provides causal ordering and, if desired, total ordering of messages on top of a reliable multicast protocol. The reliable multicast protocol requires separate acknowledgments from each destination, which limits performance. A new system that provides causal ordering on top of IP Multicasting has been implemented which is much more efficient than the old system [Clark94], and we hope to compare RMP and this new protocol soon.

The Psync protocol [PBS89] is an ingenious protocol that uses piggybacked ACKs to provide causal ordering of messages and detection of dropped packets. However, both it and the similar Trans [MSMA90] and Lansis [ADKM93] protocols require that all of the members of the group regularly transmit messages. The Trans protocol and the ToTo [DKM] protocol implemented on top of Lansis both provide total ordering of messages. These algorithms require that at least a majority of the group members be heard from before a message can be delivered, which causes latency to increase by at least

an order of magnitude. For example, for the ToTo protocol to send to a group of 8 destinations under heavy, periodic load from all sources (the best case), the latency is 23.8 ms. This increases to 114.1 ms for lightly loaded poisson sources.

The Multicast Transport Protocol (MTP) [AFM92] is an example of an asymmetric reliable multicast protocol. One site is the communication master which grants "tokens" to group members to allow them to send data. These tokens provide both flow control and total ordering of messages. This causes over dependency on the master, which limits both reliability and performance. MTP also relies exclusively on NACKs for error recovery, which limits reliability and requires extreme amounts of buffer space. Additionally MTP has not, as far as we are aware, yet been implemented.

The protocol by Crowcroft and Paliwoda [CrPa88] is one of the first protocols to propose reliable multicast over an internetwork which supports hardware multicast. The protocol provides different levels of reliability guarantees, and uses positive acknowledgments from all destinations for reliability. The paper analyzes the flooding problems that occur with simultaneous ACKs from many destinations and proposes a windowed flow control system, in some ways similar to that used in RMP, to limit these problems.

The protocol by Navaratnam, Chanson, and Neufeld [NCN88] is a centralized multicast protocol that uses a single token site to provide total ordering and reliability. It requires that each site send back a positive acknowledgment before the next packet can be sent. An implementation on the V-system takes 24.8 ms to send a multicast to four destinations. This protocol also is limited in reliability and scalability by the central server. The xAmp protocol [RoVe92] is distributed but also waits for ACKs from all destinations, and so will exhibit performance similar to NCN and ISIS.

The broadcast protocol proposed by Kaashoek et. al. [KTHB88] uses a central token site to serialize messages and NACKs for retransmissions. It piggybacks ACKs onto sent messages and has the token site regularly contact silent sites in order to limit buffer space. This protocol has reported very good latency (as low as 1.3 ms for a NULL packet) because it has been implemented on top of bare hardware. However, because each message must be transmitted twice it will fundamentally achieve lower throughput than RMP -- 600 KB/sec is a rough upper bound for a 1250 KB/sec Ethernet, as compared to 842 KB/sec for RMP. This will also limit the latency for larger messages; a 8KB packet in their protocol will spend a minimum of 13.1 ms on the Ethernet, as opposed to 6.7 ms for the message and ACK of RMP.

7.0 CONCLUSIONS

In this paper we have described the basic mechanisms of RMP, a fully distributed, totally ordered, reliable, atomic,

K-resilient fault tolerant multicast protocol. We have shown that RMP provides these features with very high performance. In a LAN, RMP provides much higher throughput to groups of two or more destinations than any protocol that does not take advantage of multicast or broadcast. In addition, it provides lower latency to groups of 3 or more destinations than most other protocols. Much work has gone into providing reliable multicast services with lower ordering guarantees because it was believed that the performance of a totally ordered multicast protocol was inherently low; RMP suggests that this is not the case, and that an efficient reliable multicast service can provide total ordering of messages for only a small latency penalty. Finally, because of its use of multiple token rings, a distributed name service, and its flow and congestion control algorithms, we expect RMP to scale gracefully and efficiently to larger groups spread over a large internetwork, and we hope to test this hypothesis in the near future.

8.0 ACKNOWLEDGMENTS

This work is supported in part by the NSF through grants CCR-9007195 and CCR-9108931, by the US Army Corps of Engineers, and by Sun Microsystems, Intel, Bull, Hewlett-Packard, DEC and Fujitsu/Open Systems Solutions. In addition, we would like to thank Alan Carroll for his work on the MessageBusI which inspired this whole project, Doug Bogia and Bill Tolone for their constant support, and Nick Maxemchuk and Jo-Mei Chang, whose work defined so much of RMP.

REFERENCES

[ADKM91] Y. Amir, D. Dolev, S. Kramer and D. Malki. "Transis: A Communication Sub-system for High Availability." Technical Report CS9113, Hebrew University of Jerusalem, Nov. 1991.

[AFM92] S. Armstrong, A. Freier, K. Marzullo. "Multicast Transport Protocol". RFC1301. (February, 1992).

[AMSM92] D. A. Agarwal, P. M. Melliar-Smith, and L. E. Moser. "Totem: A protocol for message ordering in a wide-area network." Proceedings of the First ISMM International Conference on Computer Communications and Networks (San Diego, CA, June 1992). pp. 1-5.

[ARP93] R. Aiello, E. Pagani, G. P. Rossi, "Design of a Reliable Multicast Protocol". Proceedings of IEEE INFOCOM '93 (San Francisco, March 1993). pp. 75-81.

[APR93] R. Aiello, G. P. Rossi, E. Pagani. "Casual Ordering in Reliable Group Communications." Proceedings of ACM SIGCOMM '93 (San Francisco, Sept. 1993). pp. 106-115.

[BiJo87] K. P. Birman and T. A. Joseph. "Reliable Communication in the Presence of Failures." ACM Transactions on Computing Systems. 5, 1 (Feb. 1987). pp. 47-76.

[BSS91] K. Birman, A. Schiper, P. Stephenson. "Lightweight Causal and Atomic Group Multicast." ACM Transactions on Computer Systems. 9, 3 (Aug. 1991). pp. 272-314.

[Carroll93] Alan Carroll. "ConversationBuilder: A Collaborative Erector Set." Ph.D. Thesis,
Department of Computer Science, University of Illinois, 1993.

[ChMa83] J. M. Chang and N. F. Maxemchuk. "A Broadcast Protocol for Broadcast Networks." Proceedings of GLOBCOM (Dec. 1983).

[Chang84] J. M. Chang. "Simplifying Distributed Database Systems Design by Using a Broadcast Network." Proceedings of SIGMOD (June 1984). pp. 223-233.

[ChMa84] J. M. Chang and N. F. Maxemchuk. "Reliable Broadcast Protocols." ACM Transactions on Computer Systems. 2, 3 (Aug. 1984). pp. 251-273.

[ChZw85] D.R. Cheriton and W. Zwaenepoel. "Distributed Process Groups in the V-Kernel." ACM Transactions on Computer Systems. 3, 2 (May 1985). pp. 77-107.

[Clark94] Tim Clark. ISIS work in progress. Cornell University, Department of Computer Science, 1994.

[CrPa88] J. Crowcroft, K. Paliwoda. "A Multicast Transport Protocol". Proceedings of ACM SIGCOMM '88. pp. 247-256.

[Deering89] S. Deering. "Host Extensions for IP Multicasting". STD 5. RFC1112. (August 1989).

[DKM93] D. Dolev, S. Kramer, D. Malki. "Early Delivery Totally Ordered Multicast in Asynchronous Environments." 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS). (Toulouse, France, June, 1993). pp. 544-553.

[Jacob88] V. Jacobson. "Congestion Avoidance and Control." Proceedings of ACM SIGCOMM '88 Symp (Sept. 1988). pp. 314-329.

[KTHB89] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, H. E. Bal. "An Efficient Reliable Broadcast Protocol." Operating Systems Review. 23, 4 (Oct. 1989), pp. 5-19.

[Lamp78] L. Lamport. "Time, Clocks, and the Ordering of Events in a Distributed System." Communications of the ACM. 21, 7

[LuGl90] S. W. Luan and V. D. Gligor. "A Fault Tolerant Protocol For Atomic Broadcast." IEEE Transactions on Parallel and Distributed Systems. 1, 3 (July 1990). pp. 271-285.

[MaCh84] N. F. Maxemchuk and J. M. Chang. "Analysis of the Messages Transmitted in a Broadcast Protocol." Proceedings of the International Computer Conference (Amsterdam, May 1984). pp. 1263-1267.

[MeBo76] R. M. Metcalf. and D. R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." Communications of the ACM. 19, 7 (July 1976). pp. 395-404.

[MMA90] P. M. Meillar-Smith, L. E. Moser, V. Agrawala. "Broadcast Protocols for Distributed Systems." IEEE Transactions on Parallel and Distributed Systems. 1, 1 (Jan. 1990). pp. 17-25.

[PBS89] L. L. Peterson, N. C. Buchholz, R.D. Schlichting. "Preserving and Using Context Information in Interprocess Communication". ACM Transactions on Computer Systems. 7, 3 (Aug. 1989). pp. 217-246.

[RaLi93] K. Ravindran and X. T. Lin. "Structural Complexity and Execution Efficiency of Distributed Application Protocols." Proceedings of ACM SIGCOMM '93. (San Francisco, Sept. 1993). pp. 160-169.

[Verissmo92] Verissmo. "xAMp: A Multi-primitive Group Communications Service. Proceedings of the 11th Symposium on Reliable Distributed Systems. (Houston, TX. Oct. 1992).