

A Reliable Dissemination Protocol for Interactive Collaborative Applications

Rajendra Yavatkar, James Griffioen, and Madhu Sudan
Department of Computer Science
University of Kentucky
Lexington, KY 40506
{raj,griff,madhu}@dcs.uky.edu
(606) 257-3961

ABSTRACT

The widespread availability of networked multimedia workstations and PCs has caused a significant interest in the use of collaborative multimedia applications. Examples of such applications include distributed shared whiteboards, group editors, and distributed games or simulations. Such applications often involve many participants and typically require a specific form of multicast communication called *dissemination* in which a single sender must reliably transmit data to multiple receivers in a timely fashion. This paper describes the design and implementation of a reliable multicast transport protocol called *TMTP* (Tree-based Multicast Transport Protocol). TMTP exploits the efficient best-effort delivery mechanism of IP multicast for packet routing and delivery. However, for the purpose of scalable flow and error control, it dynamically organizes the participants into a hierarchical control tree. The control tree hierarchy employs *restricted nacks with suppression* and an *expanding ring search* to distribute the functions of state management and error recovery among many members, thereby allowing scalability to large numbers of receivers. An Mbone-based implementation of TMTP spanning the United States and Europe has been tested and experimental results are presented.

KEYWORDS

Reliable Multicast, Transport Protocols, Mbone, Interactive Multipoint Services, Collaboration

INTRODUCTION

Widespread availability of IP multicast [6, 2] has substantially increased the geographic span and portability of collaborative multimedia applications. Example ap-

plications include distributed shared whiteboards [15], group editors [7, 14], and distributed games or simulations. Such applications often involve a large number of participants and are interactive in nature with participants dynamically joining and leaving the applications. For example, a large-scale conferencing application (e.g., an IETF presentation) may involve hundreds of people who listen for a short time and then leave the conference. These applications typically require a specific form of multicast delivery called *dissemination*. Dissemination involves 1xN communication in which a single sender must reliably multicast a significant amount of data to multiple receivers. IP multicast provides scalable and efficient routing and delivery of IP packets to multiple receivers. However, it does not provide the reliability needed by these types of collaborative applications.

Our goal is to exploit the highly efficient best-effort delivery mechanisms of IP multicast to construct a scalable and efficient protocol for reliable dissemination. Reliable dissemination on the scale of tens or hundreds of participants scattered across the Internet requires carefully designed flow and error control algorithms that avoid the many potential bottlenecks. Potential bottlenecks include host processing capacity [18] and network resources. Host processing capacity becomes a bottleneck when the sender must maintain state information and process incoming acknowledgements and retransmission requests from a large number of receivers. Network resources become a bottleneck unless the frequency and scope of retransmissions is limited. For instance, loss of packets due to congestion in a small portion of the IP multicast tree should not lead to retransmission of packets to all the receivers. Frequent multicast retransmissions of packets also wastes valuable network bandwidth.

This paper describes the design and implementation of a reliable dissemination protocol called *TMTP* (Tree-based Multicast Transport Protocol) that includes the following features:

1. TMTP takes advantage of IP multicast for efficient

packet routing and delivery.

2. TMTP uses an *expanding ring search* to dynamically organize the dissemination group members into a *hierarchical control tree* as members join and leave a group.
3. TMTP achieves scalable reliable dissemination via the hierarchical control tree used for flow and error control. The control tree takes the flow and error control duties normally placed at the sender and distributes them across several nodes. This distribution of control also allows error recovery to proceed independently and concurrently in different portions of the network.
4. Error recovery is primarily driven by receivers who use a combination of *restricted negative acknowledgements with nack suppression* and periodic positive acknowledgements. In addition, the tree structure is exploited to restrict the scope of retransmissions to the region where packet loss occurs; thereby insulating the rest of the network from additional traffic.

We have completed a user-level implementation of TMTP based on IP/UDP multicast and have used it for a systematic performance evaluation of reliable dissemination across the current Internet Mbone. Our experiments involved as many as thirty group members located at several sites in the US and Europe. The results are impressive; TMTP meets our objective of scalability by significantly reducing the sender's processing load, the total number of retransmissions that occur, and the end-to-end latency as the number of receivers is increased.

Background

A considerable amount of research has been reported in the area of group communication. Several systems such as the ISIS system [1], the V kernel [4], Amoeba, the Psynch protocol [17], and various others have proposed group communication primitives for constructing distributed applications. However, all of these systems support a general group communication model (NxN communication) designed to provide reliable delivery with support for atomicity and/or causality or to simply support an unreliable, unordered multicast delivery. Similarly, transport protocols specifically designed to support group communication have also been designed before [13, 5, 3, 19, 9]. These protocols mainly concentrated on providing reliable broadcast over local area networks or broadcast links. Flow and error control mechanisms employed in networks with physical layer multicast capability are simple and do not necessarily scale well to a wide area network with unreliable packet delivery.

Earlier multicast protocols used conventional flow and error control mechanisms based on a *sender-*

initiated approach in which the sender disseminates packets and uses either a *Go-Back-N* or a *selective repeat* mechanism for error recovery. If used for reliable dissemination of information to a large number of receivers, this approach has several limitations. First, the sender must maintain and process a large amount of state information associated with each receiver. Second, the approach can lead to a *packet implosion* problem where a large number of ACKs or NACKs must be received and processed by the sender over a short interval. Overall, this can lead to severe bottlenecks at a sender resulting in an overall decrease in throughput [18].

An alternate approach based on *receiver-initiated* methods [19, 15] shifts the burden of reliable delivery to the receivers. Each receiver maintains state information and explicitly requests retransmission of lost packets by sending negative acknowledgements (NACKs). Under this approach, the receiver uses two kinds of timers. The first timer is used to detect lost packets when no new data is received for some time. The second timer is used to delay transmission of NACKs in the hope that some other receiver might generate a NACK (called *nack suppression*).

It has been shown that the receiver-initiated approach reduces the bottleneck at the sender and provides substantially better performance [18]. However, the receiver-initiated approach has some major drawbacks. First, the sender does not receive positive confirmation of reception of data from all the receivers and, therefore, must continue to buffer data for long periods of time. The second and most important drawback is that the end-to-end delay in delivery can be arbitrarily large as error recovery solely depends on the timeouts at the receiver unless the sender periodically polls the receivers to detect errors [19]. If the sender sends a train of packets and if the last few packets in the train are lost, receivers take a long time to recover causing unnecessary increases in end-to-end delay. Periodic polling of all receivers is not an efficient and practical solution in a wide area network. Third, the approach requires that a NACK must be multicast to all the receivers to allow suppression of NACKs at other receivers and, similarly, all the retransmissions must be multicast to all the receivers. However, this can result in unnecessary propagation of multicast traffic over a large geographic area even if the packet losses and recovery problems are restricted to a distant but small geographic area¹. Thus, the approach may unnecessarily waste valuable bandwidth.

In this paper we present an alternative approach that achieves scalable reliable dissemination by reducing the processing bottlenecks of sender-initiated approaches

¹ Assume that only a distant portion of the Internet is congested resulting in packet loss in the area. One or more receivers in this region may multicast repeated NACKS that must be processed by all the receivers and the resulting retransmissions must also be forwarded to and processed by all the receivers.

and avoiding the long recovery times of receiver-initiated approaches.

OVERVIEW OF OUR APPROACH

Under the TMTP dissemination model, a single sender multicasts a stream of information to a *dissemination group*. A *dissemination group* consists of processes scattered throughout the Internet, all interested in receiving the same data feed. A session directory service (similar to the session directory *sd* from LBL [12]) advertizes all active dissemination groups.

Before a transmitting process can begin to send its stream of information, the process must create a dissemination group. Once the dissemination group has been formed, interested processes can dynamically join the group to receive the data feed. The dissemination protocol does not provide any mechanism to insure that all receivers are present and listening before transmission begins. Although such a mechanism may be applicable in certain situations, we envision a highly dynamic dissemination system in which receiver processes usually join a data feed already in progress and/or leave a data feed prior to its termination. Consequently, the protocol makes no effort to coordinate the sender and receivers, and an application must rely on an external synchronization method when such coordination is necessary.

For the purposes of flow and error control, TMTP organizes the group participants into a hierarchy of subnets or *domains*. Typically, all the group members in the same subnet belong to a domain and a single *domain manager* acts as a representative on behalf of the domain for that particular group. The domain manager is responsible for recovering from errors and handling local retransmissions if one or more of the group members within its domain do not receive some packets.

In addition to handling error recovery for the local domain, each domain manager may also provide error recovery for other domain managers in its vicinity. For this purpose, the domain managers are organized into a *control tree* as shown in Figure 1. The sender in a dissemination group serves as the root of the tree and has at most K domain managers as children. Similarly, each domain manager will accept at most K other domain managers as children, resulting in a tree with maximum degree K . The value of K is chosen at the time of group creation and registration and does *not* include local group members in a domain (or subnet). The degree of the tree (K) limits the processing load on the sender and the internal nodes of the control tree. Consequently, the protocol overhead grows slowly, proportional to the $\text{Log}_K(\text{Number_Of_Receivers})$.

Packet transmission in TMTP proceeds as follows. When a sender wishes to send data, TMTP uses IP multicast to transmit packets to the entire group. The transmission rate is controlled using a sliding window based protocol described later. The control tree ensures reliable delivery to each member. Each node of

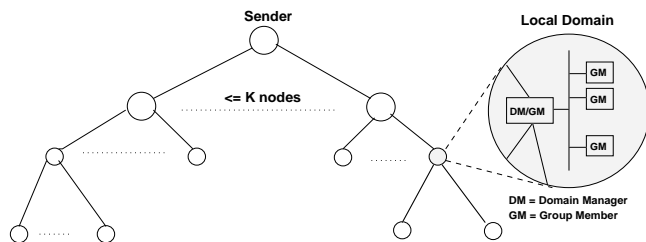


Figure 1: An example control tree with the maximum degree of each node restricted to K . Local group members within a domain are indicated by **GM**. There is no restriction on the number of local group members within a domain.

the control tree (including the root) is only responsible for handling the errors that arise in its immediate K children. Likewise, children only send periodic, positive acknowledgments to their immediate parent. When a child detects a missing packet, the child multicasts a NACK in combination with nack suppression. On the receipt of the NACK, its parent in the control tree multicasts the missing packet. To limit the scope of the multicast NACK and the ensuing multicast retransmission, TMTP uses the *Time-To-Live (TTL)* field to restrict the transmission radius of the message. As a result, error recovery is completely localized. Thus, a dissemination application such as a world-wide IETF conference would organize each geographic domain (e.g., the receivers in California vs. all the receivers in Australia) into separate subtrees so that error recovery in a region can proceed independently without causing additional traffic in other regions. TMTP's hierarchical structure also reduces the end-to-end delay because the retransmission requests need not propagate all the way back to the original sender. In addition, locally retransmitted packets will be received quickly by the affected receivers.

The control tree is self-organizing and does not rely on any centralized coordinator, being built dynamically as members join and leave the group. A new domain manager attaches to the control tree after discovering the closest node in the tree using an *expanded ring* search. Note that *the control tree is built solely at the transport layer and thus does not require any explicit support from, or modification to, the IP multicast infrastructure inside the routers.*

The following sections describe the details of the TMTP protocol.

GROUP MANAGEMENT

The session directory provides the following group management primitives:

CreateGroup(GName,CommType): A sender creates a new group (with identifier *GName*) using the CreateGroup routine. *CommType* specifies the type of communication pattern desired and may be ei-

ther *dissemination* or *concast*². If successful, CreateGroup returns an IP multicast address and a port number to use when transmitting the data.

JoinGroup(Gname): Processes that want to receive the data feed represented by *GName* call JoinGroup to become a member of the group. Join returns the transport level address (IP multicast address and port number) for the group which the new process uses to listen to the data feed.

LeaveGroup(Gname): Removes the caller from the dissemination group *GName*.

DeleteGroup(GName): When the transmission is complete, the sending process issues a DeleteGroup request to remove the group *GName* from the system. DeleteGroup also informs all participants, and domain managers that the group is no longer active.

CONTROL TREE MANAGEMENT

Each dissemination group has an associated control tree consisting of domain managers. Over the lifetime of the dissemination group, the control tree grows and shrinks dynamically in response to additions and deletions to and from the dissemination group membership. Specifically, the tree grows whenever the first process in a domain joins the group (i.e., a domain manager is created) and shrinks whenever the last process left in a domain leaves the group (i.e., a domain manager terminates).

There are only two operations associated with control tree management: *JoinTree* and *LeaveTree*. When a new domain manager is created, it executes the JoinTree protocol to become a member of the control tree. Likewise, domain managers that no longer have any local processes to support may choose to execute the LeaveTree protocol.

Figures 2 and 3 outline the protocols for joining and leaving the control tree. The join algorithm employs an *expanding ring search* to locate potential connection points into the control tree. A new domain manager begins an expanding ring search by multicasting a SEARCH_FOR_PARENT request message with a small time-to-live value (TTL). The small TTL value restricts the scope of the search to nearby control nodes by limiting the propagation of the multicast message. If the manager does not receive a response within some fixed timeout period, the manager resends the SEARCH_FOR_PARENT message using a larger TTL value. This process repeats until the manager receives a WILLING_TO_BE_PARENT message from one or more domain managers in the control tree. All existing domain managers that receive the SEARCH_FOR_PARENT message will respond with a

```
While (NotDone) {
  Multicast a SEARCH_FOR_PARENT msg
  Collect responses
  If (no responses)
    Increment TTL /* try again */
  Else
    Select closest respondent as parent
    Send JOIN_REQUEST to parent
    Wait for JOIN_CONFIRM reply
    If (JOIN_CONFIRM received)
      NotDone = False
    Else /* try again */
  }
}
```

(A) New Domain Manger Algorithm

```
Receive request message
If (request is SEARCH_FOR_PARENT)
  If (MAX_CHILDREN not exceeded)
    Send WILLING_TO_BE_PARENT msg
  Else
    /* Do not respond */
Else If (request is JOIN_REQUEST)
  Add child to the tree
  Send JOIN_CONFIRM msg
```

(B) Existing Domain Manger Algorithm

Figure 2: The protocol used by domain managers to join the control tree. A new domain manager performs algorithm (A) while all other existing managers execute algorithm (B).

²Although this paper focuses on dissemination, TMTP also supports efficient concast style communication[10].

```

If (I_am_a_leaf_manager)
  Send LEAVE_TREE request
  to parent
  Receive LEAVE_CONFIRM
  Terminate
Else /* I am an internal manager */
  Fulfill all pending obligations
  Send FIND_NEW_PARENT message to children
  Receive FIND_NEW_PARENT reply from all children
  Send LEAVE_TREE request to parent
  Receive LEAVE_CONFIRM
  Terminate

```

Figure 3: The algorithm used to leave the control tree after the last local group member terminates.

WILLING_TO_BE_PARENT message unless they already support the maximum number of children. The new domain manager then selects the closest domain manager (based on the TTL values) and directly contacts the selected manager to become its child. For each domain, its manager maintains a *multicast radius* for the domain, which is the TTL distance to the farthest child within the domain. The domain manager keeps the children informed of the current multicast radius. As described later in the description of the error control part of TMTP, both parent and its children in a domain use the current multicast radius to restrict the scope of their multicast transmissions.

Before describing the LeaveTree protocol, note that a domain manager typically has two types of children. First, a domain manager supports the group members that reside within its local domain. Second, a domain manager may also act as a parent to one or more children domain managers. We say a manager is an *internal manager* of the tree if it has other domain managers as children. We say a manager is a *leaf manager* if it only supports group members from its local domain.

A domain manager may only leave the tree after its last local member leaves the group. At this point, the domain manager begins executing the LeaveTree protocol shown in Figure 3. The algorithm for leaf managers is straightforward. However, the algorithm for internal managers is complicated by the fact that internal managers are a crucial link in the control tree, continuously servicing flow and error control messages from other managers, even when there are no local domain members left. In short, a departing internal node must discontinue service at some point and possibly coordinate children with the rest of the tree to allow seamless reintegration of children into the tree. Several alternative algorithms can be devised to determine when and how service will be cutoff and children reintegrated. The level of service provided by these algorithms could range from “unrecoverable interrupted service” to “temporarily interrupted service” to “uninterrupted service”. Our current implementation provides “probably unin-

terrupted service” which means children of the departing manager continue to receive the feed while they reintegrate themselves into the tree. However, errors that arise during the brief reintegration time might not be correctable. We are still investigating alternatives to this approach.

After a departing manager has fulfilled all obligations to its children and parent, the departing manager instructs its children to find a new parent. The children then begin the process of joining the tree all over again. Although we investigated several other possible algorithms, we chose the above algorithm for its simplicity. Other, more static algorithms, such as requiring orphaned children to attach themselves to their grandparents, often result in poorly constructed control trees. Forcing the children to restart the join procedure ensures that children will select the closest possible connection point. Other more complex dynamic methods can be used to speed up the selection of the closest connection point but, in our experience, the performance of our simple algorithm has been acceptable.

DELIVERY MANAGEMENT

TMTP couples its packet transmission strategy with a unique tree-based error and flow control protocol to provide efficient and reliable dissemination. Conventional flow and error control algorithms employ a sender- or receiver-initiated approach. However, using the control tree, TMTP is able to combine the advantages of each approach while avoiding their disadvantages. Logically, TMTP’s delivery management protocol can be partitioned into three components: data transmission, error handling, and flow control. The following sections address each of these aspects.

The Transmission Protocol

The basic transmission protocol is quite simple and is best described via a simple example. Assume a sender process S has established a dissemination group X and wants to multicast data to group X. S begins by multicasting data to the $\langle IP_multicast_addr, port_no \rangle$ representing group X. The multicast packets travel directly to all group members via standard IP multicast. In addition, all the domain managers in the control tree listen and receive the packets directly.

As in the sender-initiated approach, the root S expects to receive positive acknowledgments in order to reclaim buffer space and implement flow control. However, to avoid the *ack implosion* problem of the sender initiated approach, the sender does not receive acknowledgments directly from all the group members and, instead, receives ACKs only from its K immediate children. Once a domain manager receives a multicast packet from the sender, it can send an acknowledgment for the packet to its parent because the branch of the tree the manager represents has successfully received the packet (even though the individual members may not have received the packet). That is, a domain manager

does not need to wait for ACKs from its children in order to send an ACK to the parent. In addition, each domain manager only periodically sends such ACKs to its parent. This feature substantially reduces ACK processing at the sender (and each domain manager).

Error Control

Before describing the details of TMTP's error control mechanism we must define an important concept called *limited scope multicast* messages. A limited scope multicast restricts the scope of a multicast message by setting the TTL value in the IP header to some small value which we call the multicast radius. The appropriate multicast radius to use is obtained from the expanding ring search that domain managers use to join the tree. Limited scope multicast messages prevent messages targeted to a particular region of the tree from propagating throughout the entire Internet.

TMTP employs error control techniques from both sender and receiver initiated approaches. Like the sender initiated approach, a TMTP traffic source (sender) requires periodic (unicast) positive acknowledgements and uses timeouts and (limited scope multicast) retransmissions to ensure reliable delivery to all its immediate children (domain managers). However, in addition to the sender, the domain managers in the control tree are also responsible for error control after they receive packets from the sender. Although the sender initially multicasts packets to the entire group, it is the domain manager's responsibility to ensure reliable delivery. Each domain manager also relies on periodic positive ACKs (from its immediate children), timeouts, and retransmissions to ensure reliable delivery to its children. When a retransmission timeout occurs, the sender (or domain manager) assumes the packet was lost and retransmits it using IP multicast (with a small TTL equal to the multicast radius for the local domain so that it only goes to its children).

In addition to the sender initiated approach, TMTP uses *restricted NACKs with NACK suppression* to respond quickly to packet losses. When a receiver notices a missing packet, the receiver generates a negative acknowledgment that is multicast to the parent and siblings using a restricted (small) TTL value. To avoid multiple receivers generating 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 sibling during the delay period, it suppresses its own NACK. This technique substantially reduces the load imposed by NACKs. When a domain manager receives a NACK, it immediately responds by multicasting the missing packet to the local domain using a limited scope multicast message.

Flow Control

TMTP achieves flow control by using a combination of rate-based and window-based techniques. The rate-based component of the protocol prohibits senders from

transmitting data faster than some predefined maximum transmission rate. The maximum rate is set when the group is created and never changes. Despite its static nature, a fixed rate helps avoid congestion arising from bursty traffic and packet loss at rate-dependent receivers while still providing the necessary quality-of-service without excessive overhead.

TMTP's primary means of flow control consists of a window-based approach used for both dissemination from the sender and retransmission from domain managers. Within a window, senders transmit at a fixed rate.

TMTP's window-based flow control differs slightly from conventional point-to-point window-based flow control. Note that retransmissions are very expensive because they are multicast. In addition, transient traffic conditions or congestion in one part of the network can put backpressure on the sender causing it to slow the data flow. To oversimplify, TMTP avoids both of these problems by partitioning the window and delaying retransmissions as long as possible. This increases the chance of a positive acknowledgement being received and it also allows domain managers to rectify transient behavior before it begins to cause backpressure.

TMTP uses two different timers to control the window size and the rate at which the window advances. $T_{retrans}$ defines a timeout period that begins when the first packet in a window is sent. Since the transfer rate is fixed, $T_{retrans}$ also defines the window size. A second timer, T_{ack} , defines the periodic interval at which each receiver is expected to unicast a positive ACK to its parent.

The sender specifies the value of T_{ack} based on the RTT to its farthest child. $T_{retrans}$ is chosen such that $T_{retrans} = n \times T_{ack}$, where n is an integer, $n \geq 2$. Both $T_{retrans}$ and T_{ack} are fixed at the beginning of transmission and do not change. A sender must allocate enough buffer space to hold packets that are transmitted over the $T_{retrans}$ period.

Figure 4 illustrates the windowing algorithm graphically. The sender starts a timer and begins transmitting data (at a fixed rate). Consider the packets transmitted during the first T_{ack} interval. Although the sender should see a positive ACK at time T_{ack} , the sender does not require one until time $T_{retrans}$. Instead, the sender continues to send packets during the second and third interval. After $T_{retrans}$ amount of time, the timer expires. At this point, the sender retransmits all unACK'd packets that were sent during the first T_{ack} interval. Retransmissions continue until all packets in the T_{ack} interval are acknowledged at which point the window is advanced by T_{ack} . On the receiving end, packets continue to arrive without being acknowledged until T_{ack} amount of time has expired³.

³However, a receiver may generate a *restricted NACK* as soon as it detects a missing packet.

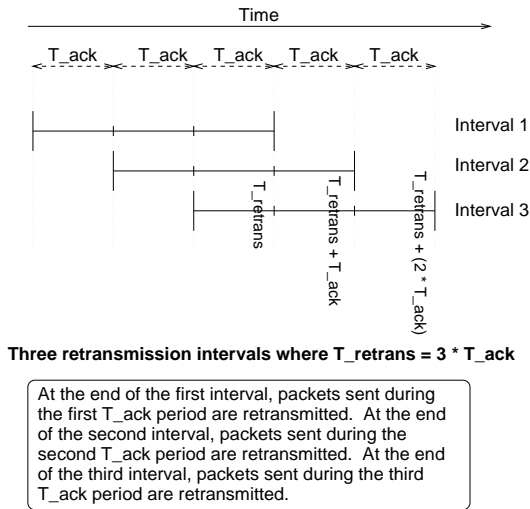


Figure 4: Different Stages in Sending Data

A domain manager must continue to hold packets in its buffer until all of its children have acknowledged them. If the children fail to acknowledge packets, the domain manager's window will not advance and its buffers will eventually fill up. As a result, the domain manager will drop and not acknowledge any new data from the sender, thereby causing backpressure to propagate up the tree which ultimately slows the flow of data.

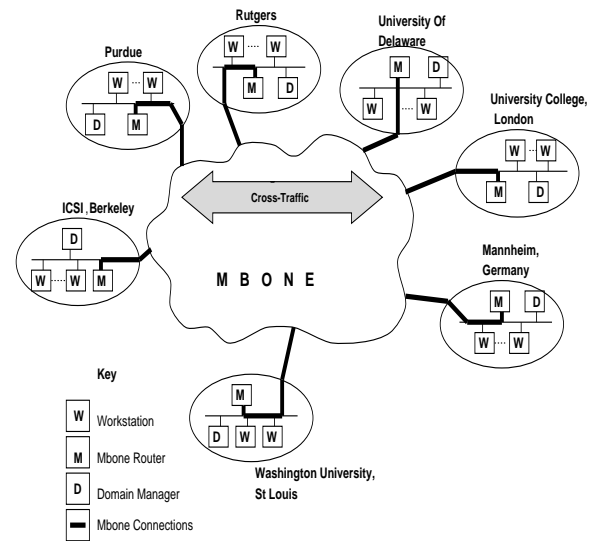
There are three reasons for using multiple T_{ack} intervals during a retransmission timeout interval ($T_{retrans}$). First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. First, by requiring more than one positive ACK during the retransmission interval, TMTP protects itself from spurious retransmissions arising from lost ACKs. Second, a larger retransmission interval gives receivers sufficient time to recover missing packets using receiver-initiated recovery when only one (or a few) packets in a window are lost. This avoids unnecessary multicast retransmissions of a window full of data. Third, multiple T_{ack} intervals during the retransmission interval provide sufficient opportunity for a domain manager to recover from transient network load in its part of the subtree without unnecessarily applying backpressure to the sender.

We have chosen the value of the multiplying factor n to be 3 based on empirical evidence; the appropriate value depends on several factors including expected error rates, variance in RTT, and expected length of the intervals with transient, localized congestion. Further study is necessary to determine whether value of n should be chosen dynamically using an adaptive algorithm.

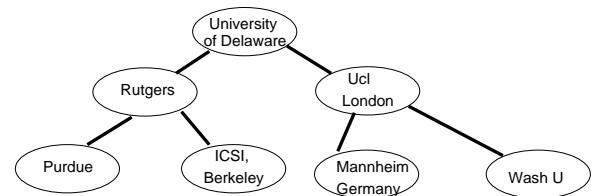
RESULTS

The Test Environment

Figure 5a illustrates the environment in which the experiments were run. Our tests involved seven geograph-



(5a) The Internet Mbone Used



(5b) The Control Tree

Figure 5: Figure 5a shows the test environment consisting of seven geographically distant sites connected by the Mbone. Figure 5b shows the corresponding control tree configuration used in the experiments.

ically distinct Internet Mbone sites across the United States and Europe: Washington University in St. Louis, Purdue University, the International Computer Science Institute at Berkeley, Rutgers University, the University of Delaware, University College at London, and the University of Mannheim in Germany. All of our experiments were conducted using standard IP multicast across the Internet Mbone and thus experienced real Internet delays, congestion, and packet loss.

As a point of comparison, we implemented a standard sender-initiated reliable multicast transport protocol both with and without window-based flow control (called *WIN_BASEP* and *BURST_BASEP* respectively). Under both protocols, the sender maintains state information for all receivers, expects positive ACKs from each receiver, and uses timeouts and global multicast retransmissions to recover from missing acknowledgments. The two BASEP protocols illustrate the performance bottlenecks related to processor load and end-to-end latency. All three protocols used the same packet size (1 Kbytes). TMTP and WIN_BASEP used a window size of 5. TMTP uses a transmission rate of 10 packets per

second, while both BASEP protocols transmit packets as fast as possible (up to the window size in the case of WIN_BASEP). Both BASEP protocols set the retransmission timeout period to be twice the RTT to the farthest site (approx. 2 seconds in our tests). TMTP uses a retransmission period of $T_{retrans} = N \times T_{ack}$. T_{ack} is dynamically set based on the RTT to the farthest group member (approximately 1.1 seconds for our tests). After some preliminary evaluation of different setting for N , our empirical results indicated that $N = 3$ provides sufficient time for local domains to recover without delaying acks unnecessarily or consuming too much buffer space. Consequently, $T_{retrans}$ was approximately 3.3 seconds in our tests. The following sections describe the performance measures used and detail the actual experiments performed.

Performance Measures

To evaluate the performance of our protocol, we identified two important measures of performance: *end-to-end delay* and *processing load*. In addition, we monitored the total number of retransmissions to estimate the amount of network traffic generated by TMTP.

From the application’s perspective, the primary concern is the delay in reliably delivering the entire data feed (e.g., video, audio, or file data) to the multiple recipients of the group. To measure the end-to-end delay, we required that each receiving application send back a single positive acknowledgment (a GOT_IT message) to the sending application when the entire data transmission was complete. The sending application then calculated the end-to-end delay as the time between the beginning of the transmission and the time at which the last group member’s final GOT_IT message is received.

From the network’s perspective, the primary concern is network load and scalability of the algorithm. If the protocol provides low end-to-end delay but consumes large amounts of network resources, the protocol will not scale well, congesting the Internet by consuming shared resources required by other Internet users. There are two aspects to network load: processing load and bandwidth consumption. To measure the processing load at the sender, receivers, and domain managers, we monitored the following processing activities:

- receiving and processing a selective positive acknowledgment
- receiving and processing a negative acknowledgment
- handling a timer event (such as a retransmission timeout)
- performing a retransmission

Because it is hard to measure the amount of processing time needed for each of the events listed above (and highly dependent on the operating system and architecture), we have chosen to simply count the total number

of such events at the sender to estimate the processing load generated by a protocol.

The second important measure of network load is bandwidth consumption. The precise amount of bandwidth consumed by each protocol is much harder to quantify since we were unable to collect traces of traffic across the Mbone to determine the number of links traversed and the amount of bandwidth consumed over each link. However, our results indicate that TMTP generated far fewer retransmissions than the BASEP protocols, and most TMTP retransmissions are local to a particular domain. For example, under the BASEP protocols most timeouts/retransmissions occurred as a result of dropped ACKs. TMTP’s hierarchy substantially reduced the number of lost ACKs, experiencing only 6 local retransmissions totaled across all domain managers (four occurring concurrently) as opposed to 9 global retransmission for BURST_BASEP (out of thirty 1K messages).

Experiments Performed

Each of our experiments measured the performance of a single dissemination group consisting of many processes evenly distributed across the seven sites pictured in Figure 5a. The total number of processes acting as receivers was varied between five and thirty processes. The five process case used only five domains while all other cases used seven domains. In each experiment, a sending process created a dissemination group, waited for the receiving processes to join the group and organize their domains into a control tree. Multiple tree configurations are possible depending on when, and in what order, domain managers join the tree. However to ensure consistency across tests, we held the tree configuration constant across all tests (see Figure 5b). After all receivers joined the group, the sender disseminated a data file to the group, and then waited for the final GOT_IT message from all receivers. The values reported for each test are averaged over at least five runs taken during weekdays at roughly the same time so that the observed Internet traffic conditions remain similar across tests.

To gauge the scalability of the protocol, we monitored the changes in processing load at the senders, receivers, and managers. To measure the effective throughput, we measured the changes in end-to-end delay as perceived by the sender. Both processing load and end-to-end delay were recorded under a variety of workloads. In the first set of tests, the sender transmitted a 30 Kbyte file to a varying number of receivers. The dissemination was considered complete when all the receivers correctly receive the entire file. In the second set of tests, the number of processes was fixed at 30 and we incrementally increased the file size from 3K to 30 Kbytes. The end-to-end delay is measured as the time between the beginning of the file transfer and the time at which the last group member’s final GOT_IT message is received.

To measure the processing load, we counted the total

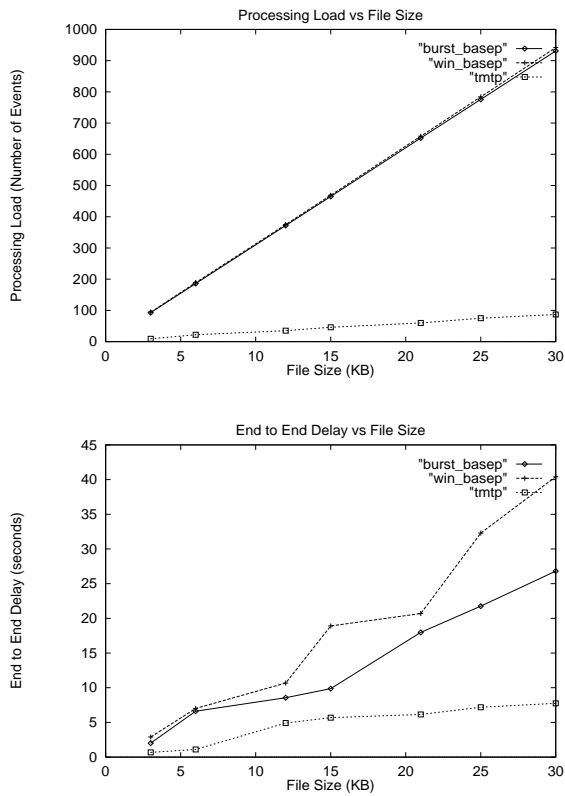


Figure 6: (a) Effect of the amount of data transmitted on the processing load. (b) Effect of the amount of data transmitted on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken with a dissemination group of size 30.

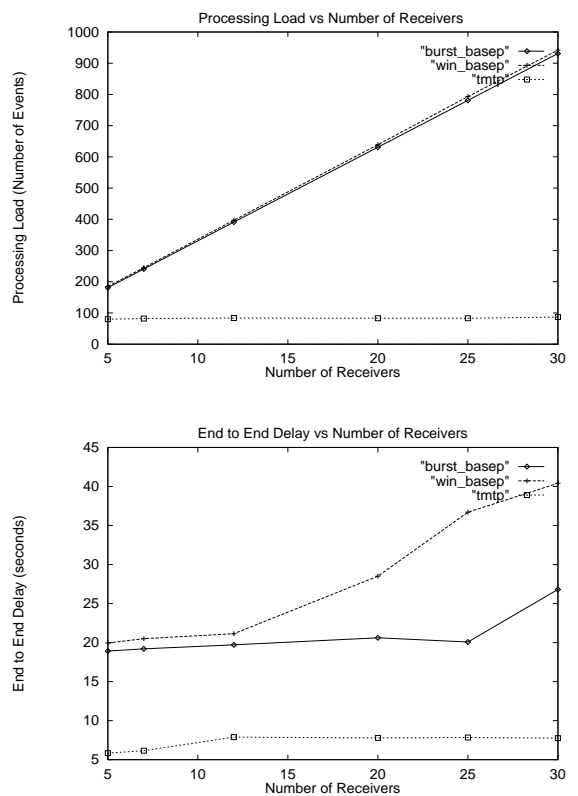


Figure 7: (a) Impact of group size (no. of receivers) on the processing load. (b) Impact of group size (no. of receivers) on the end-to-end delay. Figure b shows the time for the file transfer to complete at all the receivers. All measurements were taken for a dissemination of a 30 KB file.

number of events at the sender that contribute to the processing load. Similarly, we recorded the number of events at each domain manager. Figure 6 only shows the number of events processed at the sender. However, the balanced nature of our control tree meant the event processing load was spread equally among the sender and all domain managers. Consequently, the number of events processed at each domain manager is approximately the same as the number of events processed at the sender. Variations occurred based on the number of NACKs received.

Figures 6 and 7 show the results for each of the experiments performed. From these results we draw the following observations:

Impact of the Data Size

Figures 6a and 6b show how the file size affects the processing load and end-to-end delay. As the file size increases, the number of packets transmitted increases, thereby increasing the number of events (such as ACK/NACK processing or timer events) that affect the processing load at the sender (or a domain manager). Similarly, end-to-end delay is likely to increase due to time needed to deliver all the packets and due to increased probability of packet loss.

As the plots show, both the versions of the BASEP benchmark protocol show a significant increase in the processing load at the sender and the end-to-end delay. Note that the delay for WIN_BASEP (with flow control) is actually higher than BURST_BASEP (no flow control). This occurs because the WIN_BASEP sender expects acknowledgments from all its receivers before advancing the flow control window.

In the case of TMTP, the processing load shows only a small increase because the work is distributed among many nodes in the control tree. Consequently, the sender does not have to process acknowledgments or retransmission requests from all the receivers. TMTP's end-to-end delay is substantially lower than that of the BASEP protocols for all file sizes. Although all three protocols experience an increase in end-to-end delay resulting from larger data transmissions, packet losses, and retransmissions, TMTP's end-to-end delay rises at a significantly lower rate than that of the BASEP protocols. This occurs because error recovery in TMTP proceeds concurrently in different parts of the control tree rather than sequentially as in the BASEP cases.

Impact of the Group Size

Figures 7a and 7b show how the number of receivers (group size) affects the processing load and end-to-end delay.

Again, as the plots show, two versions of BASEP protocol show sharp increases in processing load with increase in number of receivers because the sender solely shoulders the responsibility for processing acknowledgments and retransmission requests (or timeouts) from each receiver. In the case of TMTP, the processing load

at the sender (and each domain manager) is limited by the maximum number of immediate children in the control tree and, therefore, shows almost no increase as the number of receivers is increased. This results from the fact that the number of domains remains at seven for more than seven receivers. An increase in the number of domains participating in the dissemination group would cause a slight load increase on domain managers who adopt the new children.

Figure 7a shows that the end-to-end delay of both BASEP protocols is significantly higher than that of TMTP. The primary reason for this difference stems from TMTP's receiver-initiated capabilities that respond to and correct errors quickly. In contrast, the BASEP protocols will not correct an error until a retransmission timeout occurs.

In the case of TMTP end-to-end delays increase gradually because error recovery proceeds concurrently and independently in different parts of the control tree as explained earlier. Figure 7b shows that the end-to-end delay stabilizes to almost a constant value beyond a point. That is, to a small extent, an artifact of our tests in which we did not add any new domains to the control tree, but rather only added new processes to the existing tree. However, in other experiments involving varying number of domains, we have observed a similar trend of gradual increase in end-to-end delays with increasing number of receivers at additional domains.

RELATED WORK

A considerable amount of work has been reported in the literature regarding reliable multicast [13, 5, 3, 18, 12, 1, 4, 19, 15, 8, 11, 16]. Most of the earlier approaches achieve reliable delivery using a *sender-initiated* approach which is not suitable for large-scale, delay-sensitive, reliable dissemination.

Pingali and others[18] recently analyzed and compared both sender- and receiver-initiated approaches to demonstrate the limitations of the sender-initiated approach for large-scale dissemination. Our work is also motivated by similar observations, but combines the elements of both the approaches to achieve fast, local error recovery.

The reliable multicast protocol used in LBL's whiteboard tool (*wb*) [15, 8] and the log-based reliable multicast protocol [11] are two recent examples of the receiver-initiated approach for reliable delivery. Unlike TMTP, these protocols do not combine sender-initiated with receiver-initiated approaches and differ significantly in flow control mechanisms and buffering mechanisms. Our work is related to the *wb* work in that the *wb* protocol also uses a *NACKs with NACK suppression* mechanism. The *wb* protocol reduces state management overhead and achieves high degree of fault tolerance by relying solely on the receiver to recover from a packet loss. However, the protocol incurs the overhead of global (sometimes redundant) multicasts; a receiver

multicasts a *repair request* to the entire group and one or more receivers in the group who have missing data (irrespective of their proximity to the complaining receiver) will multicast the missing packet(s) to the entire group even though the loss (or congestion) is restricted to a small region of the group topology. TMTP restricts the scope of multicast NACKs and retransmissions to the local domain to avoid generating redundant multicast transmissions over a wider region. Similar to TMTP, receivers using the wb protocol delay their NACKs to suppress duplicate NACKs in case another receiver multicasts a NACK. However, in the wb protocol, each receiver delays its NACK (and the response) by a random amount that depends on the RTT to the original sender. This can result in higher latency in recovering from packet losses. TMTP, on the other hand, uses localized recovery and, thus, the amount of random delay is bounded by the largest RTT between the local domain manager and one of the receivers in the domain. In addition, TMTP allows recovery from different errors to proceed concurrently in different domains to allow faster and efficient recovery.

Cheriton et. al.[8] have recently proposed a collection of strategies (called log-based receiver-reliable multicast or LRBM) for achieving large-scale, reliable multicast delivery. Some elements of LRBM are similar to TMTP's mechanisms to some extent. LRBM uses a hierarchy of logging servers with a primary log server responsible for sending positive acknowledgments to the multicast source. The primary log server stores the packets as long as an application desires and the receivers must recover from errors by contacting a logging server. A secondary server at each site may log received packets and satisfy local retransmission requests to reduce load on the primary server. Deployment of LRBM in the Internet is necessary to evaluate its performance in achieving reliable delivery in a wide area network environment.

Recently Paul et. al. [16] have proposed and are examining three multicast alternatives with features similar to those of TMTP. In contrast to these protocols, TMTP uses a multi-level hierarchical control tree and a dynamic group management protocol, as opposed to a static two-level hierarchy, to evenly distribute the protocol processing load and allow finer grained independent and concurrent error recovery. TMTP targets a best-effort multicast system such as IP multicast rather than an ATM-like network with allocated resources. TMTP imposes no additional load on network-level routers and requires no modification to the network-level routers, but yet incorporates both local retransmissions and combined acknowledgments. Furthermore, TMTP employs receiver-initiated recovery techniques (*restricted negative acknowledgments with nack suppression*) combined with periodic positive acknowledgments) and a unique flow control mechanism that can provide quick recovery from transient congestion and lost acknowledg-

ments.

CONCLUSION

Based on our experimental results, we believe that TMTP can scale well to provide reliable delivery on a large scale without sacrificing end-to-end latency. Under TMTP, the network processing load increases very gradually, indicating that the protocol will scale well as the number of receivers increases. Moreover, TMTP provides significantly better application-level throughput because of the concurrency resulting from local retransmissions as shown by the end-to-end measurements.

References

- [1] Ken Birman and Thomas Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Feb 1987.
- [2] S. Casner and S. Deering. First IETF Internet Audiocast. *ACM Computer Communication Review*, 22(3):92–97, July 1992.
- [3] J. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.
- [4] David. R. Cheriton and W. Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.
- [5] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proceedings of ACM SIGCOMM '88*, pages 247–256, August 1988.
- [6] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.
- [7] Prasun Dewan. A Guide to Suite: Version 1.0. Technical Report SERC-TR-60-P, Software Engineering Research Center, Purdue University, West Lafayette, IN, February 1990.
- [8] S. Floyd, V. Jacobsen, S. McCanne, C-G Liu, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *sigcomm95*, 1995. to appear.
- [9] I. Gopal and J. Jaffe. Point-to-multipoint Communication over Broadcast Links. *IEEE Transactions on Communications*, 32, September 1984.
- [10] James Griffioen and Rajendra Yavatkar. Clique: A Toolkit for Group Communication using IP Multicast. In *Proceedings of the Workshop on Services in Distributed and Networked Environments*, June 1994.

- [11] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *sigcomm95*, 1995. to appear.
- [12] Van Jacobson. *SD: Session Directory*. Lawrence Berkeley Laboratory, March 1993.
- [13] M Frans Kaashoek, A.S. Tanenbaum, S.F. Hummel, and H.E. Bal. An Efficient Reliable Broadcast Protocol. *ACM Operating Systems Review*, 23(4), October 1989.
- [14] Amit Mathur and Atul Prakash. Protocols for integrated audio and shared windows in collaborative systems. In *Proceedings of ACM Multimedia '94*, October 1994.
- [15] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Technical report, Real Time Systems Group, Lawrence Berkeley Laboratory, Berkeley, CA, September 1992. unpublished report.
- [16] S. Paul, K. Sabnani, and D. Kristol. Multicast Transport Protocols for High Speed Networks. In *IEEE Int. Conf. on Network Protocols*, 1994 Oct.
- [17] L. Peterson, N. Buchholz, and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.
- [18] Sridhar Pingali, Don Towsley, and James F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of ACM SIGMETRICS '94*, volume 14, pages 221–230, 1994.
- [19] S. Ramakrishnan and B.N. Jain. A Negative Acknowledgement Protocol with Periodic Polling Protocol for Multicast over Lans. In *Proceedings of IEEE INFOCOMM '87*, pages 502–511, March-April 1987.