RMTP: A Reliable Multicast Transport Protocol

John C. Lin*
Department of Computer Sciences
Purdue University
West Lafayette, Indiana 47907
lin@cs.purdue.edu

Sanjoy Paul
AT&T Bell Laboratories
Holmdel, New Jersey 07733
sanjoy@research.att.com

(*Proceedings of IEEE INFOCOM '96, March 1996, pp. 1414-1424.*)

Abstract

This paper describes the design and implementation of a multicast transport protocol called RMTP. RMTP provides sequenced, lossless delivery of bulk data from one sender to a group of receivers. RMTP achieves reliability by using a packet based selective repeat retransmission scheme, in which each acknowledgment (ACK) packet carries a sequence number and a bitmap. ACK handling is based on a multi-level hierarchical approach, in which the receivers are grouped into a hierarchy of local regions, with a Designated Receiver (DR) in each local region. Receivers in each local region periodically send ACKs to their corresponding DR, DRs send ACKs to the higher-level DRs, until the DRs in the highest level send ACKs to the sender, thereby avoiding the ACK-implosion problem. DRs cache received data and respond to retransmission requests of the receivers in their corresponding local regions, thereby decreasing end-to-end latency and improving resource usage. This paper also provides the measurements of RMTP's performance with receivers located at various sites in the Internet.

1 Introduction

Multicasting provides an efficient way of disseminating data from a sender to a group of receivers. Instead of sending a separate copy of the data to each individual receiver, the sender just sends a single copy to all the receivers. The technique used for distributing the data to individual receivers is transparent to the sender. In fact, there is an underlying *multicast delivery system* which forwards the data from the sender to every receiver. Conceptually, the multicast delivery system forms a multicast tree connecting the sender and the receivers, with the sender as the root node and the receivers as the leaf nodes (see Figure 1). Data generated by the sender flows through the multicast tree, traversing each tree edge exactly once. When receivers join or leave a multicast group, the multicast tree is dynamically reconfigured.

Multicasting of real-time multimedia information has recently been receiving a great deal of attention [12, 14, 15]. Also, researchers have demonstrated multicasting real-time data over the

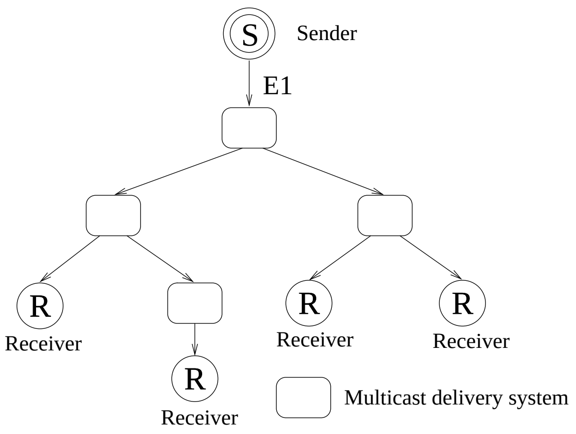


Figure 1: Illustration of a multicast delivery system forwarding data from a sender to a group of receivers

Internet using a multicast delivery system called *MBone* [2, 4]. Since most real-time applications can tolerate some data loss but cannot tolerate the delay associated with retransmissions, they either accept some loss of data or use forward error correction for minimizing such loss. The main objective of multicast protocols for transporting real-time data is to guarantee quality of service by bounding end-to-end delay at the cost of reliability. In contrast, the objective of our protocol is to guarantee *complete reliability* at the expense of delay.

There are several multicast applications which require sequenced and lossless delivery of data. Such applications cannot tolerate data loss, but can tolerate delays due to retransmissions. Examples include: a software house distributing the latest release of a software to its clients, a financial institution disseminating market data to its subscribers, a publisher distributing books electronically to bookstores, and a hospital sending patients' medical image data to physicians in other hospitals.

For such applications, a separate TCP [19] connection can be established between the sender and each receiver, delivering every packet individually to each receiver. However, such a scheme is very inefficient because every packet needs to be duplicated as many times as the number of receivers. For example, in Figure 1,

*This work was supported in part by AT&T Bell laboratories and Uni-Forum Association.

when the sender uses TCP to transmit a packet to each receiver, the same packet traverses path E1 four times. However, when the sender *multicasts* a packet to all receivers, the packet traverses path E1 only once. Thus, for utilizing network resources efficiently, there is a need for a reliable multicast transport protocol.

Reliable multicast protocols are not new in the distributed systems environment [8, 9, 10, 13]. However, these protocols apply to local area networks and do not scale well in wide area networks, mainly because the entities involved in the protocol need to exchange several control messages for coordination among themselves. In addition, they do not address fundamental issues of acknowledgment implosion [11] and propagation delays in wide area networks.

The goal of this paper is to describe the design and implementation of a reliable multicast transport protocol for wide area networks. Called RMTP, the protocol provides sequenced, lossless delivery of bulk data from a sender to a group of receivers. The basic ideas of RMTP are derived from the Designated Status Protocol (DSP) described in [16]. In particular, RMTP uses the hierarchical approach and hence the notion of *Designated Receiver* from DSP to avoid the ACK-implosion problem. In addition, the periodic sending of status messages by RMTP receivers is adopted from [7, 16]. RMTP builds on top of a *best-effort* network layer such as IP [18]. We will describe RMTP in detail in the following sections.

The remainder of this paper is organized as follows. Section 2 discusses the features of RMTP and the assumptions in the design. Section 3 describes the design and implementation of RMTP. Section 4 describes how the MBone technology has been used to implement the multicast packet delivery system used by RMTP. Section 5 describes RMTP's performance with receivers located at various sites in the Internet. Section 6 discusses future work and summarizes the contributions of the paper.

2 RMTP Features and Assumptions

Given the desired goal of providing efficient reliable delivery of data to a potentially large and diverse group of receivers, some engineering efforts are needed. In particular, emphasis has been put for providing the features of: reliability, scalability, and heterogeneity.

Reliability means that all receivers receive an exact copy of the file transmitted by the sender. The main technique used for achieving reliability is periodic transmission of status by the receivers, and a selective-repeat retransmission mechanism. However, in case of network-related problems, such as partitions, RMTP does not guarantee reliability. In such cases, the receiving application is notified. In addition, RMTP does not ensure reliable delivery of data to receivers that voluntarily or involuntarily (due to a crash or network partitioning) leave a multicast group. We assume the existence of a *Session Manager* who is responsible for detecting such anomalies and taking necessary actions.

Three design features make RMTP scalable. First, the state information maintained at each multicast participant is independent of the number of participants. Therefore, when a receiver joins or leaves a multicast group, it does not affect the state information of the sender or other receivers. Second, RMTP uses a *receiver-driven* approach; it places the responsibility of ensuring sequenced, lossless data reception on each individual receiver. Thus, the sender is relieved of the burden of tracking the status of each receiver. Third, RMTP groups receivers into local regions and uses a Designated Receiver (DR) in each such region, so that the responsibilities of processing ACKs and performing retransmissions is distributed among several DRs and the sender. Receivers in each local region send their ACKs to the DR, who is responsible for caching packets received from the sender and retransmitting any packets lost in the region. Thus, RMTP avoids the ACK implosion problem when the receiver population becomes large. Currently, the DRs are statically chosen for a given multicast session. Ideally, the DRs should be selected dynamically based on the topology of the multicast tree. This issue is currently being investigated.

RMTP makes no assumption about the processing power or the link bandwidth at each receiving node. It uses windowed flow control with congestion avoidance to avoid overloading slow receivers and links with low bandwidth. It also allows users to set an upper bound on the sender's data transmission rate. We assume that the Session Manager is responsible for choosing and setting the maximum transmission rate. To cope with receivers with varying speed, RMTP takes a conservative approach by using the progress of slow receivers to determine how much more data needs to be transmitted by the sender. However, techniques to prevent a slow receiver from slowing down faster receivers are being investigated.

As we will see in the following sections, the design and implementation of RMTP is fairly simple.

3 Protocol Description

RMTP provides sequenced, lossless delivery of bulk data from one sender to a group of receivers. The sender divides the data to be transmitted into fixed-size data packets, with the exception of the last one. A data packet is identified by packet type `DATA`, and type `DATA_EOF` identifies the last data packet. The sender assigns each data packet a sequence number, starting from 0. A receiver periodically informs the sender about the packets it has correctly received by sending ACKs. An ACK includes a sequence number L and a bitmap V. Sequence number L indicates that a receiver has correctly received all the packets with sequence number less than L. A 0 in the bitmap corresponds to a missed (or incorrectly received) packet while a 1 corresponds to a correctly received packet. For example, an ACK with $L = 15$ and $V = 01101111$[1] indicates that the receiver has correctly received all the packets up to sequence number 14 and has requested for the retransmission of packets 15 and 18.

Table 1 lists the packet types currently defined in RMTP. Their function will be described in the following subsections.

[1]The most significant bit of the bitmap is on the right.

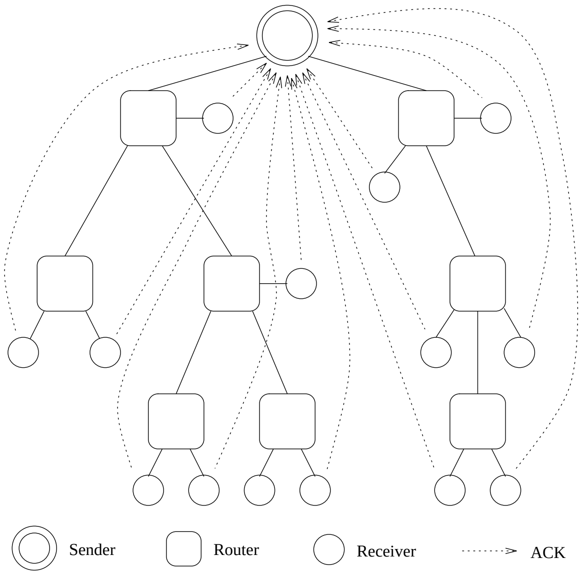


Figure 2: Illustration of a group of receivers sending ACKs to a sender

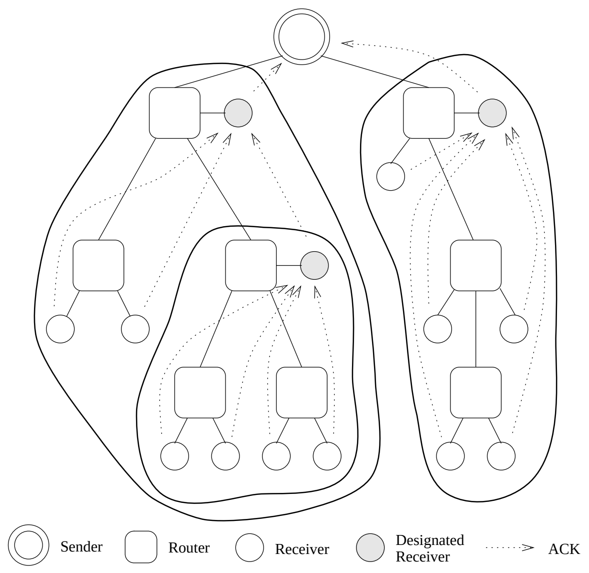


Figure 3: Using Designated Receivers to assist the sender in processing ACKs

Packet Types	
`ACK`	ACK packet
`ACK_TXNOW`	ACK - immediate transmission req.
`DATA`	Data packet
`DATA_EOF`	Last data packet
`RESET`	Packet to terminate a connection
`RTT_MEASURE`	Packet to measure round-trip time
`RTT_ACK`	ACK to `RTT_MEASURE` packet
`SND_ACK_TOME`	Packet for selecting an AP

Table 1: RMTP packet types

Connection Parameters	
W_r	receive window size in packets
W_s	send window size in packets
T_{dally}	delay after sending the last packet
T_{retx}	time interval to process retx requests
T_{rtt}	time interval to measure RTT
T_{sap}	time interval to send `SND_ACK_TOME`
T_{send}	time interval to send data packets
$Packet\_Size$	data packet size in octets
$Cache\_Size$	sender's in-memory data cache size
$CONG_{thresh}$	congestion avoidance threshold
$MCAST_{thresh}$	multicast retransmission threshold

Table 2: RMTP connection parameters

3.1 Designated Receivers

ACKs are needed from receivers in order to determine which packets need to be retransmitted. However, ACKs from a large number of receivers may overwhelm the sender (i.e., the ACK implosion problem). Furthermore, large number of ACK packets destined for the sender and retransmitted data packets generated from the sender may congest the sender's neighboring routers and local networks. In order to avoid this situation, RMTP uses *Designated Receivers* (DRs) [16]. The DRs assist the sender in processing ACKs and in retransmitting data. The concept of using DRs is best illustrated by Figures 2 and 3. Figure 2 shows a multicast tree with 14 receivers and a sender. In the absence of DRs, the sender processes ACKs from all 14 receivers. Figure 3 shows the same multicast tree with DRs. The multicast tree is partitioned into three subtrees, and the receiver rooted at each subtree has been selected as the DR of the subtree. In this case, the sender processes ACKs from only 2 receivers.

A DR is a special receiver; it caches received data, emits ACKs, and processes ACKs from the receivers in its subtree. Conceptually, a DR hides a subtree of receivers from its up-tree DR or the sender. Note that when a DR multicasts retransmissions, only the receivers in the DR's subtree receive the retransmissions. Thus, using DRs not only reduces the number of ACKs the sender has to process, but also improves the usage of network resources.

A DR serves the dual function of emitting ACKs (like any other receiver) and processing ACKs (like the sender). In order to refer to the entities that receive ACKs and process them, we will use the term AP (ACK Processor) from now on. Thus, an AP denotes either the sender or a DR. The term *receiver* refers to either a designated receiver or a normal receiver. Subsection 3.10 describes how a receiver selects its AP.

3.2 Connection and Connection Parameters

An RMTP connection is identified by a pair of *endpoints*: a source endpoint and a destination endpoint. The source endpoint consists of the sender's network address and a port number; the destination endpoint consists of the multicast group address and a port number. Each RMTP connection has a set of associated *connection parameters* (see Table 2). RMTP assumes that the *Session Manager* is responsible for providing the sender and the receiver(s) with the associated connection parameters. RMTP uses default values for any connection parameter that is not given.

3.3 Connection Establishment and Termination

Once the Session Manager has provided the sender and the receivers with the session information, receivers initialize the connection control block and stays in an unconnected state; the sender starts transmitting data. On receiving a data packet from the sender, a receiver goes from the unconnected state to the connected state and starts emitting ACKs at T_{ack} interval. (T_{ack} is dynamically adjusted based on the round-trip time (RTT) measured from a receiver to its AP to minimize unnecessary retransmissions; see subsection 3.8 for a detailed description.)

Connection termination is timer based. After it transmits the last data packet, the sender starts a timer that expires after T_{dally} seconds. (A DR also starts the timer when it has correctly received all data packets.) When the timer expires, the sender deletes all state information associated with the connection (i.e., it deletes the connection's control block). Time interval T_{dally} is at least twice the lifetime of a packet in an internet. Any ACK from a receiver resets the timer to its initial value. A normal receiver deletes its connection control block and stops emitting ACKs when it has correctly received all data packets. A DR behaves like a normal receiver except that it deletes its connection control block only after the T_{dally} timer expires[2].

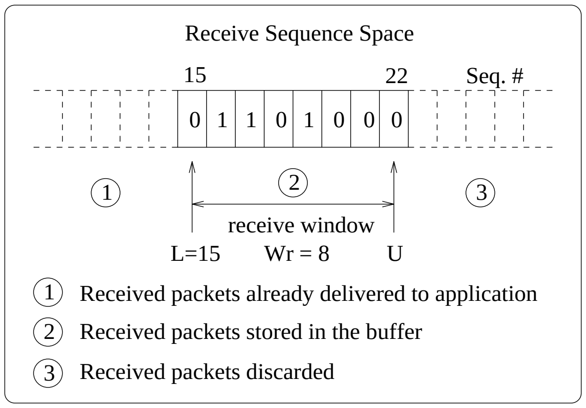
Because receivers periodically emit ACKs after connection establishment and the time interval between consecutive ACKs is much smaller than T_{dally}, the sender uses ACK reception during T_{dally} to deduce whether all receivers have received all data. If it does not receive any ACK during T_{dally}, the sender assumes either all receivers have received every packet, or something exceptional has happened that prevents the receivers from sending ACKs. Possible "exceptional" situations include: network partition and receivers voluntarily or involuntarily leaving the multicast group. As described in section 2, RMTP assumes that the Session Manager is responsible for detecting such situations and taking necessary actions.

In addition to normal connection termination, RESET packets can be used to terminate connections. For example, when RMTP detects that the sending application has aborted before data transfer is complete, it uses RESET to inform all the receivers to close the connection.

3.4 Flow Control

RMTP uses windowed flow control. The sender has a send window of W_s packets, and all receivers have a receive window of

[2]To prevent data from old session from mixing with that of the current session, the Session Manager must enforce a predefined delay before reusing a connection.



Receive Sequence Space
15
22
Seq. #
0 1 1 0 1 0 0 0
1
2
3
receive window
L=15
Wr = 8
U
1 Received packets already delivered to application
2 Received packets stored in the buffer
3 Received packets discarded

Figure 4: A receiver's receive window and the associated bitmap

W_r packets. W_s indicates the maximum number of packets the sender can transmit (without receiving acknowledgment from the receivers), and W_r determines a receiver's receive buffer size. The receive window must be no smaller than the send window to prevent receivers from dropping packets because of the lack of buffer space.

In addition to windowed flow control, RMTP allows users to set an upper bound on the sender's data transmission rate. The sender initiates data transmission at a fixed interval T_{send}. The number of packets transmitted during each interval depends on the amount of buffered user data, the usable send window, and the congestion window (described in subsection 3.9). Thus, the sender's maximum transmission rate measured over T_{send} is $W_s * Packet\_Size/T_{send}$. To set a multicast session's maximum data transmission rate, the Session Manager simply sets the parameters W_s, $Packet\_Size$ and T_{send} accordingly.

Receivers use a bitmap of W_r bits to record the existence of correctly received packets stored in the buffer. As Figure 4 illustrates, each bit corresponds to one packet slot in the receive buffer. Bit 1 indicates a packet slot contains a valid data packet. For example, Figure 4 shows a receive window of 8 packets; packets 16, 17 and 19 are received correctly and stored in the buffer. When a receiver sends an ACK to its AP, it includes the left edge of the receive window L and the bitmap. The receiver delivers packets to the application in sequence. For example, if the receiver receives packet 15 from the sender and does not receive packet 18, it can deliver packets 15, 16 and 17 to the application and advance L to 18.

3.5 Retransmissions and ACK Processing

If a multicast packet is lost, one or more receivers may miss the same packet. RMTP provides mechanisms for an AP to determine whether the lost packet should be retransmitted using unicast or multicast. Three parameters are included in the design for this purpose: T_{retx}, $MCAST_{thresh}$, and a *retransmission queue*. During T_{retx}, an AP processes ACKs from the receivers in its local region. If an ACK contains retransmission requests, the sequence numbers of the requested packets are added to the retransmission queue.

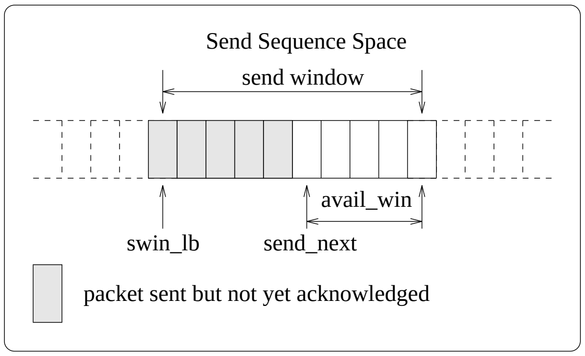


Figure 5: A sender's send window and related variables

A retransmission queue element contains the sequence number of a packet to be retransmitted, a counter C that counts the number of receivers that have requested the packet, a table $AddrTab$ that records the requesting receivers' network addresses, and a pointer to the next queue element. At the end of interval T_{retx}, an AP processes each element in the retransmission queue. If C exceeds a threshold $MCAST_{thresh}$[3], the AP delivers the packet using multicast; otherwise, the AP delivers the packet to each receiver in $AddrTab$ using unicast.

In order to determine whether a packet must be retransmitted by unicast or by multicast, an AP has to wait for T_{retx} time interval to collect retransmission requests from several receivers. This may introduce a delay in delivering the requested packets to the waiting receivers. However, there is an option for the Session Manager to set T_{retx} to 0. In that case, an AP will immediately process each received ACK and retransmit packets using unicast. There is an obvious trade-off here between efficiency and latency.

The sender uses three variables, $swin\_lb$, $send\_next$, and $avail\_win$ in the connection control block for managing the send window. As Figure 5 illustrates, variable $swin\_lb$ records the lower bound of the send window, $send\_next$ indicates the next sequence number to use when sending data packets, and $avail\_win$ is the available window size for sending data. The sender increases $send\_next$ and decreases $avail\_win$ after sending data. When ACKs acknowledging the receipt of packets with sequence number $swin\_lb$ are received, $swin\_lb$ is increased and so is $avail\_win$.

One of RMTP's design decisions is to use the progress of slow receivers to determine how much more data to transmit. Thus, the sender computes the smallest L (L_{min}) among the L values of all the ACKs received during T_{send}[4]. If L_{min} is greater than $swin\_lb$, it increases $avail\_win$ by $(L_{min} - swin\_lb)$ and sets $swin\_lb$ to L_{min}.

3.6 Immediate Transmission Request

Since RMTP allows receivers to join anytime during an ongoing session, a receiver joining late will need to catch up with the rest. In addition, some receivers may temporarily fall behind because of various reasons such as network congestion or even network partition. The immediate transmission feature is introduced in RMTP to allow lagging receivers to come up to the speed of the group.

A receiver uses an ACK_TXNOW packet to request its AP for immediate transmission of data. An ACK_TXNOW packet differs from an ACK packet only in the packet type field. When an AP receives an ACK_TXNOW packet from a receiver R, it checks bitmap V. If V indicates missed packet(s), it immediately transmits the missed packet(s) to R using unicast. If V indicates no missed packet, it computes the sequence number of the packet that R expects to receive next (call it $recv\_next$). If $send\_next > recv\_next$, it immediately unicasts $Min(W_s, (send\_next - recv\_next))$ packets of data to R[5]. Since R sends an ACK_TXNOW packet once per RTT, the AP's transmission rate in response to ACK_TXNOW packets is bounded by W_s/RTT_{R-AP}, where RTT_{R-AP} is the RTT between the requesting receiver R and its AP.

3.7 Data Cache

RMTP allows receivers to join an ongoing session at any time and still receive the entire data reliably. However, this flexibility does not come without a price. In order to provide this feature, the senders and the DRs in RMTP need to buffer the entire file during the session. This allows receivers to request for the retransmission of any transmitted data from the corresponding AP. A two-level caching mechanism is used in RMTP. The most recent $Cache\_Size$ packets of data are cached in memory, and the rest are stored on file system. In-memory cached packets contain application data as well as protocol headers to improve efficiency.

Although keeping all the transmitted data increases the size of state information maintained at each AP, the state size is not a function of the number of multicast participants (i.e., state size is still independent of the number of multicast participants). The buffered data size at the sender and the DRs can be reduced by determining how much data can be discarded based on the ACKs from the receivers. However, in such a scheme, the protocol will not be able to guarantee complete reliability to the receivers joining late, or to the receivers rejoining an ongoing session after being temporarily disconnected (partitioned). It will also require each AP to maintain a complete list of receivers in its region. RMTP does not impose these restrictions.

3.8 Measuring RTT and Calculating T_{ack}

Recall that the receivers in RMTP send ACKs periodically. If these ACKs are sent too frequently, the AP may end up retransmitting the same packet multiple times without knowing if the first retransmitted packet was received correctly by the receivers. In order to prevent such redundant retransmissions, RMTP requires each receiver to measure the round-trip time (RTT) to its AP dynami-

[3]The sender and DRs can have different $MCAST_{thresh}$ values.

[4]The L value in an ACK_TXNOW packet, which will be described in section 3.6, is not used to compute L_{min}.

[5]Because a DR does not have $send\_next$, it uses the left edge of the receive window, L in Figure 4, for the calculation.

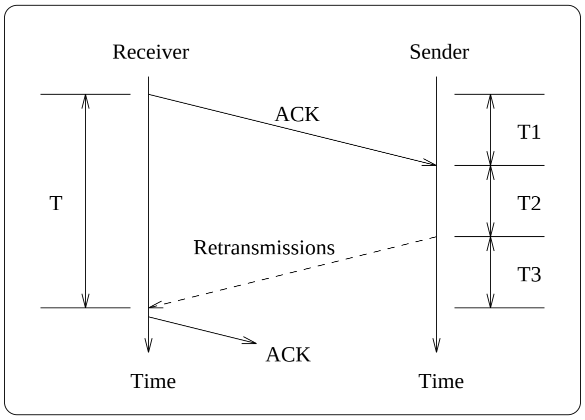


Figure 6: The components in calculating T_{ack}

cally. The measured RTTs allow each receiver to compute T_{ack}, the interval of sending ACKs.

A receiver uses RTT_MEASURE packet to measure the RTT between itself and its AP. A receiver sends the first RTT_MEASURE packet right after connection establishment. Subsequent RTT_MEASURE packets are sent at a fixed interval, T_{rtt}. To measure RTT, a receiver R includes a local timestamp in an RTT_MEASURE packet and sends the packet to its AP. When the AP receives the RTT_MEASURE packet, it immediately changes the packet type to RTT_ACK and sends the packet back to R. Upon receiving the RTT_ACK packet, R calculates RTT as the difference between the time at which the RTT_ACK packet is received and the timestamp stored in it.

RTT measurements allow a receiver to calculate T_{ack}, the interval of sending ACK. As Figure 6 illustrates, a receiver can reduce the probability of causing redundant retransmissions by sending one ACK at beginning of T and sending the next ACK shortly after the end of T. T is the sum of $T1$, $T2$ and $T3$; RTT is the sum of $T1$ and $T3$. Interval $T2$ is the delay incurred in an AP owing to the processing of ACKs. T_{ack} is computed based on T using a TCP-like scheme [1, 5].

3.9 Congestion Avoidance

RMTP provides mechanisms to avoid flooding an already congested network with new packets, without making the situation even worse. The scheme used in RMTP for detecting congestion is described below.

RMTP uses retransmission requests from receivers as an indication of possible network congestion [6, 5]. The sender uses a congestion window $cong\_win$ to reduce data transmission rate when experiencing congestion. During T_{send}, the sender computes the number of ACKs, N, with retransmission request. If N exceeds a threshold, $CONG_{thresh}$, it sets $cong\_win$ to 1. Since the sender always computes a *usable* send window as $Min(avail\_win, cong\_win)$, setting $cong\_win$ to 1 reduces data transmission rate to at most one data packet per T_{send} if $avail\_win$ is nonzero. If N does not exceed $CONG_{thresh}$ during T_{send}, the sender increases $cong\_win$ by 1 until $cong\_win$ reaches W_s[6]. The procedure of setting $cong\_win$ to 1 and linearly increasing $cong\_win$ is referred to as *slow-start*. The sender begins with a slow-start to wait for the ACKs from far away receivers to arrive.

3.10 Selecting An ACK Processor (AP)

Although the DRs are chosen statically for a multicast group, a receiver uses a mechanism to dynamically choose a DR as its AP. The idea is to choose the DR least upstream in the multicast tree with respect to a receiver as its AP. Note that a DR may fail. In that case, a receiver must be able to dynamically pick another DR as its AP. This is achieved in RMTP using a special packet called SND_ACK_TOME packet.

The selection of an AP is done as follows. The sender and all DRs multicast a SND_ACK_TOME packet along their subtrees periodically at a fixed interval T_{sap}. All SND_ACK_TOME packets have identical initial time-to-live (TTL) value in the header. Because routers decrement the TTL value when forwarding packets and all SND_ACK_TOME packets have identical initial TTL, the SND_ACK_TOME packet with maximum TTL value contains the address of the nearest uptree AP. A receiver stores the address of the nearest uptree AP and the associated TTL in its connection control block. It replaces the selected AP and the associated TTL value whenever a SND_ACK_TOME packet with a higher TTL value is received. To recover from the failure of the selected AP, a receiver associates the selection with a timeout. Each SND_ACK_TOME packet from the selected AP resets the timeout; expiration of the timeout causes the receiver to set the stored TTL value to 0 and start selecting a new AP.

A receiver selects the sender as its initial AP. The associated TTL value is set to 0, so that the receiver can select the nearest uptree AP. To allow a receiver to set its AP shortly after connection establishment, DRs multicast the first SND_ACK_TOME packet when they receive the data packet that establishes their connections. To prevent the loss of the initial SND_ACK_TOME packet, they subsequently multicast a SND_ACK_TOME packet for each data packet received, until the number of SND_ACK_TOME packet transmitted exceeds a constant X. Currently, X is set to 3.

Recall that a receiver calculates T_{ack} based on the measured RTTs between it and its AP. If the AP changes, the receiver needs to update its RTT and hence T_{ack}. Therefore, whenever a receiver selects a new AP, it immediately sends a RTT_MEASURE packet to the new AP and resets T_{ack} to a default value. Eventually, T_{ack} is computed based on the new measurements.

3.11 Overall View of the Protocol

Having described the various features of RMTP, there is a need to tie things together into a coherent form. In this subsection, a high-level perspective of the overall organization of the protocol

[6]If the sender and all the receivers are located in an environment in which the sender's maximum data rate is unlikely to cause congestion, one can bypass RMTP's congestion avoidance scheme by setting $CONG_{thresh}$ to "∞."

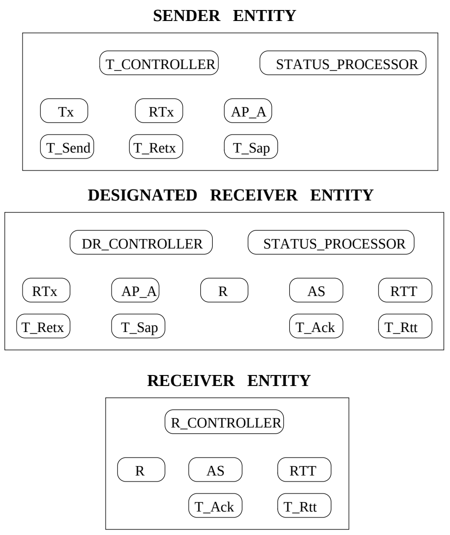


Figure 7: Block Diagram of RMTP

is provided. The protocol has three main entities: (1) Sender, (2) Receiver and (3) Designated Receiver. A block diagram description of each of these entities can be found in Figure 7.

The Sender entity has a controller component called T_CONTROLLER, which decides whether the sender should be transmitting new packets (using the Tx component), retransmitting lost packets (using the RTx component), or sending messages advertising itself as an ACK Processor (using the AP_A component). There is another component called STATUS_PROCESSOR, which processes ACKs (status) received from receivers and updates relevant data structures.

Also, note that there are several timer components: T_Send, T_Retx and T_Sap in the Sender entity, to inform the controller about whether the Tx component, the RTx component or the AP_A component should be activated.

The Receiver entity also has a controller component called R_CONTROLLER which decides whether the receiver should be delivering data to the receiving application (using the R component), sending ACK messages (using the AS component), or sending RTT_measure packets (using the RTT component) to dynamically compute the round-trip time (RTT) between itself and its corresponding ACK Processor.

Note that there are two timer components: (1) T_Ack and (2) T_Rtt to inform the controller about whether the AS or the RTT component should be activated. The component R is not timer driven. It is activated asynchronously when there are packets in sequence at the protocol buffer.

The DR entity is, in fact, a combination of the Sender entity and the Receiver entity.

4 MBone Technology and RMTP

RMTP uses MBone technologies to deliver multicast packets. MBone consists of a network of multicast capable routers and

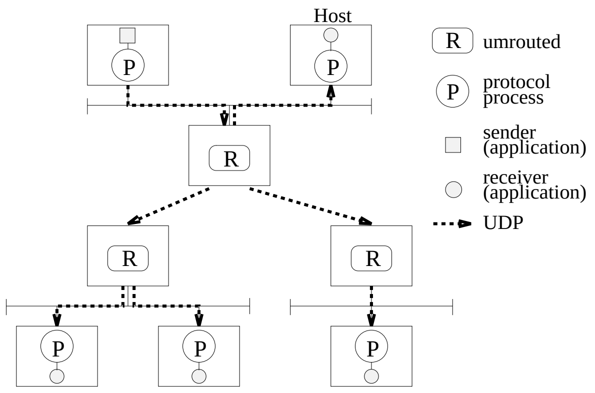


Figure 8: Multicast packet delivery from a sending application to a group of receiving applications using UDP

hosts. MBone routers use IP tunnels to forward multicast packets to IP routers that cannot handle multicast packets. An MBone router consists of two functional parts: a user-level process called `mrouted` and a multicast kernel. An `mrouted` exchanges routing information with neighboring `mrouted`s to establish a routing data structure in the multicast kernel. The multicast kernel then uses the routing data structure to forward multicast packets. To deliver multicast packets to receivers on a local subnet, an MBone router uses data-link layer multicasting (e.g., Ethernet multicasting).

4.1 User-level Implementation of RMTP

To make prototyping faster and debugging easier, we implemented multicast packet forwarding and RMTP protocol processing at user level. We modified `mrouted` to incorporate the routing functions of a multicast kernel. (We refer to the modified `mrouted` as `umrouted`.) Communications among `umrouted`s are via User Datagram Protocol (UDP) [17]. Thus, multicast packets travel on UDP-tunnels among `umrouted`s. By executing `umrouted`, a host with unicast kernel becomes a user level multicast router.

A user-level *protocol process* implements the RMTP protocol. Application-level receivers and senders use UDP to communicate with the RMTP protocol process. To deliver multicast packets to protocol processes on a local subnet, a `umrouted` uses UDP unicast instead of data-link multicast (see Figure 8).

A protocol process uses a configuration file to learn about the location of the `umrouted` that handles its multicasting requests. When a protocol process wishes to join a multicast group, it sends an Internet Group Management Protocol (IGMP) [3] Host Membership Report packet via UDP to its `umrouted`. The Host Membership Report message requires an acknowledgment from the `umrouted`. Thus, a `umrouted` builds a list of protocol processes' host addresses that it handles. A `umrouted` periodically sends an IGMP Host Membership Query message to each protocol process it handles using UDP unicast. Note that protocol processes and `umrouted`s do not follow the IGMP protocol standards to obtain multicast group membership information because they encapsulate IGMP messages in UDP and do not use data-link

W_s	15	packets
W_r	32	packets
T_{send}	600	milli-second
T_{retx}	300	milli-second
T_{rtt}	1	second
T_{sap}	3	seconds
T_{dally}	250	seconds
$Packet\_Size$	512	octets
$Cache\_Size$	128	packets
$CONG_{thresh}$	0	
$MCAST_{thresh}$	3	

Table 3: Connection parameters used

No.	Throughput (Kb/sec)			Retransmissions (%)		# Slow Start
	min.	avg.	max.	sender	DR1	
1	90.33	90.35	90.38	0.00	0.00	1
2	91.33	91.37	91.38	0.00	0.05	1
3	90.62	90.64	90.65	0.00	0.00	1
4	81.38	81.40	81.41	0.00	0.00	1
5	73.05	73.08	73.10	0.00	1.37	1
6	86.92	86.93	86.95	0.00	0.00	1
7	91.78	91.78	91.79	0.00	0.00	1
8	92.26	92.29	92.31	0.00	0.00	1
9	91.62	91.65	91.68	0.00	0.00	1
10	91.07	91.22	91.31	0.00	0.00	1

Mean throughput: 88.07 Kb/sec.
* No duplicate reception observed.

Table 4: Results of multicasting to area A1

multicast.

In essence, we built a multicast delivery system at user level using MBone technologies. Since routing is done at the user level, it is easy to implement local (subtree) multicasting (described next), which would otherwise require some support from the routers.

4.2 Router Support for Subtree Multicasting

Multicast routers compute a multicast delivery tree based on the sender's network address and the multicast group address. Thus, if there are two senders with different network addresses multicasting to the same group, two different multicast trees will be set up. In RMTP, the DRs need to multicast along the subtree of the multicast tree rooted at the sender. Achieving this objective without some support from the router is a difficult task. In this section, we describe the trick used to circumvent this problem.

Our implementation uses IP encapsulation and a new IP packet type `SUBTREE_MCAST` to inform a router that *subtree multicasting* needs to be done. To achieve subtree multicasting, a DR forms a multicast packet with the *sender*'s address as source and the group address as destination, encapsulates the packet in an IP packet with type `SUBTREE_MCAST`, and delivers the packet to a nearby router. When the router receives the packet, it decapsulates the packet, and performs multicast forwarding based on the source and destination addresses of the exposed packet as if the exposed packet originated from the sender.

Subtree multicasting results in the reception of duplicate packets for the DR who sends out the encapsulated IP packets. However, the DR can discard such duplicates.

5 Measurements on the Internet

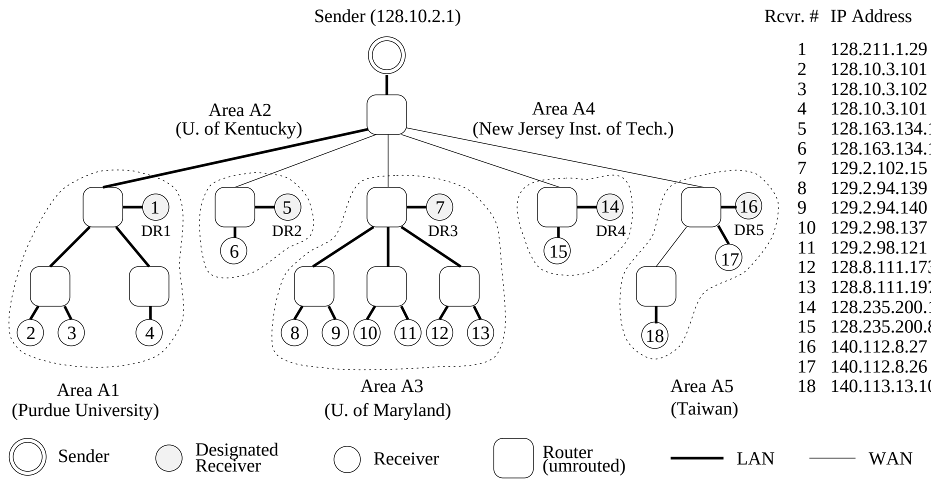
We measured the prototype implementation's performance with 18 receivers located at 5 geographic areas. Figure 9 shows the network configuration used. We implemented a simplified version of the Session Manager (SM) and its clients. Each receiving host executes the client process and the protocol process in the background, and the SM uses TCP to transport session-related information (e.g., session ID, connection parameters) to each client. Upon receiving the information, the client process invokes an application-level receiver process and informs the protocol process about the session information. Each client reports back to the SM when the application-level receiver process is ready. SM starts the sender when all the application-level receiver processes are ready.

Table 3 shows the connection parameters assigned by the SM. A maximum data rate of 100 Kbits/second (Kbps) is chosen to avoid overloading the Internet links of the test sites. The $CONG_{thresh}$ is set to 0 so that the sender invokes slow-start whenever it receives a retransmission request from a receiver. DRs are chosen by using a configuration file. Note that the sender only processes the ACKs from the DRs.

We conducted 10 experiments. Each experiment consists of 3 measurements of multicast file transfer in different network environments — M1: the sender multicasts to area A1, a LAN environment; M2: the sender multicasts to areas A1 through A4, a WAN environment; M3: the sender multicasts to all areas, a WAN environment including an international link with 512 Kbps bandwidth. For each measurement, the sender reads a 1 megabyte file from file system and multicasts it to the receivers. Receivers store the received data in a file for integrity check. Each receiver computes throughput independently after successful reception of the file. We also measured, in each area, the total number of retransmitted packets and duplicate packets by examining the log files created by the sending or receiving protocol processes.

All the experiments were conducted between January 25 and January 28, 1995. The first 3 experiments are conducted between 9:00 and 12:00 EST; the second 3 experiments are conducted between 12:00 and 17:00 EST; and the rest are conducted between 21:00 and 24:00 EST. The hosts used in the experiments are all workstation-class computers (e.g. Sun IPC, Sun IPX, Sun Sparc10). The experiments were conducted with the normal user processes running on them. No special treatments were given to the hosts running RMTP.

The results of the experiments are categorized by their measurement types (i.e., M1, M2, or M3). Tables 4 to 6 show the results. The average throughput is plotted in Figure 10. Since each receiver computes its own throughput independently, the Tables show the minimum, average, and maximum throughput among the throughput numbers reported by receivers. Note that the Tables report the



Rcvr. #	IP Address
1	128.211.1.29
2	128.10.3.101
3	128.10.3.102
4	128.10.3.101
5	128.163.134.10
6	128.163.134.11
7	129.2.102.15
8	129.2.94.139
9	129.2.94.140
10	129.2.98.137
11	129.2.98.121
12	128.8.111.173
13	128.8.111.197
14	128.235.200.1
15	128.235.200.8
16	140.112.8.27
17	140.112.8.26
18	140.113.13.101

Figure 9: Network configuration used for measuring RMTP's performance

No.	Throughput (Kb/sec)			Total # of Retransmissions (%)					Total # of Duplicates (%)				# Slow Start
	min.	avg.	max.	sender	DR1	DR2	DR3	DR4	A1	A2	A3	A4	
1	36.43	36.46	36.48	4.93	0.10	0.29	8.30	0.83	0.10	0.00	3.56	0.00	32
2	24.50	24.53	24.56	14.11	0.00	2.44	28.12	2.20	0.00	0.00	5.32	0.05	113
3	11.46	11.48	11.49	54.88	0.44	19.48	83.35	2.00	0.00	0.00	12.35	0.00	599
4	21.40	21.42	21.44	10.64	0.05	0.34	14.31	3.71	0.10	0.00	3.22	0.15	136
5	28.14	28.15	28.15	6.49	0.59	0.59	9.23	2.49	0.00	0.00	1.76	0.00	73
6	28.57	28.57	28.58	6.59	14.11	0.44	10.55	2.69	0.00	0.00	2.83	0.00	65
7	38.79	38.82	38.91	2.93	0.00	0.54	8.94	0.54	0.00	0.00	0.20	0.10	29
8	41.09	41.10	41.11	3.81	0.00	0.24	10.06	1.07	0.00	0.00	2.34	0.00	23
9	39.72	39.73	39.74	3.61	0.49	0.24	8.54	0.10	0.00	0.00	2.39	0.00	26
10	41.91	41.93	41.93	3.71	0.00	0.05	11.87	0.78	0.00	0.00	0.63	0.05	22

Mean throughput: 31.22 Kb/sec.

Table 5: Results of multicasting to areas A1, A2, A3, and A4.

No.	Throughput (Kb/sec)			Total # of Retransmissions (%)						Total # of Duplicates (%)					# Slow Start
	min.	avg.	max.	sender	DR1	DR2	DR3	DR4	DR5	A1	A2	A3	A4	A5	
1	20.81	20.81	20.82	19.29	0.88	0.34	17.09	1.66	18.56	0.00	0.00	1.03	0.00	0.15	161
2	21.27	21.34	21.36	16.80	0.24	0.98	14.36	2.20	15.43	0.29	0.00	1.90	0.00	0.15	144
3	20.67	20.71	20.72	18.55	0.15	0.20	17.58	1.22	19.53	0.00	0.00	1.46	0.00	0.10	153
4	18.17	18.22	18.23	20.41	0.10	0.54	29.88	1.03	24.12	0.00	0.00	1.03	0.00	0.05	212
5	18.27	18.85	18.97	23.54	0.68	0.20	21.29	2.20	20.90	0.00	0.00	3.03	0.00	0.10	206
6	25.47	25.53	25.55	13.48	1.03	0.10	10.55	2.39	14.26	0.00	0.15	0.88	0.20	0.00	100
7	16.62	16.62	16.63	24.76	0.05	0.00	14.70	0.68	35.21	0.00	0.00	1.42	0.00	0.00	264
8	17.52	17.57	17.58	27.00	0.15	0.20	10.55	0.73	35.30	0.00	0.00	0.83	0.00	0.10	232
9	19.28	19.30	19.30	22.02	0.10	0.05	8.25	0.68	29.79	0.00	0.00	1.17	0.00	0.00	197
10	20.67	20.83	20.89	15.77	0.00	0.20	7.96	0.20	24.66	0.00	0.00	2.73	0.00	0.05	153

Mean throughput: 19.98 Kb/sec.

Table 6: Results of multicasting to all areas

total number of retransmitted data packets observed by each AP, and the *total* number of duplicate data packets observed in each area. Thus, the numbers depend on the number of receivers in each area. As described in subsection 4.2, a DR receives duplicate packets from router when it uses subtree multicasting to deliver retransmitions. The total number of duplicate data packets reported in the Tables does not include these duplicates. The numbers in percentage are calculated as the number of packets divided by 2048 (i.e., the size of the data file in number of packets).

From the results, we observe the following:

1. DRs play a major role, as expected, in caching received data, processing ACKs, and handling retransmissions. This is obvious from the "Total # of Retransmission" columns in Tables 4, 5 and 6. In particular, note that in Table 6, seven out of ten numbers in the column corresponding to DR5 are greater than those of the sender. This means that the DR in A5 (Taiwan) retransmitted more packets to its area than did the sender (in Purdue) to all areas. That means, if the DR were not there, all these retransmissions would have to be done by the sender. Effectively, the DRs shield the sender from handling local retransmission requests and provide faster response to the requests.

2. The small difference between the "max." and the "min." values of all the throughput measurements in Tables 4,5 and 6, indicates that receivers, regardless of their geographic location, take about the same time to correctly receive the file. This shows that RMTP is able to adapt to receivers in various network environments.

3. In a heterogeneous environment, slow receivers and links with low bandwidth limit RMTP's performance. For example, with the same connection parameters, RMTP achieved a mean throughput of 88.07 Kbps in M1 (a LAN environment), and a mean throughput of 19.98 Kbps in M3 (a WAN environment with a low bandwidth international link). On the one hand, it indicates RMTP has achieved its design decision of not overrunning slow receivers and not wasting network bandwidth. On the other hand, it shows suboptimal throughput for fast receivers.

4. Low number of duplicate packets reported in areas A1, A2, A4 and A5 shows the effectiveness of RMTP's T_{ack} calculation. The main cause for A3's high number of duplicates is DR3 uses multicast for delivering retransmitted packets. It can be explained by a simple example. Suppose that $MCAST_{thresh}$ is set to 3. Now if, 4 out 6 receivers in A3 miss the same packet, DR3 will use subtree multicasting for retransmission of the missed packet. If all 6 receivers correctly receive the retransmission, two receivers will report duplicate reception.

6 Summary and Future Work

We have described the design and implementation of RMTP, and measured its performance. RMTP is designed for use as a reliable transport protocol by non-realtime applications that use multicasting to disseminate bulk data. It provides a rate-limited, windowed flow control with congestion avoidance to adapt to various network environments. Designated receivers allow hierarchical caching and avoid ACK implosion. Putting the responsibilities on each receiver to ensure sequenced, lossless reception allows the state information maintained at each multicast participant to be independent of the number of participants.

The multicast delivery system, based on MBone technology, used for the implementation of RMTP was also described. This framework is useful for doing research on Internet multicasting at user-level using UDP, without relying on the availability of multicast capable routers and hosts. The prototype implementation is being used for our continued research on evaluating multicast congestion control schemes, comparing techniques for dynamically choosing DRs for a given multicast tree, finding ways to achieve subtree multicasting without router support, and investigating the trade-offs of network resource usage and protocol performance in a heterogeneous environment with receivers of varying speeds.

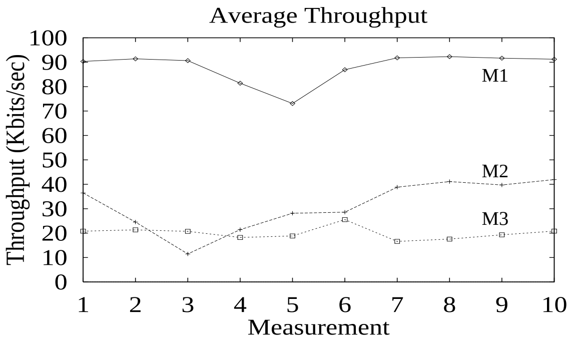


Figure 10: Average throughput among a group of receivers measured in various network environments

7 Acknowledgments

The authors are grateful to the following people and their associated institutions for providing assistance and resources that make the measurement of RMTP's performance possible: Prof. Douglas Comer, Prof. D. Karvelas, Prof. Tsern-Hui Lee, Yi-Ching Liaw, Chueng-Hsun Lin, Yi-Jan Lin, Prof. Der-Ming Liou, Shun-Jee Liu, Prof. Janche Sang, Tai-Kang Shing, Prof. Ben-Jei Tsuang, and Prof. Raj Yavatkar.

References

[1] R. Braden, "RFC-1122: Requirements for Internet Hosts – Communication Layers," *Request For Comments*, October 1989.

[2] S. Casner, and S. Deering, "First IETF Internet Audiocast," *ACM Computer Communication Review*, 22(3), July 1992.

[3] S. Deering, "RFC-1112: Host Extension for IP Multicasting," *Request For Comments*, Aug. 1989.

[4] H. Eriksson, “MBone: The Multicast Backbone,” *Communications of the ACM*, (8):54–60, Aug. 1994.

[5] V. Jacobson, “Congestion Avoidance and Control,” *Proceedings of ACM SIGCOMM ’88*, pages 314–328, Aug. 1988.

[6] R. Jain, “A Timeout-Based Congestion Control Scheme for Window Flow-Controlled Networks,” *IEEE Journal on Selected Areas in Communications*, SAC-4(7):1162–1167, Oct. 1986.

[7] A. N. Netravali, W. D. Roome, and K. Sabnani, “Design and Implementation of a High-Speed Transport Protocol,” *IEEE Transactions on Communications*, 38(11), Nov. 1990.

[8] J. Chang, and N. F. Maxemchuk, “Reliable Broadcast Protocols,” *ACM Transactions on Computer Systems*, 2(3), Aug. 1984.

[9] K. P. Birman and T. A. Joseph, “Reliable Communication in the Presence of Failures,” *ACM Transactions on Computer Systems*, 5(1), Feb. 1987.

[10] H. Garcia-Molina, and A. Spauster, “Ordered and Reliable Multicast Communication,” *ACM Transactions on Computer Systems*, 9(3), Aug. 1991.

[11] B. Rajagopalan, “Reliability and Scaling issues in Multicast Communication,” *Proceedings of ACM SIGCOMM ’92*, pages 188–198, Aug. 1992.

[12] R. Yavatkar, and L. Manoj, “Optimistic Approaches to Large-Scale Dessemination of Multimedia Information,” *Proceedings of ACM MULTIMEDIA ’93*, Aug. 1993.

[13] R. Aiello, E. Pagani, and G. P. Rossi, “Design of a Reliable Multicast Protocol,” *Proceedings of IEEE INFOCOMM ’93*, pages 75–81, Mar. 1993.

[14] N. Shacham, and J. S. Meditch, “An Algorithm for Optimal Multicast of Multimedia Streams,” *Proceedings of IEEE INFOCOMM ’94*, pages 856–864, Jun. 1994.

[15] F. Adelstein, and M. Singhal, “Real-Time Causal Message Ordering in Multimedia Systems,” to appear in *Proceedings of the 15th. ICDCS ’95*, Jun. 1995.

[16] S. Paul, K. K. Sabnani, and D. M. Kristol, “Multicast Transport Protocols for High Speed Networks,” *Proceedings of International Conference on Network Protocols*, pages 4–14, 1994.

[17] J. Postel, “RFC-768: User Datagram Protocol,” *Request For Comments*, Aug. 1980.

[18] J. Postel, “RFC-791: Internet Protocol,” *Request For Comments*, Sept. 1981.

[19] J. Postel, “RFC-793: Transmission Control Protocol,” *Request For Comments*, September 1981.