MTCP: Scalable TCP-like Congestion Control for Reliable Multicast

Injong Rhee   Nallathambi Balaguru   George N. Rouskas

TR-98-01

January 2, 1998

Abstract

We present MTCP, a congestion control scheme for large-scale reliable multicast. Congestion control for reliable multicast is important, because of its wide applications in multimedia and collaborative computing, yet nontrivial, because of the potentially large number of receivers involved. Many schemes have been proposed to handle the recovery of lost packets in a scalable manner, but there is little work on the design and implementation of congestion control schemes for reliable multicast. We propose techniques that can effectively handle instances of both short-term and persistent congestion occurring simultaneously at various parts of a multicast tree.

Our protocol incorporates several novel features: (1) *hierarchical congestion status reports* that distribute the load of processing feedback from all receivers across the multicast group, (2) the *relative time delay* (RTD) concept which overcomes the difficulty of estimating round-trip times in a multicast environment, (3) a *retransmission window* that regulates the flow of repair packets to avoid causing congestion during local recovery, and (4) *congestion localization*, a unique feature of our protocol which prevents persistently congested receivers from affecting the receiving rate of non-congested receivers. MTCP quickly throttles the sender's transmission rate based on aggregated update reports, and effectively localizes persistent, isolated congestion. Short-term congestion is handled by a modified TCP-Vegas scheme, while persistent congestion is handled by isolating the congested nodes from the rest of the group. We have implemented the short-term congestion control scheme and we have obtained encouraging preliminary results through Internet experiments.

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534

1 Introduction

As the Internet becomes more diversified in its capabilities, it becomes feasible to develop and offer services and applications that were not possible under earlier generations of Internet technologies. The Multicast Backbone (MBONE) and IP-multicast are two Internet technologies that have enabled a wide range of new applications. Using multicast, large-scale conferencing involving hundreds to thousands of participants is possible over the Internet. As multicast technologies become more widely deployed, we expect to see new multicast-based applications that demand more bandwidth and higher speed. Many of these applications will require reliable data transfer. By the year 2000, the Department of Defense predicts that demand will exist for Internet collaborative applications, such as virtual classroom and distributed interactive simulation, involving more than 10,000 participants. Clearly, a reliable data transport service that scales to large numbers of receivers will be of great importance in the near future.

Congestion control is an integral part of any best-effort Internet data transport protocol. It is widely accepted that the congestion avoidance mechanisms employed in TCP [1] have been one of the key contributors to the success of the Internet. Each conforming TCP flow is expected to respond to congestion indication (e.g., packet loss) by drastically reducing its transmission rate (multiplicative decrease of the window size) and by slowly increasing its rate when there is no congestion (linear increase of the window size). This congestion control mechanism encourages the fair sharing of a congested link among multiple competing TCP flows. A data flow is said to be *TCP-compatible* or *TCP-like* if it behaves like a flow produced by TCP under congestion [2]. At steady state, a TCP-compatible flow uses no more bandwidth than a conforming TCP connection running under comparable conditions.

Many Internet-based multimedia and collaborative applications use multicast. Unfortunately, most of the multicast schemes do not employ a congestion control or avoidance mechanism. Since TCP's congestion control mechanism strongly relies on other network flows to use a similar scheme, non-TCP-compatible multicast traffic can completely lock out competing TCP flows and monopolize the available bandwidth. Furthermore, the fact that non-TCP compatible flows are insensitive to existing congestion conditions (possibly caused by their traffic) increases the chances of simultaneous congestion collapses in various parts of the Internet [3]. Because of the far-reaching negative impact of non-TCP-compatible multicast traffic, it is highly unlikely that transport protocols for large-scale reliable multicast will become successful or widely accepted unless they employ TCP-like congestion control mechanisms.

The main challenge of congestion control for reliable multicast is scalability. To respond to congestion occurring at many parts of a multicast tree within a TCP time-scale, the sender needs to receive immediate feedback regarding the receiving status of all receivers. However, because of the potentially large number of receivers involved, the transmission of frequent updates from the receivers directly to the sender becomes prohibitively expensive and non-scalable. A fundamental question that we address in this paper is how to reconcile these two conflicting requirements.

Another challenging issue is the provision of mechanisms for isolating the effects of persistent congestion. Since a single multicast tree may span many different parts of the Internet, a TCP-like congestion control scheme will tend to reduce the sender's transmission rate upon indication of congestion in any part of the tree. While such a feature certainly fosters fairness among different flows (inter-fairness), it does nothing for fairness among the receivers in the same multicast group (intra-fairness). Specifically, it would be unfair for receivers whose paths from the sender do not include a congested link to be subject to a low transmission rate just because some isolated links are congested. A second question that we consider is how to ensure that only those receivers sharing the same congested links are subject to a reduced transmission rate. We note that any mechanisms for localization of persistent congestion must be transparent to receivers not affected by the congestion.

In this paper, we introduce a new congestion control protocol that addresses the above questions and overcomes the shortcomings observed in previously proposed schemes. Our protocol is based on a multi-level tree where the root is the sender, and other nodes in the tree are receivers. The sender multicasts data to all receivers, and the latter send positive and negative acknowledgments (ACKs and NACKs, respectively) to their parents in the tree. Internal tree nodes, called *sender's agents* (SAs), are responsible for handling feedback generated by their children and for retransmitting lost packets. Several tree-based protocols [4, 5, 6, 7, 8] have been proposed in the literature, but, to the best of our knowledge, they do not incorporate scalable TCP-like congestion control mechanisms that respond to congestion within a TCP time-scale. Our protocol, on the other hand, has several unique features, as follows.

Hierarchical congestion status reports. Each SA monitors the congestion level of its children *by independently maintaining a dynamic congestion window* using the ACKs and NACKs received from them. SAs for leaf receivers send a summary of congestion to their parents; this summary is simply the size of their window. In turn, SAs higher in the tree report to their parents their summaries, which are constructed by taking the minimum of the sizes of their windows and the window sizes contained in the summaries received from their children. This recursive construction of summaries effectively distributes the load of measuring the congestion level across the tree. The sender regulates its transmission rate based on its own summary.

Use of relative time delay. Maintaining a congestion window at each SA is not trivial because the SA is not the sender of the packets being acknowledged. For instance, the round trip time (RTT) of a packet cannot be measured as in TCP because the sender's clock and the SA's clock are not synchronized. Instead, we introduce the concept of *relative time delay* (RTD), defined as the difference between the clock value of the sender when a packet is sent and the clock value of the internal node (SA) when the corresponding ACK is received. We have modified the congestion control procedures (e.g., slow start, fast retransmission and recovery, and congestion avoidance) of TCP-Vegas [9] to use RTD instead of RTT.

Use of retransmission window. Many previously proposed protocols assume that there is always extra bandwidth available for retransmission, and that lost packets can be retransmitted without being subject to any congestion control. This assumption is overly optimistic, especially when a large number of packets is lost. The problem is further complicated by the fact that nodes other than the sender (i.e., the SAs) do the retransmission. We solve this problem by having each SA maintain another TCP-like congestion window for retransmissions. This new window measures the bandwidth available between an SA and its children, and it is used to regulate the flow of retransmissions.

Congestion localization. While short-term congestion can be handled by regulating the sender's transmission rate, persistent congestion at even a single link of the multicast connection will severely affect the sender's rate and, consequently, all receivers. To overcome this problem, we propose to remove receivers experiencing persistent congestion from the original group into a different multicast group, as follows. When an SA detects persistent congestion at its children (by examining their RTDs), it creates a new multicast group for its congested children. The SA forwards data received from the sender to this group, at a slower rate suitable for those receivers. Hence, the sender does not have to reduce its transmission rate for the segregated receivers, meaning that congestion does not affect receivers in other parts of the network. We also provide mechanisms so that, when congestion subsides, the segregated receivers may rejoin the original multicast group.

This paper is organized as follows. In Section 2, we discuss related work on reliable multicast. In Section 3 we present a suite of congestion congestion control mechanisms for scalable, tree-based reliable multicast protocols. In Section 4 we discuss an initial implementation of some of these mechanisms, and we present preliminary experiments and results. In Section 5 we describe work in progress, and in Section 6 we conclude the paper.

2 Related Work

Many reliable multicast protocols have been proposed in the literature [10, 11, 12, 13, 14, 15, 5, 6, 4, 16, 17, 18, 19, 20]. For the purposes of our discussion, we classify these protocols into three broad categories: *unstructured*, *structured* and *hybrid*. We examine the protocols in each category with an emphasis on their congestion control techniques.

Unstructured protocols do not impose any structure among receivers, and Pingali et al. [21] further classify them into *sender-based* [10, 12, 17, 19, 18, 20] and *receiver-based* protocols [15, 16]. In sender-based protocols, every receiver sends ACKs or NACKs directly to the sender, and the sender retransmits lost packets reported in NACKs. The main problem with sender-based protocols is the *feedback implosion* problem: if many receivers send ACKs or NACKs to the sender at the same time, the sender may quickly become overloaded. This problem is especially severe when losses occur near the sender, in which case a large number of receivers will experience packet loss. In a system involving more than a few receivers, the load imposed by the storm of acknowledgments limits the function of the sender.

In a receiver-based protocol, each receiver multicasts NACKs to all members of the group, and any receiver that has received the requested packets multicasts them to the group. Typically, the protocols incorporate randomized NACK and retransmission suppression timers to reduce the number of duplicate NACKs and retransmissions. We find three main shortcomings with receiver-based protocols.

First, Yajnik et al. [22] report that most packet loss in the MBONE occurs not at the backbone, but near end receivers, and that even excluding packet loss occurring near the sender, a small, but still significant, amount of loss (about 1% to 30%) involves more than two receivers. This study suggests that it is highly likely for two receivers not sharing common multicast routes to lose the same packets. A randomized NACK suppression technique may cause some uncorrelated NACKs to suppress correlated NACKs which actually report about a congested link. Since NACKs are multicast, the sender would get NACKs, but possibly from a different receiver each time: it may appear to the sender that NACKs are completely uncorrelated. Thus, the sender may not distinguish correlated packet losses from uncorrelated ones. It is unclear whether the sender should respond to all the NACKs by controlling its rate or ignore "seemingly" uncorrelated NACKs. Either approach seems unreasonable.

Second, in most receiver-based protocols that primarily use NACKs to detect congestion, the absence of NACKs is considered as no congestion or congestion clearance. Some Internet studies [22, 23], however, reveal that almost every experiment trace includes one or more extremely long bursts of packet loss lasting from a few seconds up to a few minutes. During these long loss bursts, no packets are received. As a result, receivers do not detect any packet loss, and do not send any NACKs. A similar scenario arises when the return path from receivers to the sender is congested, so that all feedback is lost. In either case, the sender would incorrectly translate the lack of feedback as no congestion.

Third, the randomized NACK suppression techniques employed by the receiver-based protocols require each receiver to estimate the round trip time (RTT) to every receiver in the group. This approach requires $O(n^2)$ RTT estimations by every receiver, thus imposing limits on scalability. Grossglauser [24] proposed a distributed deterministic timeout estimation protocol that does not require global information. However the protocol assumes that the end-to-end delay variation is bounded and *a priori* known to all receivers.

Structured protocols impose a logical structure among group members. Two commonly studied structures are *rings* and *trees*. In ring protocols [13, 14], a logical ring of group members is formed. Typically, a token is passed around the ring and only the process with the token may send feedback to the sender. Transis [13] implements a static sliding window algorithm that regulates the flow of data. Although the system later evolved to include a hierarchical ring, no flow or congestion control scheme was developed for the hierarchical ring. RMP [14] supports TCP-like congestion control based on both ACKs and NACKs. However, since only the token holder can send an ACK, it is unclear how the ACKs are used for purposes of congestion control when there is a large number of nodes in the ring. In RMP, since NACKs are also multicast to suppress other NACKs, the protocol suffers from problems similar to those arising in receiver-based protocols.

In a tree protocol [5, 6, 4, 7, 8], a logical tree structure is imposed on the multicast group, with internal nodes acting as representative receivers for the group. While the sender multicasts data to the entire group, a receiver sends feedback only to its parent. The representatives buffer packets received from the sender, and retransmit any packets reported lost by their children. Since the maximum degree of each node is fixed to a small constant, each node, including the sender, receives only a small amount of feedback within a round trip time. In the following, we discuss the congestion control schemes of RMTP [5] and TMTP [6], since the LBRM [4], LGC [7] and LORAX [8] tree protocols do not incorporate (or do not give much detail about) a congestion control scheme.

The main problem with RMTP is that it does not provide end-to-end feedback. The sender only gets feedback from its own children (called *designated receivers* (DR)) about their receiving status. Hence, the sender has little information about the congestion status of leaf receivers. When congestion occurs at leaf receivers, it may not be possible for the sender to detect the congestion, especially if the DRs and the leaf receivers do not share the same network path. In this case, the sender will continue to transmit at the same rate, aggravating the existing congestion. As a result, RMTP traffic can be completely unresponsive to congestion and may cause congestion collapse.

TMTP also does not provide end-to-end feedback. This protocol relies on a back pressure effect caused by the lack of buffers at representative nodes (called *domain managers* (DM) in TMTP terminology). In TMTP, DMs store the packets received from the sender until they receive ACKs for the packets from their children. When the buffers at a DM fill up because of congestion, the DM drops the next incoming packet. Its parent will continue to retransmit the packets not acknowledged by the DM until the parent's buffers also fill up. The sender detects congestion only when the buffers of all DMs between the sender and the congested nodes are completely full. So the congestion is completely neglected until the sender feels the pressure. Since each DM typically maintains a large number of buffers to reduce the number of ACKs returned to it, it may take a long time before the sender feels the pressure and reduces its rate. The fact that TMTP continues to transmit at a fixed rate despite the congestion is unfair to TCP-compatible flows which reduce their rates at the first indication of congestion.

Hybrid protocols [25, 26] combine the packet recovery techniques used in structured and unstructured protocols. As in receiver-based protocols, a receiver can multicast NACKs suppressing other NACKs, while other receivers may respond to the NACKs by retransmitting lost packets. In addition, a small number of representative receivers multicast their feedback immediately without any delay or suppression. The sender uses this feedback to control its transmission rate.

Delucia and Obraczka [25] have proposed a hybrid congestion control technique in which the size of the representative set is fixed, but the actual nodes in the set change over time based on the congestion status of receivers. Assuming that a small set of bottleneck links always causes the majority of the congestion problem, the protocol solves the feedback implosion problem, as well as other problems associated with SRM [15] (such as the RTT estimation problem). The scalability and utility of the protocol highly depend on this basic assumption, namely, that the representative set is always small. This assumption may not be realistic, however, since it is possible that several group members be independently and simultaneously congested although they do not share the same congested links. The protocol provides no safeguard against this situation.

Handley [26] has also proposed a hybrid congestion control architecture. The technique, though still at the stage of work-in-progress, roughly works as follows. A small set of representative receivers is selected based on their loss characteristics, and each representative forms a subgroup along with receivers that share similar loss characteristics. For each subgroup, one relay receiver is chosen to receive data from the sender and play them out at a slower rate suitable for the receivers in the subgroup. The idea of representatives is similar to that in [25], but the subgroup idea is new and promising. However, the overhead, complexity, and efficacy of dynamic subgroup formations have yet to be explored, justified or proven. In addition, since the group structure is essentially two-level, it is not clear whether the protocol is scalable to very large numbers of receivers.

Other types of protocols that do not fall within the above categories include receiver-driven layered multicast protocols [27, 28, 29]. These protocols implement congestion control by encoding the transmitted data into multiple layers and transmitting each layer to a different multicast group. By joining and leaving different multicast groups, each receiver can control its own receiving rate. Initially, the layering technique was proposed for continuous multimedia data streams which can tolerate some loss. Recently the technique was applied to a reliable bulk data multicast by Vicisano, Rizzo and Crowcroft [28]. However, the technique is applicable only when a large portion of the data is available for encoding prior to transmission, but not when data is generated in real-time such as during synchronous collaborative conferences.

3 MTCP: A New Congestion Control Scheme for Reliable Multicast

In this section we describe *Multicast TCP* (MTCP), a new congestion control scheme for reliable multicast based on a tree-structured protocol. Tree-structured protocols are not new and have been studied by many researchers [8, 5, 6, 4, 7]. However, little work has been done on TCP-like congestion control for these protocols. Instead, most previous work focuses on the issues of error recovery and feedback implosion. Levine and Garcia-Luna-Aceves [30, 8] analytically show that tree-based protocols can achieve higher throughput than any other class of protocols, and that their hierarchical structure is the key to reducing the processing load at each member of the multicast group. However, their analysis does not consider the effect of congestion control. Although tree-based protocols such as RMTP [5] and TMTP [6] incorporate congestion control schemes, these schemes suffer from the problems mentioned in Section 2. Hofmann [7, 31] also proposes to use a tree structure for feedback control, and provides a detailed description of how to construct such a tree. But he does not provide many details on congestion control.

Our TCP-like congestion control protocol takes advantage of a hierarchical tree structure imposed on the group members [1]. Since much work exists on how to construct such a tree dynamically (e.g., see [7, 31]), we omit the details on tree construction.

We have designed MTCP with the following goals in mind:

- **Compatibility with existing TCP congestion control.** MTCP's congestion control mechanism can correctly detect congestion shortly after its onset, and it can effectively reduce the transmission rate so that the bandwidth of the congested link can be shared fairly among the multicast flow and other TCP-compatible flows using the link (*inter-fairness* property). If congestion is transient and it later disappears, the mechanism can effectively increase the transmission rate to its pre-congestion levels.

- **Fairness among receivers.** In the presence of persistent congestion (typically caused by non-TCP-compatible flows) at one or more links of the multicast tree, MTCP is capable of achieving a high overall throughput by allowing non-congested receivers to advance faster than congested ones. We will refer to this feature as the *intra-fairness* property of MTCP, since it allows the various receivers to proceed at a rate that roughly corresponds to the resources available along their corresponding paths from the source.

- **Scalability.** By employing a logical multi-level tree, using hierarchical congestion status reports, and locally retransmitting repair packets, MTCP has the potential to scale to large numbers of receivers.

Compatibility with TCP traffic is an important requirement since (a) TCP is the most commonly used transmission protocol in the Internet [32], and (b) the utility of TCP depends on all other network flows being no more aggressive than TCP congestion control (i.e., multiplicative decrease of the congestion window on congestion occurrence, and linear increase at steady state). Non-TCP-compatible flows can easily lock out TCP traffic and monopolize the available bandwidth. Fairness among the receivers of the multicast application is essential in order to efficiently utilize all available resources in the network: one severely congested link should not affect the receiving rates of receivers whose paths do not include that link. Finally, scalability is necessary because the target applications of reliable multicast may involve a very large number of receivers.

In the following two subsections we describe the congestion control mechanisms used to achieve the stated goals.

[1] This is a *logical* tree that is not related to the tree used by IP-multicast for routing the packets.

3.1 TCP-compatibility and Scalability

We consider a reliable multicast scenario in which the sender transmits data to the receivers by using the services of an IP-multicast routing technology [33]. In order to employ TCP-like congestion control, the transmission rate of the sender needs to be adjusted when any part of the multicast routes is congested. To control the transmission rate within a TCP time-scale, the sender needs to have up-to-date information regarding the congestion status of all receivers in the multicast group. However, sending feedback directly to the sender creates an implosion problem. We need a scheme that distributes the load of processing feedback equally over the multicast group, while promptly notifying the sender of congestion occurring in the multicast routes. A tree-based protocol provides the most effective means to achieve this goal.

The basic operation of MTCP is as follows. The nodes participating in the multicast session are organized in a multi-level tree, where the root is the sender and other nodes in the tree are receivers. The sender multicasts data to all receivers, and the latter send positive and negative acknowledgments (ACKs and NACKs, respectively) to their parents in the tree. Internal tree nodes, called *sender's agents* (SAs), are responsible for handling feedback generated by their children and for retransmitting lost packets. Each SA independently monitors the congestion level of its children using the ACKs and NACKs received from them. When an SA sends an ACK to its parent, it includes in the ACK a summary of the congestion level of its children. Its parent then summarizes the congestion level of its own children (taking into account the summaries received from them), and sends the summary to its parent, and so on. The sender regulates its rate based on its own summary.

A congestion summary is used to estimate the *minimum* bandwidth available along the paths to receivers contained in the subtree rooted at the SA that sends the summary. Thus, the summary computed at the sender represents the current available bandwidth in the most congested link on the paths to all receivers in the multicast group. Since data is sent to all receivers at the same rate using multicast, as far as congestion control is concerned, the sender treats the entire multicast group as a single receiver. By sending only as much data as the bottleneck link can accommodate, the sender will not aggravate congestion anywhere in the network.

In order to estimate the minimum bandwidth available in each subtree, two mechanisms are adopted. First, each SA maintains a *dynamic congestion window* whose size indicates the amount of available bandwidth between the sender and the SA's children [2]. (Details on the window adjustment protocol will be presented shortly.) Second, each SA sends to its parent a congestion summary

[2] This window is different from the receive window of RMTP [5] which simply records received and missing packets.

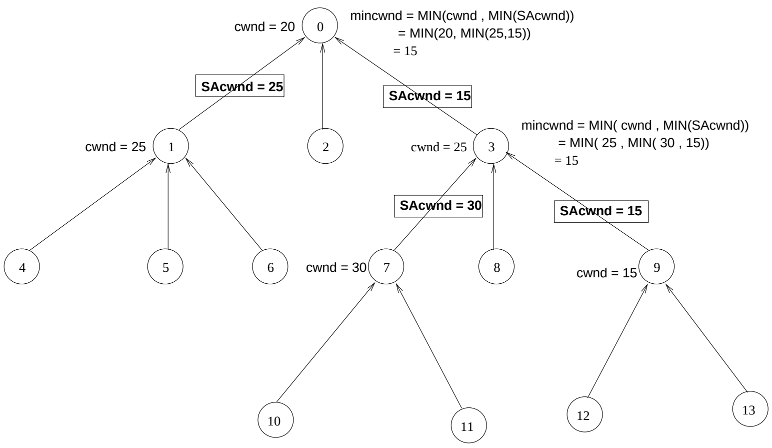

Figure 1: A tree where the circles represent the nodes in the multicast group, the boxes represent congestion summaries, and the arrows show how the summaries are propagated.

consisting of the minimum of its window size and the window sizes reported by its children. Figure 1 illustrates how congestion summaries propagate to the sender. Note that the computation of the minimum bandwidth within the subtree rooted at an SA is based only on the minimum bandwidth reported in the congestion summaries by the children of the SA. Since the maximum number of the children is typically fixed to a small constant, this scheme achieves good load-balancing.

We note that the delay between the time when an SA detects congestion and the time when the sender reduces its transmission rate in response to the congestion, may be longer than the TCP time-scale: since the congestion status report (in the form of a summary) has to travel all the way to the root of the tree, this delay can be larger than a round trip delay. However, unless congestion is reported directly to the sender (an approach that can lead to ACK implosion), this extra delay is unavoidable. Furthermore, as Floyd pointed out in the recent reliable multicast meeting [34], extra delays up to a few seconds can be tolerated because network links where a single flow can create severe transient congestion are likely to employ an appropriate queue management mechanism such as random early detection (RED) [2, 35]. We have observed through a small-scale experiment that the delay in our protocol is indeed well within this range; more details will be given in Section 4 (also refer to Figure 3).

We have modified TCP-Vegas [9] to adjust the congestion windows of the SAs and the sender. There are, however, several important differences between our protocol and TCP-Vegas.

Relative time delay (RTD). In TCP, the sender maintains the congestion window based on feedback received from a single receiver. The window algorithm in TCP is a closed loop feedback control system where the sender always receives ACKs for the packets it sent out. However, in our setting, it is not a closed loop system since SAs have to adjust their windows based on ACKs for packets that another node (the sender) transmitted. The main problem with this open loop system is that a SA cannot accurately estimate the round trip time (RTT) of a packet, which in TCP-Vegas is the main means to detect congestion and estimate the available bandwidth in the network path to a receiver. This problem arises due to the unpredictable delay variance in the network and the fact that the sender's and SA's clocks are not synchronized.

In MTCP, instead of measuring RTT, we measure the difference between the clock value taken at the sender when a packet is sent, and the clock value taken at the SA when the corresponding ACK is received from a child node. We call this time difference the *relative time delay* (RTD). The RTD to a child receiver can be easily measured by having each ACK carry the transmission time of the packet being acknowledged [3]. RTDs are used in the same way that RTTs are used in TCP-Vegas. During a congestion avoidance period [1, 9], the difference between the minimum measured RTD and the currently measured RTD to a child node is used to estimate the number of packets in transit (i.e., the actual throughput) from the sender to the node. Also, a weighted average of RTDs is used to estimate the retransmission timeout value (RTO). RTOs are used to detect congestion, and they trigger retransmissions and the slow start mechanism [1].

Using the RTD for these purposes is appropriate because MTCP uses only the relative differences in the RTDs. If, for instance, the sender's clock runs much faster than the SA's, it is possible that the RTO for a node be negative. In MTCP, however, the retransmission timer of a packet is set to expire only after the sum of the send time of the packet [4] (according to the sender's clock) plus the RTO of the node becomes smaller than the current clock value of the SA. Since in this case the sender's clock value is larger than the SA's, the sum gives the correct time at which the timer should expire.

An SA also estimates the RTT between itself and each of its children by periodically polling the children. These RTT estimations are used to set the RTO for *retransmitted* packets. Since SAs retransmit lost packets to their children, it makes sense to use these RTTs for the RTOs of retransmitted packets (i.e., after the initial timers which are set by RTDs expire and the packets are retransmitted).

Retransmission window for fast retransmission. In MTCP we adopt a selective ACK scheme (SACK) [36], along with a delayed ACK scheme in order to reduce the number of retransmissions and ACKs. These mechanisms, however, require some changes in the fast retransmission and recovery algorithms of TCP [1]. Consider a SA which receives a NACK reporting a list of lost

[3] Thus, MTCP requires that each packet multicast by the source carry its send time, which is copied in the ACK.

[4] The SA knows the send time of the packet since it also receives all packets multicast by the source.

packets. If many packets are lost in a long loss burst and the SA retransmits them immediately without knowing the available bandwidth between itself and its children (the SA only knows the bandwidth between its children and the sender) it may cause another instance of congestion. To overcome this problem, in MTCP each SA maintains another window, called the *retransmission window*, and used only for retransmitted packets. Since SAs receive ACKs for the packets it retransmitted anyway, maintaining the retransmission window does not incur much overhead. To the best of our knowledge, this technique is unique to MTCP in that most other protocols [15, 5, 6, 14] multicast repair packets regardless of the bandwidth available in the network.

3.2 Congestion Localization

TCP-like congestion control mechanisms such as the ones described in the previous subsection clearly foster inter-fairness among competing flows. However, any scheme which responds to some isolated congested link by simply imposing the same reduced transmission rate to all receivers, unfairly penalizes receivers with more network resources. When congestion is only short-term, intra-fairness may not be an issue. In fact, ensuring intra-fairness for short-term congestion would be an overkill since transient congestion can be promptly eliminated by TCP-like congestion control. However, persistent congestion, especially congestion caused by non-TCP-compatible flows, can prolong the period during which receivers are unfairly subject to a low transmission rate.

To ensure intra-fairness under persistent congestion, MTCP employs a scheme that can effectively isolate the effects of congestion so that only those receivers whose paths share a congested link are subject to a reduced rate. Our scheme does not require knowledge of routing paths or of the group membership, and it can work with any tree-based protocol. The scheme works in three stages: *detection*, *localization*, and *recovery*. We note that the congestion localization mechanism involves only those receivers experiencing persistent congestion, and it is completely transparent to all other receivers.

Detection. During this stage, each SA is monitoring the network for signs of persistent congestion. We are currently investigating a number of detection techniques. One approach is to have each SA count the number of slow starts caused by each child within a given period. If a child frequently causes slow starts, the link between the sender and the child is likely to be persistently congested. However, this technique alone may not detect all congestion, because the sender would reduce the transmission rate drastically when congestion occurs. Once the sender matches its rate to the bandwidth available in the congestion link, it maintains the rate until more bandwidth becomes available. Thus, although SAs may not initiate many slow starts, many other receivers are still unfairly subject to the reduced rate for a long time. Another promising technique is to use the measured RTDs for each child. After the sender's rate is reduced during a slow start period, if the estimated RTD for a child node is much larger than the minimum RTD measured over a long time period, the node is clearly experiencing persistent congestion. This is because even under the reduced rate, the congested link would carry data at its full capacity causing long delays (and thus, high RTD values), while uncongested links carry a light load of traffic. A combination of the two techniques is also possible.

Localization. When an SA determines that one or more of its children suffer from persistent congestion, it directs these children to leave the original multicast group and to join a new one called the *local multicast group*. Subsequently, the SA forwards packets received from the sender to the new local group at a reduced rate. (The SA continues to retransmit packets lost by its non-congested children, those which are still part of the original group, as before.) The transmission rate to the local group is estimated by maintaining a separate window, called the *local window*. This window is used for transmissions to the local group only, and is maintained in a way similar to the original congestion window. Feedback from receivers in the local group are not used to adjust the original congestion window of the SA, but are used only to adjust the local window. The SA only reports the size of the original congestion window to its parent, as before. By moving congested children into a new group, the persistent local congestion is concealed from the sender. As a result, the latter can continue transmitting data at a rate higher than if it had to take congested nodes into consideration.

An SA cannot continue to transmit to its local group at a rate lower than the sender's rate for ever. If congestion at the local group continues, the SA will run out of buffer space, and it will start dropping incoming packets. Dropped packets will cause a slow start for the SA's parent, and eventually for the sender. In order to have the sender quickly recover from this slow start, we recursively allow a congested SA's parent to form a new local multicast group that includes the congested SA (as well as other congested children of the parent, if any), but *not* the nodes that were in the local group formed by the congested SA. If the problem continues for a very long time, the initial congestion will cause a chain of local groups that will propagate to the sender itself. Eventually, the sender's buffers will begin to fill up, in which case the sender has no choice but to reduce its transmission rate to the original multicast group to the level of the rate within its own local group. As we can see, if congestion at a single link persists for a long time, eventually the transmission rate to the group will be determined by the receiving rate of the slowest receiver. This scenario cannot be avoided without infinite buffers at the sender or the SAs. What our localization feature attempts to do is to avoid slowing down the sender for as long as possible, hoping that in the meantime the congestion will subside and that congested receivers will start catching up with the rest of the group (more on this shortly).

This use of back pressure is similar in spirit to that of TMTP [6]. However, there is one fundamental difference, namely, that we use back pressure only within the context of congestion localization. In MTCP, most occurrences of congestion (especially, short-term congestion) are handled by throttling the sender's rate immediately. TMTP, on the other hand, does not handle short-term congestion at all, and does not have any congestion localization feature. Our technique also bears some similarity to Cheung and Ammar's destination set grouping [37]. The main difference is that in their scheme, the sender always forms local groups, whereas we have SAs form a local group. Hence, no extra load is imposed on the sender when local groups are created by SAs.

Recovery. When persistent congestion subsides, a receiver in a local group may re-merge into the original group. When the RTDs of the receiver get closer to its minimum RTD, the SA of the receiver can instruct it to leave the local group and join the original multicast group. However, since the receiver has been receiving at a rate lower than the sender's rate, it is possible that the receiver has missed many packets that the sender already sent out. Hence, the merging may cause the receiver's SA to retransmit a large number of packets so that the receiver can catch up with the rest of the original group. This retransmission, however, will not cause extra congestion because the flow of retransmission is also regulated by the retransmission window of the SA. Also, during retransmission, the SA goes into a slow start which causes the sender to reduce its transmission rate as well. The retransmission window and the TCP-like congestion control mechanisms employed in MTCP allow for a seamless integration of the segregated receivers into the original group.

4 Preliminary Experiments

We have modified the Collaborative Computing Transport Layer (CCTL) [20] to incorporate our congestion control scheme. CCTL provides a sender-based reliable multicast service on top of UDP. We have extended CCTL to work with a multi-level tree structure. Our implementation involves only the hierarchical TCP-like congestion control mechanism and the retransmission window. Other features such as the congestion localization and membership operations have not been implemented yet. Since a dynamic membership protocol that automatically constructs a tree of receivers is not implemented within CCTL (CCTL is a sender-based protocol), we manually set up all the different trees used in the following experiments.

4.1 Scalability and Response Time

The first set of experiments involved a source transmitting a 70 MB file to multiple destinations. During each experiment we measured the receiving rate at each receiver, and we also recorded the congestion window sizes. The purpose of these experiments was to obtain preliminary evidence on (1) whether the congestion control protocol can be scalable (i.e., how many children can a SA accommodate?), and (2) whether the protocol can respond to a congestion within a TCP time-scale. Note that we do not claim that the experiments presented here are sufficient to provide conclusive answers to these questions. In order to reach meaningful conclusions regarding the behavior and properties of the congestion control mechanism employed in MTCP, we are currently in the process of conducting a thorough analytical and simulation study, to be followed by real-world experimentation once the protocol is fully implemented (a discussion on our work-in-progress is presented in the following section).

In order to investigate the scalability of the protocol, we set up a one-level tree (rooted at the sender, with all receivers on the same level) and measured the throughput and CPU load at the sender as we added more receivers. All the machines used in the experiment were Ultra-Sparc Model 250 attached to a 100 Mbit/s LAN. Figure 2 shows the throughput and CPU load against the number of receivers. The CPU load represents the percentage of time that the CPU is used during the transfer of the file. As more receivers are added, the throughput does decrease, but not significantly. At the same time, we see that the CPU load also decreases with the number of receivers. This behavior can be explained by observing that as the number of receivers increases, the sender spends a greater amount of time waiting for ACKs, and thus total transfer time also increases. These results indicate that even if the sender and the SAs have as many as 16 children, the processing of ACKs does not pose a problem. In view of the fact that the experiment was performed in a high-speed LAN (where the sender can transmit at fast rate and also receives ACKs at a fast rate), the number 16 appears to be a reasonable upper bound on the number of children in the tree for large-scale implementation.

We then investigated the time delay involved in responding to congestion. To this end, we set up a four-level tree and examined how long it takes for the congestion window of the sender to be adjusted in response to changes in the congestion window of SAs in the path to the congested receiver. The tree involves one machine from each of the following sites: NCSU (the sender), Emory University (the first SA, SA1), Georgia Tech (the second SA, SA2), and University of Texas, Austin (the leaf receiver). Figure 3 shows a five second segment of the experiment. In this experiment we found Texas to be the bottleneck, which caused SA2 (at Georgia Tech) to have the smallest window size. The sender's window size is the largest. Recall that in our protocol, the sender regulates its transmission rate based on the minimum of all the reported congestion summaries and its own window. Let us call this minimum the *transmission window*. As we can see, the transmission window closely follows the window of SA2. Furthermore, we observe that whenever any site runs into a slow start, the transmission window reduces its size drastically within about 200 ms to 250 ms. For example, in Figure 3 we see that SA2 initiated a slow start at around 43 seconds, and that about 250 ms later the transmission window also dropped to match the window of SA2.

4.2 Inter-fairness

The second set of experiments involves the tree in Figure 4, and its purpose is to test the inter-fairness of MTCP in a realistic environment. The tree consists of five different sites, totally 23 receivers, and one sender at the root. The types of machines used are an assortment of SPARC Ultra's and SPARC 5's and 20's.

For the routing of MTCP packets, we have a special process at each site, called *mcaster*, much like `mroutd` in MBONE. The packets generated from the sender at the root of the tree are routed along the tree using mcasters. An mcaster simply "tunnels" incoming packets by first multicasting them to its own subnet via IP-multicast, and then forwarding them to the mcasters of its child sites in the tree via UDP.

An inter-fair protocol uses no more bandwidth than a conforming TCP traffic would use on the same link. To test the inter-fairness of MTCP, we run an independent TCP traffic over each WAN route. Note that since MTCP packets are routed over WAN via UDP, the TCP and MTCP traffics between the same sites in our experiments take the same WAN routes. This setup suits our purpose well because WAN links are mostly likely to be a bottleneck.

We run four different experiments. The first experiment involves areas A1 and A4. The second experiment involves areas A1, A2, A3, and A4. The third experiment involves the entire tree. In these experiments, the MTCP sender and TCP senders transmit data as fast as it is allowed by their congestion control protocols. Each TCP sender also starts transmitting at the approximately same time as the sender. In these experiments, we expect MTCP to match its sending rate to the minimum bandwidth available in the tree, so every involved MTCP receiver should receive at the approximately same rate as the TCP receiver on the bottleneck connection in the tree. The fourth experiment involves A1 and A4. In this experiment, while MTCP is transmitting, we run three

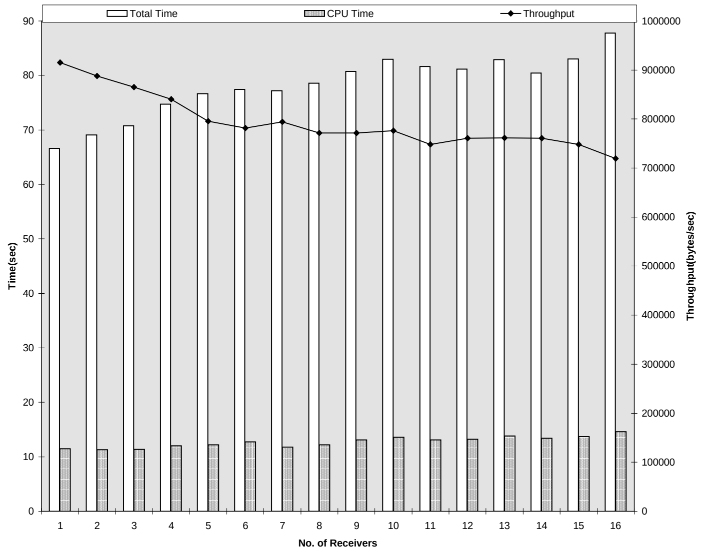

Figure 2: One level scalability test

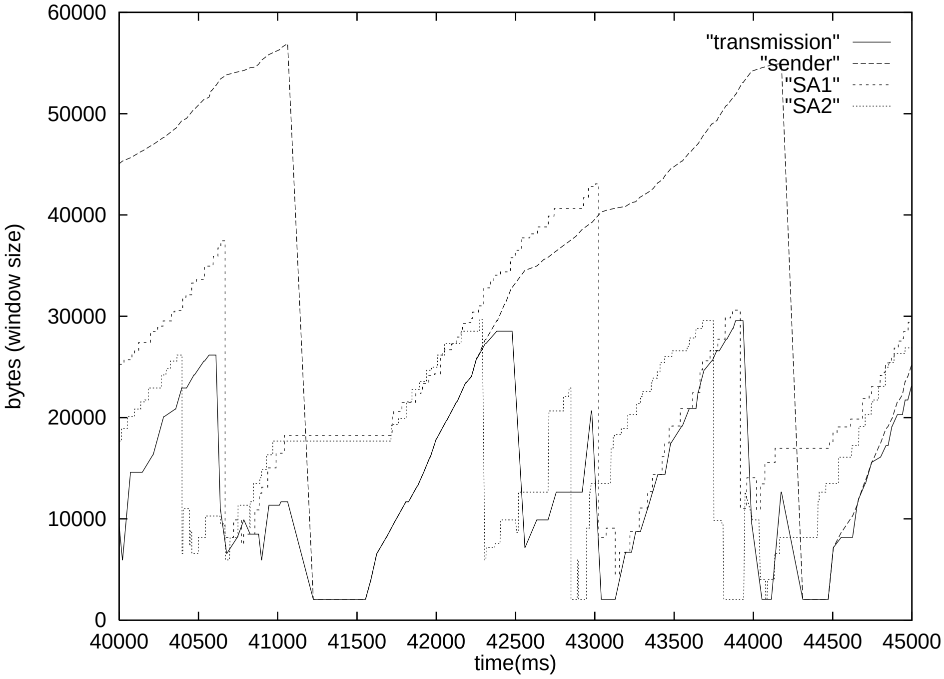

Figure 3: Multi-level response time test

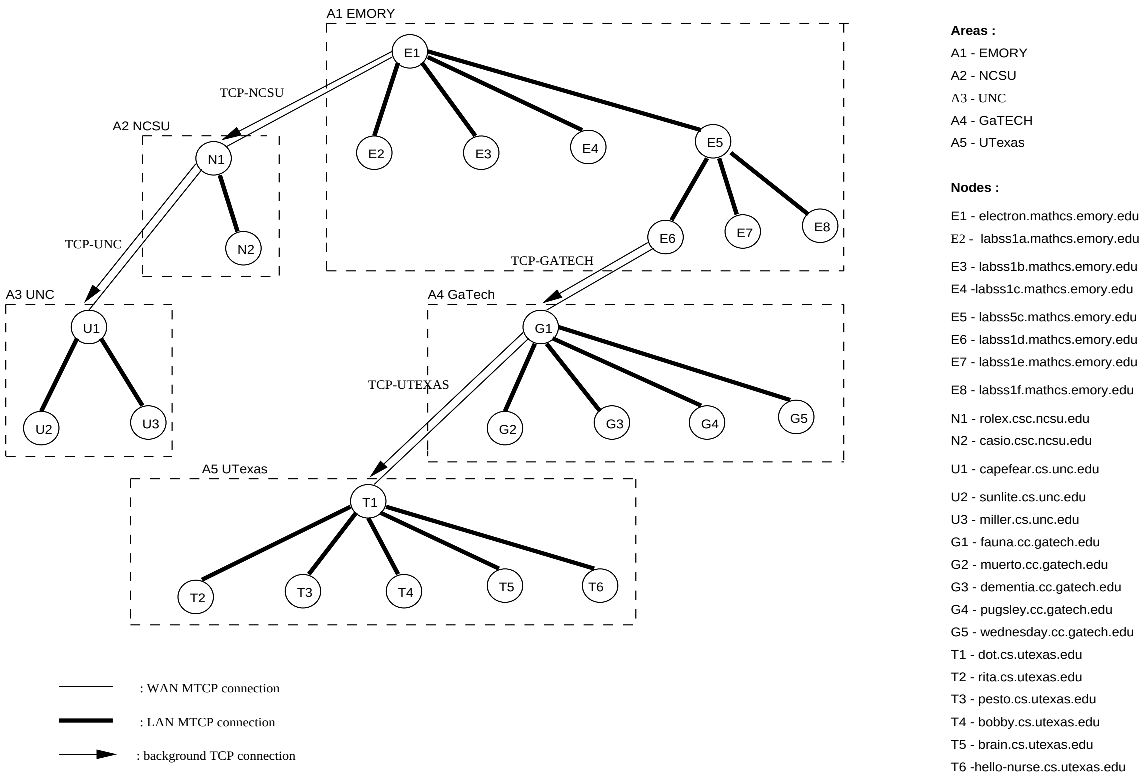

Figure 4: Trees used for experiments

TCP traffics from Emory University to Georgia Tech, each of which is started at a different time. We expect to see that MTCP adjusts its rate to match the current level of bandwidth available over the link between Emory University to Georgia Tech. In all experiments, each receiver records its receiving rate at every second. We present both the recorded rates and their timed average over each 5 second period.

The result of the first experiment is shown in Figures 5 and 6. We run MTCP and TCP traffics over 300 seconds and recorded the receiving rates of MTCP and TCP receivers. Figure 5 shows the timed averaged receiving rates of MTCP and TCP. Figure 6 shows the recorded receiving rates taken at every second. Both graphs show that that MTCP and TCP share approximately the same bandwidth around 280 KB/s.

The result of the second experiment is shown in Figures 7 and 8. In the figures, it is clear that the route from Emory University to the NCSU is the bottleneck because the TCP connection between these two sites gives the minimum receiving rates. MTCP matches the TCP receiving rate over this bottleneck route around 70 KB/s.

The result of the third experiment is shown in Figures 9 and 10. In the figures, it is clear that the route from Georgia Tech to the University of Texas is the bottleneck because the TCP connection between these two sites gives the minimum receiving rates. MTCP's receiving rate is no more than TCP's over this route which is around 60 KB/s.

These three experiments indicate that MTCP uses no more bandwidth than a TCP traffic uses on the bottleneck route of a given tree configuration. Although a bottleneck link is located several levels away from the root, MTCP still adjusts its rate according to the available bandwidth on that link. In all experiments, the fluctuation of MTCP's receiving rate is not perfectly synchronized with that of TCP's. This is because MTCP and TCP are not the same protocol, and the way that they detect congestion is different. In addition, MTCP reacts to every instance of congestion within a tree while TCP reacts to congestion only within two end points.

Figure 11 shows the result of the fourth experiment. TCP1, TCP2, and TCP3 are started around 200, 250, and 410 seconds respectively after MTCP is started. All TCP connections are made between different host machines to eliminate the effect of computational overhead. The same type of machines are used.

When MTCP runs alone, its receiving rate reaches around 400 KB/s. When TCP1 is added, MTCP reduces its rate from 400 KB/s to 300 KB/s while TCP1 traffic slowly increases its rate to be around 300 KB/s. As soon as TCP2 is added, both TCP1 and MTCP reduce their rates. TCP1 goes down to 180 KB/s while MTCP matches its rate with TCP2 around 240 KB/s. As TCP3 is added, both MTCP and TCP2 reduce their rates slightly. MTCP still does not use more bandwidth than TCP2. As TCP1 finishes its transmission, MTCP's rate bounces up to match that of TCP2. TCP3 also increases its rate. It appears that TCP3 always uses less bandwidth than the second TCP. The difference is about 50KB. There could be a couple of reasons for this difference. First, their end points are different although they use the same WAN route. So there could be other background job activities on the end points of TCP3. Second, TCP itself sometimes can be too conservative to estimate the available bandwidth. As TCP2 ends, both TCP3 and MTCP increase their rates quite a bit. MTCP settles around 330 KB while TCP3 goes up to 260 KB. The difference is close to that between the receiving rates of TCP2 and TCP3. When TCP3 ends, MTCP restores its rate quickly to 400 KB/s. From this experiment, we observe that MTCP seems to adjust its rate as quickly as TCP, according to the current available bandwidth on the bottleneck link in a given tree.

5 Work In Progress

Although the experiments presented in the previous section provided encouraging results regarding the scalability potential of the short-term congestion control scheme of MTCP, the largest number of receivers in any of the experiments was 23. In order to investigate the behavior of MTCP under reliable multicast scenarios involving large numbers (hundreds) of receivers spanning a very large geographical area, where congestion instances can arise simultaneously in many parts of the tree, we are pursuing two directions. First, we are trying to obtain access to more sites in North America and Europe to run larger scale Internet experiments. However, we are limited by the overhead associated with managing and updating the various sites. Therefore, we are also working on developing a simulation tool for testing MTCP. We are using the `ns` simulation package since it can efficiently support networks of several hundred nodes (an important feature since we are interested in the scalability of MTCP), and it requires a modest amount of effort to support our new protocol. The simulation experiments will also be used to determine appropriate values for various protocol parameters as well as heuristic methods to be used in the implementation. For instance, we will evaluate various techniques for maintaining the retransmission window (see Section 3.1) and for detecting persistent congestion in parts of the tree (refer to Section 3.2).

We are also working towards implementing the localization mechanism for handling persistent congestion. Recall from Section 3.2 that the mechanism consists of three stages, namely, detection of congestion, localization through the creation of new local groups for congested receivers, and recovery whereby previously congested receivers are returned to the original multicast group. We are investigating a number of approaches to efficiently implementing these features of MTCP.

We are also in the process of developing analytical techniques to show that, by creating new multicast groups for congested receivers, a tree-based reliable multicast protocol can achieve a higher throughput than by simply letting the slowest receiver determine the transmission rate of the source. These techniques are an extension of the analysis in [30] as follows. The maximum throughput analysis of tree-based protocols in [30] is based on the assumption that all receivers behave identically. Specifically, packet loss is modeled by a single parameter p which represents the probability that a receiver will not receive a packet independently of other receivers. This probability is determined by the loss rate along the path to the slowest receiver. We are extending this analysis to a tree-based protocol that maintains two multicast groups, one for a set of receivers with loss probability p_1, and one for receivers with loss probability $p_2 < p_1$. We expect our analysis to show that the weighted maximum throughput for the two sets of receivers is higher than the maximum throughput obtained by the analysis of [30] for a single multicast group with loss probability of p_1.

Finally, we have developed a new fairness index (hereafter referred to as *intra-fairness* index) to measure the performance of MTCP in terms of intra-fairness. This index is similar in spirit to the fairness index used to measure inter-fairness [38]. Let a_i denote the available bandwidth along the path to receiver i, and let b_i denote the bandwidth used by the protocol along that path. Also, let $a^{(min)} = \min\{a_i\}$ denote the bandwidth available along the path to the slowest (congested) receiver. Our intra-fairness index is defined as:

$$F = \frac{\left(\sum_{i=1}^{N} \frac{b_i}{a_i}\right)^2}{N \sum_{i=1}^{N} \left(\frac{b_i}{a_i}\right)^2} \tag{1}$$

We note that, an ideal reliable multicast protocol would allow each receiver to proceed at its own pace (i.e., it would achieve $b_i = a_i \ \forall \ i$, in which case $F = 1$). On the other hand, protocols proceeding at the pace of the slowest receiver are such that $b_i = a^{(min)} \ \forall \ i$. In this case, the fairness index becomes $F = \frac{\left(\sum_{i=1}^{N} \frac{1}{a_i}\right)^2}{N \sum_{i=1}^{N} \left(\frac{1}{a_i}\right)^2}$, which, by the Cauchy inequality, is at most equal to one. In the special case where $a_i = a \ \forall \ i$, the last expression reduces to $F = 1/N$, meaning that when the transmission rate of the source is tied to the slowest receiver, unfairness increases linearly with the size of the receiver set, certainly an undesirable behavior. The intra-fairness index can be easily calculated during simulation or real-world Internet experiments.

6 Concluding Remarks

We have presented MTCP, a set of congestion control mechanisms for tree-based reliable multicast protocols. MTCP was designed to effectively handle instances of both short-term and persistent congestion occurring simultaneously at various parts of a multicast tree. We have implemented the short-term congestion control scheme of MTCP and we have obtained encouraging preliminary results through Internet experiments. We are currently working on implementing the persistent congestion control scheme, and on developing a simulation model that will be used to investigate the scalability of MTCP.

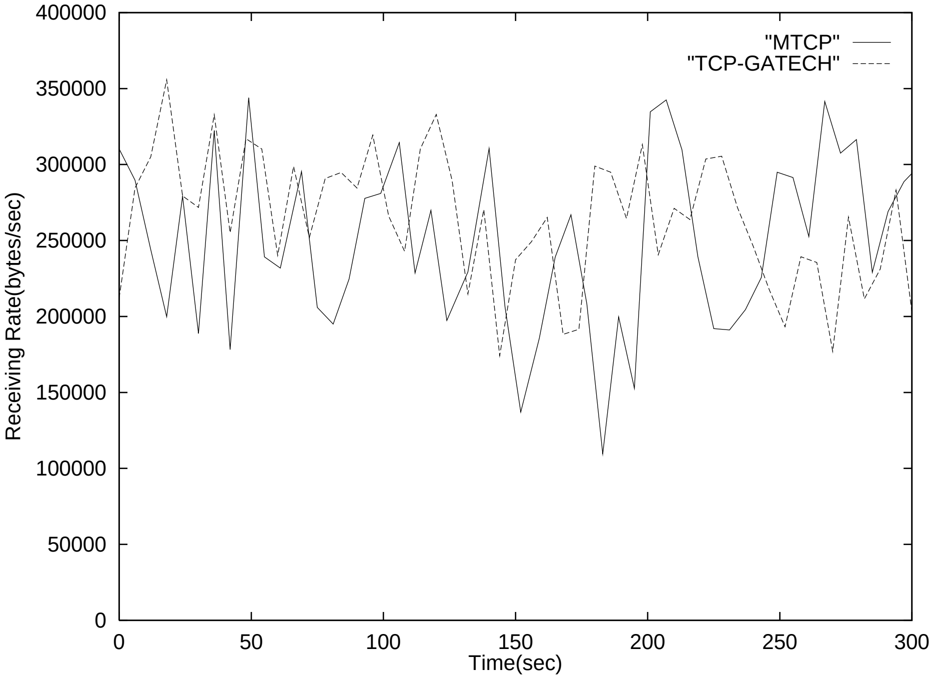

Figure 5: Receiving rates averaged over every 5 second in the first experiment involving areas A1 and A4

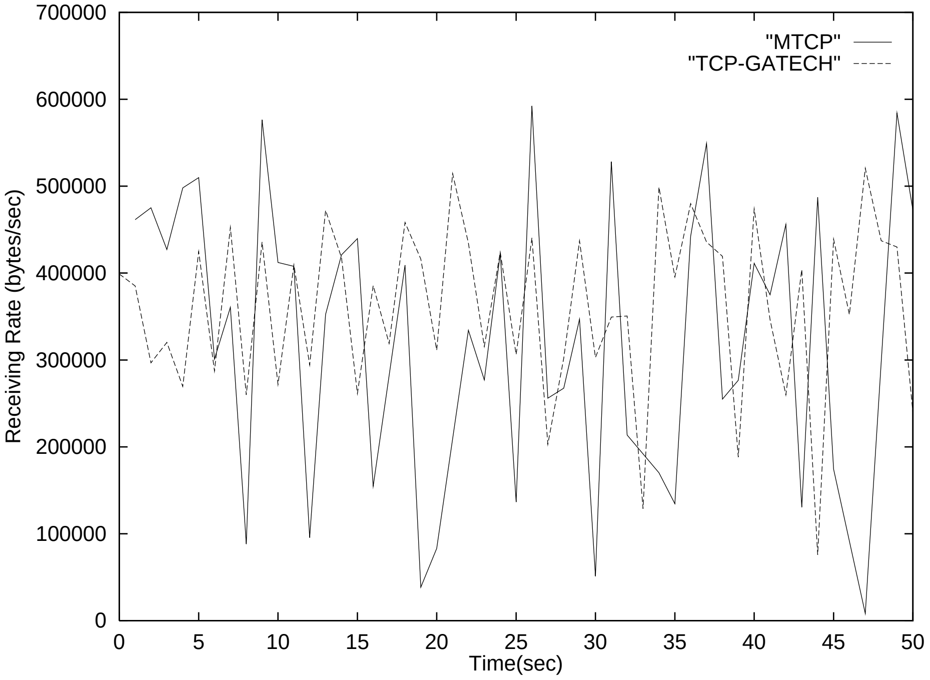

Figure 6: Receiving rates recorded at every second in the first experiment involving areas A1 and A4

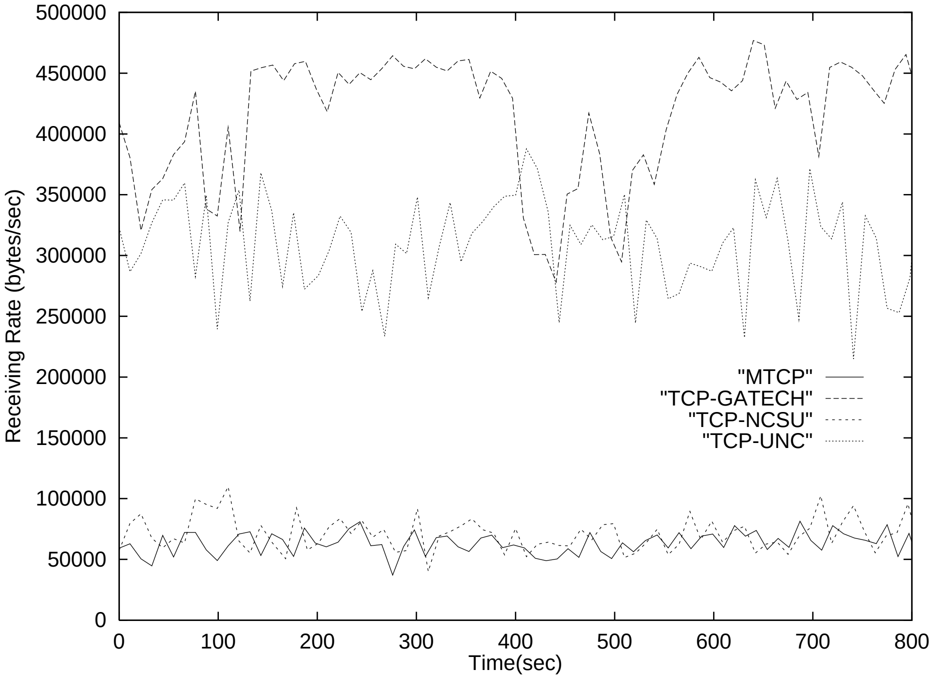

Figure 7: Receiving rates averaged over every 5 second in the first experiment involving areas A1, A2, A3, and A4

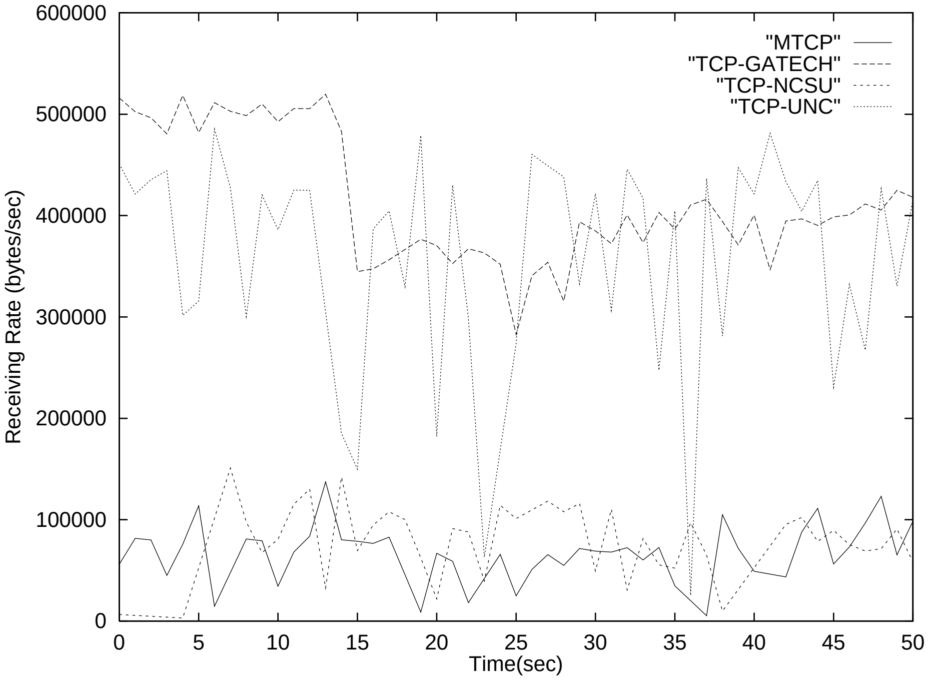

Figure 8: Receiving rates recorded at every second in the first experiment involving areas A1, A2, A3, and A4

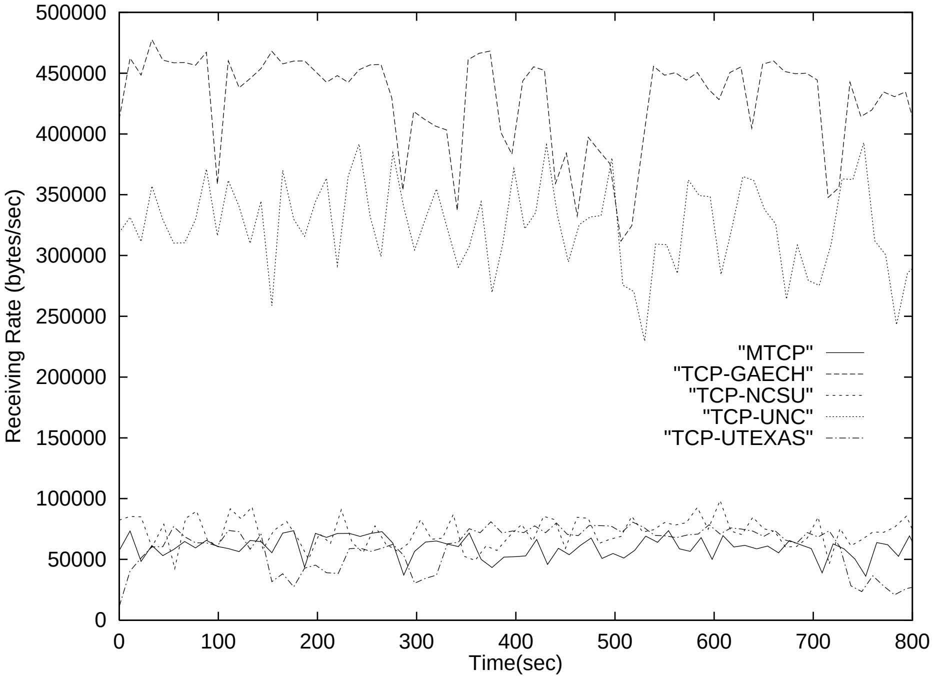

Figure 9: Receiving rates averaged over every 5 second in the first experiment involving areas A1, A2, A3, A4, and A5

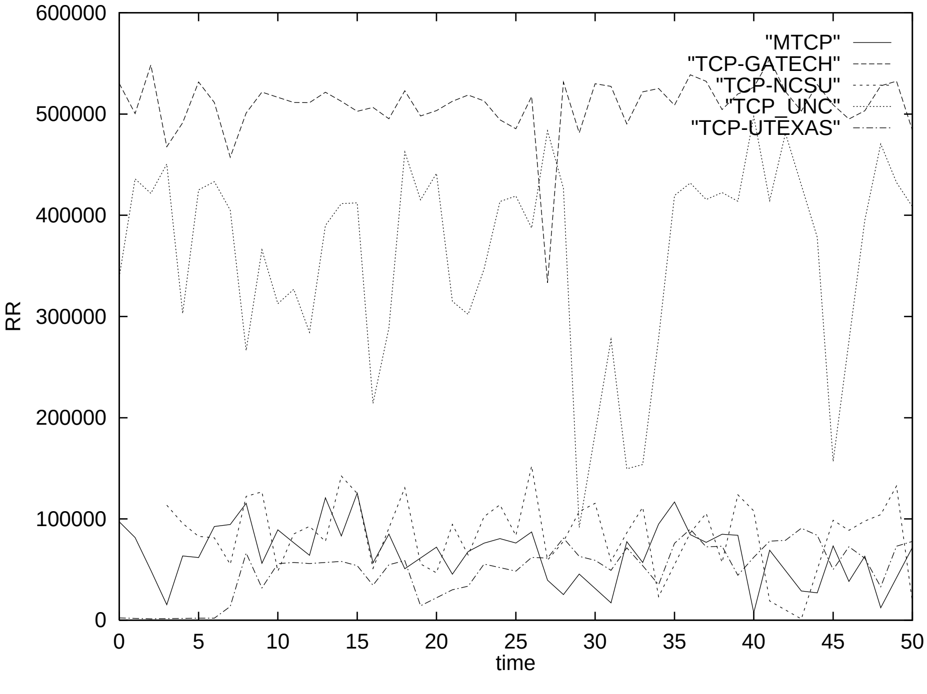

Figure 10: Receiving rates recorded at every second in the first experiment involving areas A1, A2, A3, A4, and A5

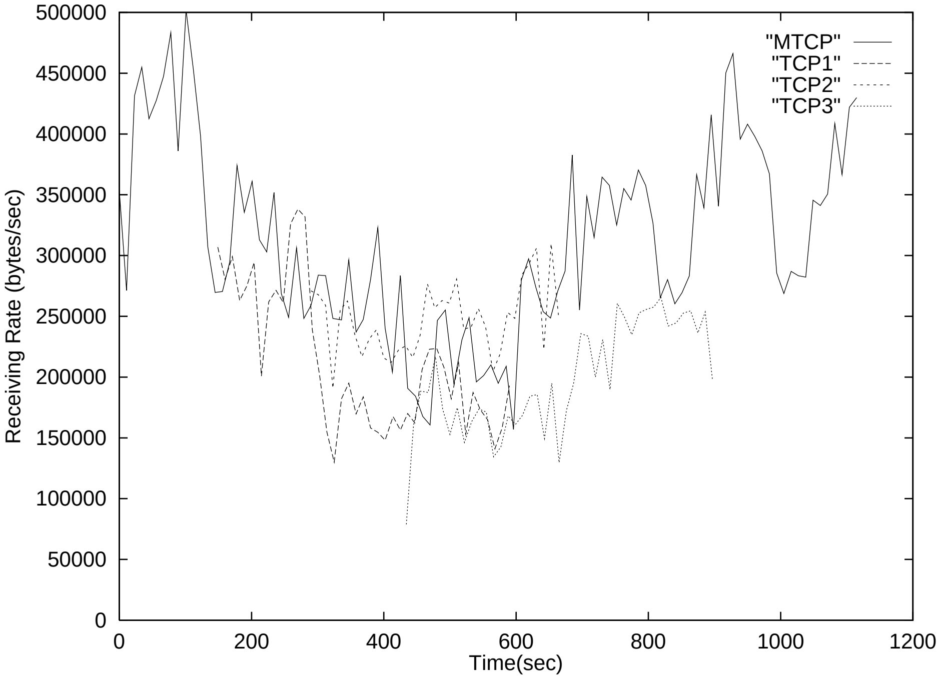

Figure 11: The timed average of receiving rates when three new TCP connections are added at different times

References

[1] V. Jacobson. Congestion avoidance and control. In *Proceedings of SIGCOMM*, pages 314–329. ACM, August 1988.

[2] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommedations on queue management and congestion avoidance in the Internet. *Internet Draft*, March 1997.

[3] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. Technical report, Lawrence Berkeley Laboratory, February 1997.

[4] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of SIGCOMM*, pages 328–341. ACM, August 1995.

[5] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (RMTP). In *Proceedings of INFOCOM '96*. IEEE, March 1996.

[6] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of Multimedia 1996*. ACM, 1996.

[7] M. Hofmann. A generic concept for large-scale multicast. In *B. Plattner (ed.), Broadband Communications, Proceedings of International Zurich Seminar on Digital Communications (IZS '96)*. LNCS 1044, Springer Verlag, February 1996.

[8] B. N. Levine, D. B. Lavo, and J. J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proceedings of Multimedia 1996*, pages 365–376. ACM, 1996.

[9] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of SIGCOMM '94*, pages 24–35. ACM, May 1994.

[10] Alfred C. Weaver. *Xpress Transport Protocol Version 4*. University of Virginia.

[11] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol. In *Internet Request for Comments RFC 1301*, February 1992.

[12] K. P. Birman and T. Clark. Performance of the Isis distributed computing toolkit. Technical Report 94-1432, Cornell University's Computer Science Department, June 1994.

[13] D. Dolev and D. Malki. The Transis approach to high availability cluster communication. *Communications of the ACM*, April 1996.

[14] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems*. LNCS 938, Springer Verlag.

[15] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of SIGCOMM '95 Conference*, pages 342–356. ACM, October 1995.

[16] B. Sabata, M. J. Brown, and B. A. Denny. Transport protocol for reliable multicast: TRM. In *Proceedings of the IASTED International Conference on Networks*, pages 143–145, January 1996.

[17] R. Van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(9):64–70, April 1996.

[18] J. R. Cooperstock and S. Kotsopoulos. Why use a fishing line when you have a net? an adaptive multicast data distribution protocol. In *Proceedings of 1996 USENIX Technical Conference*, January 1996.

[19] A. Koifman and S. Zabele. A reliable adaptive multicast protocol. In *Proceedings of INFOCOM '96*, pages 1442–1451. IEEE, March 1996.

[20] I. Rhee, S. Y. Cheung, P. W. Hutto, and V. S. Sunderam. Group communication support for distributed multimedia and CSCW systems. In *Proceedings of 17th Interanational Conference on Distributed Computing Systems*. IEEE, June 1997.

[21] S. Pingali, D. Towsley, and J. F. Kurose. A comparision of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of SIGMETRICS*. ACM, 1994.

[22] M. Yajnik, J. Kurose, and D. Towsley. Packet loss correlation in the MBONE multicast network. Technical Report CMPSCI 96-32, University of Massachussetts, Amherst, MA.

[23] V. Paxson. End-to-end routing behavior in the Internet. In *Proceedings of SIGCOMM '96*. ACM, August 1996.

[24] M. Grossglauser. Optimal deterministic timeouts for reliable scalable multicast. In *Proceedings of INFOCOM '96*, pages 1425–1432. IEEE, March 1996.

[25] D. DeLucia and K. Obraczka. Multicast feedback supression using representatives. In *Proceedings of INFOCOM '97*. IEEE, April 1997.

[26] M. Handley. A congestion control architecture for bulk data transfer. In *The Reliable Multicast Research Group Meeting in Cannes*, September 1997.

[27] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of SIGCOMM*, pages 117–130. ACM, August 1996.

[28] L. Vicisano, L. Rizzo, and J. Crowcroft. TCP-like congestion control for layered multicast data transfer. In *Proceedings of INFOCOM '98 (to appear)*. IEEE, April 1998.

[29] T. Turletti, J. C. Bolot, and I. Wakeman. Scalable feedback control for multicast video distribution in the Internet. In *Proceedings of SIGCOMM '94*. ACM, August 1994.

[30] B. N. Levine and J. J. Garcia-Luna-Aceves. A comparision of known classes of reliable multicast protocols. In *Proceedings of International Conference on Network Protocols*. IEEE, October 1996.

[31] M. Hofmann. Adding scalability to transport level multicast. In *Proceedings of Third COST 237 Workshop - Multimedia Telecommunications and Applications*. Springer Verlag, November 1996.

[32] K. Thompson, G. J. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network Magazine*, 11(6):10–23, November/December 1997.

[33] S. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[34] S. Floyd. Requirements for congestion control for reliable multicast. *The Reliable Multicast Research Group Meeting in Cannes*, September 1997.

[35] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[36] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *Computer Communications Review*, July 1996.

[37] S. Y. Cheung and M. H. Ammar. Using destination set grouping to improve the performance of window-controlled multipoint connections. Technical Report GIT-CC-94-32, Georgia Institute of Technology, August 1994.

[38] D-M. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.