Failure Handling in a Reliable Multicast Protocol for Improving Buffer Utilization and Accommodating Heterogeneous Receivers

Gunjan Khanna, John Rogers, Saurabh Bagchi
Electrical and Computer Engineering Department, Purdue University
465 Northwestern Avenue, West Lafayette, IN 47907.
Phone: 765-494-3362. Fax: 765-494-2706.
Email: {gkhanna, jsrogers, sbagchi}@purdue.edu

[Submitted to PRDC '04 as a regular paper]

Abstract

Reliable multicast protocols are an important class of protocols for reliably disseminating information from a sender to multiple receivers in the face of node and link failures. A Tree-based Reliable Multicast Protocol (TRAM) provides scalable reliable multicast by grouping receivers in hierarchical repair groups and using a selective acknowledgment mechanism. In this paper, we present an improvement to TRAM to minimize the resource utilization at intermediate hosts and to localize the effect of slow or malicious receivers on normal receivers. We present an evaluation of the existing protocol on a campus-wide wide area network (WAN) with respect to end-to-end parameters (latency, jitter, data rate), and resource utilization (buffer utilization and memory usage at sender and intermediate hosts). A high-quality streaming mpeg video application is used as the workload for the study. The evaluation brings out the effect of scaling the number of receivers on the parameters. The study is done for the error-free case and with message delay, drop and reordering errors. Then we present the design of an augmented protocol, called TRAM++, that optimizes the buffer requirement at intermediate hosts and improves the end-to-end parameters for normal receivers in the presence of some slow or malicious receivers. An implementation of TRAM++ is provided and it is evaluated on the campus-wide WAN with and without errors. The evaluation brings out that, given a constraint on the buffer availability at intermediate hosts of 16% of maximum buffer usage in TRAM, TRAM++ can tolerate the constraint at the expense of increasing the end-to-end latency for the normal receivers by only 3.2% in error-free cases. When slow or faulty receivers are present, TRAM++ is able to provide the same uninterrupted quality of service to the normal nodes while localizing the effect of the faulty ones without incurring any additional memory overhead.

Keywords: tree based reliable multicast, buffer utilization, message errors, slow or unresponsive receivers, evaluation of end-to-end parameters.

1 Introduction

The ability to reliably transmit and update large amounts of data and content in real-time is crucial in a large number of domains, such as for financial services, content delivery network providers, large retail chains, application service providers (ASPs), interactive TV, wireless entertainment, and e-learning providers. It can take considerable time and bandwidth if the sender must send a separate copy to each receiver. IP multicasting allows a sender to distribute data to all interested parties while minimizing the use of network resources. Many applications, however, require reliable data delivery which is resilient to failures of nodes and network links. This requirement is supported by reliable multicast protocols.

A Tree-based Reliable Multicast Protocol (TRAM) is designed to provide multicast reliability that scales to a large receiver population. TRAM ensures reliability by using a selective acknowledgment mechanism and scalability by adopting a hierarchical tree-based repair mechanism. The receivers and the sender of a multicast session dynamically form repair groups. These repair groups are linked together hierarchically to form a tree with the sender at the root of the tree, the receivers at the leaves and entities called Repair Heads (RH) at the intermediate levels of the tree. The RHs cache the messages that pass through them and initiate local repair by resending messages that are nacked by the receivers.

From an end-to-end perspective, we consider latency and jitter as parameters of interest for receivers of a multicast stream. We consider the scenario of a video stream being delivered by TRAM and measure its output parameter values. In addition to the performance metrics, there are resource utilization metrics which characterize a reliable multicast protocol. TRAM ensures the recovery from missing messages by nacks and retransmissions by the RHs. The RHs therefore need stable storage to buffer messages that have not yet been received by all the receivers. The aggregate buffer utilization in the system, combined over the sender and all the RHs, is taken as one indicator of resource usage in the system. In this paper, we evaluate the existing TRAM implementation with respect to performance and resource utilization, with varying number of receivers. The evaluation is performed on a campus-wide wide area network (WAN) with receivers located at different distances from the source. The ability of the reliable multicast protocol to tolerate faults is evaluated under different message error conditions. The errors injected are message drops, reorders, and delays, and the performance metrics are measured under these conditions.

It has to be considered that in a large group of multicast receivers, not all the receivers will be identical. Considering a practical scenario where different receivers are present in different subnets, each has varying traffic conditions and processing constraints. Some of the receivers will be slow in processing the messages and generating acks. Some receivers may be unresponsive for long periods of time, either because of a natural failure or performance bottleneck, or because of malicious purpose. The design point should be that the QoS measures of the normal receivers should be affected as little as possible because of the slow and unresponsive receivers. We propose a protocol based on TRAM, called *TRAM++*, which localizes the effect of slow or unresponsive receivers on the correctly functioning receivers. TRAM++ provides a mechanism to track unresponsive receivers and beyond a threshold, prune the receivers. Since the receivers are not homogenous, maintenance of synchrony among the receivers is sacrificed as a deliberate design choice to provide fairness to normal receivers. Pruning is considered a necessary step to keep the system performance above a tolerable limit, bound the resource usage and to eliminate malicious receivers.

TRAM++ also optimizes the aggregate buffer usage in the system under normal conditions without any degradation of the performance parameters. It achieves this by dropping the guarantee of recovery of missing messages from the immediate RH to which the receiver is connected. The RH may drop messages from its buffer in order to accommodate new messages. The guarantee of recovery is provided by the sender which buffers messages in stable storage till they have been acknowledged by every receiver.

The study shows that both TRAM and TRAM++ scale well with respect to latency as the number of receivers is increased, up to a total of 30. TRAM is not successful in isolating the normal receivers from the effect of faulty or malicious ones. TRAM++ is able to achieve this through its protocol of differentiated acks and buffer management. TRAM++ under a constraint of 16% of the TRAM buffer availability at the RH is able to maintain the latency within an overhead of 3.2% in the error free scenario. It is also able to prune malicious receivers faster because of local decision making ability at the RH without any additional memory overhead. It is also able to isolate and prune a malicious receiver that constantly joins and leaves the multicast group in an attempt to cause denial of service in the system. The rest of the paper is organized as follows. Section 2 presents previous related work. In section 3, we present the TRAM and the TRAM++ protocols. Section 4 details the

implementation of TRAM++. Section 5 gives the experiments and results, under no errors and with errors. Section 6 concludes the paper.

2 Related Work

IP multicast is a protocol that defines a mechanism for one or more senders to send data to a group of receivers —a collection of one or more hosts identified by a single class D IP address [ERI94]. The abstraction of multiple hosts into a single entity provides several advantages over unicast, or "point-to-point," protocols: the sender(s) only manages a connection with the group, not all of its members; and data can be sent to all the receivers without multiple transmissions from the sender(s). In situations where similar data is sent to multiple hosts, multicast IP can provide bandwidth conservation, reduced resource requirements for the server, and an increase in overall scalability.

Though the general idea of IP multicast is simple, the message delivery is purely best-effort and there is no guarantee that the data is going to reach all the multicast receivers. The loss of messages may be due to congestion, failures of nodes, failures of links, etc. To address these concerns, an enhanced category of multicast IP called reliable multicast has evolved. It defines mechanisms to monitor host population, congestion, transfer speeds, and data loss on a per-host basis, and mechanisms to counter the loss, thus providing a more reliable multicast environment.

Due to the wide variety of applications which require reliable multicast, it is considered that a “one size fits all” protocol is infeasible. Therefore, three classes of protocols have been proposed in the literature – (i) NACK oriented protocols; (ii) Tree-based ACK oriented protocols; (iii) Asynchronous layered coding protocols that use forward error correction (FEC). Of these classes, the second or a hybrid of the first two classes is of relevance to our current work. Several protocols have been presented and studied [FLO96, PAU96, YAV95]. We chose TRAM as the representative protocol for study since it had openly available source code and a large active user community which was extremely useful in setting up and troubleshooting the system. The detailed evaluation of TRAM presented here is expected to shed light on the other protocols too since many design decisions are similar, such as local recovery at the repair head (or the counterpart in the particular protocol), and ack aggregation. The concern about constraining resource usage at intermediate hosts and localizing the effect of slow or malicious receivers applies to all reliable multicast frameworks.

TRAM was first presented by Chiu *et al* in [CHI98]. This study evaluated some of the parameters considered here, namely, rate, loss and cache occupancy, but using a simulation model. The simulation model made several simplifications – all RHs were at a fixed distance from the sender, no node failures were simulated and only a small subset of the links (between the RH and the receivers) were injected with failures. The congestion control mechanism in TRAM was studied in a recent paper by Chiu *et al* [CHI02]. The two aspects of congestion control – receiver feedback based windowing and server data rate based traffic shaper – are studied using an implementation for a LAN and an emulator for a WAN. The study showed how to dynamically adjust the data rate used to schedule packet transmission at the sender to smooth the transmission. The authors in [CHI00] examine the issue of pruning decisions in multicast transport protocols. The decision boils down to choosing a minimum data rate and pruning receivers that fail to meet the minimum rate. It is mentioned that the minimum rate has to be chosen carefully so as not to prune genuine receivers experiencing occasional network bottlenecks. Determining an optimal data rate for the system is dependent on the kind of network and its traffic conditions, and is a complex decision. The rate of the entire group is controlled by the slowest receiver in the unpruned set of receivers. This results in slowdown observed by normal receivers even in completely disjoint parts of the multicast tree. The scheme in [CHI00] is a simplification of the more general scheme called *optimal pruning* described in [JIA00]. In that paper, the authors propose multiple subgroups of receivers with a utility function for individual nodes in the subgroup and one for the subgroup as a whole. The algorithm presented aims to pick the subgroup that maximizes the sum of node utility and subgroup utility. We believe that the algorithm can work if it is possible to assign utility functions for all possible subgroups. In the practical scenario considered in our paper, it was not possible to come up with a utility assignment. Also, the pruning decision needs to be rapid to isolate malicious receivers and the choice algorithm in Jiang's work runs in exponential time.

The design point of providing stable storage only at end points has been proposed and implemented in the context of publish-subscribe systems for a system called Gryphon to provide reliable exactly-once message delivery in the face of node failures [BHO02]. The work assumed the extreme design point of no stable storage available at the intermediate nodes whereas in our study this is a parameter that can be tuned.

A promising approach to building reliable distributed systems is group communication [RAY96,HAY98]. Reliable group communication is an important paradigm to build distributed applications that require multi-peer interaction with different levels of consistency. At least a decade of research has gone into designing and improving group communication protocols to provide membership, multicast and ordering services. However, such protocols are overkill for our goal. First, the members of a group are considered homogeneous and group joins and leaves are made visible to all the group members. Second, the design principle in our system is not to enforce synchronicity among the group members which is what the group communication protocols are designed to achieve. Finally, group communication paradigm is particularly suited to local area networks with tight bounds on end-to-end latency which may not be achievable in the wide area network considered here.

3 Protocol Description

3.1 TRAM

The detailed description of TRAM can be found in [CHIU98]. We provide an overview here and present details of the features relevant to the study. TRAM is distributed as a part of the Java Reliable Multicast Service (JRMS) by Sun Microsystems [SUN03]. JRMS is a set of libraries and services for building multicast-aware applications. TRAM is designed for high scalability targeted towards multicasting streaming data from a single sender to a large number of receivers. TRAM ensures reliability by using a selective acknowledgement mechanism. An ack is sent in the form of an offset and a bit vector once every ack window (32 packets). It provides scalability by adopting a hierarchical tree-based repair mechanism. The receivers and the data source of a multicast session in TRAM interact with each other to dynamically form repair groups. These repair groups are linked together in a hierarchical manner to form a tree with the sender at the root of the tree. Figure 1 shows a typical TRAM repair tree. The nodes participating in TRAM play three roles, some nodes playing multiple roles – sender, receiver and repair head (RH). Every repair group has a receiver that functions as a group head; the rest function as group members which are said to be affiliated with their head. All members receive data multicast by the sender. The group members report lost and successfully received messages to the group head using a selective acknowledgement mechanism. Every ack message contains a start message number indicating the first missing message, and a bit vector, with a 1 denoting a missing packet and a 0 denoting a received packet. If no packets are missing, the message number indicates all messages prior to and including this one has

been received and the bit vector is of zero length. An ack message is sent after every *ack window* worth of packets has been received, or an *ack interval* timer goes off. The RHs cache every message sent by the sender and provide repair service for messages that are reported as lost by the members. The RH's maintain a high and low water mark for monitoring cache occupancy. If the amount of buffer occupied by the packets goes beyond the high water mark, an attempt is made to purge the cache. Failure to do so is taken as an indication of congestion in the network. The RHs aggregate acks from all its members and send an aggregate ack up to the sender to avoid the problem of ack implosion. The data rate sent out by the sender is bounded by maximum and minimum rates configured at the sender. Receivers that cannot keep up with the minimum data rate can be pruned from the repair tree.

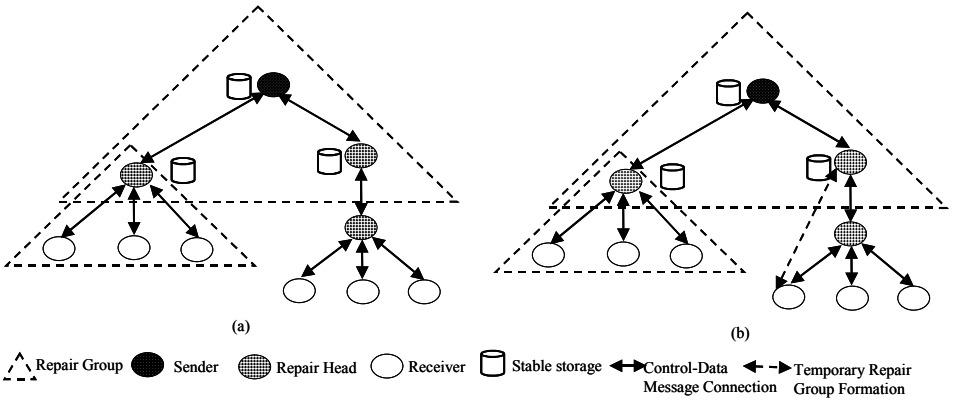

Figure 1. (a) TRAM hierarchical repair tree (from [CHIU98]) (b) TRAM++ hierarchical repair tree with temporary reaffiliation

Figure 1(a) shows a TRAM deployment with a sender, two levels of RHs and multiple receivers connected through links over which bi-directional data and ack messages flow. Two examples of repair groups are shown, one involving the sender and the three RHs at the first level, and the second showing a RH and its receivers.

3.2 TRAM++

TRAM++ builds upon TRAM with the following two goals

1. Reduce the resource requirement at the repair heads, chiefly cache utilization, but also processing.
2. Handle slow or malicious receivers in the environment while localizing their effect on correctly functioning receivers.

To achieve these goals, TRAM++ introduces the changes described below. A figure of the hierarchical structure in TRAM++ with a sender, receivers and repair heads is shown in Figure 1(b).

Buffer management at RH: The design point in TRAM++ is that the RHs may be spread over a wide area and have constraints on available buffer, while the sender has higher, though not infinite, buffer capacity. TRAM++ optimizes the buffer requirement at the RHs by pruning old messages even if they have not been acknowledged by all its receivers. The advantage is that this frees up the buffer resources at the RH for accommodating new messages which are required for the well-behaved receivers to make progress. Consequently, a nack from a receiver may not always be satisfied locally at the immediate RH. A message is not discarded from the sender's storage till it has been acked by *all* the receivers. Therefore, a nack can always be satisfied by the sender. When a RH cannot satisfy a nack, it indicates to the receiver to initiate a *temporary re-affiliation* with a RH at a higher level. This is shown through the dotted arrow in Figure 1(b), where the receiver re-affiliates temporarily for recovering the messages its RH does not have. The reaffiliation is transient and lasts for the duration of recovery of the single packet. This process is repeated recursively if recovery is not successful at the higher level, till the receiver reaffiliates with the sender at which point its nack is guaranteed to be satisfied.

Handling Slow or Malicious Receivers: TRAM lets the data rate be driven by the slowest receiver. Therefore, the effect of a slow receiver is visible to the correctly functioning receivers all across the network. Even if the pruning feature of TRAM is turned on (which it is not for most deployments), the threshold minimum tolerable data rate is set quite conservatively and there are likely to be large periods of slowdown to the normal receivers. On the contrary, TRAM++ localizes the disruption to the part of the tree where the lagging receivers reside. In TRAM++, the RH uses two types of acks – a *greedy ack* and a *permanent ack*. The maximum sequence number of the packet that the sender should send down is sent piggybacked with acks. The greedy ack is sent by the RH upwards when its buffer reaches the low water mark. The purpose of sending the greedy ack is to indicate to the sender to send new data down even though all the receivers have not acked yet. The permanent ack is sent once all the receivers have acked. The role of this ack is to let the sender know that reclamation of storage is possible. In TRAM, the sequence number sent upwards is determined by the slowest receiver thus affecting the data rate observed by all the receivers, normal or laggard. In TRAM++, the sequence number is determined by the RH's available buffer capacity. Incorporating the additional ack requires additional computation at the sender and the RHs which is the same cost as for the basic ack determination in TRAM. But in a failure free system where all

receivers are keeping up with the data rate, the greedy acks are not sent and therefore, the additional processing overhead is not observed.

TRAM++ has the functionality to prune receivers which are considered lagging beyond an acceptable degree. The metric used for the pruning decision is the percentage of retransmission requests which cannot be locally satisfied as a fraction of the total number of packets. When the metric exceeds a tunable threshold parameter, the receiver is pruned. This is an effective means of removing malicious receivers which may increase the processing load in the system by requesting repeated retransmissions. This may serve as an indication to the receiver to disassociate from the current RH because of its resource constraints and reaffiliate with a more resource rich RH.

4 Implementation

In this section, we present the details of the TRAM implementation which are of importance to the protocol and the changes that have been made in TRAM++.

4.1 TRAM Implementation

The TRAM code is multi-threaded. These threads are responsible for carrying out the group management functions in addition to basic sending and receiving of data packets. *GroupMgmtThread* is the main thread which is responsible for starting up TRAM, initiation of the *beacon messages* by the sender and affiliation of the receivers to the senders or repair heads. The beacon messages are used to advertise the session and invite nodes to join the multicast session. This thread performs the task of sending periodic hello messages among the receivers and its head. Each receiver also maintains a backup list of heads which it can switch to if the current head resigns or fails. Once the data transmission phase starts, *InputDispThread* and *OutputDispThread* come into picture. *OutputDispThread* transmits the packets. *InputdispThread* gives the packet to all the listeners and hence, each entity calls its received packet method to get the desired packet. The sender and the repair head's sending functionality use *HeadAck* class to receive ack packets. The receivers use *MemberAck* class to receive data packets and to send acks. Repair head uses *MemberAck* class, as it is a receiver for the sender above, to send cumulative acks. Figure 2 shows pictorially the threads or methods which are used for upstream and downstream communication in TRAM.

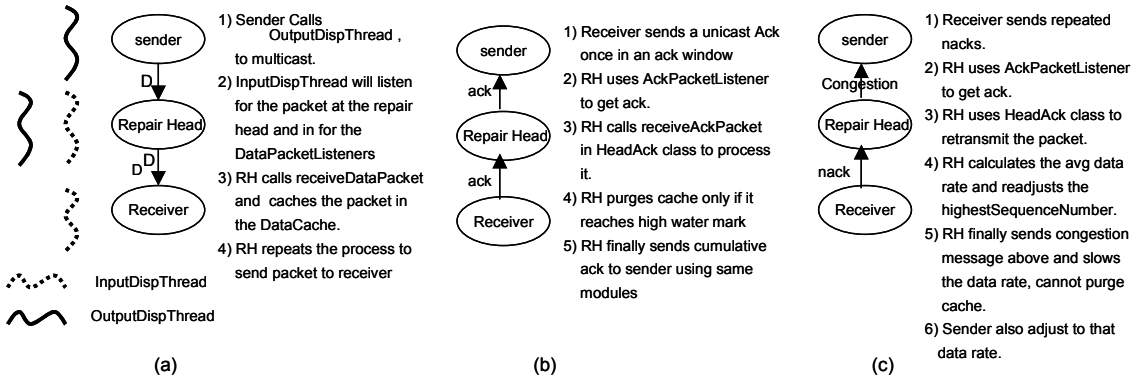

Figure 2. TRAM message processing (a) downstream with no errors, (b) upstream with no errors, (c) upstream with message, node or link errors

In the case of errors, the downstream path is identical to the error free case. In the upstream path, the receiver sends nacks to the RH which transmits the requested packets; the RH adjusts the data rate and sends to the sender which finally adjusts the data rate.

4.2 Modifications for TRAM++

To create TRAM++ from TRAM, we have added new message types to the existing message and sub-message types of TRAM. We have tried to minimize the changes to the basic TRAM structure and be able to add a separate layer of functionality which transforms TRAM to TRAM++. The processing of messages downstream and acks upstream in the no error scenario is shown in Figure 3.

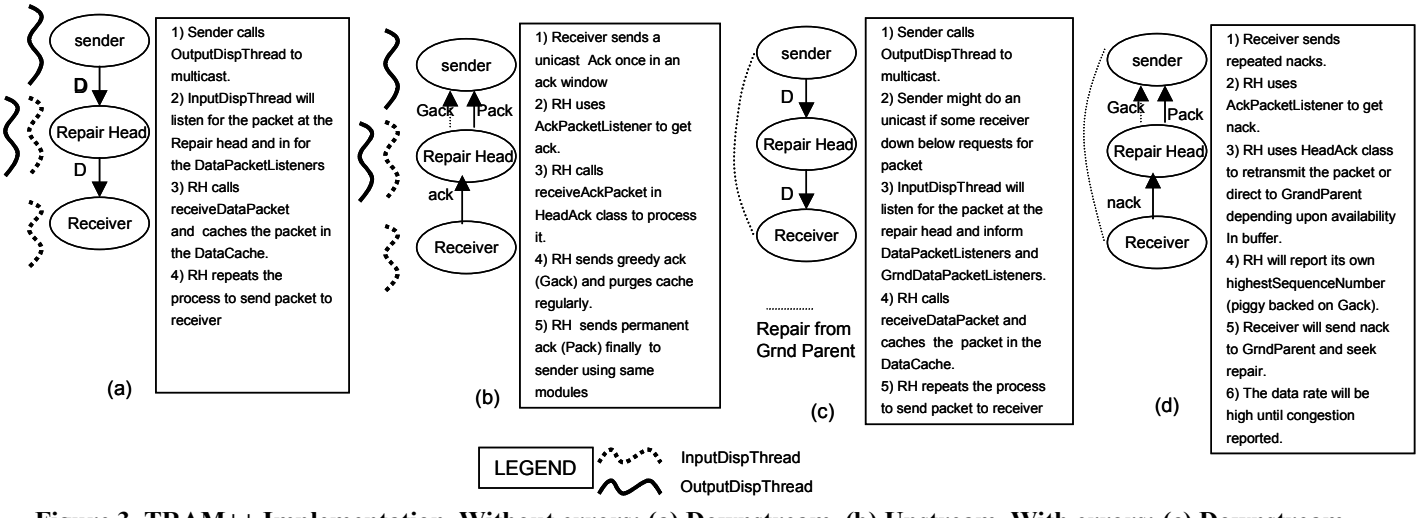

Figure 3. TRAM++ Implementation. Without errors: (a) Downstream. (b) Upstream. With errors: (c) Downstream. (d) Upstream.

Recollect that a RH may not be able to satisfy a receiver's nack. In order for the RH to indicate to the receiver that it doesn't have the packet in cache, we have introduced a sub-message type called NOT_REPAIRED.

This sub-message type forces the receiver to go for *temporary reaffiliation*. The data packet sent by the RH in response to the above type of nack, has a payload which only contains information about the higher-level repair head, also called *grand repair head (GRH)*. The receiver uses this information and sends a request for the missing packet to the new GRH. The GRH unicasts the requested data packet to the receiver. In TRAM, only multicast data exists so there is no notion of sending data through unicast. To enable this, we added a message type called UCAST_DATA. A flag is added to the ack packet of TRAM to differentiate between greedy and permanent ack types.

The ack processing mechanism of TRAM is modified for TRAM++. In TRAM++ as the ack (implicit nack) is received by the repair head, it checks which packets the receiver has asked for retransmission. If the packet is not in the cache, the repair head sends a packet to the receiver giving it the information that the requested packet could not be repaired and provides it with the address and port of the GRH. The execution steps of TRAM++ with errors are shown in Figure 3. The downstream processing is identical to the TRAM case except that the sender may send a unicast if some receiver had directly requested a retransmission through the temporary reaffiliation process.

5 Experiments and Results

TRAM and TRAM++ are installed on a campus-wide WAN and experiments are conducted on the testbed. The experiments have two broad goals:

1. Evaluate the scalability and robustness of TRAM with respect to different types of message errors.
2. Evaluate the improvement provided by TRAM++ in the event of failures and overhead incurred by TRAM++ under failure-free conditions.

5.1 Testbed Setup

The testbed has a sender, multiple repair heads and varying number of receivers. The computing machines are distributed across campus in the Mathematics (clusters *al-zn*), Electrical Engineering (named *pegasus* and *lyra*, together with its cluster) and Materials Science (*msee190*) buildings. The machine named msee190 is always used exclusively as the sender. All the machines run RedHat Linux. It is important to note that the implementation is done and the experiments performed on a production campus-wide wide area network with normal traffic coexisting with the reliable multicast traffic.

MSEE machine (Sender)	Pentium IV 1.5 GHz machine with 1 GB of memory
EE machines	Pentium 4 2.26 GHz processor with 1 GB memory, 533 FSB and 512 KB cache
Math machines	450 MHz Pentium II machines with 256 MB of memory divided into 13 clusters with 48 machines each
Rtr_{MSEE}	Cisco 5505 switch
RtrMATH	Cisco 6509 switch
Links	Intra-cluster links 100 Mbps, inter-cluster links 1 Gbps

Table 1. Hardware configuration

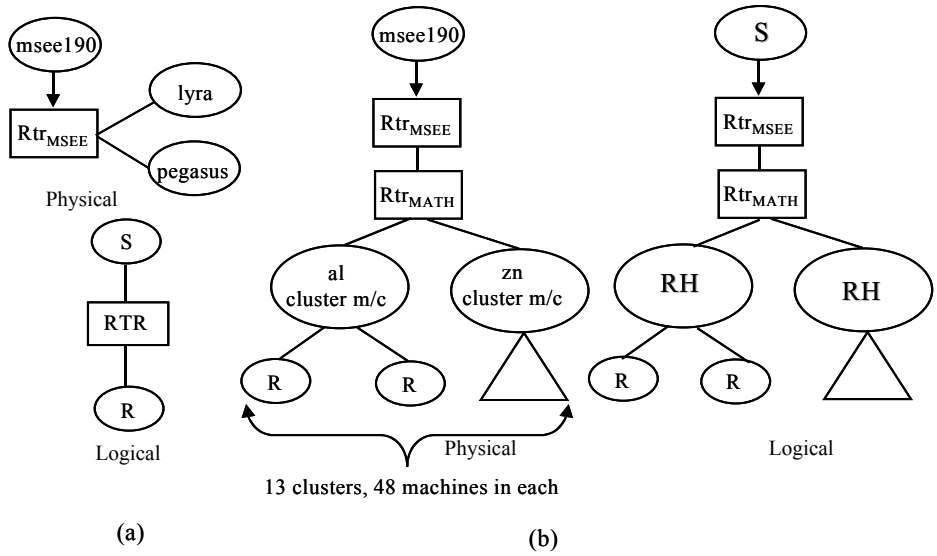

Figure 4. Sample physical and logical configuration for deployment with (a) 2 hops (b) 4 hops between sender & receiver

Different experiments are conducted with the receivers at different hops distance from the sender. In Figure 4, we show the physical and logical views of two different hop topologies. The physical topologies show the routers as well as the protocol participants (sender, receiver or RH), while the logical topologies only show the participants. A *hop* is defined as a link in the physical topology. This is different from the traditional definition of TTL in networking. For example, traversal from a receiver within a Math cluster machine to its RH in the same cluster does not decrement the TTL, but is considered a hop. We felt this to be reasonable because there is a cost of processing both at protocol participants and at the routers.

We use the reliable multicast infrastructure to send a high bandwidth Mpeg-2 video data feed from the sender to the receivers. A 40 kbps feed is sent in 1316 bit payload packets, leaving space for the TRAM and the IP headers to fit within 1500 Ethernet MTU. The number of packets sent is at least 8000, with a larger number if the initial transients, which are discarded from our results, are longer. The parameters of the protocol set at the participants are shown in Table 2.

Parameter	Value	Parameter	Value
(Max, Min) data rate	(40 kbps, 1 kbps)	Stable storage size at RH (TRAM) Low water mark: High water mark (num packets)	1200 packets 400:800
Stable storage size at RH (TRAM++) Low water mark: High water mark (num packets)	200 packets 32:120	Pruning of receiver (% of packets asking for reaffiliation) (TRAM++)	0.08%

Table 2. TRAM and TRAM++ configuration settings

Output measures: The output metrics studied in the experiments are shown in Table 3 along with a brief interpretation of each.

Latency	End-to-end measure between sender and receiver. Clocks on the machines are synchronized using the *nntp* service.
Jitter	The difference in latencies between successive packets. This metric is important for a smooth video playback.
Data rate	The data rate computed at the sender which should lie between the max and the min data rates specified. If a slow receiver tends to reduce it beyond the limit, it will be pruned.
Buffer utilization	Stable storage used at the sender or the RH (which one it is will be clarified in the context).
Memory utilization	The amount of main memory used by the RH process. This is taken as an indicator of the processing resources needed at the RH since the processor utilization stays very close to zero.

Table 3. Output metrics used in the evaluation

Normal and Error Injection Runs: A single run of the experiment is defined as the transfer of at least 8,000 packets of the video feed. A normal run is one where no errors are injected, though the variability of the environment may create congestion and transient instability such as spikes in latency. In the error injection runs, three kinds of message errors are simulated in the network – drops, delays and reordering. The first type of error is message drop where a message is dropped on the downward link between the RH and a single receiver. In one error injection run, a single receiver is identified as the faulty (or, malicious) one and the error injector works on its link. For simulating message delays, the ack packet from the faulty receiver is delayed on the upstream link to the RH. For injecting message reorderings, we vary the inter-message gap (M_d) between the messages which are to be reordered. Recollect that the ack window (W_a) is the number of packets for which one ack gets sent. If $M_d < W_a$, then TRAM buffers the out-of-order packet and delivers it when an ordered sequence of packets can be created. This results in behavior identical to the error free case since the element in the ack bit vector is reset to 0 when the message is delivered in-order. If $M_d \geqslant W_a$, then the receiver sends a nack in the next ack window for the message whose place was taken by the out-of-order packet. This is identical to the message drop case and hence we do not separately show results for the message reordering case.

5. Evaluation of TRAM

5.1.1 Error-free Case

In this experiment, we investigate how TRAM scales with the number of receivers. The latency curves in Figure 5(a) show that the latency is substantially less for the single hop case since the message is on a direct connection and does not have to traverse a router. The latency for 2, 3 or 4 hops is comparable. None of the latencies become substantially worse with increasing number of receivers. The jitter for our environment is found to be below 1 ms on an average which is lower than the granularity of our timing mechanism.

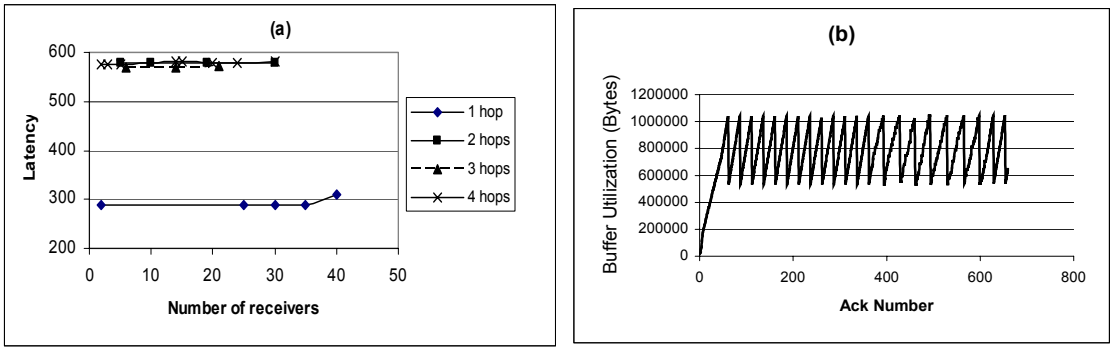

Figure 5. Evaluation of TRAM in error-free case

The next experiment investigates the buffer utilization at the RH. TRAM is set up to use one-thirds of maximum buffer capacity (1200 packets) as the low water mark and two-thirds as the high water mark. Therefore, the buffer utilization oscillates between 400 and 800 packets, i.e., between 526.4 KB and 1.0528 MB. This is found to be true as the number of receivers is increased and is also independent of whether the utilization is at the RH or the sender. When we measure the data rate against time (or equivalently the number of the ack packet), it is found that in the normal case, after the initial transient, the data rate picks up and tends towards the max data rate specified in the configuration.

5.1.2 Error Injection Case

For TRAM, the effect of the error on the faulty receiver and the normal receiver will be identical, and therefore no distinction is made in the presentation. Figure 6 shows the variation of latency with the number of receivers for different message drop rates. The drop rates considered here are 1 every 50 packets, 3 every 50 and 5 every 50. The dropped packets are all consecutive. Figure 6(a), (d) show the variation of latency and Figure 6 (b), (e) show data rate variations against time for 2 different packet drop rates – 3 out of 50 and 5 out of 50. The buffer utilization at the RH (Figure 6(c),(f)) shows an interesting trend. The purging of the buffer begins

when its size reaches the high water mark (800 packets), and while under error-free conditions, the purging would have been able to reduce the buffer utilization to 400 packets, here the reduction is only down to about 750 packets. If the drop rate is increased to 5 out of every 50 packets, the affected receiver is pruned. On pruning, the buffer is purged completely since the packets were being buffered to accommodate the receiver that just got pruned. Therefore the utilization comes down to zero. Immediately after pruning, the sender tries to increase the data rate and the utilization goes back to the usual oscillation between 400 and 800 packets till the pruning happens next.

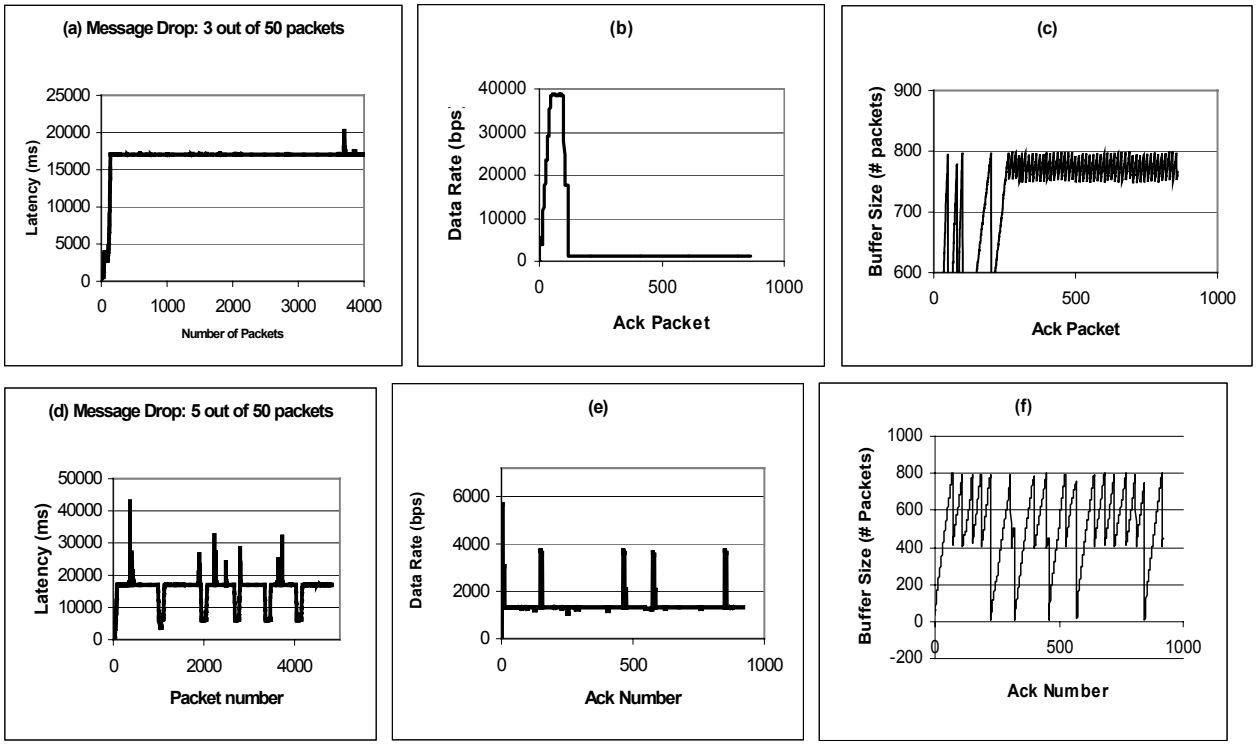

Figure 6. Effect of message drop on TRAM: (a)&(d) Latency for 3 and 5 packets dropped out of 50, (b)&(e)Respective Data Rates, (c)&(f)Respective Buffer Occupancy

In Figure 7, we show the effect of introducing message delays in TRAM. In the first experiment, a delay of 8,000 ms is introduced. The latency shows a regular spike of 8,000 ms and this spike repeats roughly every 32 packets since the ack window is set to 32 and a delay is introduced for every ack. The data rate shows congestion control at work. The sender tries to increase the data rate to the max rate (40 kbps), but is forced back because of the delayed ack. It reduces the data rate, but is able to sustain a rate greater than 1 kbps, and as a result, the receiver does not get pruned. The buffer utilization varies between the low and high water mark as in the error free case (Figure 5(b)) and is therefore not shown here again. It is found that a delay of 10,000 ms

causes pruning of the slow receiver. Thus, it is seen that in TRAM, once pruning of misbehaving receivers happens, the remaining receivers continue to see performance as in the error free case.

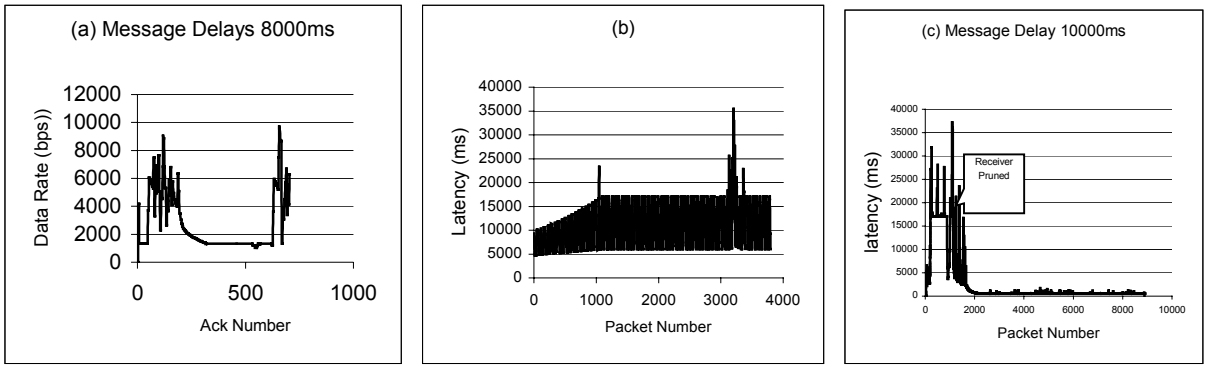

Figure 7. Effect of message delay on TRAM: (a)-(b) – 5 receivers, delay of 8000 ms, no pruning; (c) – 2 receivers, delay of 10000 ms, pruning.

5.2 Evaluation of TRAM++

5.2.1 Error-free Case

The results of the scalability test of TRAM++ are shown in Figure 8. It is observed that the protocol is scalable like TRAM in the range under consideration (5-30 receivers). The variation in latency in this range is about 1.01%. The average latency in the range is 584.44 ms, which gives a 3.2% overhead over TRAM. As the buffer constraint is varied in TRAM++, the relative overhead is found to remain constant in the error free case. This is expected since the buffer is not completely utilized in the absence of errors. The overhead is ascribed to three main reasons: extra message schema matching due to the introduction of new message types, aggressive cache pruning to satisfy the storage constraints at the RH, and additional control messages – two kinds of acks being sent upstream by the RHs.

Regarding the comparative resource usage, the CPU utilization is very close to zero for the maximum data rate that the network infrastructure can support and therefore cannot form a meaningful point of comparison between the two protocols. The main memory utilization for both TRAM and TRAM++ vary between 7.5% and 30.0%. The sender buffer utilization curve shown in Figure 8(b) oscillates between 400 and 800 packets as in TRAM's buffer utilization. In TRAM++ a maximum of 5 reaffiliations are allowed per receiver. If a receiver tries to reaffiliate more than that, it is pruned assuming malicious nature. The RH buffer utilization varies between the maximum pruning level (32 kB) and the high water mark (120 kB) and is shown in Figure 8(c). In the no error case, this buffer utilization does not depend on the number of receivers. A system designer can set

the buffer upper bound knowing the stable storage constraints and TRAM++ will operate under the bound. The data rate supported by TRAM++ also approaches the max data rate parameter set in the system (40 kbps).

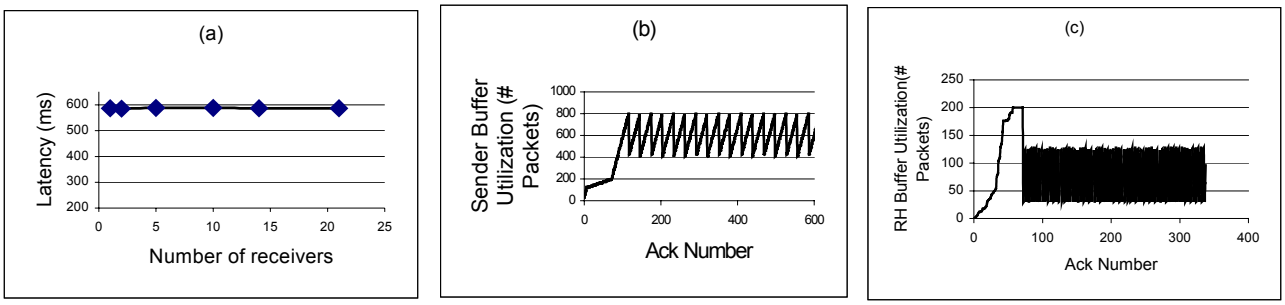

Figure 8. Evaluation of TRAM++ in error-free case: (a) Receiver latency (b) Sender buffer utilization and (c) Repair Head buffer utilization for 27 receivers

5.2.2 Error Injection

For the error scenarios in TRAM++, the metrics are expected to be improved for the normal receivers compared to the faulty receivers.

For message drops, we conduct two sets of experiments, one with 5 packets out of 50 being dropped, and the second with 2 out of 50. For the 5 of 50 case (Figure 9), it is observed that the malicious receiver gets pruned at around 650 packets, after which the system behaves as in the error free case. The sender's buffer utilization once drops to zero because of purging of entire cache at the time of pruning. Then the utilization oscillates as usual between 400 and 800 packets, and the data rate also tends towards the maximum. However, the buffer utilization at the RH goes up to the maximum buffer space, before purging occurs. For 2 out of 50 packets being dropped (Figure 10), the receiver is repeatedly pruned and rejoins the multicast group. The normal or non-faulty node is not affected at all and its latency remains around the no error scenario value. This achieves the important design goal of TRAM++ of isolating the effect of a malfunctioning receiver to its part of the repair tree. Contrast this to the behavior in TRAM shown in Figure 6 where the latency of the normal receiver is shown to go above 10,000 ms for similar drop rates. We can see in Figure 10(b) that the malicious receiver latency shows spikes followed by drop to zero latency which indicates the point when the receiver is disconnected. We had the malicious receiver repeatedly reconnect to the multicast group to test the robustness of TRAM++ to this kind of malicious behavior. It can be argued that once a receiver has been detected as misbehaving, then its subsequent attempts to reconnect will be denied by keeping track of IP addresses of such receivers. However, it is well known that IP spoofing can defeat such a scheme. Therefore, we decided to use this error model to simulate a real-world

malicious receiver. The sender data rate fluctuates between a rise towards the maximum and a steep descent to near zero as each pruning operation occurs. The sender buffer utilization remains as in the higher drop rate case.

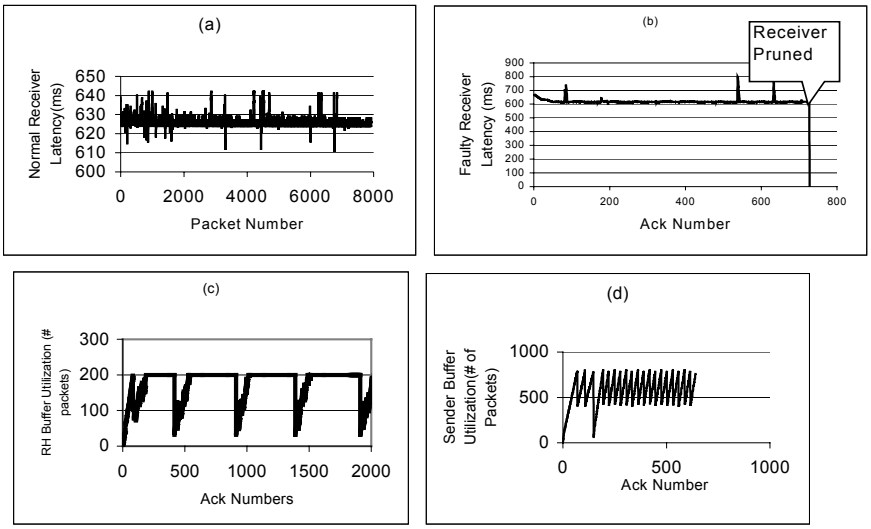

Figure 9. Evaluation of TRAM++ under message drop rate of 5 out of 50 packets.

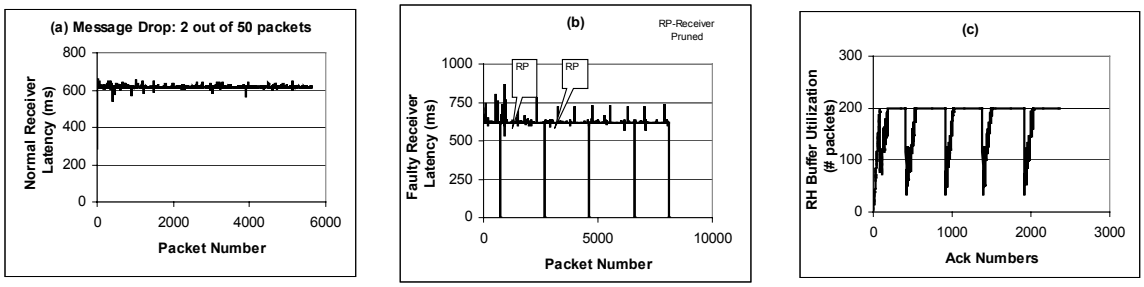

Figure 10. Evaluation of TRAM++ under message drop rate of 2 out of 50 packets

In the case of message delays, the pruning is found to happen for delays of 8000 ms and above. Figure 11 shows a scenario where pruning is not done (delay = 1000 ms) and Figure 12 shows a pruning scenario (delay = 8000 ms). For the no-pruning scenario, the sender data rate and buffer utilization behave as in the error free case (see Figure 5(b) & Figure 8(b) respectively). The latency of the normal receiver remains unaffected though the sender detected congestion causes its latency to rise towards the end of the experimental run. The malicious receiver has a saw-tooth latency pattern with the peak separated from the base by the delay amount.

5.3 Highlights of Result

Some of the important results coming out of the study are summarized here.

1. Both TRAM and TRAM++ scale well with respect to latency as the number of receivers is increased, up to a total of 30. Also, the jitter is negligible even for 4 hops spread across campus. Under error-free conditions,

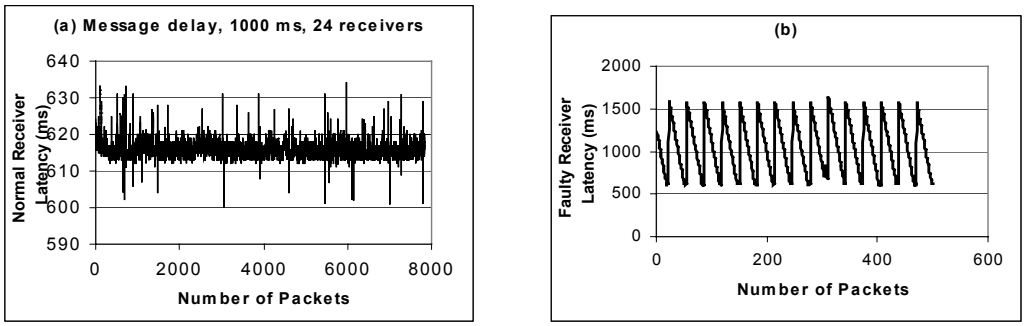

Figure 11. Effect of message delay on TRAM++ for 1000 ms delay

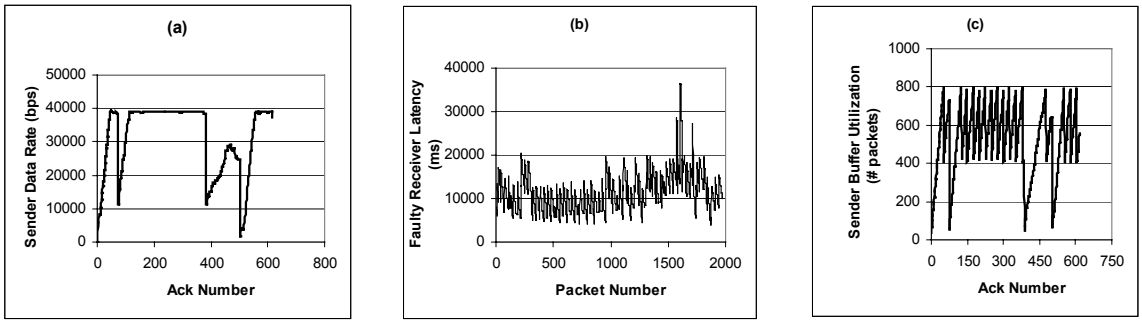

Figure 12. Effect of message delay on TRAM++ for 8000 ms delay (Faulty receiver is pruned)

both were able to maintain a streaming video of data rate 40 Kbps, but the next higher step of 60 Kbps that we tried was not stable. This is because the testbed network was shared and though the rated bandwidth was 100 Mbps, it could not support jitter free 60 Kbps traffic.

2. TRAM is not successful in isolating the normal receivers from the effect of faulty or malicious ones. TRAM++ is able to achieve this through its protocol of differentiated acks and buffer management.

3. TRAM++ under a constraint of 16% of the TRAM buffer availability at the RH is able to maintain the latency within an overhead of 3.2% in the error free case. TRAM++ achieves this without any additional memory overhead. In cases with errors, the latency of normal receivers in TRAM++ is better by a factor of up to 30. TRAM++ is also able to prune malicious receivers faster because of local decision-making ability at the RH based on temporary reaffiliations.

6 Conclusion

In this paper, we have presented an evaluation of a reliable multicast protocol based around hierarchical trees, called TRAM. An implementation of TRAM was installed on machines situated on different parts of the campus, and then an evaluation study was performed. The evaluation brings out its performance in the absence as well as presence of failures. Two design improvements to TRAM were presented and the new protocol called

TRAM++ is evaluated under similar conditions to TRAM. The comparison brings out that both protocols are scalable in latency in the range in which the experiments are done. TRAM++ incurs a minor latency penalty in failure-free conditions. If constraints are enforced on the intermediate nodes, then TRAM++ can enforce the conditions and yet localize the disruption to the system due to a few slow or malicious nodes. TRAM++ also includes an algorithm for pruning nodes deemed too slow or suspected to be malicious.

We are currently working on distributed monitor architecture to detect failures or intrusions and work collaboratively with the TRAM++ participants to isolate the failed or compromised entity. The goal is to be able to diagnose participants that are failed or malicious with low detection latency.

Acknowledgements

This work is supported in part through a grant from the Purdue Research Foundation. Installation support for TRAM and TRAM++ was provided by Dale Talcott and Casey Carlson of Purdue ITAP.

References

[BHO02] S. Bhola, R. Strom, S. Bagchi, Y. Zhao and J. Auerbach, "Exactly-once Delivery in a Content-based Publish-Subscribe System". In Proceedings of the International Conference on Dependable Systems and Networks (DSN'02), pp. 7-16, June 2002.

[CHI98] Dah Ming Chiu, Stephen Hurst, Miriam Kadansky and Joseph Wesley, “TRAM: A Tree-based Reliable Multicast Protocol”, Sun Technical Report TR 98-66, July 1998.

[CHI00] Dah Ming Chiu, Miriam Kadansky, Joe Provino, Joseph Wesley and Haifeng Zhu, “Pruning algorithms for multicast flow control”, Proceedings of NGC 2000 on Networked group communication, pp. 83-92, 2000.

[CHI02] Dah Ming Chiu, M. Kadansky, Joe Provino, J. Wesley, H. Bischof, Haifeng Zhu, “A congestion control algorithm for tree-based reliable multicast protocols”, Proceedings of INFOCOMM ’02, pp.1209-1217, 2002.

[ERI94] H. Eriksson, “MBone: The Multicast Backbone”, Communications of the ACM, pp.54-60, August 1994.

[HAY98] Mark Hayden, “The Ensemble System,” Cornell University Technical Report, TR98-1662, January 1998.

[JIA00] T.Jiang, M.Ammar, and E.Zegura, "On the Use of Destination Set Grouping to Improve Inter-receiver Fairness for Multicast ABR Sessions", in Proceedings of INFOCOMM'00, 2000.

[PAU96] Sanjoy Paul and John C. Lin, “RMTP: A Reliable Multicast Transport Protocol”, INFOCOMM 1996, pp.1414 1424.

[REN96] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis, “Horus, a flexible Group Communication System,” Communications of the ACM, April 1996.

[SRM97] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, “A reliable multicast framework for lightweight session and application layer framing,” IEEE/ACM Trans. Networking, vol. 5, Volume 5, Number 6, pp. 784-803.

[SUN03] Java Reliable Multicast Service. At: http://www.experimentalstuff.com/Technologies/JRMS/

[YAV95] R. Yavatkar, J. Griffioen, and M. Sudan, “A Reliable Dissemination Protocol for Interactive Collaborative Applications”, In Proceedings of the ACM Multimedia '95 Conference, November 1995.