TRAM: A Tree-based Reliable Multicast Protocol

Dah-Ming Chiu, Stephen Hurst,
Miriam Kadansky, Joseph Wesley

SML TR-98-66 July 1998

Abstract:

This paper describes TRAM, a scalable reliable multicast transport protocol. TRAM is designed to support bulk data transfer with a single sender and multiple receivers. It uses dynamic trees to implement local error recovery and to scale to a large number of receivers without seriously impacting the sender. It also includes flow control, congestion control, and other adaptive techniques necessary to operate efficiently and fairly with other protocols across the wide variety of link and client characteristics that make up the Internet as well as intranets. TRAM has been successfully used to implement several bulk data delivery applications. TRAM has been tested and simulated in a number of network environments.

M/S MTV29-01
901 San Antonio Road
Palo Alto, CA 94303-4900

email addresses:
dahming.chiu@east.sun.com
stephen.hurst@east.sun.com
miriam.kadansky@east.sun.com
joseph.wesley@east.sun.com

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@eng.sun.com>.

TRAM: A Tree-based Reliable Multicast Protocol

Dah-Ming Chiu, Stephen Hurst, Miriam Kadansky, Joseph Wesley

Sun Microsystems Laboratories
2 Elizabeth Drive
Chelmsford, MA 01824
U.S.A

1. Introduction

Reliable multicast holds great promise for bulk data transfer applications. Distributing significant amounts of identical data from a single sender to multiple clients can take considerable time and bandwidth if the sender must send a separate copy to each client. Multicasting over a network allows a sender to distribute data to all interested parties while minimizing the use of network resources. However, in order to take advantage of the increasing availability of multicast networking infrastructures, multicast applications require a reliable transport layer.

Multicast scalability with reliability has been a primary design goal of TRAM. TRAM ensures reliability by using a selective acknowledgement mechanism and scalability by adopting a hierarchical tree-based repair mechanism. The hierarchical tree not only overcomes implosion-related problems but also enables localized multicast repairs.

The receivers and the sender 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. These types of trees have been shown to be the most scalable way of supporting reliable multicast transmissions [SURVEY]. Every repair group has a receiver that functions as a group head; the rest function as group members. These members are said to be affiliated with their head. Except for the sender, every repair group head in the system is a member of some other repair group. 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 similar to TCP's [SACK]. The repair heads cache every message sent by the sender and provide repair service for messages that are reported as lost by the members. The dynamic nature of the tree allows it to react to changes in the underlying network infrastructure without sacrificing reliability.

TRAM has intentionally been kept as lightweight as possible. TRAM has been developed as part of a larger project, the Java™ Reliable Multicast Service™ [JRMS]. JRMS includes support for a wide range of features desirable for reliable multicast: group management, security, receiver customization of data, session advertisement, address allocation, etc. JRMS also includes a protocol-independent API, designed to support multiple transport protocols.

This paper is organized as follows:

- Section 2 describes related work in reliable multicast.
- Section 3 describes the major features of TRAM.
- Section 4 details tree-building methods of TRAM.
- Section 5 describes flow and congestion control, acknowledgements, and repairs.
- Section 6 describes our implementation and sample applications.
  Section 7 summarizes the limitations of TRAM.
- Section 8 describes some future possibilities for enhancing TRAM.

Our conclusions are presented in section 9.

2. Related Work

A multitude of models such as SRM [SRM], RMTP [RMTP], and TMTP [TMTP], have been proposed to overcome the challenges of reliable multicast.

SRM[SRM] uses global NACKs to report missing packets, and transmits repairs globally. Timer-based random backoff mechanisms are used to suppress duplicate NACKs and repairs. Data is discarded from caches on a timer basis. SRM supports application-level framing which allows an additional reliability mechanism to be built into applications. SRM has been deployed in shared-data applications with multiple senders, such as wb (a whiteboard application). Recent work [SCALE, LOCAL] has focused on increasing scalability by localizing session messages and error recovery.

RMTP [RMTP, RMTP-II] uses a hierarchical tree-based approach to overcome ACK/NACK implosion at the sender. RMTP supports local repair via localized multicast or unicast, reducing the reach of unsolicited repairs. RMTP's tree formation relies on static configuration. Able repair nodes are designated in advance, then selected by the rest of the members. RMTP is targeted at bulk and live data applications.

TMTP [TMTP] also uses a tree-based approach to overcome ACK/NACK implosion and support local repair. It supports dynamic tree formation in bi-directional multicast environments. TMTP is targeted at bulk and live data applications.

3. Features of TRAM

The TRAM model is tree-based and adopts many of the techniques presented in RMTP and TMTP. The tree formation process is dynamic as in TMTP. The TRAM tree formation algorithm works in both bi-directional multicast environments and uni-directional multicast environments (such as satellite links) with unicast-only backchannels. Repair nodes are elected based on a wide spectrum of criteria, and the tree is continuously optimized based on the receiver population and network topology. The ACK reporting mechanism is window-based like RMTP, with the addition of optimizations to reduce burstiness and processing overhead. The flow control mechanism is rate-based and adapts to network congestion. The sender senses and adjusts to the rate at which the receivers can accept the data. Receivers that cannot keep up with a minimum data rate can be dropped from the repair tree.

Some of the major features of TRAM are:

Reliability: TRAM guarantees delivery of data to any receiver that joins the tree and is able to keep up with the minimum transmission speed specified by the sender. While this level of guarantee cannot ensure applications against delivery failure, features of JRMS [JRMS] can be used to closely keep track of individual members' status.

Avoiding acknowledgement implosion: Point-to-point transports achieve reliability by acknowledging receipt of data directly to the sender. Unfortunately, this does not scale over multicast. A sender trying to field acknowledgements from many receivers will quickly become overloaded. TRAM avoids this implosion by dynamically designating a receiver to be a repair head for a group of members. The repair head fields acknowledgements from each of its group members and supplies them with repair packets. To avoid overload, each repair head is responsible for only a limited number of members.

Local repair: TRAM builds the tree so that repair heads are close to their members. This enables repair heads to perform repairs using small time-to-live (TTL) values, which not only minimizes network bandwidth consumption but also avoids unnecessary processing at receivers not requiring repair.

Automatic configuration: Different applications, networks, or user groups may perform better using different types of trees. TRAM provides for building different types of trees optimized for these different circumstances without adding complexity to the network infrastructure itself. As a receiver group changes, TRAM also provides for ongoing optimization of an existing tree, for instance, enabling a member to find a better repair head. A particular tree's formation method may also be changed at any time during the transmission at the discretion of the sender. TRAM's tree management works in networks providing only unidirectional multicast, as well as those supporting bi-directional multicast.

Rate-based flow control and congestion avoidance: TRAM schedules packet transmission according to a data rate. This data rate is dynamically adjusted based on congestion feedback from the receivers. Congestion feedback is aggregated by repair heads through the tree. The algorithm used to adjust the rate works in different network topologies. The rate is bounded by maximum and minimum rates configured at the sender.

Feedback to the sender: Each member of the tree periodically reports statistics to its repair head. This includes statistics that assist in building the tree (for instance, the number of available repair heads on the tree) as well as reports on congestion conditions, which allow the sender to adapt its data rate to network conditions. This information is aggregated at each level of the tree in order to reduce control traffic to the sender.

Controlling memory use: Each repair head is responsible for ensuring that the data is received by all of its members. This means that a repair head must cache data until it is sure that all of its members have received it. TRAM requires positive acknowledgements from members when data is received. This enables repair heads to reclaim cache buffers containing data that has been received by all members.

Repair group management: Both members and repair heads monitor each other to detect unreachability. Non-responsive members can be dropped from the repair group and corresponding cache buffers can be reclaimed. Non-responsive repair heads can be abandoned by their members in favor of an active repair head. Repair heads are also responsible for detecting receivers which cannot keep up with the minimum transmission rate specified by the sender. While such members cannot be dropped from the multicast group, they can be denied repair head support and receive no repairs.

Scalability: TRAM has been designed to be scalable in many situations, such as large numbers of receivers and sparsely or densely populated receiver groups. TRAM also accommodates wide ranges of receiver capabilities. Control message traffic is designed to be limited in all of these cases.

4. Tree Formation and Management

Figure 1 shows a typical hierarchical repair tree of TRAM with repair heads caching the multicast data sent by the sender and performing localized repairs to their group members.

A hierarchical repair tree can be constructed using either a top-down or a bottom-up approach. In the top-down approach, the tree begins to form from the sender. That is, the sender will first start accepting members for its repair group and each receiver, upon becoming a member of some group, can start to form its own repair group. A receiver has to be part of the tree to form a repair group of its own. In the bottom-up approach, the receivers can independently start forming their own repair groups even before attaching to the tree.

The top-down approach guarantees caching of data

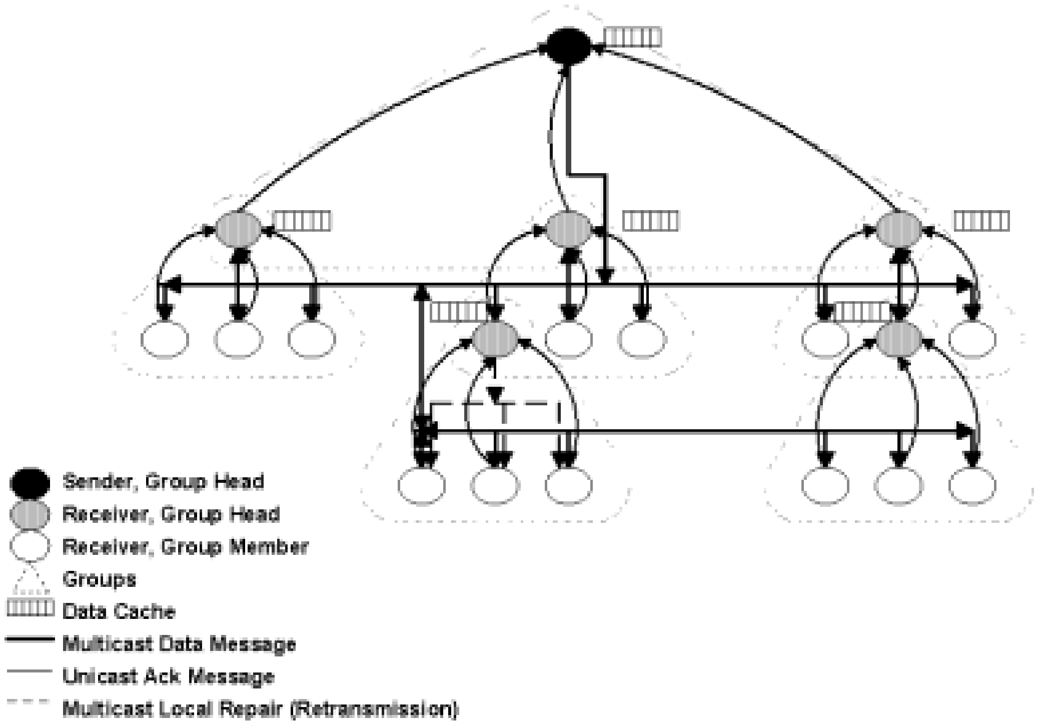

Figure 1: Typical Hierarchical Repair Tree in TRAM

somewhere in the tree hierarchy because the repair heads accept members only after attaching to the tree. The same level of guarantee cannot be achieved with the bottom-up approach, as a repair head that is still unattached may receive repair requests for data for which the repair head itself needs to seek repair. The data guarantee as stated here assumes the use of an acknowledgement-based mechanism to manage the data cache. Repair groups may be formed more quickly using the bottom-up approach due to parallelism. TRAM primarily uses the top-down approach, but incorporates the advantages of the bottom-up approach in its optimization for LANs (see the section on LAN-based tree formation below).

Tree Construction Techniques Supported In TRAM

To dynamically construct the repair tree, a protocol has to have mechanisms to enable receivers to attach and detach from the tree. Since TRAM uses the top-down approach, a receiver has to first attach to the tree before it can form its own repair group. In order to attach to the tree, a receiver has to discover a repair head that can accept it as its member. The discovery process can be done in one of two ways:

- the current repair heads in the system periodically multicast an advertisement message to solicit new members, or

- the receivers that are seeking a repair head multicast a periodic advertisement message.

From an efficiency and network resource usage standpoint, the second option is more attractive because the advertisements occur only when there are receivers that are seeking a repair head and will stop when all the receivers in the system are attached to the tree. The major disadvantage of the first option is that head advertisements have to persist for the entire multicast session as repair heads have no way of discovering that all receivers have attached to the tree.[1] The only disadvantage of the second option is that it requires the underlying network to support bi-directional multicast service. [2]

To be flexible and independent of network-related constraints, TRAM supports the following tree construction techniques:

- Repair tree construction when the network supports only unidirectional multicast.
- Repair tree construction when the network supports bi-directional multicast.
- Combination of unidirectional and bi-directional multicast tree construction techniques.
- Repair tree construction when multiple receivers are on a LAN (LAN-based Tree Formation). This method can be used locally on a LAN, with any of the above methods used to connect the LAN tree to the rest of the group.

Irrespective of the choice of repair tree construction technique, some of the generic policies governing the tree construction process in TRAM are:

1. The repair tree construction process is initiated by the sender by multicasting either a periodic control message (called the *beacon* message) or data to the multicast group using the session TTL scope.
2. The advertisement messages (the head seeking member advertisement and the receiver seeking head advertisement) are multicast using the Expanding Ring Search (ERS) mechanism [TMTP]. This is done to control the scope of these messages. TRAM allows the maximum value of this scope to be limited in order to control advertisement traffic.
3. Receivers affiliate with the closest repair head. Closeness is determined by the shortest TTL distance from the repair head to the receiver. This policy promotes localized repairs.[3]
4. Each repair head maintains a repair TTL that is large enough to reach all its members.
5. The repair heads in TRAM have to cache the received multicast data messages, perform repairs, and also process acknowledgements from the group members. Not every receiver in the system may be suitable to perform this role. A receiver may be unsuitable for reasons relating to hardware, software, resource limitations, or administrative restrictions. TRAM allows receivers to be configured with the following preferences:

   - EagerHead
   - ReluctantHead.

[1] See the JRMS Overview [JRMS] for methods of determining receiver status.

[2] It is possible to have a configuration in which there is no guarantee of multicast support from every receiver back to the sender, but there is enough support for any member to find a head using the second option.

[3] Other prioritizations of parameters, such as round-trip time, or tree level, result in the formation of different kinds of trees. For instance, a repair head may be chosen based on its proximity to the sender.

- MemberOnly

  An EagerHead will attempt to form its own repair group upon attaching itself to the tree. A ReluctantHead will also attempt to form its own repair group; however, the rules of repair head selection will prevent it from attracting members unless no eager repair heads are available. Note that the default value for receiver preference is ReluctantHead; this allows a tree to be formed without specifically configuring a receiver preference for each member. A MemberOnly receiver will never attempt to perform the role of a repair head.

6. The sender of the multicast session is an EagerHead and is always considered to be part of the tree.
7. To reduce the processing load on repair heads, TRAM supports mechanisms to restrict the number of members that an individual repair head can accept.

The details of the various tree construction techniques are discussed in the following sub-sections.

Repair Tree Construction Under Unidirectional Multicast

There are many existing networks that support multicast with some restrictions. These restrictions can be either technological (hardware or software limitations) or administrative in nature. A typical administrative restriction can be a network that is configured to forward incoming multicast messages and discard all outgoing multicast messages originating within the network. Constructing a dynamic repair tree with multicast restrictions can be tricky and complex. The following tree construction technique is adopted by TRAM when the underlying network has only unidirectional multicast.

All receivers tuned to the multicast group are inactive until the sender initiates the tree formation process. The sender triggers the process by multicasting either the beacon or data. Data as well as beacon messages contain the *HeadAdvertisementInterval* information. The HeadAdvertisementInterval specifies the rate at which *HeadAdvertisement* messages are to be sent out by the affiliated repair heads in the system. The sender alters the HeadAdvertisementInterval as the number of advertising repair heads in the system changes. This restricts the HeadAdvertisement traffic to a constant bandwidth. The number of advertising repair heads in the system is propagated up in the tree hierarchy and is made available to the sender. The sender computes the HeadAdvertisementInterval using the following formula

```
HeadAdvertisementInterval  =  max(
      1/2 second,
      AdvertisingHeadCount * HASize / MaxHABW,
      AdvertisingHeadCount / MaxPacketRate)
```

Where

HASize = the size of the HeadAdvertisement message in bits. This is typically 416 bits (including the UDP & IP headers).

MaxHABW = The maximum allowed HeadAdvertisement bandwidth in bits per second. The application controls this value which can be different before the data starts and after the data starts. Typically, only a small portion of the data bandwidth is reserved for HeadAdvertisement messages after data starts.

MaxPacketRate = Currently 30 packets/second.

Affiliated receivers whose receiver preference is either EagerHead or ReluctantHead send out periodic HeadAdvertisement messages. The sender, like any receiver that is a potential repair head and is already part of the tree, sends out periodic HeadAdvertisement messages. HeadAdvertisements are sent with increasing TTL scope at the interval specified by the sender (HeadAdvertisementInterval). Repair heads, including the sender, stop sending HeadAdvertisement messages when they can no longer accommodate new members.

Reception of a beacon or data will cause any unaffiliated receivers to wait for one HAListenInterval to receive a HeadAdvertisement message. The HAListenInterval is given by the following minimization factor:

```
HAListenInterval =
         Min((3*HeadAdvertisementInterval),60secs)
```

At the end of one HAListenInterval, if no HeadAdvertisement message is received, the receiver continues to listen for another HAListenInterval. The listening process continues until one or more HeadAdvertisement messages are received. When multiple HeadAdvertisement messages are received, the receiver will have to perform a selection process to choose the best-suited repair head. The repair head selection criteria is based on the following parameters and are listed in the order of preference:

- Smallest TTL distance from the repair head to the member
- EagerHead over ReluctantHead
- Highest reported group member count

Upon selecting a repair head, the receiver sends a unicast *HeadBind* message to the repair head. The HeadBind message is sent to the repair head's unicast port number that is specified in the HeadAdvertisement message.

A repair head, upon receiving a HeadBind message, verifies that the receiver can be accommodated as a member and responds with a unicast *AcceptMember* or with a unicast *RejectMember*. Typically, a RejectMember message is sent when the repair head can no longer accept members or when the repair head is attempting to relinquish its head duties. If a RejectMember message is sent, an appropriate reason code is included in the message. The unicast port to which the repair head sends the response is obtained from the member's HeadBind message. The member then tries the next best repair head, or goes back to listening for HeadAdvertisements if none are available.

After sending a HeadBind, the receiver waits for a predetermined interval of time to receive a response. Failing to receive a response causes the receiver to retransmit the HeadBind message. After a number of successive failed attempts, the member tries the next best repair head, or goes back to listening for HeadAdvertisements if none are available.

If an AcceptMember message is received, the receiver considers itself affiliated with the repair head and starts to send periodic HeadAdvertisement messages if its receiver preference requires.

In a typical scenario, with values for AdvertisingHeadCount of 500 and MaxHABW of 200kbps in the first equation, the HeadAdvertisementInterval yields a value of 16 seconds. Using this value in the second equation yields a value of 48 seconds for HAListenInterval. This is the minimum time that it will take a member to join the tree. The minimization factor of equation 2 limits the maximum time that a member waits to choose a repair head to 60seconds.

The main advantage of this technique is that it does not require the network to support bi-directional multicast, since the tree construction mechanism only relies on multicast service from the repair head to the receiver.

Some of the disadvantages of this technique are:

- The bandwidth consumed by HeadAdvertisement messages can be significant, especially when there are many receivers with receiver preference set to either EagerHead or ReluctantHead and the incremental increases of TTL scope ultimately cause the HeadAdvertisement messages to go beyond where they are needed.

- The HeadAdvertisement messages may have to continue for the entire multicast session if the repair heads have no way of knowing when all the receivers have joined the tree in order to stop sending the HeadAdvertisement messages.

- The excess HeadAdvertisement traffic can contribute to network congestion and reduce the scalability of the protocol. In the current version of TRAM, HeadAdvertisement traffic is not subject to congestion control.

Repair Tree Construction Under Bidirectional Multicast

Constructing a repair tree when the underlying network has no directional restrictions can be efficient and straightforward. Except for the difference in the HeadAdvertisement triggering mechanism, this tree construction technique is very similar to the preceding one. The receivers seeking repair heads use a *MemberSolicitation* message to trigger the repair heads to generate HeadAdvertisement messages. The following describes the tree construction process.

Receivers listen for beacon or data messages from the sender before trying to join the tree. Once a receiver hears from the sender, it multicasts a MemberSolicitation message to the multicast group using the ERS mechanism. After sending the MemberSolicitation message, the receiver waits to receive one or more HeadAdvertisements from the potential repair heads in the neighborhood. This wait period is referred as ResponseInterval.

Any repair head that is already part of the tree and can accommodate new members will respond to the MemberSolicitation message by multicasting a HeadAdvertisement message. Any repair head that cannot accommodate the member or a member with receiver preference configured to MemberOnly will ignore the MemberSolicitation message.

If the receiver's ResponseInterval expires and it has not received any HeadAdvertisement messages, the

MemberSolicitation message is resent to a larger TTL. This process continues until one or more HeadAdvertisement messages are received from the repair heads that received the MemberSolicitation message. If multiple HeadAdvertisement messages are received, the receiver will have to perform a selection process to choose the best-suited repair head. The repair head selection criteria are the same as in the previous technique. Upon selecting a repair head, the receiver sends a HeadBind message to the repair head.

The processing of a HeadBind message at the repair head and dispatching of an AcceptMember or RejectMember message is same as in the previous technique.

After sending the HeadBind message, the receiver expects to receive a response within a fixed interval of time. Failing to receive a response causes the receiver to retransmit the HeadBind message. After a number of successive failed attempts, the member tries the next best repair head, or sends another MemberSolicitation if no other repair heads are available.

If an AcceptMember message is received, the receiver considers itself affiliated with the repair head.

The main advantages of this approach are:

- MemberSolicitation avoids continuous advertisement by repair heads because it is used only when there are receivers attempting to join the tree. This tends to reduce overall bandwidth consumption by tree formation activities.

- The ERS mechanism used by the MemberSolicitation message provides isolation when receivers are seeking repair heads in different parts of the network.

The main disadvantage is that this technique does not work with unidirectional multicast. Also, the bandwidth consumed by the MemberSolicitation messages can be significant when many receivers from a region attempt to attain membership at the same time, especially in a LAN environment. The LAN optimizations described below helps to alleviate this problem.

Combined Tree Formation Technique

This technique is a hybrid of the earlier two techniques. Before the multicast data starts to flow, TRAM builds its repair tree using the unidirectional multicast tree construction technique. Once the multicast data starts to flow, TRAM switches to using the bi-directional multicast tree construction technique.

The reasoning behind this strategy is that in the absence of the data, the HeadAdvertisement messages can utilize the bandwidth expected to be used by data. Typically, the bandwidth expected to be used by the data is specified by the maximum data rate at the sender. The sender takes into account the number of advertising repair heads in the system when computing the HeadAdvertisementInterval. This allows the sender to control the advertising overhead, limiting it to no more than the MaxHABW.

However, once the data starts to flow, the sender may not want to have HeadAdvertisement traffic in addition to data and local repairs. At this point, it, changes the tree construction technique to bi-directional multicast by setting the HeadAdvertisementInterval to 0.

This technique has its limitations and can fail to construct a repair tree if data starts before all members are affiliated. Some members may not be able to affiliate with the tree if their multicast MemberSolicitation messages cannot reach any available repair heads.

LAN-based Tree Formation

When a large number of receivers are located on one LAN, the tree construction techniques described above are not optimal for the following reasons:

- Most of the LAN members will affiliate with repair heads that are not on the LAN, resulting in increased link traffic.

- All LAN members that are potential repair heads will start advertising for members, causing increased traffic on the LAN.

TRAM eliminates these problems by adopting the following optimization technique.

The LAN-based tree formation optimization adopts the bottom-up approach for building a subtree on the LAN before affiliating it with the rest of the tree. The result is a subtree in which only one repair head on the LAN affiliates off of the LAN. Also, only certain repair heads on the LAN will multicast HeadAdvertisement messages.

The LAN-based tree construction technique requires the receivers on a LAN to elect a LAN head. The elected LAN head, known as the *root LAN head*, is then responsible for attaching to the repair tree using whatever tree-formation method is in use and performing repairs of lost packets on the LAN. While the root LAN head attaches itself to the rest of the repair tree, the rest of the receivers on the LAN become members of the root LAN head. If the number of receivers on the LAN exceeds the root LAN head's membership limit, its members collaborate to elect additional LAN heads.

LAN-based tree formation uses the bidirectional method described above. One additional field is added to the HeadAdvertisement message: a LAN state field indicating the advertising node's LAN state. Nodes willing to be a LAN head use this field to volunteer. After one HeadAdvertisementInterval, the best repair head, based on receiver preference and member limit, is elected the root LAN head.

Once the root LAN head is elected, it solicits members as if it were already affiliated with the tree. Other LAN members will naturally choose the root LAN head since it is the closest one. Meanwhile, the root LAN head affiliates itself to an off-LAN repair head.

If the root LAN head reaches its member limit, additional LAN heads are elected from those already affiliated under the root LAN head.

The following two diagrams show the advantage of using LAN-based tree formation. These diagrams are generated by TreeTest, a simulator and animator of the tree-building process, further described in section 6. Each diagram shows a sender in the center; the rest of the nodes are repair heads or reluctant repair heads on a LAN. In the first example, without LAN-based tree formation, most of the receivers on the LAN affiliate with the sender. In the second diagram, with LAN-based tree formation, members on the LAN affiliate with the root LAN head, which affiliates with the sender.

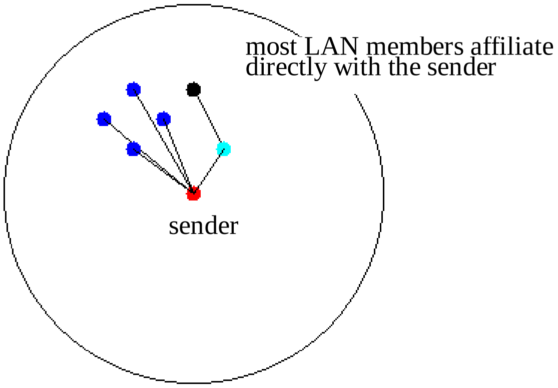

Figure 2: Tree formation without LAN optimization

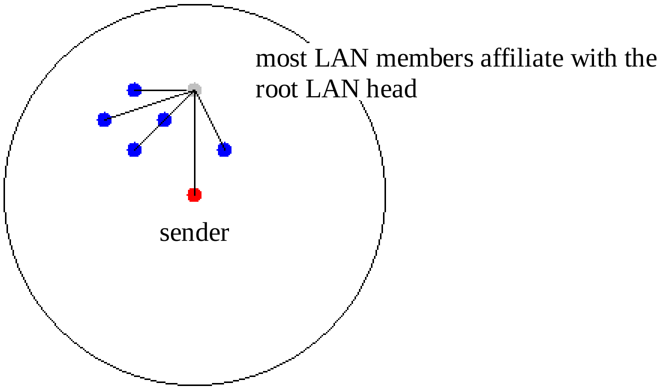

Figure 3: Tree formation with LAN optimization

Tree Management

The constructed repair tree has to be maintained to ensure reliable data delivery and proper protocol operation. TRAM supports a robust and rich tree management framework. Some of the functions supported by tree management are

(1) group member and repair head monitoring
(2) repair group TTL computation and management
(3) member reaffiliations and repair head resignations
(4) pruning of unsuitable members
(5) aggregation and propagation of protocol related information up the tree hierarchy to the sender

The different tree management messages used in TRAM are

- **Hello**: a multicast message sent to the group address with a TTL scope limited to the repair group scope. Repair heads that have one or more group members multicast this message.

- **Hello-Unicast**: a unicast hello message from a repair head to a specific group member.
- **ACK**: a unicast control message sent by a member to its repair head. This message includes a bitmap indicating received and lost packets, indications of inadequate or excess Hello TTL values, and congestion indications.
- **Beacon:** a multicast message sent by the sender to the session scope to trigger tree-building and as a keep-alive in the absence of data.

The details of how these messages are used in the tree management operation are described in the following sections.

Group member and repair head monitoring

In TRAM, the sender distributes the burden of reliable delivery among a subset of receivers that perform the role of repair heads. To ensure reliable delivery, it is required that every repair head cache the received multicast data until every member in the group acknowledges it. Since a repair head caches data until all the members acknowledge, it is very important that the repair head monitor its group members. The monitoring mechanism enables the repair heads to detect and react to conditions such as a member exiting the multicast session or becoming unresponsive. Absence of such a monitoring mechanism may lead to indefinite caching of messages at the repair heads.

From the member's perspective, it is important to ensure that lost messages are safely cached at the repair head, or further up in the tree. Repair can only be ensured when the members find the affiliated repair head to be reachable and responsive. If a repair head becomes unresponsive, the availability of lost messages cannot be guaranteed unless the member detects this condition and quickly reaffiliates with a different repair head. The monitoring mechanism enables members to detect the repair head loss condition.

The details of how the monitoring operation is carried out in TRAM are discussed below.

Monitoring of repair head by the group members

Each repair head periodically multicasts a Hello message (when necessary) to the group address with a session scope large enough to reach all of its members. The Hello period is determined by the following:

Hello Period = Max (1 Second, 1 AckInterval)

The AckInterval is computed at runtime and represents the time that a member has to wait to receive a window of packets at the rate of data transfer that is currently in use. The details of computing the AckInterval are discussed later.

The reception of the repair head's Hello message ensures that the repair head is operational. If the member does not receive a Hello message for more than one Hello period, the member sets a flag indicating that Hellos are not being received in its next ACK message to the repair head. If two such ACK messages go unanswered, the member gives up on the repair head and attempts to reaffiliate with a different repair head.

In response to a member's ACK message indicating that Hellos are not being received, the repair head sends out a Hello-Unicast message. The Hello-Unicast message serves to fill in for the lost Hello message.

Repairs performed by a repair head can also indicate to its members that it is operational. But, this cannot replace the Hello mechanism as the repairs are not guaranteed to be regular, and some repair heads may not perform any repairs for many Hello periods. However, TRAM can exploit repairs to optimize the repair head monitoring mechanism. The following are some of the optimizations adopted:

- Since the sender is always sending data or beacons, it does not need to send out periodic Hello messages. The sender sends out Hello messages only when it is essential, such as for TTL computation, or to demand acknowledgements.
- Repair heads do not send Hello messages when they have performed repairs within the latter half of the preceding Hello period.
- Members reset their Hello monitoring timers whenever a repair packet from the repair head is received.

Monitoring of group members by the repair head

Repair heads use the following mechanism to monitor members.

Each member of a repair group is expected to send at least one ACK message to its repair head for every acknowledgement interval. The details of computing the acknowledgement interval are discussed later. If a repair head does not receive an ACK message from a member for a period longer than 1 acknowledgement interval, the repair head adds the member to a list. The next time the repair head builds the Hello message, the entries in the list are included in the message. A special flag in the Hello message is set to demand ACK messages from the listed members. If more than three such Hello messages go unanswered by a member, the repair head disowns the member and reclaims all the cache buffers containing data unacknowledged by the member.

A member finding itself listed in a Hello message responds immediately by sending an ACK message to the repair head.

Repair Group TTL Computation

To conserve network bandwidth, it is important for repair heads to perform localized repairs, that is, with a TTL scope that is just large enough to reach all of their members. For the localized repair strategy to function properly, it is critical that the repair heads performing the repair compute the appropriate TTL to use to perform repairs. If the computed repair TTL is too small, the repair may not reach all of the repair requestors. On the other hand, if the TTL is larger than required, the repair will unnecessarily burden other receivers affiliated with other repair heads.

Initially, in the HeadBind message, each member recommends a TTL based on the received HeadAdvertisement message. The repair head uses the largest recommended TTL as its repair TTL. Should the path from the repair head to the member change, the computed repair TTL might become inappropriate. The following mechanisms help the repair heads track and adjust the repair TTL:

- When the repair TTL becomes insufficient, some group members fail to receive the Hello messages. These members report the condition to the repair head via ACK messages.

- The repair head responds to such ACK messages by first sending a Hello-Unicast message. The Hello-Unicast message assures the member that the repair head is functioning. The repair head then increases the repair TTL by a default step value. The next Hello message sent by the repair head goes to the new repair TTL scope. Note that the repair TTL is never expected to be greater than the session TTL.

The above two steps continue until the member stops reporting that Hellos are not being received.

Each member reports in the ACK message the excess TTL from the repair head's Hello messages. If the repair head determines that it is using a larger TTL than required to reach all of its members, it reduces the repair TTL.

Member Reaffiliations And Repair Head Resignations

Reaffiliation in TRAM is triggered when a member decides that it wants to affiliate with a different repair head. This may occur because its old repair head is resigning, or is not responding, or because the member has decided that another repair head would be better (see the section below on tree optimization).

The following describes the steps involved in the reaffiliation process.

- The member attempting to reaffiliate can either directly send a HeadBind message to the preferred repair head (learned from Hello and possibly HeadAdvertisement messages), or perform repair head selection based on received HeadAdvertisement messages, or seek a repair head by multicasting MemberSolicitation messages using the scheme described earlier. When the reaffiliating member is a repair head, the member has to avoid selecting a repair head that is at or below its level/depth in the tree hierarchy in order to avoid loops and disconnection from the rest of the tree.

- The member uses the normal TRAM affiliation mechanisms to affiliate with the new repair head (sending a HeadBind and receiving an AcceptMember or RejectMember message). If this affiliation fails, it may start over with another repair head.

- Once the member has affiliated to the new repair head, it maintains its affiliation with its old repair head until it has successfully received repairs for any packets for which repairs are not guaranteed by the new repair head. During this interval (the *TransitionInterval*), the member sends ACKs to both the old and the new repair heads, unless the old repair head is not responding. If the new repair head becomes unresponsive during this interval, the member may start over with another repair head.

- Once the TransitionInterval has been completed, the member sends an ACK with the Terminate Membership flag set to its old repair head. At this point, reaffiliation is complete and the member and the old repair head forget about each other. If the member is itself a repair head, it continues to function as a repair head during reaffiliation.

A repair head that is reaffiliating may not accept new members until its reaffiliation process is complete. Repair heads support all other normal repair head operations during reaffiliation.

A repair head typically resigns when the application exits or when it has determined itself to be redundant in the region. The reception of Hello messages from neighboring repair heads enables a member or repair head to determine that there are multiple repair heads in the region. Reducing the number of repair heads in a region leads to a reduction in management traffic.

Aggregation and Propagation of Session Information

The monitoring mechanism supported in TRAM establishes a management session between every repair head and its members. This session can be used as a means to aggregate and propagate various protocol-related events and information up the tree hierarchy. This is an efficient mechanism as it avoids problems related to implosion. The following are some of the events and information aggregated and propagated in the tree:

- Congestion events from the receivers in the system.

- Informatory details such as the number of receivers that are currently being served by the tree. The total number of receivers being serviced is computed as follows: every repair head in the system reports its direct member count and also the number of members that its members are serving. This information gets aggregated at each level and is reported to the next level in the tree hierarchy and eventually to the sender. Other information aggregated in this way includes the number of advertising repair heads in the tree (used to calculate the HeadAdvertisementInterval).

- Information related to the tree depth at which a member is operating. The repair heads utilize this information to avoid tree loops at the time of reaffiliation.

Tree Optimization Techniques

The optimality of the repair tree can drastically affect the scaling benefits of using a repair tree in the first place. A randomly built tree with many overlapping repair groups can cause repair heads to perform repairs with large TTL scope, which causes spillover of repair traffic to regions that do not require the repairs. The spillover is undesirable as it affects scalability, exhibits poor use of network resources and may also contribute to network congestion. The characteristics of an optimal multicast repair tree and the details of achieving the same are discussed below.

The characteristics of an optimal repair tree in TRAM may vary depending on the distribution of the receiver base. We consider two types of receiver populations. In *sparse* distributions, receivers are widely scattered with few concentrations (in terms of TTL) of members. In *dense* distributions, receivers are closely located to each other in terms of TTL.

Tree optimization for sparse distributions

When the receivers are sparsely distributed, the typical characteristics of an optimal tree are

- multiple repair groups that have few overlapping repair regions and
- every member in a group is affiliated to the closest repair head (in terms of the TTL distance from the repair head to the member).

The following examples illustrate poorly built and well built trees using the above criteria of optimality. In the first example, all members have affiliated with the sender, forming one large repair group. Any repair will reach all of the members. In the second example, localized groups are formed, limiting the scope of repairs.

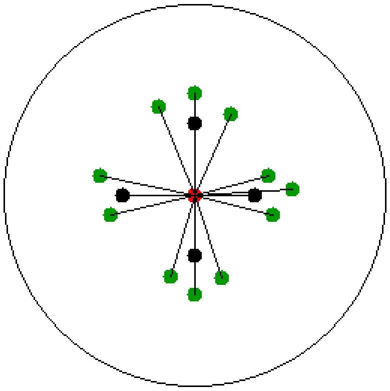

Figure 4: Poorly-built tree for sparse distribution

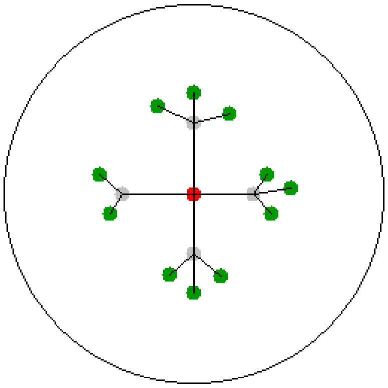

Figure 5: Well-built tree for sparse distribution

Due to the unpredictable times at which receivers join the group, it is likely that the initial tree constructed in TRAM is sub-optimal. To achieve tree optimality, every member and every repair head in the system should continuously pursue tree optimization. Tree optimization consists of determining when optimization is possible and acting upon this opportunity.

The following are some of the tree optimization techniques:

- When a member hears a Hello from a different repair head in the region that is closer than its current repair head, the member attempts to reaffiliate to the closer repair head. Note that a repair head has to perform loop avoidance checks before choosing to reaffiliate with the closer repair head. Without loop avoidance checks, improper tree formation (or tree disintegration) may result when a repair head chooses to affiliate with a repair head that is at or below its level/depth in the tree hierarchy.

- When two repair heads are found to be close to each other, one of the repair heads can volunteer to resign in favor of the other repair head. Typically the repair head that is better suited can continue to be a repair head while the other repair head can resign. In situations where both the repair heads are found to be suitable, tie breaker techniques such as the repair head that has fewer members, or the repair head that has the lowest unicast address and port number combination, can be used to resolve the condition.

- To ensure quick and smooth reaffiliations of its members, a resigning repair head can include the details of any backup repair head (network address, unicast port number, worst case TTL distance from the backup repair head to the members) in the Hello message. The backup repair head details are informatory in nature and members with better alternatives can choose to ignore this information and reaffiliate with a different repair head. The details of the backup repair head help the members that do not hear Hellos from any other repair head other than the currently affiliated repair head.

- The repair head can adopt a strategy wherein the members that are considered to be farthest are repaired using unicast and those that are considered closer are repaired using multicast. When this strategy is in use, the repair heads can be limited to accept only a few unicast members.

Tree optimization for dense distributions

When all receivers are densely located in one region (say an intranet with few TTL hops between the sender and all the receivers), it may be difficult to form repair groups that have a low degree of overlap. In this situation, if the number of repair heads in the system is not controlled, much of the bandwidth can be consumed by the management traffic. Hence in the dense case, the characteristic of an optimal tree is to have as few repair heads in the system as possible.

The tree optimization technique in the dense case focuses on reducing the number of repair heads rather than finding a closer repair head. A member tries to affiliate to a repair head that is already performing the role of repair head. A new repair head is accepted only when all the current repair heads are no longer accepting members. Monitoring Hellos and Head-Advertisements is not necessary.

The dense mode optimization algorithm is simple, straightforward and quick. The sparse mode optimization algorithm is complex and has to be pursued throughout the multicast session to form localized repair groups that have fewer overlapping regions. It is to be noted that the sparse mode optimization is unsuitable for multicast sessions that span a short duration, since some time is required to optimize the tree.

By default, TRAM supports sparse mode optimization.

5. Flow Control

TRAM is designed to transfer bulk data from one sender to many receivers. The data is transmitted at a rate that adjusts automatically between a specified minimum and maximum. Sequence numbers in the data packets allow receivers to identify missing packets. Each member is bound to a repair head that retransmits lost packets when requested. Acknowledgements sent by receivers contain a bitmap indicating received and missing packets. Missing packets are repaired by the repair head; packets acknowledged by all members are removed from the repair head's cache. The following sections describe in detail the mechanisms for transmitting data, requesting retransmission of lost packets, detecting congestion, adjusting the transmit rate, late join requests, and handling end of transmission.

Data Transmission

The sender in a TRAM application transmits data packets to every receivers in the multicast group. TRAM sends the packets at a specified rate. Each packet is given a unique sequence number starting with 1. Receivers use these numbers to detect out of order and missing packets.

Acknowledgments

Members acknowledge one window of packets at a time to their repair head. This window is known as the ACK window. The ACK window size is configurable; the default is 32. For example, in the default case, members send ACKs every 32 packets.

To avoid multiple members sending ACK messages at the same time, and to possibly expedite repairs, ACK messages are distributed over the window. Each member selects a random packet between 1 and the ACK window size to start sending ACK messages. For example, one member may send ACKs at packets 32, 64, 96, etc., while another sends ACKs at packets 10, 42, 74, etc.

Acknowledgments are also sent if a timer equal to 1.5 times the estimated ACK interval expires. The estimated ACK interval is computed at each receiver when an ACK is sent. It estimates the amount of time it takes to receive an ACK window's worth of packets. The formula is:

ACK interval = ACK window *
(Time since last ACK / Packets since last ACK)

This timer is canceled if an ACK is sent using the triggering mechanism described above. If this timer expires, it indicates that the sender has paused and allows members to report and recover any lost packets without having to wait for the sender to start sending new data.

Each ACK message contains a start sequence number and a bit map length. If no packets were missing, the bit map length is 0 and the sequence number indicates that all packets prior to and including this packet were successfully received. The repair head saves this information and uses it to remove packets from its cache.

If there are one or more missing packets, the start sequence number indicates the first missing packet. A bit map must follow. Each bit in the map represents a packet sequence number starting with the start sequence number. If the bit is set, that packet is missing and must be retransmitted. A bit map length indicates how many valid bits are present.

When the repair head receives an ACK message with a missing packets bit map, the sequence number specified minus 1 is saved for this member. This indicates that all packets prior to and including this sequence number have been received successfully. The repair head then scans the bit map looking for missing packets. It immediately places these packets onto the transmit queue unless they have recently been retransmitted or are already on the queue from another request.

Data Retransmissions

When a repair head receives a request to retransmit a packet, it retransmits it as soon as possible. Retransmissions take priority over new data packets. Retransmitted packets are sent at the current rate used for new data packets from the sender. Each repair head computes the average data rate of all packets it receives and sends retransmissions at this rate.

Duplicate Retransmission Avoidance

When several members request retransmission of the same packet, TRAM sends the packet immediately for the first request. Subsequent requests are ignored if they are received within 1 second of the first request.

Occasionally, packets are waiting for retransmission due to rate limitations. If a new request for a packet is received and that packet is awaiting retransmission, the request is ignored.

Every repair head in TRAM keeps track of the lowest packet sequence number that all members have received. Before a repair head retransmits a packet that has been waiting to be retransmitted, it again checks the sequence number of the packet to be retransmitted against this lowest packet number. If the retransmission packet sequence number is lower, the repair head skips this retransmission because all of its members have already acknowledged the receipt of the packet. This can happen when multiple repair heads retransmit the same packets and their transmission range overlaps.

Rate-based Flow Control

Even with unicast transports, flow and congestion control is a delicate problem. The study of flow and congestion problems for multicast transports is at its infancy. Several talks [MCA1], [MCA2], [MCA3] were given at a recent workshop, which were somewhat different than the schemes in TRAM.

The flow control in TRAM is rate-based and is similar to NETBLT [NETBLT] , a rate-based unicast protocol.

TRAM's packet scheduler computes the amount of time to delay each packet in order to achieve the desired data rate. The delay is computed with the formula:

packet size / desired rate

The overhead in processing the packet is subtracted from this delay. TRAM then sleeps for the calculated period, sends the packet, and the cycle continues. This is similar to the widely known token bucket algorithm.

TRAM's flow control uses various algorithms such as slow start and congestion control to dynamically adapt to network conditions. A maximum and a minimum rate can be specified to limit the operation of these algorithms. The minimum rate effectively defines the receiver population. Any receiver that cannot keep up with the minimum rate will be *pruned* (no longer guaranteed repairs).

If packets are lost or ACKs are not reaching a repair head, its cache will start to fill up. If the sender's cache fills up above a threshold value, it stops sending new data until it can free some buffers from its cache. This bounds how much the sender and receivers are allowed to get out of synch.

TRAM sessions go through two phases of flow control: *slow start* and *congestion control*. The slow start phase is the initial phase during which TRAM carefully tests the network to find an appropriate operating point. This is analogous to TCP's slow start. After the slow start phase, TRAM will have established some boundaries for its operation and enters the *congestion control* phase.

Slow Start

During the slow start phase, the initial data rate starts at 10% of the maximum, or the minimum rate if that is greater. Every two ACK windows this rate is increased another 10% of the maximum data rate. This process continues until the maximum rate is reached or congestion causes the rate to decrease.

Congestion Reporting

Congestion is detected at the receivers and repair heads:

- Receivers detect and report congestion based on missing packets.
- Repair heads detect and report congestion based on their cache usage.

At receivers
Receivers detect and report congestion based on an increase in the number of missing packets between two ACK windows. For example, if a receiver detects 5 missing packets during an ACK window, and has 10 packets missing in the next window, a congestion message is sent to its repair head. The congestion message contains the highest sequence number received. When the repair head receives the congestion message, it determines whether this is a new congestion report and if so, forwards it immediately up to its repair head. Each repair head will forward one congestion packet from its members for each ACK window. The repair head computes the ACK window from the sequence number specified in the congestion message with the formula:

sequence number / ACK window size

Once a congestion message for an ACK window has been forwarded up the tree, congestion reports for its ACK window and previous ACK windows will be ignored. The sender does not react to multiple congestion reports for the same window.

At repair heads

Repair heads also generate congestion messages when their data caches begin to fill up. Each repair head maintains a low and high water mark on its cache. When the number of packets in the cache reaches the high water mark, an attempt is made to purge the cache. If it can't purge the cache below the high water mark because a member has not acknowledged these packets, a congestion message is forwarded up the repair tree. In this situation the repair head temporarily increases its high water mark to the current value plus the number of packets in an ACK window. The repair head performs the same test when this new temporary high water mark is reached. If the cache is exhausted, new packets are dropped without acknowledging them.

The sequence number in a repair-cache-generated congestion message is the highest sequence number the repair head has received.

At the sender

The sender also maintains a cache for its immediate group members. If its cache fills up to the high water mark and can't be reduced, it reacts as if it received a congestion message for that window. It also temporarily increases its high water mark to the current value plus the size of an ACK window. If the cache fills to this new level and can't be reduced, it reacts again.

This process continues until the cache is full. At this point the sender blocks any new data from the application and attempts to solicit an ACK from the members that are causing the cache to fill up. If these members do not respond quickly, they may be pruned.

When some cache buffers are freed, the application can start sending data again.

In experiments, we observe that the repair head's cache-generated congestion messages almost always follow the receivers' loss-generated congestion messages. This is logical as the build-up in the repair head's cache is invariably caused by the lost packets at the receivers.

Congestion Control

The sender reacts to congestion feedback as follows:

- react to selected congestion reports
- decrease the rate in the face of congestion
- increase the rate in the absence of congestion

The only significant experience with real network congestion control is based on TCP traffic and the algorithms implemented in the TCP protocol. It is not surprising that the natural approach is to try to adopt TCP' s algorithms for reacting to network congestion [TCPREQ], [TCPFC]. One of the key ingredients of TCP's algorithm is to follow the additive increase/multiplicative decrease rule [CA2], [CA1].

The increase amount

Since TRAM transmission is rate-based, an immediate problem is determining the right rate increase in the absence of congestion. TCP is a window-based protocol in which increases are done by incrementing the congestion window by one, a dimensionless parameter. In the spirit of TCP, the correct amount to increase TRAM' s rate would be a small fraction of the bottleneck bandwidth. Plus, this amount would need to be adopted by all the flows sharing the same bottleneck. This increase is not easily determined.

A constant increase amount will always be wrong for some topologies. Although we used 10% of the maximum rate for slow start, it does not seem suitable since the maximum rate may be far from the current bottleneck rate. Instead, TRAM derives the increase amount dynamically as follows. TRAM keeps track of the *historically highest achieved rate* (HHR). After each rate decrease, a new increase amount is calculated as

Increase = (HHR – current_rate) / 4

The rationale is to use HHR as a local estimate of the bottleneck bandwidth. Within a small number of steps (in this case 4), TRAM tries to return to that level if there is no congestion. The constant 4 will require further experimentation; a larger number might be adopted for potentially more equitable bandwidth-sharing by many flows in exchange for slower convergence to the optimal rate.

An immediate reaction to this formula would be to replace HHR with something that adapts to the current network condition, for instance, using the *most recently achieved high* (MRH), or a moving average of MRH, or the session average data rate. Each of these cases resulted in diminishing increases for some common topologies, eventually leading to a rate close to zero.

Decrease

The receipt of a congestion report causes the data rate to drop by a percentage (50%, we are also experimenting with 25%). This is the same as TCP. Adjusting the rate by a percentage is very appealing since the adjusted amount is dimensionless, hence there are no calibration problems.

Synchronizing feedback and control

The next important aspect of a congestion control scheme is how to keep the feedback and control in synchrony with each other. Ideally, each control action is based on the feedback from the network that reflects the consequence of the previous control. From systems theory, we know that there is a chance of building a control that leads to optimal and stable behavior.

If our feedback does not capture the consequence of the previous control, then we are likely to end up with a control that reacts to the same conditions repeatedly and leaves the system in wide oscillation. This oscillation is indeed what we observed in our simulation experiments if we increase the rate for *each* window when there is no congestion, and decrease for *each* window when there is congestion.

In order to make sure that each congestion feedback includes the previous control actions, the congestion control algorithm must wait for *several* windows before acting on the feedback. This tends to make the system less responsive to topologies with a small number of receivers.

As a compromise, TRAM increases the rate *every other* ACK window in the absence of congestion reports. When a congestion report is received, TRAM decreases the rate right away and records the window in which the congestion happened. Another increase or decrease in rate is not allowed until after N windows where N is

$$N = 4 * (rate / HHR)$$

Where 4 is simply 2 times the normal schedule of reacting to every 2 windows.

The reasons for using a variable schedule are:

- At lower rates, the risk of congestion is less hence it is fairly safe to increase the rate sooner.
- At lower rates, the time penalty for waiting an additional window is longer.

Accounting For Retransmission

To further improve the congestion control algorithm, TRAM includes mechanisms to take into account the effect of retransmission when determining the rate during periods of congestion.

In the congestion message, the receivers report an estimate of how long it will take to do local repairs. This information is aggregated back to the sender. In reaction to a congestion message, the sender not only reduces its rate, but also pauses briefly to let the local repairs complete.

This mechanism has further smoothed out the oscillations in our simulation experiments.

Congestion Control Simulation and Results

To support the design of TRAM, we built a simulation model using the NS simulation tool [19]. The TRAM protocol is modeled using the simulation tool, NS [NS]. The object-oriented simulation environment, the scripting language support, plus the companion animation tool, NAM, enable effective experiments.

We tried many different network topologies. The results from two of the variants are shown here. The basic network consists of 201 nodes. The sender agent runs on one node, repairer agents run on 24 of the nodes, and pure receiver agents run on 168 of the nodes. The other 8 nodes are router-only nodes. The whole network is symmetric. Each repairer is 2 hops away from the sender, and each receiver is 1 hop away from its parent (repair head). The links from the sender to the first tier routers are 1.5Mb/s links with 50msec delay. The rest of the links are 1.5Mb/s links with 10msec delay, except 3 of these links are 0.5Mb/s links with 10msec delay. These 3 slow links are the bottleneck of the multicast session. The following figure shows this topology, rendered by NAM.

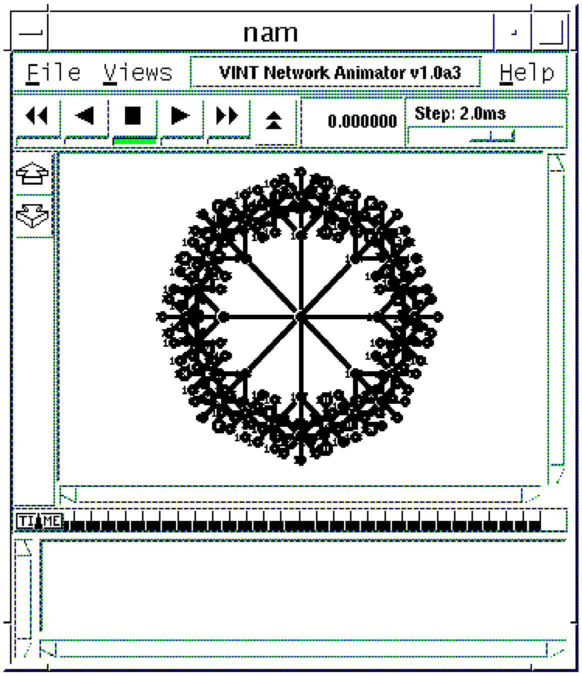

Figure 6: Network topology

In the variation of this topology, the 3 slow links are further programmed to deterministically go up and down. Every 3 seconds, they go down for 0.05 seconds. On average the down time is less than 2%.

The experiment is to send 1000 packets, 1400 bytes each. The sending starts at 1.5 seconds from the beginning of the simulation. Since the limiting bandwidth is 0.5Mb/s, assuming ideal scheduling the whole transmission should take 22.4 and 23 seconds respectively.

The following two figures should how TRAM did for each of the two cases. The X axis is time in units of seconds. The Y axis scale is used for a number of things:

- Rate: in 10Kb/s (i.e. 50 = 0.5Mb/s)
- Loss: number of packets lost at ACK time
- Cache occupancy: number of packets

Figure 7 shows results for a network without up/down dynamics; figure 8 shows results for a network with up/down dynamics.

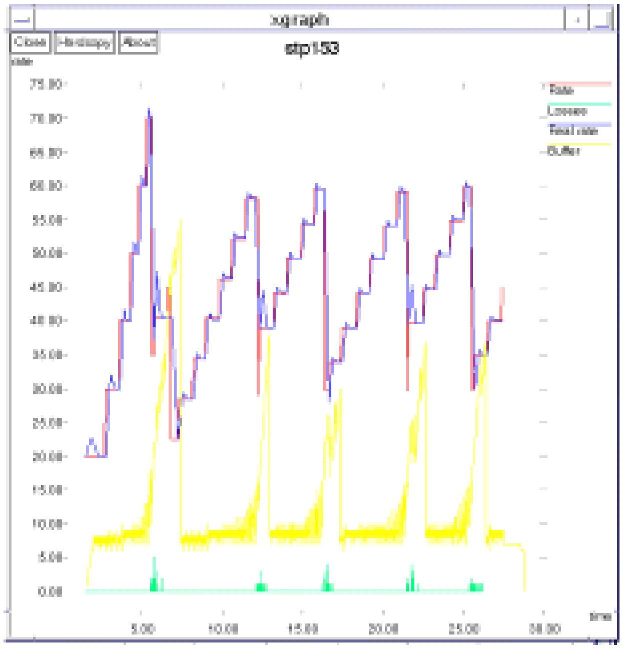

Figure 7: Static 201 node network

The top curves represent the monitored rate and the send rate. The next curve is the cache occupancy observed at a repair head that is responsible for a receiver that is behind the slow link. Losses are shown along the bottom. As can be seen, the buffer occupancy shoots up soon after the losses start to occur.

For the static network in Figure 7, the maximum possible transmission rate, as limited by the bottleneck bandwidth, is 50 (x10Kb/s). TRAM manages to keep the rate oscillating between 30 and 60. The initial spike is bigger, the result of slow start when there is no hint what the possible maximum rate is. The subsequent improved performance is what we expected.

The fact that the cache occupancy oscillates at around 7 in the steady state is normal. In these tests, the ACK window is set to 8. With ACK staggering, the cache usage is expected to stay around 8, instead of going from 0 to 8 and back to 0 in a saw-tooth form.

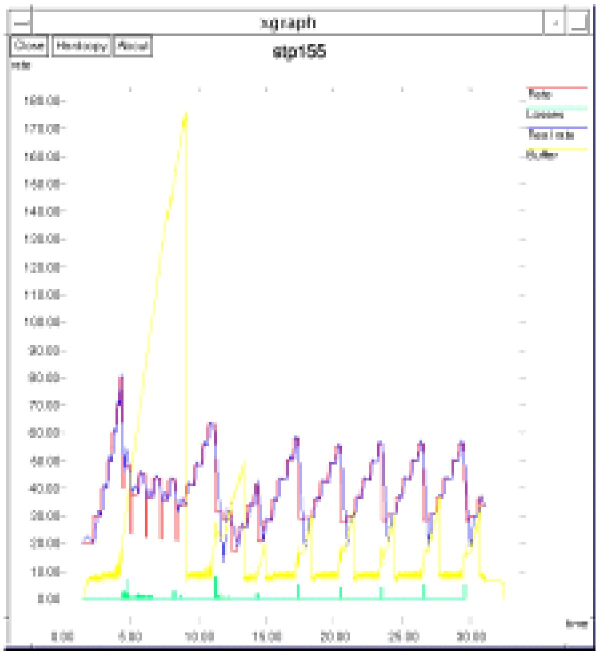

Figure 8: Adding up/down dynamics to some links

In figure 8, the link up/down dynamics clearly induce very periodic losses at the times the links turn off (every 3 seconds). TRAM adapts quite well, except right after the first link down, when the losses induced a high cache occupancy. This is because a retransmission failed to get to the receiver behind the faulty link for quite a long time. After it overcame the initial difficulty, the rate oscillated between 30 and 60 as in the static network case. The time of completing the test is only marginally longer than the static network case.

Late Joins

Receivers joining the multicast group after data transmission has started have two options for recovering data previously sent:

- Recover as much data previously sent as possible. This option allows the receiver to request retransmission of all the previously sent data that its repair head has cached. A repair head typically has at least the last 50 packets sent in its cache.

- Don't recover anything sent before the receiver joined. This option doesn't attempt to recover any previously sent packets. The first data packet received after the new member joins the repair tree is handed up to the application. All previously sent packets are ignored.

Both of these options require that the receiver join the multicast repair tree before any data is given to the application. This insures that all subsequent data can be received reliably.

End Of Transmission

Receivers must be able to determine when the session has completed to ensure they have received all of the data before exiting.

When the sender application completes, end of transmission is signaled throughout the multicast group. The sender notifies all members of session completion with a beacon packet that has the TXDONE flag set. This packet also includes the sequence number of the last data packet sent. The sender transmits this packet periodically until all of its immediate members acknowledge the receipt of all packets sent. The sender can then exit.

When a member receives the beacon packet with the TXDONE flag set, it immediately sends an ACK message to its repair head indicating whether it has received all the packets transmitted or requires more retransmissions. TRAM notifies the application when it receives all of the packets.

When a repair head receives the beacon packet with the TXDONE flag set, it communicates with its repair head just as a receiver does. The repair head must wait for all of its members to acknowledge all packets before it can close the session. If a member requires retransmission, the repair head must retransmit all the packets required of its members prior to closing itself. If the beacon from the sender with the TXDONE flag set is received but one or more members have not acknowledged all packets, a Hello message is sent to these members with the same information contained in the beacon packet. Members receiving this Hello message must respond in the same way that they would if they received the beacon. If the repair head still doesn't hear from its members after sending the Hello, it retries several times. After a period of time, it gives up on the member and removes it from the member list.

When all members have either acknowledged all packets to the repair head or have been removed from the member list because they were not responsive, the repair head can close its session.

Pruning

In a multicast data distribution setup, the source of a multicast stream can operate in a mode that is sensitive or insensitive to the data reception feedback from the receivers. The drawbacks of being insensitive are a lack of response to network congestion and an inability to deliver the data to as many receivers as possible. Being sensitive makes the multicast distribution mechanism overcome the above drawbacks but also introduces a new drawback that may make the sender operate at a rate that is slower than what is desired by the application. To overcome this drawback, it is necessary that the multicast delivery system support some sort of a pruning mechanism which enables receivers that do not meet the reception criteria to be isolated and removed from the repair mechanism.

In TRAM, the reception characteristics of all the receivers is distributed knowledge. The sender knows of the reception characteristics of its immediate members while some other repair heads in the system know the reception characteristics of some other set of receivers. Due to the distributed nature of reception characteristics, TRAM adopts a collaborative pruning technique that involves the sender and all the repair heads in the system.

The technique requires the sender to orchestrate the pruning operation by providing a MinimumDataRate signal. The signal is included in the header of multicast data and beacons sent by the sender. The signal is set to *off* when no congestion is being reported from the sender.

As a result of receiving congestion feedback information from one or more receivers, the sender attempts to reduce the rate of transmission to accommodate the slow receivers.

The sender sets the MinimumDataRate signal *on* when the sender is operating at the minimum rate specified by the application. The MinimumDataRate signal informs repair heads in the distribution tree to prune any poorly performing receivers. The repair heads may respond to receiving the Minimum-DataRate signal by pruning members.

Pruned members can be members that are slow, members that are requesting excessive repairs, or members that have become unresponsive as a result of a network partition or for some other reason. The members that are pruned are notified of membership termination via the Hello-Unicast message. The repair head may stop honoring repair requests from members that are pruned.

Note that repair heads can independently perform the pruning operation (i.e., without a sender signal). This may result in premature pruning of the members, as the repair heads may not know whether or not the sender is operating at the configured minimum rate.

6. Implementation And Sample Applications

TRAM is currently implemented in Java and requires JDK 1.1 or higher.

Several sample applications have been developed to test TRAM's capabilities:

- **Slinger** – a one-to-many file transfer protocol useful for testing bulk data transmission speeds and tree-building
- **Bricks** – a one-to-many transfer protocol whose GUI graphically indicates packet reception and repair ordering
- **Stock** – a one-to-many stock ticker program useful for testing live data
- **TreeTest** – a GUI built on top of the TRAM classes for testing tree-building. TreeTest is the source for the tree-hierarchy diagrams in this document.

7. Limitations

Realistic TTL Values

Tree formation in TRAM relies on TTLs. Current multicast routing protocols may not provide effective TTL values for this purpose.

Heterogeneous Receiver Population

Although TRAM adjusts itself to the capabilities of the receivers as much as possible, a particular slow receiver may end up without repair service if it cannot keep up with the minimum speed specified by the sender.

8. Future Work

Support For Live And Resilient Data

With a few adjustments, TRAM could support live data transmissions. The primary difference between bulk data and live data is the requirement to view live data in real time. Bulk data can afford to delay repair of a single packet; live data cannot. For TRAM to support live data, mechanisms must be in place to repair missing packets more quickly, not just on window boundaries.

Full Data Recovery For Late Joins

Currently TRAM has no guaranteed support for late joiners requiring full data recovery. If a member comes late to a session, it may not find a repair head that has all of the existing session data in its cache. A possible solution is to acquire the missing data from the sender, another repair head, or a specially designated complete repair cache.

Emerging Multicast Routing Support

Some promising work has been proposed to improve reliable multicast by adding more capability at the routing layer. Subcasting (instructing routers to direct multicast only downward through the tree), randomcast [RANDOM] (randomly picking a single link for a subcast), AIM [AIM], and LMS [LMS] improve local recovery by suppressing duplicate replies, improving repair time, and reducing the scope of multicast repairs. Although TRAM works well now, these added capabilities would be welcome. As these features become available, TRAM may use them.

More LAN Tree Optimizations

This paper details methods for forming efficient trees on LANs by ensuring only one repair head on the LAN affiliates off of it. LAN trees could be further improved by using multicast negative acknowledgements instead of unicast ACKs. This would enable members experiencing the same loss to suppress negative acknowledgements, as in [SRM].

Congestion Control

Once TRAM is more fully deployed and we have more results concerning the performance of TRAM's congestion control, we expect to make more improvements. In particular, interaction with other network traffic, for instance, TCP, needs further study.

Repair Tree Management

Some situations call for more controlled tree formation. Some possibilities include

- Static assignments of members to repair heads
- Static assignment of standby repair heads
- Recording a tree formation for later use

Multiple Senders

Future versions of TRAM may support multiple senders.

Security

Future versions of TRAM will address receiver access control, source authentication, data integrity and confidentiality, and certain denial of service attacks, such as an unauthorized sender flooding the multicast group with data.

9. Conclusion

Multicasting holds great potential to drastically improve the efficiency of data transmission. Making multicast reliable, however, proves to be a hard problem because of the wide spectrum of requirements from applications.

In this paper, we have described a reliable multicast transport protocol that is based on a repair tree formed by the receivers. This repair tree is used not only for the purpose of distributing the load of repair, but also for the purpose of efficiently funneling feedback to the sender. Such a protocol is suitable for a wide range of applications that need to scale to large receiver populations, require almost simultaneous data delivery, and require a high degree of reliability by the receivers that can meet a minimum rate of data transmission.

While some parts of the design of this protocol are similar to other reliable multicast transport protocols, our contributions described in this paper focus on the areas of automatic repair tree formation and rate-based flow and congestion control. Our congestion control incorporates several new ideas verified by simulation results.

Finally, this protocol is not only simulated, but also implemented. It is part of a framework for integrating different multicasting facilities and applications as described in [JRMS].

10. Acknowledgements

The authors gratefully acknowledge the many contributions of Steve Hanna, Phil Rosenzweig, and Radia Perlman.

11. References

[AIM] B. Levine, J. Garcia-Luna-Aceves, *Improving Internet Multicast with Routing Labels*, University of California, Santa Cruz, 1997

[CA1] Jacobson et al, *Congestion Avoidance and Control*, Proceedings of ACM SIGCOMM 1988 pp. 314-329, August, 1988

[CA2] Chiu and Jain, *Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks*, Computer Networks and ISDN Systems Vol 17 pp 1-14, 1989

[JRMS] S. Hanna, M. Kadansky, P. Rosenzweig, *Java Reliable Multicast Service Overview*, Sun Microsystems Laboratories, SMLI TR-98-68, September 1998

[LMS] C. Papadopoulos, G. Parulkar, G. Varghese, *An Error Control Scheme for Large-Scale Multicast Applications*, Washington University St. Louis, 1997

[LOCAL] C. Liu, D. Estrin, S. Shenker, L. Zhang, *Local Error Recovery in SRM: Comparison of Two Approaches,* January 1997

[MCA1] Sano, Shiroshita, *Congestion Control for Bulk RM*, RM Workshop in Cannes, September, 1997

[MCA2] Vicisano et al, TCP-Link *Congestion Control for Layered Multicast Data Transfer,* RM Workshop in Cannes, September, 1997

[MCA3] Todd Montgomery, *A Loss Tolerant Rate Controller for Reliable Multicast,* RM Workshop in Cannes, September, 1997

[NETBLT] Clark, Lambert, Zhang, *NETBLT: A High Throughput Transport Protocol*, Proceedings of ACM SIGCOMM 1987, pp 353-359, August, 1987

[NS] Network Simulator – ns (version 2), http://www-mash.cs.berkeley.edu/ns

[RANDOM] A. Costello*, Search Party: An Approach to Reliable Multicast with Local Recovery,* University of California, Berkeley, 1997

[RMTP] S.Paul, J.C.Lin. *RMTP: A Reliable Multicast Transport Protocol,* Proceedings of IEEE INFOCOM'96, pp 1414-1424

[RMTP-II] B. Whetten, M. Basavaiah, S. Paul, T. Montgomery, N. Rastogi, J. Conlan, T. Yeh *The RMTP-II Protocol,* draft-whetten-rmtp-ii-00.txt, Internet Draft, IETF, April, 1998

[SACK] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgement Options*, RFC 2018, October 1996

[SCALE] P. Sharma, D. Estrin, S. Floyd, L. Zhang, *Scalable Session Messages in SRM,* February 1998

[SRM] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang, *A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing*. To appear in IEEE/ACM Transactions on Networking, Nov 1996

[SURVEY] B. Levine, J. Garcia-Luna-Aceves, *A Comparison of Known Classes of Reliable Multicast Protocols.* University of California, Santa Cruz. 1996

[TCPFC] Mahdavi, Floyd, *TCP-friendly Unicast Rate-Based Flow Control*, Technical note, End-to-End Internet Research Task Force mailing list, January 8, 1997

[TCPREQ] Floyd et al, *Requirements for Congestion Control for Reliable Multicast*, RM Workshop in Cannes, September, 1997

[TMTP] R. Yavatkar, J. Griffioen, M. Sudan, *A Reliable Dissemination Protocol for Interactive Collaborative Applications*, University of Kentucky, 1995

About the Authors

Dah Ming Chiu is a Senior Staff Engineer at Sun Microsystems Laboratories, Chelmsford. He has a Ph.D. degree from Harvard University, and has worked for AT&T Bell Labs and DEC before joining Sun. His current research interests include modeling and simulation of flow control algorithms and other resource allocation algorithms in networking protocols, secure multicast, and Internet quality of service and economy.

Steve Hurst is a Staff Engineer at Sun Microsystems Laboratories, Chelmsford. He is currently working on the design and development of a reliable multicast protocol in Java. He is also working on the design and development of multicast security. Prior to joining Sun, Steve worked in the UNIX Networking group and Network Products group at Digital Equipment Corporation. He has a B.S. in Computer Engineering from the University of Connecticut.

Miriam Kadansky is a Senior Staff Engineer at Sun Microsystems Laboratories, Chelmsford. She has a B.A. in Mathematics from Harvard College, and worked for several networking vendors before joining Sun. Her current interests include secure reliable multicasting and Java.

Joseph S. Wesley is a Member of Technical Staff at Sun Microsystems Laboratories, Chelmsford. His current interests include reliable multicast and multicast security. Prior to joining Sun, he worked in the router development group at BBN Systems and Technologies, Inc. in Cambridge, Massachusetts. He has an M.S. in Computer Engineering from the University of Massachusetts, Lowell, and a B.E. in Electronics and Communication from the University of Mysore, India.