

An Overview of Reliable Multicast Transport Protocol II

Brian Whetten, Talarian Corporation
Gursel Taskale, Reuters

Abstract

This document provides an overview of the Reliable Multicast Transport Protocol II, RMTP-II. RMTP-II is a reliable multicast protocol, designed to reliably and efficiently send data from a few senders to large groups of simultaneous recipients. It works over both symmetric networks and asymmetrical network topologies such as those provided by satellite, cable modem, or ADSL carriers. Before sending, each sender must connect with a trusted top node to receive permission and control parameters for its data stream. The top node provides network managers with a single point of control for the senders, allowing them to monitor and control the traffic being sent. RMTP-II builds on a rich field of existing work, and adds to it the following novel contributions. It differentiates the roles of the nodes in the protocol, provides algorithms for smoothing and control of the return (TRACK) traffic, and provides explicit support for highly asymmetrical networks. It provides explicit network management controls through a centralized point of control, a fully distributed membership protocol that enables positive confirmation of data delivery, and fault recovery algorithms which are integrated to the reliability semantics of the protocol. It includes a novel reliability level called time bounded reliability, and offers a unique combination of TRACKs, NACKs, and FEC for increased scalability and real-time performance. Finally, it integrates distributed algorithms for RTT calculation to each receiver, and provides automatic configuration of receiver nodes.

P multicast is an Internet Engineering Task Force (IETF) standard that allows a single packet to be sent, unreliably, to up to many thousands of simultaneous receivers. It is both selective (it is delivered only to receivers who have subscribed to the multicast address) and efficient (a packet is never transmitted on a link more than once). However, most applications require some form of reliability to recover from link-level packet losses and from the packet losses that are an inherent by-product of Internet congestion control [1].

Reliable Multicast Transport Protocol II (RMTP-II) [2] is a transport protocol for IP multicast, which draws heavily on the original RMTP protocol [3, 4]. The RMTP protocol provided groundbreaking work in the use of a tree topology for acknowledgment (ACK) aggregation and local recovery, but is less suited to real-time delivery because it does not support negative ACKs (NACKs) or forward error control (FEC). Also, it only provides rudimentary control on the amount of ACK traffic it generates, and its automatic tree configuration algorithms are not suited to all environments. Finally, it does not include many features such as explicit asymmetrical network support, explicit network management, counted group membership, or

time bounded reliability. These limitations motivated us to extend the core ideas of RMTP to create RMTP-II.

The next section details a small sampling of the work that has been done on reliable multicast, and the primary work on which RMTP-II has drawn. Given the wealth of rich research, we approached the design of RMTP-II as largely a task in selecting engineering trade-offs between complexity and functionality, to provide a solution that met real market requirements. Some of the primary driving applications for RMTP-II are real-time data (e.g., financial market data), file transfer, multimedia streaming (including large-scale distance learning), publish/subscribe middleware, and server replication. More details on typical multicast application requirements can be found in [5]. These applications share the demands of typically having a few senders and up to tens of thousands of receivers. They require that a protocol provide good throughput of application data to receivers, and may also require packet-level or application-level delivery confirmation. They do not require any total ordering of messages. They may require limited recovery of data for late joining receivers, but this can be implemented more easily at a higher level. The applications need to be deployable on a wide range of networks, including

not only terrestrial intranets and the Internet, but also highly asymmetrical networks, such as satellite downlinks with terrestrial return paths, wireless networks, and networks with thin clients for receivers. Finally, standard many-to-many multicast is not available at the network layer in many of the target networks. RMTP-II attempts to address all of these requirements.

A key requirement for many of these applications is the need for group membership and positive confirmation of data delivery to receivers. Many of the present driving applications for one-to-many multicast are for sending high-value data, and wish to get assurances at the sender that all data was delivered to all the receivers. Tree-based ACKs (TRACKs) appear to be the best proposal to date for providing these guarantees in a real-time scalable fashion. The core of RMTP-II is a set of algorithms that provide and manage this TRACK traffic. While it has been shown that TRACK protocols have the highest scalability of any non-FEC-based protocol [6], it has also been shown that FEC is required in order to scale in the face of independent loss, and we show below that NACKs are essential for providing low-latency delivery in the face of packet loss. For these reasons, RMTP-II also includes optional NACK and FEC algorithms.

One of the primary barriers to deploying multicast applications is the concerns of network managers that these applications will hurt their networks, because of the large potential fanout of each application and the high bandwidth of multimedia applications. The IETF also shares these concerns, and emphasized in RFC 2357 [7] the need for standards-track reliable multicast protocols to provide congestion control that is fair with TCP. By differentiating between applications (senders and receivers) and network elements (top nodes and designated receivers), RMTP-II can provide network managers with tools for monitoring and controlling the performance and network utilization of large applications. In addition, the extra feedback provided by TRACKs (as opposed to NACKs) allows RMTP-II to solve two of the most difficult problems in congestion control: scalable round-trip time (RTT) measurements and slow start [8].

While TRACK-based protocols make it easier to solve the above set of diverse requirements, they come with very real trade-offs. These include the risk of generating more control traffic than NACK-only protocols, and the problems — a single potential point of failure and potential bottleneck — of having a centralized top node (TN) in the tree. Even more important, there is considerable difficulty in configuring the topology of the hierarchy in a way that is approximately congruent with the underlying physical network topology.

To address the first of these, RMTP-II provides a set of smoothing and control algorithms to manage and limit the TRACK control traffic. These algorithms do not eliminate the control traffic trade-off, but allow it to be explicitly monitored and controlled. RMTP-II also provides an optional hot backup to the TN to eliminate the potential single point of failure. It minimizes the amount of work done by a TN (including restricting it from doing any retransmission of packets) in order to minimize the risk of it becoming the bottleneck in the system.

RMTP-II provides an algorithm for automatically configuring the tree if there is only a single-level hierarchy, which can be sufficient for real-time applications of up to 100 or more receivers and non-real-time applications of up to 1000 or more receivers. For larger deployments, RMTP-II assumes the existence of manual configuration files or a separate session manager component, to handle the configuration of interior tree nodes (designated receivers, DRs). Also, while RMTP-II provides integrated window-based flow control and rate-based sender limits, it does not seek to provide TCP-friendly congestion control. Both of these components — automatic tree configuration and TCP-friendly congestion control — are of

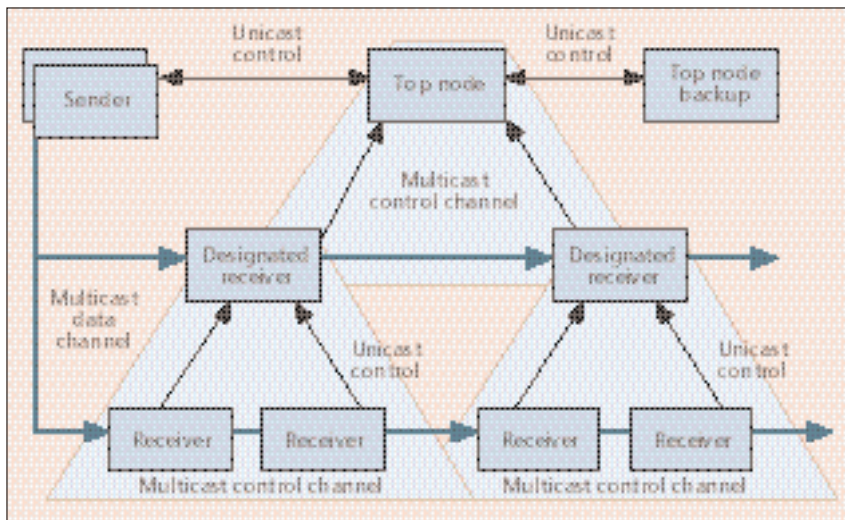
sufficient complexity and difficulty that we have left them out of the core design. This has allowed the protocol to address more of the core features needed for reliable delivery without becoming overly complicated, and allows the “best of breed” of these separate components (both of which are the subject of very active research) to be used as they become mature.

Related Work

RMTP was the first protocol to propose using a hierarchical structure to achieve scalability. TMTP was the first to demonstrate the advantage of combining both ACKs and NACKs together [9]. Reference [10] showed the benefits of integrating FEC, in both reactive and proactive forms, to reduce the cost of repairing independent losses across multiple receivers and to reduce the average latency of the protocol. SHARQFEC [11] quantified the substantial advantages of combining both hierarchical mechanisms and FEC. MTP and MTP-2 [12, 13] showed the value of imposing explicit roles on the different nodes of the protocol, which allows the protocol to be simplified and provides more control to the network manager. SRM provided fundamental work on how to make NACKs efficient. It showed how stochastic algorithms for setting the delay before transmitting NACKs, combined with delays on responding to NACKs, can provide effective suppression of NACKs [14]. This technique required the computation of the RTT between each node and each sender, which is impractical in many environments, and prompted additional work on NACK backoff policies. Reference [15] proposed the use of an exponential backoff for NACKs, and showed that this is extremely effective in suppressing redundant NACK traffic across a very wide range of receiver group sizes. LORAX was the first protocol to advocate the use of shared hierarchical control trees for multiple senders [16]. MFTP was the first protocol to demonstrate the importance of supporting asymmetrical networks, such as satellite networks, with congested or low-bit-rate return paths [17]. RMTP-II has borrowed heavily from all of these important works. Finally, LORAX [18], LMS [19], and PGM [20] all proposed the use of new protocol software in the routers, to (among other things) facilitate the discovery of the router tree and to provide efficient suppression of NACK transmission and repairs. While RMTP-II does not require new router software to be added, these protocols demonstrated the importance of allowing for future router acceleration, and RMTP-II was designed with this in mind.

There have been many dozens of individual reliable multicast protocols proposed over the years. Recently, a rough consensus on a general taxonomy has been reached among some of the community, which is discussed in [21] and [5]. Under this taxonomy, RMTP-II is a tree-based acknowledgment (TRACK) protocol. The original TRACK protocols were RMTP [4] and TMTP [9]. Recently, the TRAM protocol was proposed [22], which appears to draw heavily on RMTP and has high similarity to RMTP-II, but integrates a new algorithm for automatic tree configuration.

The second class of protocols is the NACK-only protocols, such as SRM and MDPv2 [23]. As mentioned above, the original SRM protocol had significant scalability limitations [24, 25] and required many-to-many multicast. Recent work has been done to reduce these limitations. These changes include the integration of FEC and optional use of exponentially timed NACKs. Some experimentation was also done with constructing hierarchically scoped local groups. MDPv2 is a one-level NACK-only protocol, which uses the exponential NACKs proposed in [15], along with integrated FEC. NACK-only protocols like MDPv2, which use exponential NACKs and do not attempt to construct a hierarchical topology, are roughly equiv-



■ Figure 1. An RMTP-II tree.

alent to RMTP-II's NACK algorithms for any one parent, as described later. Compared with the hierarchical version of SRM, the primary difference is that SRM assumes that the end clients perform the hierarchically scoped functions, while RMTP-II assumes that manually configured servers perform this functionality. This NACK-only class of protocols does not provide confirmed data delivery. However, they are simpler to implement, and the lack of hierarchy in most versions of these NACK-only protocols removes the need for tree configuration. The flow control provided by the positive ACKs in TRACK protocols such as RMTP-II allows them to provide higher maximum throughput than NACK-only protocols.

The third class of protocols are the router assist protocols, such as PGM, LORAX, and LMS. These protocols use new software in the routers to provide NACK suppression and more tightly scoped retransmission to the receivers that need them. This allows them to provide much of the scalability of a TRACK protocol, with automatic tree configuration, since the protocols are able to discover and use the underlying multicast topology. However, the long deployment cycle of protocols such as IP multicast demonstrate the difficulty of deploying new protocols that require both new host and router software.

The fourth class of protocols are open loop protocols, which only use FEC encoding techniques for reliability. These protocols do not provide real-time delivery and inherent confirmed data delivery, but have the highest theoretical scalability of any protocol class.

Design Goals and Network Assumptions

The following is a list of the top RMTP-II design goals:

- **Few-to-many delivery:** The protocol supports up to a few senders per transport session, but without any ordering guarantees across these senders. Its primary focus is on scaling the number of receivers.
- **Receiver scalability:** This is a primary design goal. Scalability is achieved through the combination of the following:
 - Hierarchical positive ACKs, with strict management of the amount of return traffic
 - Local retransmission from DRs
 - Use of optional FEC to reduce the effects of independently distributed loss
 - Use of optional NACKs to allow TRACKs to be sent very infrequently; NACKs scoped in a hierarchical fashion, as opposed to sent to the entire group
- **Strong reliability guarantees:** The protocol provides a fully distributed group membership and acknowledgment algorithm,

which allows TRACKs to give positive confirmation when all receivers have delivered a packet.

- **Explicit network management:** All senders must connect to the TN in order to send to the group. The TN provides a centralized point of management and control for network managers.
- **Asymmetrical network support:** The protocol differentiates between the control channel and the data channel, to allow the protocol to work in fundamentally asymmetrical environments, such as with a satellite downlink and a terrestrial return path.
- **Optional router assist:** It has been well demonstrated that special software in the routers can increase the scalability of a protocol. However, if IP multicast is an example, this software tends to take an extremely long time to deploy widely.

Hence, the protocol is designed to take advantage of generic router assist when it becomes available, but without requiring it for scalability.

- **Support of congestion control:** RMTP-II is designed to support TCP-friendly congestion control. Its hierarchical positive ACKs allow it to solve two of the most difficult problems: slow start and scalable RTT measurements. Its use of hierarchy also allows it to take advantage of restricted worst edge calculations, which ameliorate the drop to zero problem in RMCC. [1]
- **No requirement of many-to-many multicast:** Some multicast deployments do not support receivers being able to source multicast packets to the same group a sender is using. RMTP-II is designed not to require this feature.
- **Simplicity:** While more complex than some protocols, RMTP-II strives to be as simple as possible. One way it does this is by differentiating the roles of the group members.

A Protocol Overview

RMTP-II provides sequenced reliable delivery of data from a few senders to a large group of receivers. RMTP-II consists of a network that has one or more sender nodes (SDs), many receiver nodes (LNs), a TN, and zero or more DRs. It may also have a TN backup (TN-B).

In order to provide optimal support for asymmetrical networks, RMTP-II breaks operations in to a data channel and a control channel. Data is multicast by a sender on the data channel. Data may be retransmitted by the sender on the data channel, or multicast or unicast on a local control channel by a DR. The control channels are organized into a tree with the TN at the top of the tree and the receivers at the bottom of the tree. Designated receivers are always in the middle of the tree.

The simplest configuration consists of a single sender and a TN on one host, and receivers on multiple other hosts connected to the network. An implementation could allow multiple nodes of different types to run in a single process, allowing a host to act as a sender and a receiver, a sender and a TN, a receiver and a DR, or other combinations.

Figure 1 illustrates an RMTP tree with multiple control nodes. In the case of a symmetrical network whose control tree topology is relatively congruent to the multicast routing tree topology, the data channel may have the same path as the control channel. In the case of an asymmetrical network, such as a one-way satellite network with a terrestrial return path, the receivers and DRs would receive the multicast data directly, but the TNs (and perhaps some of the DRs) will only receive control traffic.

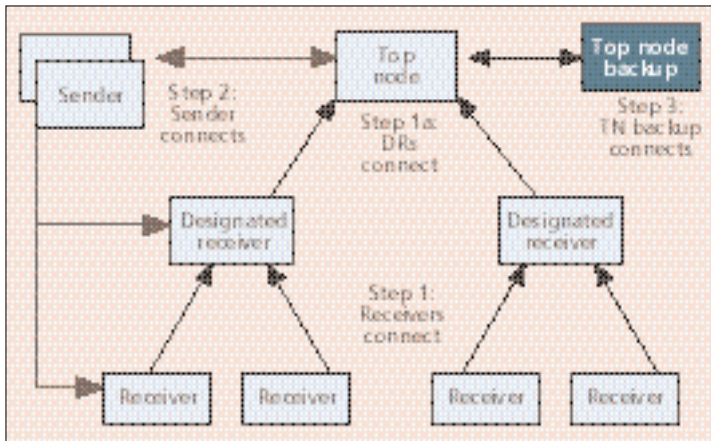


Figure 2. An example of normal operation.

A receiver joins a data channel to receive data. A receiver periodically informs its parent about the packets it has or has not received by unicasting a TRACK packet to the parent. Each parent node aggregates the TRACKs from its child nodes and unicasts a single aggregated TRACK to its parent. The TN aggregates the TRACKs from its child nodes and unicasts a single TRACK to the sender.

Each DR or TN has a local control channel, which is a multicast group that connects to all of its children. This is used for local multicast retransmission of lost packets to just the parent's children. Each control node also multicasts Heartbeat packets on this channel, which inform the child nodes that the parent node is still functioning.

A tree forms a loop from the sender to the receivers, through the control tree, and back to the sender. Data and NullData regularly exercise the data channel. Heartbeat packets exercise the downward control channel. TRACKs, NACKs, and HeartbeatResponse packets regularly exercise the control channel in the upward direction. This combination constantly checks that all of the nodes in the tree are still functioning correctly, and initiates fault recovery when required.

One RMTP-II tree can support multiple data channels, each with its own multicast address. Each data channel can be used by one or more senders to send data. A receiver must join all data channels corresponding to the streams it wishes to receive. A tree has only a single control tree, which consists of both the local multicast control channels and the unicast packets sent from children to their parents.

RMTP-II differentiates between clients (senders and receivers) and interior nodes (TNs and DRs). It assumes that the interior nodes may be operated by network managers as part of the infrastructure, and offers network management tools that allow the managers to control the rest of the tree from the interior nodes. The protocol collects statistics at each interior node, which cover the resources used by the children of that node, and makes these available through a Simple Network Management Protocol (SNMP) interface. The network manager can also change key parameters at the TN, which control the performance of the connection, and the resources it is allowed to use. If a client (sender or receiver) is misbehaving, the network manager can eject that client from the group. The network manager uses SNMP to communicate with an internal control node, and that node then issues the eject command to the misbehaving client.

Figure 2 shows an example of normal operation. At time 0, a TN and two DRs are running, but are not connected to each other. At time 1, a set of four receivers join the tree by contacting the two DRs, prompting each DR to join the TN. At time 2, a sender joins the TN, and advertises a new multicast data channel. The receivers all join the new data channel, and

the tree is now formed. At time 3, a TN-B starts and joins the TN. This produces the full topology in Fig. 2, and a fully operational, fault-tolerant tree. For now, we assume that all of the nodes have a static configuration file, which provides the address of their parent. Later, we discuss how the receivers can automatically locate their parents, using a multicast address.

With the sender quiescent, the tree is still sending control packets around. The TN and DRs each send occasional Heartbeat packets to their local multicast control channel, and their children (including the sender and TN backup) reply with Heartbeat Response packets. The sender periodically issues NullData packets to notify the receivers that it is still active, and to let them discover any data packets they might have missed (even though none have yet been sent).

At time 4, the sender starts sending a stream of data packets on the multicast data channel. The receivers and DRs each receive these packets, and respond by generating TRACKs and propagating them up the control tree, and back to the sender. When the sender gets a TRACK for a data packet it has sent, it can advance its transmission window and free that packet from memory. If a packet is missed, it will be requested implicitly in the next TRACK sent by that node, or explicitly with a NACK if they are enabled. If the node's parent has that packet, it will retransmit it to its local control channel. Otherwise, it will pass the request up the tree. If a retransmission request gets all the way to a sender, the sender retransmits the packet on the multicast data channel. If FEC is enabled, instead of requesting a single packet, a child requests how many packets it has lost out of the last FEC window of W packets. The parent then retransmits a number of parity packets to all of its children, equal to the most packets lost out of that window by any of its children.

RMTP-II Algorithms

TRACK Traffic Management

Use of hierarchical ACKs (TRACKs) fundamentally increases protocol scalability, but still leaves the protocol vulnerable to creating congestion collapse on the back channel, particularly in highly asymmetrical environments where the data channel may be much larger and less congested than the control channel. RMTP-II manages this issue with a set of global tree parameters and algorithms for TRACK management. If the optional NACKs are not enabled, there is a fundamental trade-off between the amount of control traffic generated and the speed at which losses are detected and recovered from. RMTP-II makes this trade-off explicit by creating a relationship between the amount of control traffic and data traffic, managed by the following variables:

- B — Maximum branching factor. This is a tree-wide constant for the maximum number of children that can be attached to any interior tree node.
- R — Maximum overhead ratio. This is a tree-wide constant for the maximum number of ACKs that should be received, at any interior tree node, for each data packet.
- H — TRACK window. This is the maximum period, measured in sequential sequence numbers, between TRACKs generated at a receiver. $H = \text{Ceiling}(B/R)$
- M — Child modulus. Each child is assigned a unique number by its parent, M , between 0 and $B - 1$. The child will then respond to any packets with sequence number N where $N(\text{Modulo } M) = 0$.

As an example, assume a parent has $B = 6$ children, and the desired ratio of TRACK packets received to data packets sent is $R = 2$. Then the children will respond to every $H = 6/2$

= 3 data packets. The children are separated into three groups by their M numbers (Modulo H), 0, 1, 2. The children with number 0 will respond to sequence numbers that equal 0 (Modulo 3). The children with number 1 will respond to sequence numbers that equal 1 (Modulo 3). The children with number 2 will respond to sequence numbers that equal 2 (Modulo 3). The six children are partitioned into three response classes of size $6/3 = 2$. There will be $B/H = 2$ children responding to each data packet.

```
Sequence #:      101 102 103 104 105 106 107
108
Sequence #, Mod 3: 2   0   1   2   0   1   2
0
```

```
Receiver Index #:      1, 2, 3, 4, 5, 6
Receiver Index (Modulo 3): 1, 2, 0, 1, 2, 0
```

R is the primary control for a tree's trade-off between minimizing control traffic, and minimizing latency and required buffering. Note that B refers to the maximum number of children for a DR, not the actual number of children. For cases where B dramatically overestimates the number of children, the actual control traffic will be much less than R , but this allows H to be a tree-wide parameter.

When data traffic is low, a receiver may not send a packet for a long time. This could cause long waits for packet stability and also make the receiver appear to have failed. The tree-wide constant $T_{\text{track_max}}$ guarantees that receivers respond in a timely manner. $T_{\text{track_max}}$ specifies the maximum time that can elapse between TRACKs while the stream is active. If the stream is active, a receiver sends at least one TRACK within the interval $T_{\text{track_max}}$. The value of the variable T_{track} dynamically controls the time between TRACKs. T_{track} varies as the average of the previous two inter-TRACK times, $T_{\text{track}1}$ and $T_{\text{track}2}$. The values of $T_{\text{track}1}$ and $T_{\text{track}2}$ are initially set to $T_{\text{track_max}}$. $T_{\text{track}} = \text{Min}((T_{\text{track}1} + T_{\text{track}2}) * C, T_{\text{track_max}})$, where C is a tree-wide constant that determines the responsiveness of T_{track} . The recommended value for C is 1.

The T_{track} timer is initialized when a receiver receives the first packet of a data stream. A new T_{track} value is calculated and the T_{track} timer reset each time the receiver sends a TRACK packet. If the data traffic load is high, H packets are received in time T_{track} or less and the above *rotating TRACK* algorithm determines the generation of TRACKs. If the load is low, the T_{track} timer expires before H packets are received, and T_{track} determines the generation of TRACKs. Under low load conditions, the value of T_{track} grows in an approximately exponential manner to $T_{\text{track_max}}$.

NACK Traffic Management

While TRACKs provide a constant, well managed stream of information about the packets which have and have not been received, RMTP-II's optional use of NACKs is key for supporting real-time delivery applications. As we will show later, the average recovery time for a lost packet when NACKs are enabled is usually significantly lower than when TRACKs alone are used. Like many other protocols, RMTP-II uses the exponentially weighted timer algorithms proposed in [15] to provide optimal NACK suppression. When a receiver discovers it has missed a packet, it sets a random, exponentially weighted timer. If the receiver has not yet seen a repair for that packet when the timer goes off, it unicasts a NACK packet to its parent. When the parent receives a request for a lost packet, if it has the lost packet, it multicasts the repair to all of its children on its local control channel. If it does not have the packet, it passes the request on to its parent and multicasts a control packet to its children to suppress other NACKs.

Reliability Semantics

In a reliable multicast protocol, one of the major issues is deciding when to declare to a sender that all receivers have successfully received a packet so that it can be discarded from the sender's retransmission buffers. The simplest method (often used with NACK-only protocols) is to explicitly designate how large the retransmission buffer should be. This is specified in terms of either time (the sender holds onto all packets for N s after they were first transmitted) or in space (the sender holds a maximum buffer of the last K bytes that were transmitted). However, this makes inefficient use of the retransmission buffers, and does not provide any explicit confirmation of which members received the data. RMTP-II's TRACKs come from all receivers, and thus allow it to provide positive confirmation of the data's delivery to all receivers.

As first analyzed in [26], there are multiple choices on how to do the TRACK aggregation, which trade off between latency and buffer space, and reliability in the face of persistent failures. RMTP-II offers two options: pessimistic aggregation and optimistic aggregation.

Pessimistic aggregation requires that any interior node get a positive confirmation from each of its children before it can acknowledge that packet's stability to its parent. Optimistic aggregation allows an interior node to acknowledge that a packet is stable as soon as it has received it, assuming that it is capable of performing local retransmission to all its descendants. For optimistic aggregation, the TRACK bitmap that requests retransmission of lost packets also serves to report confirmed data delivery. For pessimistic aggregation at nodes which also provide local recovery, a separate field must be used to report stability, because the stability semantics are different than the retransmission request semantics of the TRACK bitmap. This field, which denotes the highest sequential sequence number (HighestStableSeqNum) which has been delivered to all receivers, is then advertised by the sender in the data packets it sends out. This allows the interior nodes to hold onto packets for retransmission until all receivers in the tree have received it.

The advantage of optimistic aggregation is that the sender gets stability of packets before the packets are actually stable at all the receivers. This decreases the buffering requirements at the sender. The disadvantage of optimistic aggregation is that it reduces the ability of a tree to allow all receivers to recover from the failure of their parents.

Consider an RMTP tree in which a receiver has a DR as parent and a DR as the next ancestor node. The parent node receives all the packets, but the receiver misses a packet. If the receiver's parent dies before it can retransmit the missing packet and the receiver rejoins the ancestor node, the receiver likely cannot recover the missing packet. Using optimistic aggregation, the failed parent would have reported that the message had gone stable, and the ancestor and sender would have then released the packet by the time the receiver had rejoined to the ancestor. With pessimistic aggregation, neither the sender nor the ancestor would have received a stability notification for that packet yet.

Reliability and Ordering Options

As described above, RMTP-II offers two options for data delivery confirmation: pessimistic and optimistic aggregation. RMTP-II also offers three levels of reliability/ordering semantics: unordered, source-ordered, and time bounded. Source-ordered is similar to TCP: a receiver will receive all packets, in the order in which they were sent from a single source, or it will declare a failure for the stream. Unordered delivery attempts to provide reliable delivery, but delivers the packets

in the order in which they are received from the network. If a packet is unrecoverable, the receiver can choose to declare a failure or continue operating.

Time bounded reliability provides bounded reliability for real-time applications with strict jitter requirements. Recovery on a lost packet is only performed for up to T_{recover} time. Each receiver uses the control tree Heartbeats to calculate the approximate delta between its clock and the clock of the sender, as well as the one-way RTT to the sender. With time bounded reliability, the sender timestamps a packet with the time it is sent, using its local clock. When the receiver gets a packet with sequence number $N + 1$, but has not yet received packet N , it waits up to the time bound T_{recover} on packet $N + 1$, calculated using the packet's timestamp TS_{N+1} and the clock differential to the sender, C_{delta} . It will attempt recovery of any lost packets with smaller sequence numbers until time $T_{\text{max_recover}}$, after which it declares the packets lost, and delivers packet $N + 1$ to the application. $T_{\text{max_recover}} = TS_{N+1} + C_{\text{delta}} + T_{\text{recover}}$.

Finally, as part of its flow and rate control, RMTP-II also allows bounds to be set on the minimum and maximum sender throughput. When a receiver is not able to keep up with a minimum data rate, it is forced to leave the group.

Loss Notification and Recovery

RMTP-II uses TRACKs to notify the sender when a packet has gone stable and can be freed. A TRACK includes the following fields:

- High Sequence Number – The sequence number of the highest numbered packet received
- Low Sequence Number – The sequence number of the lowest numbered packet not yet received
- Stable Sequence Number – The sequence number of the highest contiguous stable packet
- Bitmap – A list of the packets between the High and Low Sequence Numbers, corresponding to whether each packet between is missing (0) or acknowledged (1)

For optimistic aggregation, the Low Sequence Number is the same as the Stable Sequence Number, and the Bitmap corresponds to the packets which have gone stable. For pessimistic aggregation, the Bitmap is used to notify the parent of packets needing to be retransmitted, and the stable sequence number is used to notify the parent when it can release retransmission buffers. If the Bitmap is larger than a given size, it can be run length encoded.

A NACK is an explicit request for retransmission of missed data. Networks that operate under low loss conditions can use NACKs, combined with reduced TRACK traffic, to get faster recovery of lost data packets with less control traffic overhead. A NACK is a TRACK packet with a special NACK flag set. After detecting a missing packet, a receiver waits for a random time Z before generating a NACK. The time, Z , is a random variable having a truncated exponential distribution on the interval $[0, T]$, according to the formulas in [15]. The parent immediately replies to a NACK by retransmitting the missing packet on its local control channel, unless it is suppressing retransmission due to a recent TRACK or NACK. When children receive a retransmission, they cancel any corresponding pending timers, resulting in NACK suppression. If a DR receives a NACK requesting retransmission of packets it does not have, it multicasts an empty retransmission to suppress other NACKs from its children, and forwards the NACK to its parent. A single NACK or empty retransmission can cover multiple packets.

TRACK and NACK packets prompt retransmission of missing packets. This retransmission of lost data packets can be done by the sender or DR nodes. When a DR gets a retransmission request, it fulfills it, if able, on its local control channel. A DR passes up retransmission requests (in a

TRACK or NACK) that it cannot fill itself. When a node sends a retransmission, it calculates a minimum period of time, $T_{\text{min_retransmit}}$, before another retransmission request will be honored for this packet. $T_{\text{min_retransmit}}$ is calculated as the local RTT of the control channel multiplied by 2 to the power of the number of retransmissions of that packet (including the current one). $T_{\text{min_retransmit}}$ can never exceed $T_{\text{max_retransmit}}$, a fixed system constant.

When a DR receives retransmissions from its parent, it forwards these retransmissions to its children via its local control channel, assuming it has a pending request for that packet. A TN that receives a NACK acts just like a DR that does not have the packets for retransmission, with the parent of the TN being the sender for that stream. However, when a sender retransmits a packet, it is assumed that a large part of the tree is missing the packet, so it is retransmitted on the data channel.

Forward Error Correction

RMTP-II supports two modes of FEC: proactive and reactive. Proactive FEC is a mechanism by which parity packets are sent along with the data packets. The receivers use the parity packets to recover the missing data packets. This is used in environments where it is known ahead of time that there is a high loss rate in the network (such as in a wireless LAN), and to support real-time applications.

Reactive FEC is a mechanism by which the sender encodes and sends parity packets only if it gets notification about missed data packets. The sender sends parity packets instead of retransmitting the data packets. The receivers are able to repair different lost packets with these parity packets. This is used in environments where receivers face independent loss. For example, if receivers are connected to the network by modem, the losses at the various receivers may be independent. If each receiver loses a different packet, in the worst case, every packet could have to be retransmitted. With reactive FEC, for a window of N packets, the retransmitter only has to retransmit a number of parity packets equal to the maximum number of lost packets in the window for any descendant.

Distributed Membership

Traditionally, multicast membership algorithms have provided either total membership knowledge or no membership knowledge [14]. The first does not scale, and the second does not allow for confirmed data delivery. RMTP-II's membership algorithm solves this by providing aggregated ACKs. Optionally, it can also provide *counted membership*.

Each node in the tree is responsible for maintaining the list of children joined to it, its parent, and any alternate backup parents. When a child joins a parent, it does so using a simple request-reply protocol to the parent. If a receiver joins a stream in process, it is only allowed to start at the current highest sequence number the DR has received. The DR relays this number in response to a join request. The parent can accept or reject requests to join, and is responsible for detecting the failure of its children.

In order to join a tree, a node must know some information about which parent in the tree it wishes to join. It can specify either the IP address and UDP port for the parent, or the IP multicast address and UDP port for the local control channel of the parent, assuming that this multicast address supports many-to-many multicast. Each control node must specify the multicast address of the local control channel it is going to use, as well as an optional UDP port that it will use to receive unicast join requests. Each node must also specify its role in the tree. All other information, including global tree configuration parameters and the address(es) of the data channel(s), are automatically propagated through the control tree. Note

Detection of	At neighbor	Packet type	Detection interval
Top node	Backup TN	Heartbeat	$F * T_{hb}$
Top node	Child	Heartbeat	$3 * F * T_{hb}$
Child	Parent	TRACK, HeartbeatResp	$2 * F * T_{hb}$
Backup TN	Top node	HeartbeatResp	$3 * F * T_{hb}$
DR	Child	Heartbeat	$2 * F * T_{hb}$
Sender	All	NullData	$2 * F * T_{nulldata_max}$

■ Table 1. *Failure detection.*

that in the case of a single-level tree (with a TN but no DRs), this allows for near-automatic configuration. Each node simply specifies its role (which the application must know) and a common multicast address/port.

When a receiver or DR wishes to leave the tree, it also does so with a request-reply protocol. Normally, a receiver will wait until the sender advertises the end of a stream, and it has delivered all packets up to the end of the stream, before it leaves the group. A DR should not leave the group until all of its children have left the group. If it needs to leave for some exceptional reason, it should send eject messages to all of its children first.

Counted membership is an optional tool for getting usage information on the tree. With this, when a node sends a TRACK to its parent, it includes the current number of descendant receivers. This is aggregated at each interior node in the tree, providing the sender with a reasonably accurate count of the number of current receivers.

Fault Tolerance

Pessimistic aggregation provides the foundation for RMTP-II's fault-tolerant semantics. The protocol is designed to allow all receivers to continue to receive a data stream in the face of the failure of any single interior node. In the face of more complex failures, the protocol is designed to degrade gracefully, shedding any receivers that are not able to recover and continuing operation with the piece of the tree that is still functioning. Pessimistic aggregation gives the fault detection and recovery algorithms time to recover from a failure, and still be able to recover lost packets.

A Fault-Tolerant Top Node — RMTP-II provides a fault-tolerant TN capability by allowing the configuration of a backup TN. The backup TN monitors the TN's Heartbeat packets, and takes over if it stops detecting these. The TN sends the backup TN's address in the Heartbeat packets to its child nodes in case the children first detect a failure.

Failure Detection — The protocol detects failures through three keep-alive packets: Heartbeat, HeartbeatResponse, and NullData. The Heartbeat packet is multicast periodically from a parent to its children on its local control channel, and the children then respond by unicasting a HeartbeatResponse packet back. NullData packets are sent by a sender when it has no data to send, to "exercise" the control tree. The key parameter for failure detection is the global tree parameter F . The higher the value of F , the more keep-alive packets must be missed before a failure is declared. Another important parameter is T_{hb} , the maximum amount of time between Heartbeats or HeartbeatResponses. Finally, $T_{nulldata_max}$ is the maximum amount of time between NullData packets.

The goals of the failure detection algorithm are:

- To detect parent failures sufficiently faster than detecting

child failures, to allow children to fail-over to another parent before confirmed data delivery for packets the child missed is declared. When a packet is confirmed as delivered, it is removed from possible retransmission.

- To have the TN backup detect a TN failure sufficiently faster than the TN's children, to allow it to take over before the children declare failure.
- To detect all failures quickly enough that the failures can be handled before the buffering limits are exceeded, which is tied in with flow control.
- To avoid mistakenly detecting failures due to normal packet loss; in particular, to have an even lower chance of mistakenly declaring failure cases that can't be recovered from, such as a DR declaring the TN or a child failed.

Table 1 lists the detection intervals that nodes use to detect the failure of their neighbors.

This algorithm can meet the above stated goals, as long as the following conditions are met:

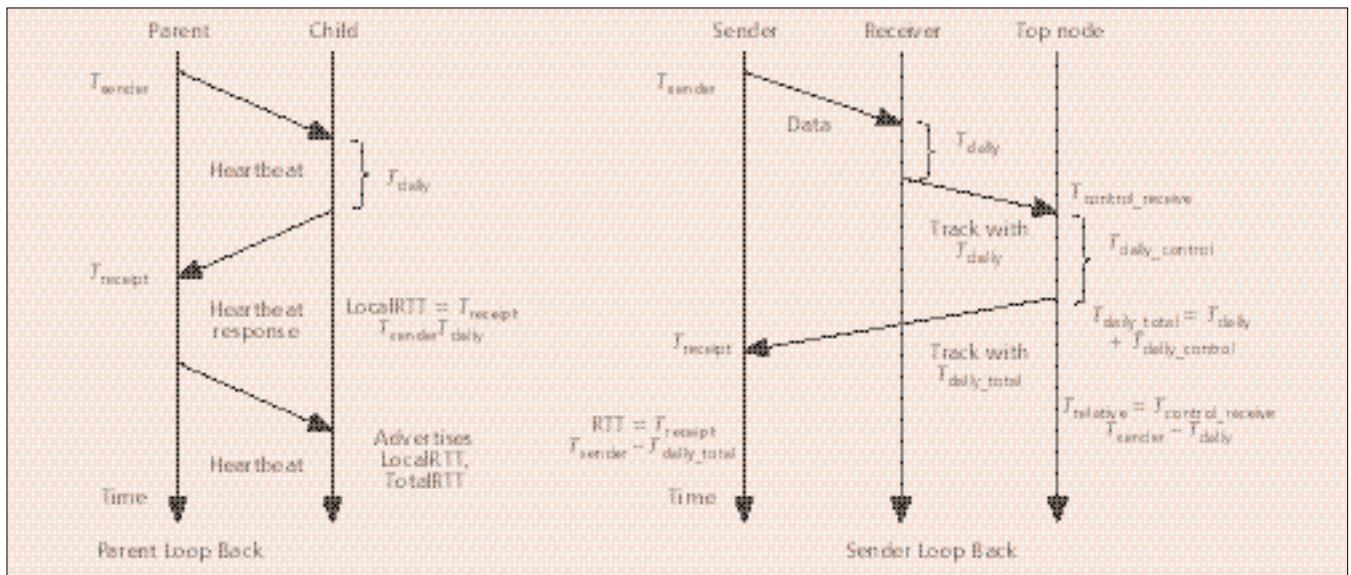
- F must be at least 3.
- The time for a receiver to switch to a new parent is no more than $2/3$ of T_{hb} .
- F is sufficiently large for the maximum loss rate and receiver scale expected for deployment.

The more receivers in a tree, and the higher the loss rate, the larger F must be in order to give the same probability that erroneous failures won't be declared. For example, take a network where the maximum (uniformly randomly distributed) packet loss rate to each receiver is 10 percent, there are 1000 receivers, and F is 5. For any period of $F * T_{hb}$ at the maximum loss rate, there is a 1 percent chance of a receiver declaring a parent DR failure and switching to a new parent. There is a 0.00001 percent chance of a DR declaring the failure of any receiver during that period.

Failure Recovery — The backup TN detects the TN's failure within an interval that is $1/3$ the interval required by the other children of the TN. The backup TN takes over the TN's responsibility and sends Heartbeat packets on the TN's multicast control channel.

When the TN's children start to receive the backup TN's Heartbeat packets, they validate the backup TN's identity and then send a request to the backup TN to rejoin the group, using the backup TN as the new TN. This allows the backup TN to construct a new list of the child DRs it will have that have survived the failure. Because there is no shared state between the TN and the backup TN, except for the list of data channel and control channel addresses, there does not have to be any special consistency between the TN and backup TN.

RMTP-II allows parents to advertise their ancestors and/or peers to their children, as part of Heartbeat packets. When a child detects failure of a DR parent, if the child node has information about other control nodes, it can try to reconnect to an alternate parent node. Like joining a backup TN, the child node sends a request to rejoin the tree to the alternate parent node. This rejoin request includes the sequence number of the highest sequentially received data packet in each stream the child was receiving. The parent can then accept the request, reject the request, or accept the request but notify the child that some of the data it missed is unrecoverable. Because a child detects a failure in approximately half the time any parent detects the same failure, there is a high probability that a child will be able to rejoin the stream where it left off, assuming there is adequate bandwidth available for delivering the missing packets. If pessimistic aggregation is enabled, the child should be able to recover all missing packets from any parent it selects in the tree, assuming no more than one interior node has failed. No interior node can release packets for retransmis-



■ Figure 3. RTT algorithms.

sion until they are smaller than the highest sequential stable sequence number (HighestStableSeqNum) advertised from the sender. Therefore, a receiver will only fail if it is not able to transfer to a new parent before its parent's failure is detected and the impact of this on the HighestStableSeqNum is propagated to the interior nodes.

A control node may fail and restart before its children can detect the failure. If a control node receives a TRACK or HeartbeatResponse packet from a child node that is not in its child list, it sends an eject notification that indicates the parent has restarted. The receiver can then decide to declare a failure, or attempt to rejoin the session with the DR, perhaps losing some packets in the process. Finally, a control node may detect the failure of its child, in which case it flushes that child from its list of children. This will cause any packets that were pending acknowledgment on just that receiver to be acknowledged in the next TRACK.

Explicit Network Management

One of the major barriers to deploying multicast applications has been the lack of tools for controlling these applications. One of the primary purposes of having a separate TN is that it provides a centralized point of monitoring and control. The control nodes provide three primary management tools: SNMP monitoring, the dynamic ability to change tree parameters, and the ability to eject misbehaving senders or receivers.

Because the control nodes receive information about all of their children, they can compute aggregated statistics on the number of descendants (using counted membership), local loss rate, cumulative retransmission rate, throughput, and so on. RMTP-II then provides an optional SNMP or RMON2 management information base (MIB) which makes these accessible to network managers with standard network monitoring tools.

In addition, the TN provides a central point of control for global tree parameters. These are all set at the beginning of a session (typically from a configuration file), and provide such things as bounds on the minimum throughput and maximum throughput, the maximum branching factor, and the overhead ratio for control traffic. Using the same MIBs, the network manager can optionally change these parameters in real time, and RMTP-II then propagates these changes to the rest of the tree.

Finally, if a control node notices that one of its children (a sender is always considered a child of the TN) is misbehaving, it can send an eject message to that child. This can be done as part of an automatic congestion control algorithm, as enforce-

ment on the sender's throughput, and as a way of removing receivers that are not able to keep up with the minimum throughput of a session. It can also be done through the network management MIB at either a DR or TN.

Flow Control and Distributed RTT Calculations

RMTP-II congestion control is a large topic of still ongoing research [27]. It is being developed in a way that makes it relatively orthogonal to the protocol, so it can be used with multiple protocols. However, RMTP-II provides some integrated features that are key tools for these flow and congestion control algorithms: distributed RTT calculation and windowed flow control. For flow control, a maximum window, MaxWindow, is set as a tree-wide parameter and passed to all nodes. Each receiver and DR node is expected to be able to allocate at least MaxWindow buffers for incoming packets from that data stream. As explained earlier, each TRACK or NACK contains a Stable Sequence Number field. Each sender keeps track of the difference between the most recently reported Stable Sequence Number and the highest sequence number it has sent. This is not allowed to be larger than MaxWindow.

Accurately measuring RTT to each node in a group is extremely important for congestion control, to allow senders to avoid spurious retransmission, but even more important for use in the TCP response function which is the heart of many of the congestion control proposals. RMTP-II provides two algorithms for distributed RTT calculations: Sender Loop Back RTT measurements and Parent Loop Back RTT measurements. Sender Loop Back is essentially a hierarchical form of the Parent Loop Back algorithm. Figure 3 illustrates how these algorithm work. The Parent Loop Back algorithm depends on the control channel feedback path from a parent to its children and back. In RMTP-II, each control node regularly sends a Heartbeat control packet to its children, and may also send retransmission packets. On each of these packets, it fills in a timestamp field T_{sender} that corresponds to the local clock at the parent at the time the parent sent the packet. The children then echo this back in their next control packet (TRACK or Heartbeat Response) to the parent, along with a T_{daily} measurement, which corresponds to the amount of time they delayed between receiving the packet with the T_{sender} field and sending the return packet. Upon receipt of the packet at time $T_{receive}$, the local RTT can then be calculated as follows:

$$\text{LocalRTT} = T_{receive} - T_{sender} - T_{daily}.$$

In addition to letting each parent find the LocalRTT to each child, this also allows the nodes in the control tree to advertise to their children the cumulative RTT to that child. This works in a hierarchical fashion. The top of the tree (which is the sender, for purposes of these RTT algorithms) advertises, as part of the control packets sent to the TN, the local RTT measurement to it. Then each control node recursively adds up the RTT value of its parents, adds the local RTT to its furthest child, and advertises this to its children in the next locally multicast packet. Optionally, the parent can advertise the actual RTT value to each of its children rather than the worst RTT, but this significantly increases the size of the locally multicast control packets, and so is only used when there is wide disparity, and is only sent as part of Heartbeat packets.

With the Sender Loop Back algorithm, the basic part of the Parent Loop Back method can be used in a hierarchical fashion. This allows the sender to calculate the worst RTT in the entire tree. To do this, the Sender puts T_{sender} in a data packet. Upon receipt of this packet, T_{sender} and T_{dally} are piggybacked on the next control packet (TRACK or NACK) sent from each receiver to its parent. At a control node, the T_{dally} value is aggregated. Each time a control node gets a packet with a set of timestamps in it, it samples its own clock, $T_{\text{control_receive}}$, and calculates the relative RTT to that child.

$$T_{\text{relative}} = T_{\text{control_receive}} - T_{\text{sender}} - T_{\text{dally}}$$

When it comes time for a control node to send a control packet to its parent, it finds the child with the worst T_{relative} value, samples its own current local clock $T_{\text{control_send}}$, and calculates a $T_{\text{dally_control}}$ value.

$$T_{\text{dally_control}} = T_{\text{control_send}} - T_{\text{control_receive}}$$

It then passes up the T_{sender} value from that child, along with $T_{\text{dally_total}}$

$$T_{\text{dally_total}} = T_{\text{dally}} + T_{\text{dally_control}}$$

This Loop Back algorithm has a number of advantages: it is relatively simple, has low overhead, is fully scalable, and can be calculated for each round of TRACK packets sent up the tree. In networks where the latency of the data and control paths is

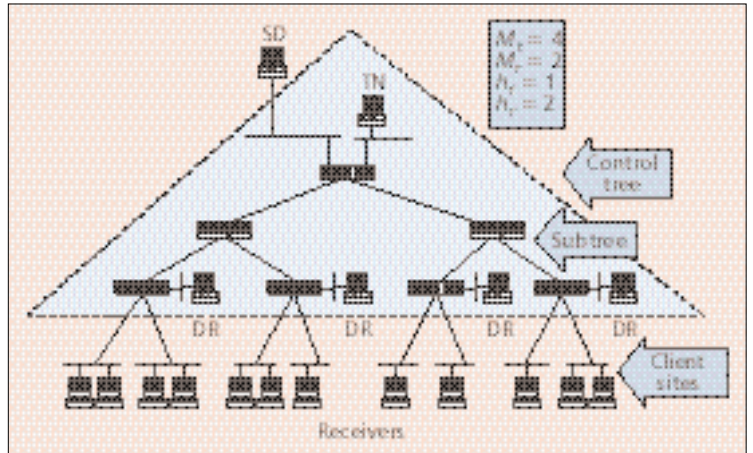


Figure 4. The network model used for protocol analysis.

very different, it still works reasonably well. In the extreme case, it errs on the side of caution — while it may calculate a value for the data path that is much higher than the actual value, it will never produce a value that is less than 50 percent of the actual value. Since it can piggyback information on NACKs and Retransmission packets, as well as TRACK and Heartbeat packets, congested receivers will tend to generate rapid updates of their RTT estimates. This allows a congestion control protocol to more rapidly adapt to congested receivers, while still making sure that each receiver creates a measurement on a regular basis.

However, it also has a number of limitations. The Sender Loop Back algorithm only calculates a single worst-case RTT value, and this is only made available to the sender. This is less than optimal for congestion control algorithms such as TFMCC [27], because it precludes the use of Worst Path or Restricted Worst Edge calculations. For the Parent Loop Back algorithm, fully distributed measurements can be created that enable the use of Restricted Worst Edge calculations, but these measurements will be generated less frequently than the Sender Loop Back algorithm. The hierarchical nature of both algorithms makes them more susceptible to clock measurement errors, making it very important to use high-resolution timers.

Protocol Analysis

This section presents a small set of the results of a performance analysis of RMTP-II [28]. This study developed detailed equations modeling the overhead and latency of RMTP-II, and validated them against both simulations of RMTP-II in OPNET and actual measurements of the protocol's performance in a testbed.

Figure 4 shows the network model used. It is broken into two halves, a control tree consisting of a tree of routers and RMTP-II control nodes, and client sites with sets of receivers running on shared LANs. The control tree consists of a series of subtrees, each of which has a TN or DR at the top and a set of DRs at the bottom. A router is assumed to be collocated with each TN or DR. There may also be zero or more levels of routers in a subtree, between the parent and child control nodes. For purposes of tractability, the control tree is assumed to be full and perfectly balanced. Also, the multicast routing topology and RMTP-II tree are assumed to be symmetrical and fully congruent. Note that this second key assumption depends on the tree configuration, which presently is a key potential weakness of RMTP-II, requiring accurate manual configuration.

The size of each subtree is determined by the router branching factor, M_r , and the number of levels of routers, h_r . A subtree connects a parent to a set of $M_r^{h_r}$ DR children, where $M_r^{h_r} = M_r^{h_r + 1}$. The control tree then consists of h_t levels of

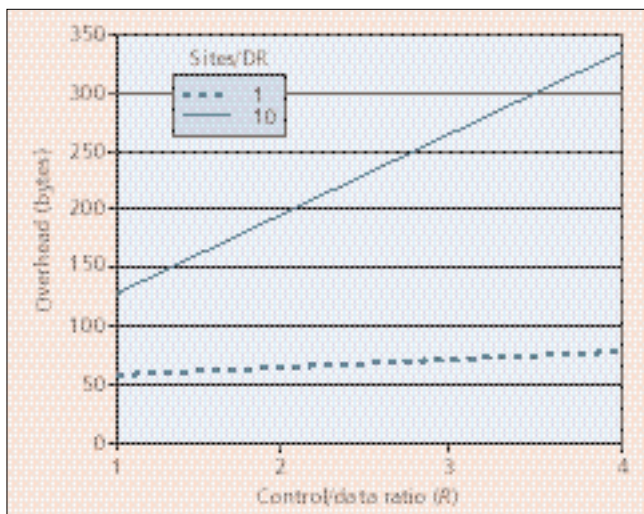
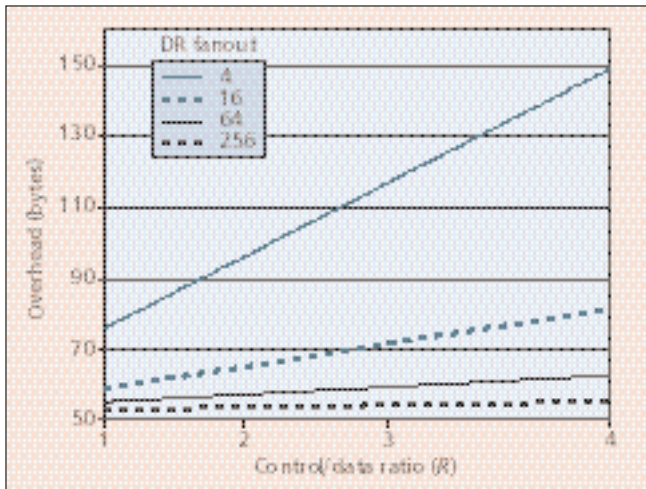
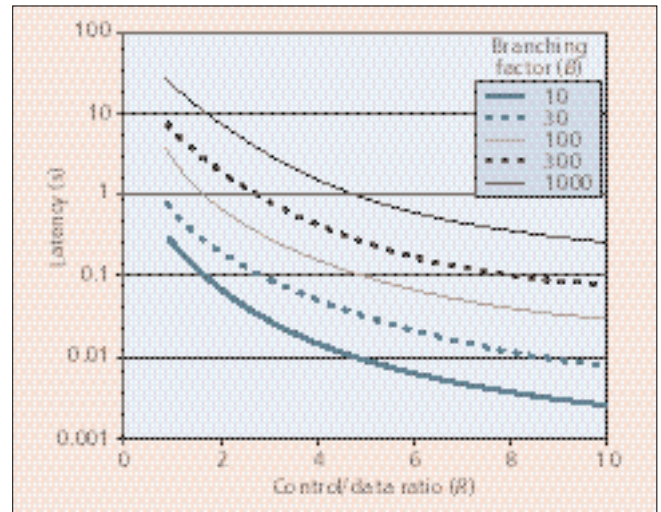


Figure 5. Client site overhead.



■ Figure 6. Control tree overhead.



■ Figure 7. TRACK-based latency.

subtrees, for a total of $N_{DR} = M_t \wedge h_t$ DRs. At the bottom of the tree, each of the N_{DR} DRs is connected to a set of $N_{Sites/DR}$ client sites. Each client site has $N_{LN/Client}$ receivers. In total, there are N_{LN} receivers, where $N_{LN} = N_{DR} * N_{Sites/DR} * N_{LN/Client}$.

As explained previously, the amount of TRACK traffic generated is highly dependent on the RMTP-II constants B (the maximum number of children per node), C (the actual number of children per node), and R (the overhead ratio). In Figs. 5–8, B is set to the maximum of M_t and $(N_{Sites/DR} * N_{LN/Client})$, so C is as close to B as possible. Given this, the amount of TRACK overhead traffic is then primarily a function of R .

The equations model the average overhead on a link, which includes the cost of retransmissions, packet headers, and control packets, but does not include the cost of link headers. They also model the latency of the time it takes to retransmit a lost packet. As with any analytical analysis of a complex process, a list of simplifying assumptions is necessary. In addition to the assumptions mentioned above, the protocol is analyzed during normal static operation without the additional overhead incurred as a result of receivers joining and leaving the control tree. It is assumed that the control channel path and data channel path are congruent, so the data and control paths use the same routers. The equations only provide for uniformly random loss rates. Each link in the control tree has an average packet loss rate of P_r , and each client site has an average packet loss rate of P_{CS} . This model reflects a combination of dependent and independent losses. It assumes that if any receiver at a client site misses a packet, all receivers at that client site miss the packet. However, if a client site attached to a DR misses a packet, this does not affect the probability that other client sites attached to the DR also miss the packet. Inside the control tree, losses are assumed to be mostly independent, although for subtrees with at least one level of interior routers, losses on the top level(s) of the subtree affect multiple receivers.

Figures 5 and 6 demonstrate the theoretical scalability of RMTP-II's control traffic, given this list of assumptions. Bytes overhead per packet are on the y axis, and the overhead ratio R (the number of control packets per data packet sent) is on the x axis. NACKs are not enabled, and the loss rate is negligible. In Fig. 5, the number of children connected to each final DR is 256. Two lines are shown, one for 1 client site per DR, and one for 10 client sites per DR. The overhead, which is an average of the overhead on all links, is higher when all of the receivers are at the same site, because all of the traffic is concentrated on a single link to the DR. Figure 6 shows the average overhead for the control tree. In this graph, there are four levels of subtrees, each with between four and 256 DRs, for a total of between 256 and 4 billion DRs in the control

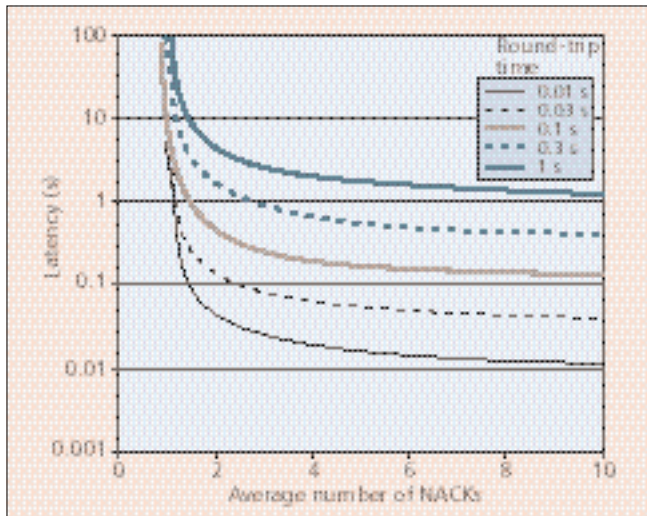
tree. Because the loss rate is negligible in these graphs, the number of subtrees has no effect on the overhead. Thus, this graph shows that the average overhead per link does not change for a tree with fixed fanout, even for very deep trees.

Figures 7 and 8 show the average latency to detect a lost packet when using only TRACKs and when NACKs are used. For both, a single subtree is analyzed. 100-byte data payloads are being sent every 50 ms. For Fig. 7, different lines are plotted for different size groups, from 10 to 1000. The latency shown in this figure does not include the latency due to physical transmission delay. For example, with $R = 5$ and $B = 100$, each client is sending a TRACK every 20 data packets, and the average detection time is 0.5 s. Note that if the period between data packets were longer, the average detection time would also increase. If you assume that the RTT of the subtree is 100 ms, it would take 0.6 s to recover from a lost packet with only TRACKs enabled, but it would take 2.6 s if the data packets were sent every 250 ms. For Fig. 8, the x axis shows the worst case average number of NACKs to be generated in response to a packet lost by 1000 children. The y axis shows the worst case average latency, assuming the worst case of only a single receiver losing the packet. Unlike Fig. 7, this graph includes the physical latency of the link, which is plotted for values between 10 ms and 1 s. These graphs show that the two types of control traffic are useful in different settings. In networks with relatively low RTTs, NACKs can provide the lowest-latency recovery, even with hundreds or thousands of receivers. In higher-latency networks, if the tree fanout is smaller and packets are being sent at a high rate, TRACKs can provide the lowest latency for recovery.

Conclusion

This article presents a high-level overview of RMTP-II, its design goals, and its algorithms. It provided some basic analysis of the scalability of the protocol, which agreed with the conclusions found in [6] about the high scalability of this class of protocols. Some of the strengths and limitations of the protocol can be seen from this article.

RMTP-II's strengths include very high theoretical scalability, a range of reliability levels including time bounded reliability, and explicit tools for monitoring and controlling data streams. It also provides explicit support of asymmetrical networks, support for both real-time and non-real-time applications, and key feedback information necessary for TCP-friendly congestion control. It does not require new router software to be deployed, although it can take advan-



■ Figure 8. NACK-based latency.

tage of generic router assist as it becomes available. It provides optional automatic configuration of receivers, which allows the protocol to be auto-configuring if a single-level topology is used, without DRs. The system provides automatic fault detection and fail-over of interior control nodes.

Like any protocol, it also has to make certain trade-offs, and these produce limitations and weaknesses. RMTP-II's top limitation is its requirement for configuration of the tree topology. At present, RMTP-II requires manual configuration of the inner tree topology. If this configuration is done in a way that causes significant incongruence with the multicast routing topology, it can cause significant performance degradation. Manual configuration of the topology can be a significant barrier to widespread deployment. In addition, if the group membership or network topology changes radically during a session, RMTP-II may not adapt to these changes well. In addition, TRACK overhead can be significant if the application needs low-latency recovery from packet loss and is not able to use NACKs. For this reason, it is recommended that NACKs and/or FEC be used for all scalable real-time applications. Finally, while the protocol attempts to be as simple as possible, it is certainly more complex than some other protocols, such as nonhierarchical NACK-only protocols.

While a tremendous amount of work has been done on RMTP-II, and a commercial implementation has been shipping for two years, much future work remains. In particular, we are working on selecting the best automatic tree configuration algorithms from the body of work that has been done on this topic. We are also doing additional scalability and performance analysis and simulations, including ones with an implementation of the TCP-Friendly Multicast Congestion Control (TFMCC) specification integrated with the protocol. We are examining other potential algorithms for distributed RTT calculations, such as [29] and a variant that creates clock differential calculations without requiring many-to-many multicast. We are looking at adding in an option for header compression, for real-time applications with small data payloads. Finally, we are analyzing the security implications of the protocol [30].

RMTP-II owes a tremendous amount to the wide range of excellent academic work it has built on, and will surely continue to do so in the future, as it continues to evolve.

References

- [1] B. Whetten and J. Conlan, "A Rate Based Congestion Control Scheme for Reliable Multicast," GlobalCast Commun. Tech.I White Paper, Nov. 1998; <http://www.talarian.com/rmtp-ii>
- [2] B. Whetten et al., "THE RMTP-II PROTOCOL," Internet draft, Apr. 1998, work in progress; <http://www.talarian.com/rmtp-ii>

- [3] J.-C. Lin and S. Paul, "RMTP: A Reliable Multicast Transport Protocol," *IEEE INFOCOM 1996*, Mar. 1996, pp. 1414-24.
- [4] S. Paul et al., "Reliable Multicast Transport Protocol (RMTP)," *IEEE JSAC*, vol. 15, no. 3, Apr. 1997.
- [5] M. Handley et al., "The Reliable Multicast Design Space for Bulk Data Transfer," Internet draft, June 1999; work in progress.
- [6] B. Levine and J. J. Garcia-Luna-Aceves, "A Comparison of Known Classes of Reliable Multicast Protocols," *Proc. Int'l. Conf. Network Protocols*, Oct. 1996.
- [7] A. Mankin et al., "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols," RFC 2357, June 1998.
- [8] B. Whetten, "A Proposal for Reliable Multicast Congestion Control Requirements," work in progress; <http://www.talarian.com/rmtp-ii/overview.htm>
- [9] R. Yavatkar, J. Griffioen, and M. Sudan, "A Reliable Dissemination Protocol for Interactive Collaborative Applications," *Proc. ACM Multimedia '95 Conf.*, Nov. 1995.
- [10] J. Nonnenmacher, E. W. Biersack, and D. Towsley, "Parity-Based Loss Recovery for Reliable Multicast Transmission," *Proc. ACM SIGCOMM '97*, Cannes, France, Sept. 1997.
- [11] R. Kermode, "Scoped Hybrid Automatic Repeat Request with Forward Error Correction," *Proc. ACM SIGCOMM '98*, Sept. 1998.
- [12] S. Armstrong, A. Freier, and K. Marzullo, "Multicast Transport Protocol," DARPA RFC 1301, Feb. 1992.
- [13] C. Bormann et al., "MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport," *Int'l. Conf. Comp. Commun. and Networks*, 1994.
- [14] S. Floyd, V. Jacobson, and S. McCanne, "A Reliable Multicast Framework for Lightweight Sessions and Application Level Framing," *Proc. ACM SIGCOMM '95*, Aug. 1995 pp. 342-56.
- [15] J. Nonnenmacher and E. W. Biersack, "Optimal Multicast Feedback," *Proc. IEEE INFOCOM 1998*, Mar. 1998.
- [16] B. Levine, D. Lavo, and J. J. Garcia-Luna-Aceves, "The Case for Concurrent Reliable Multicasting Using Shared ACK Trees," *Proc. ACM Multimedia '96 Conf.*, Nov. 1996.
- [17] K. Miller et al., "StarBurst Multicast File Transfer Protocol (MFTP) Specification," draft-miller-mftp-spec-02.txt, Jan. 1997, work in progress.
- [18] B. N. Levine and J. J. Garcia-Luna-Aceves, "Improving Internet Multicast Routing with Routing Labels," *IEEE Int'l. Conf. Network Protocols*, Oct. 28-31, 1997, pp. 241-50.
- [19] C. Papadopoulos, G. Parulkar, and G. Varghese, "An Error Control Scheme for Large Scale Multicast Applications," *INFOCOM '98*, Mar. 1998.
- [20] D. Farinacci et al., "PGM Reliable Transport Protocol Specification," Internet draft, Aug. 1998, work in progress.
- [21] B. Whetten et al., "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer," Internet draft, Oct. 1999, work in progress.
- [22] M. Kadansky, D. Chiu, and J. Wesley, "Tree-Based Reliable Multicast (TRAM)," Internet draft, Nov. 1998, work in progress.
- [23] J. Macker and B. Adamson, "Multicast Dissemination Protocol Version 2 (MDPv2)," work in progress; <http://manimac.itd.navy.mil/MDP>
- [24] C.-G. Liu, "Error Recovery in Scalable Reliable Multicast," Ph.D. dissertation, Univ. Southern CA, Dec. 1997.
- [25] M. Lucas, "Efficient Data Distribution in Large-Scale Multicast Networks," Ph.D. dissertation, Dept. of Comp. Sci., Univ. VA, May 1998.
- [26] S. Paul, K. Sabatini, and D. Kristol, "Multicast Transport Protocols for High Speed Networks," *Proc. Int'l. Conf. Network Protocols*, 1994, pp. 4-14.
- [27] M. Handley, S. Floyd, and B. Whetten, "Strawman Specification for TCP Friendly (Reliable) Multicast Congestion Control (TFMCC)," work in progress.
- [28] G. Taskale, "Performance Analysis of Reliable Multicast Transport Protocol II," Master's thesis, SUNY Stony Brook, May 1999.
- [29] V. Ozdemir, S. Muthukrishnan, and I. Rhee, "Scaleable, Low-Overhead Network Delay Estimation," work in progress.
- [30] T. Hardjorno and B. Whetten, "Security Requirements for RMTP-II," Internet draft, work in progress, June 1999.

Biographies

BRIAN WHETTEN (whetten@talarian.com) is the chief scientist for Talarian Corporation. Previously, he was the principal founder and CTO of GlobalCast Communications. His primary field of contribution has been reliable multicast protocols, and he is an author in the IETF Reliable Multicast Transport (RMT) working group. He was a co-author of the RMP and RMTP-II protocols, and was responsible for architecting GlobalCast's extensive product line. He holds Bachelor of Science and Master of Science degrees in computer science from the University of Illinois at Urbana-Champaign, and is a Ph.D. candidate at the University of California, Berkeley.

GURSEL TASKALE (gursel.taskale@reuters.com) is currently a programmer analyst at Reuters PLC. He has been investigating multicast technology for use in the near-real-time distribution of reliable data from a small set of senders to a large set of receivers as part of his Master's thesis work. RMTP was introduced at the IETF as a promising solution for this type of use. To better understand the protocol, he has developed analytical models that focused on the overhead, utilization, and scalability aspects. To validate the models he has developed an Opnet simulation of RMTP and performed tests at a testbed. The overhead and utilization results have been presented as a technical paper at SUNY Stony Brook. Recently he has been working for the chief technical officer's organization at Reuters investigating up-and-coming technologies. Prior to Reuters he worked as a software engineer at a small company in Long Island, New York.