RMT Working Group
Internet Engineering Task Force
Internet Draft
Document: draft-ietf-rmt-bb-track-02.txt
November 2002
Expires May 2003

Brian Whetten
Consultant
Dah Ming Chiu
Miriam Kadansky
Sun Microsystems
Seok Joo Koh
ETRI
Gursel Taskale
TIBCO

Reliable Multicast Transport Building Block for TRACK
<draft-ietf-rmt-bb-track-02.txt>

Status of this Memo

Abstract

This document describes the TRACK Building Block. It contains functions relating to positive acknowledgments and hierarchical tree construction and maintenance. It is primarily meant to be used as part of the TRACK Protocol Instantiation. It is also designed to be useful as part of overlay multicast systems that wish to offer efficient confirmed delivery of multicast messages.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119.

1. Introduction

One of the protocol instantiations the RMT WG is chartered to create is a TRee-based ACKnowledgement protocol (TRACK). Rather than create a set of monolithic protocol specifications, the RMT WG has chosen to break the reliable multicast protocols into Building Blocks (BB) and Protocol Instantiations (PI). A Building Block is a specification of the algorithms of a single component, with an abstract interface to other BBs and PIs. A PI combines a set of BBs, adds in the additional required functionality not specified in any BB, and specifies the specific instantiation of the protocol. For more information, see the Reliable Multicast Transport Building Blocks and Reliable Multicast Design Space documents [2][3].

As specified in [2], there are two primary reliability requirements for a transport protocol, ensuring goodput, and confirming delivery to the Sender. The NORM and ALC PIs are responsible solely for ensuring goodput. TRACK is designed to offer application level confirmed delivery, aggregation of control traffic and Receiver statistics, local recovery, automatic tree building, and enhanced flow and congestion control.

Whereas the NORM and ALC PIs run only over other building blocks, the TRACK PI has a more difficult integration task. To run in conjunction with NORM, it must either re-implement the functionality in the NORM PI, or integrate directly with the NORM PI. In addition, in order to have reasonable commercial applicability, TRACK needs to be able to run over other protocols in addition to NORM. To meet both of these challenges, the TRACK PI is designed to integrate with other transport layer protocols, including NORM, PGM [20], ALC [19], UDP, or an overlay multicast system. In order to accomplish this, there can be multiple TRACK PIs, one for each transport protocol it is specified to integrate with. The vast majority of the protocol functionality exists in this document, the TRACK BB, which in turn references the automatic tree building block [16]. For more details on the specific functionality of TRACK, please see the reference TRACK PI[21].

TRACK is organized around a Data Channel and a Control Channel. The Data Channel is responsible for multicast data from the Sender to all other nodes in a TRACK session. In order to integrate with NORM and other goodput-ensuring transport protocols, these protocols are used as the Data Channel for a given Data Session. This Data Channel MAY also provide congestion control. Otherwise, congestion control MUST be provided by the TRACK PI, through using the TFMCC or other approved congestion control building block.

This document describes the TRACK Building Block. It contains functions relating to positive acknowledgments and hierarchical tree

construction and maintenance. While named as a building block, this

document describes more functionality than the PI documents. With the exception of congestion control, almost all of the functionality is encapsulated in this document or the BBs it references. The TRACK PIs are then primarily responsible for instantiating packet formats in conjunction with the other transport protocols it uses as its Data Channel.

The TRACK BB assumes that there is an Automatic Tree Building BB [16] which provides the list of parents (known as Service Nodes within the Tree BB) each node should join to. If Receivers are used that may also serve as Repair Heads, the TRACK BB assumes the Auto Tree BB is also responsible for selecting the role of each Receiver as either Receiver or Repair Head. However, the TRACK BB may specify that a particular node may not operate as a Repair Head.

The TRACK BB also assumes that a separate session advertisement protocol notifies the Receivers as to when to join a session, the data multicast address for the session, and the control parameters for the session. This functionality MAY be provided in a TRACK PI document.

The TRACK BB provides the following detailed functionality.

... .Hierarchical Session Creation and Maintenance. This set of functionality is responsible for creating and maintaining (but not configuring) a hierarchical tree of Repair Heads and Receivers.

- Bind. When a child knows the parent it wishes to join to for a given Data Session, it binds to that parent.
- Unbind. When a child wishes to leave a Data Session, either because the session is over or because the application is finished with the session, it initiates an unbind operation with its parent.
- Eject. A parent can also force a child to unbind. This happens if the parent needs to leave the session, if the child is not behaving correctly, or if the parent wants to move the child to another parent as part of tree configuration maintenance.
- Fault Detection. In order to verify liveness, parents and children send regular heartbeat messages between themselves. The Sender also sends regular null data messages to the group, if it has no data to send.
- Fault Recovery. When a child detects that its parent is no longer reachable, it may switch to another parent. When a parent detects that one of its children is no longer reachable, it removes that child from its membership list and reports this up the tree to the Sender of the Data Session.
- Distributed Membership. Each Parent is responsible for maintaining a local list of the children attached to it.

- Data Sessions. This functionality is responsible for the reliable, ordered transmission of a set of data messages, which together constitute a Data Session. These are initially transmitted using another transport protocol, the Data Channel Protocol, which has primary responsibility for ensuring goodput and congestion control.
    - Data Transmission. The Sender takes sequenced data messages from the application, and passes them to the Data Channel Protocol for multicast transmission. It delays passing them to the Data Channel Protocol if it is presently flow controlled.
    - Flow Control and Buffer Management. Receivers and Repair Heads MAY maintain a set of buffers that are at least as large as the Senders transmission window. The Receivers pass their reception status up to the Sender as part of their TRACK messages. This MAY be used to advance the buffer windows at each node and limit the Senders window advancement to the speed of the slowest Receiver.
    - Retransmission Requests. While primary responsibility for goodput rests with the Data Channel Protocol, Receivers MAY request retransmission of lost messages from their parents.
    - Local Recovery. Repair heads keep track of retransmission requests from their children, and provide repairs to them. If a Repair Head cannot fulfill a retransmission request, it forwards it up the tree.
    - End of Stream. When a Data Session is completed, this is signaled as an End of Stream condition.

...TRACK Generation and Aggregation. This set of functionality is responsible for periodically generating TRACK messages from all Receivers and aggregating them at Repair Heads. These messages provide updated flow control window information, roundtrip time measurements, and congestion control statistics. They OPTIONALLY acknowledge receipt of data, OPTIONALLY report missing messages, and OPTIONALLY provide group statistics. The algorithms include:
    - TRACK Timing. In order to avoid ACK implosion, the Receivers and Repair Heads use timing algorithms to control the speed at which TRACK messages are sent.
    - TRACK Aggregation. In order to provide the highest levels of scalability and reliability, interior tree nodes provide aggregation of control traffic flowing up the tree. The aggregated feedback information includes that used for end-to-end confirmed delivery, flow control, congestion control, and group membership monitoring and management.
    - Statistics Request. A Sender may prompt Receivers to generate and report a set of statistics back to the Sender. These statistics are self-describing data types, and may be defined by either the TRACK PI or the application.

- Statistics Aggregation. In addition to the predefined aggregation types, aggregation of self-describing data may also be performed on Receiver statistics flowing up the tree.

. Application Level Confirmed Delivery. Senders can issue requests for application level confirmation of data up to a given message. Receivers reply to this request, and the confirmations are reliably forwarded up the tree.

- Distributed RTT Calculations. One of the primary challenges of congestion control is efficient RTT calculations. TRACK provides two methods to perform these calculations.
  - Sender Per-Message RTT Calculations. On demand, a Sender stamps outgoing messages with a timestamp. As each TRACK is passed up the tree, the amount of dally time spent waiting at each node is accumulated. The lowest measurements are passed up the tree, and the dally time is subtracted from the original measurement.
  - Local Per-Level RTT Calculations. Each parent measures the local RTT to each of its children as part of the keep-alive messages used for failure detection.

2. Applicability Statement

The primary objective of TRACK is to provide additional functionality in conjunction with a receiver reliable protocol. These functions MAY include application layer reliability, enhanced congestion control, flow control, statistics reporting, local recovery, and automatic tree building. It is designed to do this while still offering scalability in the range of 10,000s of Receivers per Data Session. The primary corresponding design tradeoffs are additional complexity, and lower isolation of nodes in the face of network and host failures.

There is a fundamental tradeoff between reliability and real-time performance in the face of failures. There are two primary types of single layer reliability that have been proposed to deal with this: Sender reliable and Receiver reliable delivery. Sender reliable delivery is similar to TCP, where the Sender knows the identity of the Receivers in a Data Session, and is notified when any of them fails to receive all the data messages. Receiver reliable delivery limits knowledge of group membership and failures to only the actual Receivers. Senders do not have any knowledge of the membership of a group, and do not require Receivers to explicitly join or leave a Data Session. Receiver reliable protocols scale better in the face of networks that have frequent failures, and have very high isolation of failures between Receivers. This TRACK BB provides Sender

reliable delivery, typically in conjunction with a Receiver reliable system.

This BB is specified according to the guidelines in [21]. It specifies all communication between entities in terms of messages, rather than packets. A message is an abstract communication unit, which may be part of, or all of, a given packet. It does not have a specific format, although it does contain a list of fields, some of which may be optional, and some of which may have fixed lengths associated with them. It is up to each protocol instantiation to combine the set of messages in this BB, with those in other components, and create the actual set of packet formats that will be used.

As mentioned in the introduction, this BB assumes the existence of a separate Auto Tree Configuration BB. It also assumes that Data Sessions are advertised to all Receivers as part of an external BB or other component.

Except where noted, this applicability statement is applicable both to the TRACK BB and to the TRACK PIs.

2.1 Application Types

TRACK is designed to support a wide range of applications that require one to many bulk data transfer and application layer confirmed delivery. Examples of applications that fit into the one-to-many data dissemination model are: real time financial news and market data distribution, electronic software distribution, audio video streaming, distance learning, software updates and server replication.

Historically, financial applications have had the most stringent reliability requirements, while audio video streaming have had the least stringent. For applications that do not require this level of reliability, or that demand the lowest levels of latency and the highest levels of failure isolation, TRACK may be less applicable.

TRACK is designed to work in conjunction with a receiver reliable protocol such as NORM, to allow applications to select this tradeoff on a dynamic basis.

2.2 Network Infrastructure

TRACK is designed to work over almost all multicast and broadcast capable network infrastructures. It is specifically designed to be able to support both asymmetrical and single source multicast environments.

Asymmetric networks with very low upbound bandwidth and a very low loss Data Channel may be better served solely through NACK based protocols, particularly if high reliability is not required. A good example is some satellite networks.

Networks that have very high loss rates, and regularly experience partial network partitions, router flapping, or other persistent faults, may be better served through NACK only protocols. Some wireless networks fall in to this category.

2.3 Private and Public Networks

TRACK is designed to work in private networks, controlled networks and in the public Internet. A controlled network typically has a single administrative domain, has more homogenous network bandwidth, and is more easily managed and controlled. These networks have the fewest barriers to IP multicast deployment and the most immediate need for reliable multicast services. Deployment in the Internet requires a protocol to span multiple administrative domains, over vastly heterogeneous networks.

2.4 Manual vs. Automatic Controls

Some networks can take advantage of manual or centralized tools for configuring and controlling the usage of a reliable multicast group. In the public Internet the tools have to span multiple administrative domains where policies may be inconsistent. Hence, it is preferable to design tools that are fully distributed and automatic. To address these requirements, TRACK provides automatic configuration, but can also support manual configuration options.

2.5 Heterogeneous Networks

While the majority of controlled networks are symmetrical and support many-to-many multicast, in designing a protocol for the Internet, we must deal with virtually all major network types. These include asymmetrical networks, satellite networks, networks where only a single node may send to a multicast group, and wireless networks. TRACK takes this into account by not requiring any many-to-many multicast services. TRACK does not assume that the topology used for sending control messages has any congruence to the topology of the multicast address used for sending data messages.

2.6 Use of Network Infrastructure

TRACK is designed to run in either single level or hierarchical configurations. In a single level, there is no need for specialized network infrastructure. In hierarchical configurations, special

nodes called Repair Heads are defined, which may run either as part of a distributed application, or as part of dedicated server software. TRACK does not specifically support or require Generic Router Assist or other router level assist.

2.7 Deployment Constraints

The two primary tradeoffs TRACK has, for the functionality it provides, are additional complexity, and decreased failure isolation. Hence, if target applications are to be deployed in networks with high rates of persistent failures, and isolation of failed Receivers from affecting other Receivers is of high importance, TRACK may not be appropriate. Similarly, if simplicity is paramount, TRACK may not be appropriate.

2.8 Target Scalability

The target scalability of TRACK is tens of thousands of simultaneous Receivers per Data Session. Dedicated Repair Heads are targeted to be able to support thousands of simultaneous Data Sessions.

2.9 Known Failure Modes

If a hierarchical Control Tree is misconfigured, so that loop-free, contiguous connection is not provided, failure will occur. This failure is designed to occur gracefully, at the initialization of a Data Session.

If the configuration parameters on control traffic are poorly chosen on an asymmetrical network, where there is much less control channel bandwidth available than data channel bandwidth, there may be a very high rate of control traffic. This control traffic is not dynamically congestion controlled like the data traffic, and so could potentially cause congestion collapse.

This potential control channel overload could be exacerbated by an application that makes overly heavy use of the application level confirmation or statistics gathering functions.

2.10 Potential Conflicts With Other Components

None are known of at this time.

3. Architecture Definition

3.1 TRACK Entities

3.1.1 Node Types

TRACK divides the operation of the protocol into three major entities: Sender, Receiver, and Repair Head. The Repair Head corresponds to the Service Node described in the Tree Building draft. It is assumed that Senders and Receivers typically run as part of an application on an end host client. Repair Heads MAY be components in the network infrastructure, managed by different network managers as part of different administrative domains, or MAY run on an end host client, in which case they function as both Receivers and Repair Heads. Absent of any automatic tree configuration, it is assumed that the Infrastructure Repair Heads have relatively static configurations, which consist of a list of nearby possible Repair Heads. Senders and Receivers, on the other hand, are transient entities, which typically only exist for the duration of a single Data Session. In addition to these core components, applications that use TRACK are expected to interface with other services that reside in other network entities, such as multicast address allocation, session advertisement, network management consoles, DHCP, DNS, overlay networking, application level multicast, and multicast key management.

3.1.2 Multicast Group Address

A Multicast Group Address is a logical address that is used to address a set of TRACK nodes. It is RECOMMENDED to consist of a pair consisting of an IP multicast address and a UDP port number. In this case, it may optionally have a Time To Live (TTL) value, although this value MUST only be used for providing a global scope to a Data Session, and not for scoping of local retransmissions. Data Multicast Addresses are Multicast Group Addresses.

TRACK MAY be used with an overlay multicast or application layer multicast system. In this case, a Multicast Group Address MAY have a different format. The TRACK PI is responsible for specifying the format of a Multicast Group Address.

3.1.3 Data Session

A Data Session is the unit of reliable delivery of TRACK. It consists of a sequence of sequentially numbered Data messages, which are sent by a single Sender over a single Data Multicast Address. They are delivered reliably, with acknowledgements and retransmissions occurring over the Control Tree. A Data Session ID uniquely identifies it. A given Data Session is received by a set of zero or more Receivers, and a set of zero or more Repair Heads. One or more Data Sessions MAY share the same Data Multicast Address (although this is NOT RECOMMENDED). Each TRACK node can simultaneously participate in multiple Data Sessions. A Receiver

MUST join all the Data Multicast Addresses and Control Trees corresponding to the Data Sessions it wishes to receive.

3.1.4 Data Channel

A Data Session is multicast over a Data Channel. The Data Channel is responsible for efficiently delivering the Data messages to the members of a Data Session, and providing statistical reliability guarantees on this delivery. It does this by employing a Data Channel Protocol, such as NORM, ALC, PGM, or Overlay Multicast. TRACK is then responsible for providing application level, Sender based reliability, by confirming delivery to all Receivers, and optionally retransmitting lost messages that did not get correctly delivered by the Data Channel. A common scenario would be to use TRACK to provide application level confirmation of delivery, and recover from persistent failures in the network that are beyond the scope of the Data Channel Protocol.

3.1.5 Data Channel Protocol

This is the transport protocol used by a TRACK PI to ensure goodput and statistical reliability on a Data Channel.

3.1.6 Data Multicast Address

This is the Multicast Group Address used by the Data Channel Protocol, to efficiently deliver Data messages to all Receivers and Repair Heads. All Data Multicast Addresses used by TRACK are assumed to be unidirectional and only support a single Sender.

3.1.7 Control Tree

A Control Tree is a hierarchical communication path used to send control information from a set of Receivers, through zero or more Repair Heads (RHs), to a Sender. Information from lower nodes is aggregated as the information is relayed to higher nodes closer to the Sender. Each Data Session uses a Control Tree. It is acceptable to have a degenerate Control Tree with no Repair Heads, which connects all of the Receivers directly to the Sender.

Each RH in the Control Tree uses a separate Local Control Channel for communicating with its children. It is RECOMMENDED that each Local Control Channel correspond to a separate Multicast Group Address. Optionally, these RH multicast addresses MAY be the same as the Data Multicast Address.

3.1.8 Local Control Channel

A Local Control Channel is a unidirectional multicast path from a Repair Head or Sender to its children. It uses a Multicast Group Address for this communication.

3.1.9 Host ID

With the widespread deployment of network address translators, creating a short globally unique ID for a host is a challenge. By default, TRACK uses a 48 bit long Host ID field, filled with the low-order 48 bits of the MD5 signature of the DNS name of the source. A TRACK PI, to match up with the goodput-ensuring protocol that TRACK PI uses as its Data Channel Protocol, MAY redefine the length and contents of this identifier.

3.1.10 Data Session ID

A Data Session ID is a globally unique identifier for a Data Session. It may either be selected by the Data Channel Protocol (i.e. NORM) or by TRACK. By default, it is the combination of the Host ID for the Sender, combined with the 16 bit port number used for the Data Session at the Sender. This identifier is included in every TRACK message.

3.1.11 Child ID

All members in a TRACK Data Session, besides the Sender, are identified by the combination of their Host ID, and the port number with which they send IP packets to their parent.

3.1.12 Message Sequence Numbers

A Message Sequence Number is a 32 bit number in the range from 1 through 2^32 û 1, which is used to specify the sequential order of a Data message in a Data Stream. A Sender node assigns consecutive Sequence Numbers to the Data messages provided by the Sender application. By default, zero is reserved to indicate that the Data Session has not yet started. A TRACK PI MAY redefine this. Message Sequence Numbers may wrap around, and so Sequence Number arithmetic MUST be used to compare any two Sequence Numbers.

3.1.13 Data Queue

A Data Queue is a buffer, maintained by a Sender or a Repair Head, for transmission and retransmission of the Data messages provided by the Sender application. New Data messages are added to the Data Queue as they arrive from the sending application, up to a specified buffer limit. The admission rate of messages to the network is controlled by the flow and congestion control algorithms. Once a

message has been received by the Receivers of a Data Stream, it may be deleted from the buffer.

At the Sender, A TRACK PI may integrate the Data Queue with the buffer used by the Data Channel Protocol.

3.2 Basic Operation of the Protocol

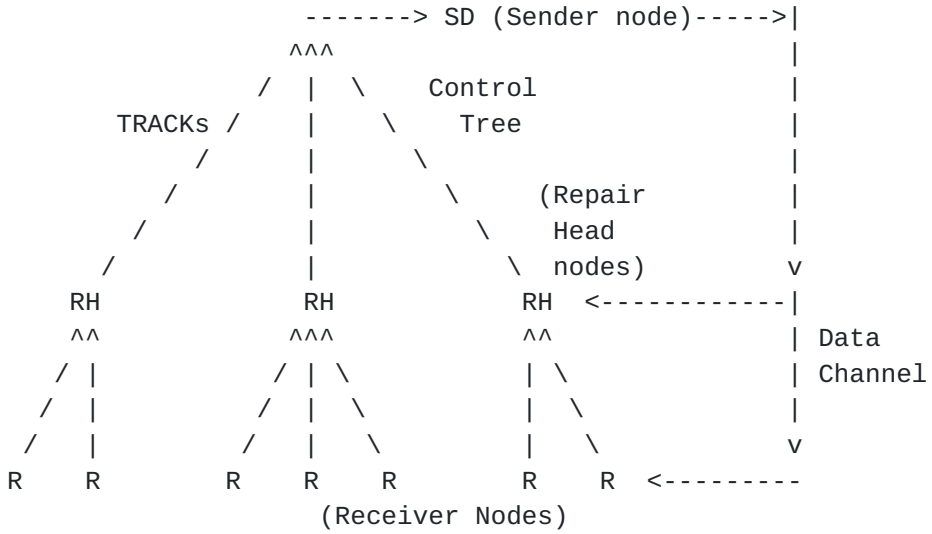
For each Data Session, TRACK provides sequenced, reliable delivery of data from a single Sender to up to tens of thousands of Receivers. A TRACK Data Session consists of a network that has exactly one Sender node, zero or more Receiver nodes and zero or more Repair Heads.

The figure below illustrates a TRACK Data Session with multiple Repair Heads.

Before a Data Session starts, a session advertisement MUST be received by all members of the Data Session, notifying them to join the group, and the appropriate configuration information for the Data Session. This MAY be provided directly by the application, by an external service, or by the TRACK PI.

A Sender joins the Control Tree and a Data Channel Protocol. It multicasts Data messages on the Data Multicast Address, using the Data Channel Protocol. All of the nodes in the session subscribe to the Data Multicast Address and join the Data Channel Protocol.

There is no assumption of congruence between the topology of the Data Multicast Address and the topology of the Control Tree.



A Receiver joins the appropriate Data Channel Protocol, and the Data Multicast Address used by that protocol, in order to receive Data. A Receiver periodically informs its parent about the messages that it

has received by unicasting a TRACK message to the parent. It MAY also request retransmission of lost messages in this TRACK. Each parent node aggregates the TRACKs from its child nodes and (if it is not the Sender) unicasts a single aggregated TRACK to its parent.

The Sender and each Repair Head have a multicast Local Control Channel to their children. This is used for transmitting Heartbeat messages that inform their child nodes that the parent node is still functioning. This channel is also used to perform local retransmission of lost Data messages to just these children. TRACK MUST still provide correct operation even if multicast addresses are reused across multiple Data Sessions or multiple Local Control Channels. It is NOT RECOMMENDED to use the same multicast address for multiple Local Control Channels serving any given Data Session.

The communication path forms a loop from the Sender to the Receivers, through the Repair Heads back to the Sender. Original data (ODATA), Retransmission (RDATA) and NullData messages regularly exercise the downward data direction. Heartbeat messages exercise the downward control direction. TRACK messages regularly exercise the Control Tree 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.

This hierarchical infrastructure allows TRACK to provide a number of functions in a scaleable way. Application level confirmation of delivery and statistics aggregation both operate in a request-reply mode. A sender issues a request for application level confirmation or statistics reporting, and the receivers report back the appropriate information in their TRACK messages. This information is aggregated by the Repair Heads, and passed back up to the Sender. Since TRACK messages are not delivered with the reliability of data messages, Receivers and Repair Heads transmit this information redundantly.

TRACK also gathers control information that is useful for improving the performance of flow and congestion control algorithms, including scaleable round trip time measurements.

Normally, goodput in ensured by lower level protocols, such as the NACKs and FEC algorithms in NORM and PGM. However, TRACKs MAY also include optional retransmission requests, in the form of selective bitmaps indicating which messages need to be retransmitted. The RH is then responsible for retransmitting these messages on the Local Control Channel to its children.

3.3 Component Relationships

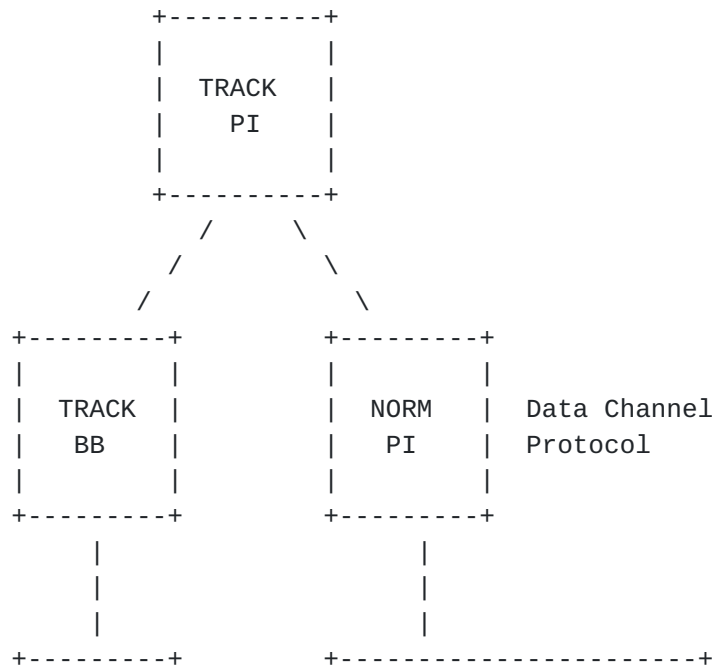
TRACK is primarily designed to run in conjunction with another transport protocol that is responsible for ensuring goodput. It is RECOMMENDED that this Data Channel Protocol also be responsible for congestion control, although the TRACK PI MAY provide this congestion control function instead, and MAY pass the congestion control statistics it collects to the Data Channel Protocol, in order to enhance the performance of the congestion control algorithms.

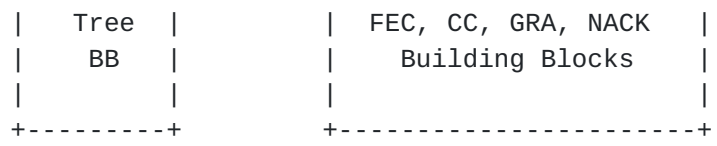
The primary Data Channel Protocol that TRACK is designed to work with is NORM. In this case, the NORM PI is responsible for interfacing with the NACK BB, the FEC BB, the Generic Router Assist BB, and the appropriate congestion control BB.

TRACK then adds additional functionality that complements this receiver-reliable protocol, such as application level confirmed delivery, retransmission in the face of persistent failures, statistics aggregation, and collection of extra information for congestion control.

The TRACK BB is responsible for specifying all of the TRACK-specific functionality. It interfaces with the Automatic Tree Building Block. The TRACK PI is then responsible for instantiating a complete protocol that includes all of the other components. It is expected that there will be multiple TRACK PIs, one for each Data Channel Protocol that it is specified to work with.

The following figure illustrates this, for the case where NORM is the Data Channel Protocol.





For more details on integration, please see the example TRACK PI over UDP [17].

4. TRACK Functionality

4.1 Hierarchical Session Creation and Maintenance

4.1.1 Overview of Tree Configuration

Before a Data Session starts reliably delivering data, the tree for the Data Session needs to be created. This process binds each Receiver to either a Repair Head or the Sender, and binds the participating Repair Heads into a loop-free tree structure with the Sender as the root of the tree. This process requires tree configuration knowledge, which can be provided with some combination of manual and/or automatic configuration. The algorithms for automatic tree configuration are part of the Automatic Tree Configuration BB. They return to each node the address of the parent it should bind to, as well as zero or more backup parents to use if the primary parent fails.

In addition to receiving the tree configuration information, the Receivers all receive a Session Advertisement message from the Senders, informing them of the Data Multicast Address and other session configuration information. This advertisement may contain other relevant session information such as whether or not Repair Heads should be used, whether manual or automatic tree configuration should be used, the time at which the session will start, and other protocol settings. This advertisement is created as part of either the TRACK PI or as part of an external service. In this way, the Sender enforces a set of uniform session configuration parameters on all members of the session.

As described in the automatic tree configuration BB, the general algorithm for a given node in tree creation is as follows.

1) Get advertisement that a session is starting
2) Get a list of neighbor candidates using the getSNs Tree BB interface, and OPTIONALLY contact them
3) Select best neighbor as parent in a loop free manner
4) Bind to parent
5) Optionally, later rebind to another parent

When a child finishes step 4, it is up to automatic tree configuration to, if necessary, continue building the tree in order to connect the node back to the Sender. After the session is created, children can unbind from their parents and bind again to new parents. This happens when faults occur, or as part of a tree optimization process. Steps 1 through 3 are external to the TRACK BB. Step 4 is performed as part of session creation. Step 5 is performed as part of session maintenance in conjunction with automatic tree building, as either an Unbind or Eject, combined with another Bind operation.

Once steps 1 through 3 are completed, Receivers join the Data Multicast Address, and attempt to Bind to either the Sender or a local Repair Head. A Receiver will attempt to bind to the first node in the tree configuration list returned by step 3, and if this fails, it will move to the next one. A Receiver only binds to a single Repair Head or Sender, at a time, for each Data Session.

The automatic tree building BB ensures that the tree is formed without loops. As part of this, when a Repair Head has a Receiver attempt to bBnd to it for a given Data Session, it may not at first be able to accept the connection, until it is able to join the tree itself. Because of this, a Receiver will sometimes have to repeatedly attempt to Bind to a given parent before succeeding.

Once the Sender initiates tree building, it is also free to start sending Data messages on the Data Multicast Address. Repair Heads and Receivers may start receiving these messages, but may not request retransmission or deliver data to the application until they receive confirmation that they have successfully bound to the tree.

4.1.2 Bind

4.1.2.1 Input Parameters

In order to join a Data Session and Bind to the tree, the following nodes need the following parameters.

A Repair Head requires the following parameters.

- Session: the unique identifier for the Data Session to join, received from the session advertisement algorithm specified in the PI.

- ParentAddress: the address and port of the parent node to which the node should connect, received from the Auto Tree BB.

- UDPListenPort: the number of the port on which the node will listen for its childrens control messages. This parameter is configured by the application.

- RepairAddr: the multicast address, UDP port, and TTL on which this node sends control messages to its children. This parameter is configured by the application.

A Sender requires the above parameters, except for the ParentAddress. A Receiver requires the above parameters, except for the UDPListenPort and RepairAddr.

4.1.2.2 Bind Algorithm

A Bind operation happens when a child wishes to join a parent in the distribution tree for a given Data Session. The Receivers initiate the first Bind protocols to their parents, which then cause recursive binding by each parent, up to the Sender. Each Receiver sends a separate BindRequest message for each of the streams that it would like to join. At the discretion of the PI, multiple BindRequest messages may be bundled together in a single message.

A node sends a BindRequest message to its automatically selected or manually configured parent node. The parent node sends either a BindConfirm message or a BindReject message. Reception of a BindConfirm message terminates the algorithm successfully, while receipt of a BindReject message causes the node to either retry the same parent or restart the Bind algorithm with its next parent candidate (depending on the BindReject reason code), or if it has none, to declare a REJECTED_BY_PARENT error. Once the node is accepted by a Repair head, it informs the Tree BB using the setSN interface.

Reliability is achieved through the use of a standard request-response protocol. At the beginning of the algorithm, the child initializes TimeMaxBindResponse to the constant TIMEOUT_PARENT_RESPONSE and initializes NumBindResponseFailures to 0. Every time it sends a BindRequest message, it waits TimeMaxBindResponse for a response from the parent node. If no response is received, the node doubles its value for TimeMaxBindResponse, but limits TimeMaxBindResponse to be no larger than MAX_TIMEOUT_PARENT_RESPONSE. It also increments NumBindResponseFailures, and retransmits the BindRequest message. If NumBindResponseFailures reaches NUM_MAX_PARENT_ATTEMPTS, it reports a PARENT_UNREACHABLE error.

When a parent receives a BindRequest message, it first consults the automatic tree building BB for approval (using the acceptChild Tree

BB interface), for instance to ensure that accepting the BindRequest

will not cause a loop in the tree. Then the parent checks to be sure that it does not have more than MaxChildren children already bound to it for this session. If it can accept the child, it sends back a BindConfirm message. Otherwise, it sends the node a BindReject message. Then the parent checks to see if it is already a member of this Data Session. If it is not yet a member of this session, it attempts to join the tree itself.

The BindConfirm message contains the lowest Sequence Number that the Repair Head has available. If this number is 0, then the Repair Head has all of the data available from the start of the session. Otherwise, the requesting node is attempting a late join, and can only use this Repair Head if late join was allowed by the PI. If late join is not allowed, the node may try another Repair Head, or give up.

Similarly, if a failure recovery occurs, when a node tries to bind to a new Repair Head, it must follow the same rules as for a late join. See Fault Recovery, below.

4.1.3 Unbind

A child may decide to leave a Data Session for the following reasons. 1) It detects that the Data Session is finished. 2) The application requests to leave the Data Session. 3) It is not able to keep up with the data rate of the Data Session. When any of these conditions occurs, it initiates an Unbind process.

An Unbind is, like the Bind function, a simple request-reply protocol. Unlike the Bind function, it only has a single response, UnbindConfirm. With this exception, the Unbind operation uses the same state variables and reliability algorithms as the Bind function.

When a child receives an UnbindConfirm message from its parent, it reports a LEFT_DATA_SESSION_GRACEFULLY event. If it does not receive this message after NUM_MAX_PARENT_ATTEMPTS, then it reports a LEFT_DATA_SESSION_ABNORMALLY event. Unbinds are reported to the Tree BB using the lostSN interface.

4.1.4 Eject

A parent may decide to remove one or more of its children from a data stream for the following reasons. 1) The parent needs to leave the group due to application reasons. 2) The Repair Head detects an unrecoverable failure with either its parent or the Sender. 3) The parent detects that the child is not able to keep up with the speed of the data stream. 4) The parent is not able to handle the load of its children and needs some of them to move to another parent. In the first two cases, the parent needs to multicast the advertisement

of the termination of one or more Data Sessions to all of its children. In the second two cases, it needs to send one or more unicast notifications to one or more of its children.

Consequently, an Eject can be done either with a repeated multicast advertisement message to all children, or a set of unicast request-reply messages to the subset of children that it needs to go to.

For the multicast version of Eject, the parent sends a multicast UnbindRequest message to all of its children for a given Data Session, on its Local Multicast Channel. It is only necessary to provide statistical reliability on this message, since children will detect the parents failure even if the message is not received. Therefore, the UnbindRequest message is sent FAILURE_DETECTION_REDUNDANCY times.

For the unicast version of Eject, the parent sends a unicast UnbindRequest message to all of its children. Each of them responds with an EjectConfirm. Reliability is ensured through the same request-reply mechanism as the Bind operation.

Ejections are reported to the Tree BB using the removeChild interface.

4.1.5 Fault Detection

There are three cases where fault detection is needed. 1) Detection (by a child) that a parent has failed. 2) Detection (by a parent) that a child has failed. 3) Detection (by either a Repair Head or Receiver) that a Sender has failed.

In order to be scaleable and efficient, fault detection is primarily accomplished by periodic keep-alive messages, combined with the existing TRACK messages. Nodes expect to see keep-alive messages every set period of time. If more than a fixed number of periods go by, and no keep-alive messages of a given type are received, the node declares a preliminary failure. The detecting node may then ping the potentially failed node before declaring it failed, or it can just declare it failed.

Failures are detected through three keep-alive messages: Heartbeat, TRACK, and NullData. The Heartbeat message is multicast periodically from a parent to its children on its Local Control Channel. NullData messages are multicast by a Sender on the Data Control Channel when it has no data to send. TRACK messages are generated periodically, even if no data is being sent to a Data Session, as described in section 7.2.

Heartbeat messages are multicast every HeartbeatPeriod seconds, from a parent to its children. Every time that a parent sends a Retransmission message or a Heartbeat message (as well as at initialization time), it resets a timer for HeartbeatPeriod seconds. If the timer goes off, a Heartbeat is sent. The HeatbeatPeriod is dynamically computed as follows:

interval = AckWindow / MessageRate

HeartbeatPeriod = 2 * interval

Global configuration parameters ConstantHeartbeatPeriod and MinimumHeartbeatPeriod can be used to either set HeartbeatPeriod to a constant, or give HeartbeatPeriod a lower bound, globally.

Similarly, a NullData message is multicast by the Sender to all Data Session members, every NULL_DATA_PERIOD. The NullData timer is set to NULL_DATA_PERIOD, and is reset every time that a Data or NullData message is sent by the Sender.

The key parameter for failure detection is the global tree parameter FAILURE_DETECTION_REDUNDANCY. The higher the value for this parameter, the more keep-alive messages that must be missed before a failure is declared.

A major goal of failure detection is for children to detect parent failures fast enough that there is a high probability they can rejoin the stream at another parent, before flow control has advanced the buffer window to a point where the child can not recover all lost messages in the stream. In order to attempt to do this, children detect a failure of a parent if FAILURE_DETECTION_REDUNDANCY * HeartbeatPeriod time goes by without any heartbeats. As part of buffer window advancement, all parents MAY choose to buffer all messages for a minimum of FAILURE_DETECTION_REDUNDANCY * 2 * HeartbeatPeriod seconds, which gives children a period of time to find a new parent before the buffers are freed. Children report parent failures to the Tree BB using the lostSN interface.

A parent detects a preliminary failure of one of its children if it does not receive any TRACK messages from that child in FAILURE_DETECTION_REDUNDANCY * TrackTimeout seconds (see discussion of how TrackTimeout is computed below). Because a failed child can slow down the groups progress, it is very important that a parent resolve the childs status quickly. Once a parent declares a preliminary failure of a child, it issues a set of up to FAILURE_DETECTION_REDUNDANCY Heartbeat messages that are unicast (or multicast) to the failed Receiver(s). These messages are spaced apart by 2*LocalRTT, where LocalRTT is the round trip time that has

been measured to the child in question (see below for description of

how LocalRTT is measured). These Heartbeat messages contain a ChildrenList field that contains the children who are requested to send a TRACK immediately.

Whenever a child receives a Heartbeat message where the child is identified in the ChildrenList field, it immediately sends a TRACK to its parent. If a parent does not receive a TRACK message from a child after waiting a period of 2*LocalRTT after the last Heartbeat message to that child, it declares the child failed, and removes it from the parents child membership list. It informs the Tree BB using the removeChild interface.

A child or a Repair Head detects the failure of a Sender if it does not receive a Data or NullData message from a Sender in FAILURE_DETECTION_REDUNDANCY * NULL_DATA_PERIOD.

Note that the more Receivers there are in a tree, and the higher the loss rate, the larger FAILURE_DETECTION_REDUNDANCY must be, in order to give the same probability that erroneous failures wont be declared.

4.1.6 Fault Notification

When a parent detects the failure of a child, it adds a failure notification field to the next TRANSMISSION_REDUNDANCY TRACK messages that it sends up the tree. It sends this notification multiple times because TRACKs are not delivered reliably. A failure notification field includes the failure code, as well as a list of one or more failed nodes. Failure notifications are aggregated up the tree and delivered to the Sender. A failure notification is not a definitive report of a node failure, as the child may have detected a communication failure with its parent and moved to a different Repair Head.

4.1.7 Fault Recovery

The Fault Recovery algorithms require a list of one or more addresses of alternate parents that can be bound to, and that still provide loop free operation.

If a child detects the failure of its parent, it then re-runs the Bind operation to a new parent candidate, in order to rejoin the tree. A node may perform a late join, i.e. binding with a Repair Head which cannot provide all the necessary repair data, only if allowed by the PI.

4.1.8 Distributed Membership.

Each Repair Head is responsible for maintaining a set of state variables on the status of its children. Unlike the Generic Router Assist, this is hard state, that only is removed when a child leaves that Repair Head gracefully, or after the Repair Head detects that a child has failed. These variables MUST include, but are not necessarily limited to, the following:

- ChildID. This is the two byte identifier assigned to the Child by the Repair Head. This uniquely identifies this Child to this Repair Head, but has no meaning outside that scope.
- GlobalChildIdentifier. This is the globally unique identifier for this Child.
- ChildRTT. This is the weighted average of the local RTT to this Child.
- LastTRACK. This is the contents of the last TRACK message sent from this Child, if any, not including options.
- LastApplicationLevelConfirmation. This is the contents of the last Application Level Confirmation sent from this Child, if any.
- Last Statistics. This is the contents of the last Statistics message sent from this Child, if any.
- ChildLiveness. This is a set of variables that keep track of the liveness of each child. This includes the last time a TRACK message was received from this child, as well as the number of Heartbeat messages that have been directed at it, and the time at which the last Heartbeat message was sent to the child. Please see Fault Detection, above, for more details.

4.2 Data Sessions.

4.2.1 Data Transmission and Retransmission

Data is multicast by a Sender on the Data Multicast Address via the Data Channel Protocol. The Data Channel Protocol is responsible for taking care of as many retransmissions as possible, and for ensuring the goodput of the Data Session. TRACK is then responsible for providing OPTIONAL flow control and application level reliability. The mechanics of an application level confirmation of delivery are handled by TRACK, including keeping track of the distributed membership list of receivers and aggregating acknowledgements up the Control Tree. Please see below for more details on flow control and application level confirmation.

A common scenario for handling recovery of lost messages is to allow the Data Channel Protocol to provide statistical reliability, and then allow TRACK to provide retransmissions for more persistent failure cases, such as if a Receiver is not able to receive any Data messages for a few minutes.

Retransmissions of data messages may be multicast by the Sender on the Data Multicast Address or be multicast on a Local Control Channel by a Repair Head.

A Repair Head joins all of the Data Multicast Addresses that any of its descendants have joined. A Repair Head is responsible for receiving and buffering all data messages using the reliability semantics configured for a stream. As a simple to implement option, a Repair Head MAY also function as a Receiver, and pass these data messages to an attached application.

For additional fault tolerance, a Receiver MAY subscribe to the multicast address associated with the Local Control Channel of one or more Repair Heads in addition to the multicast address of its parent. In this case it does not bind to this Repair Head or Sender, but will process Retransmission messages sent to this address. If the Receivers Repair Head fails and it transfers to another Repair Head, this minimizes the number of data messages it needs to recover after binding to the new Repair Head.

4.2.2 Local Retransmission

If a Repair Head or Sender determines from its child nodes TRACK messages that a Data message was missed, the Repair Head retransmits the Data message. The Repair Head or Sender multicasts the Retransmission message on its multicast Local Control Channel. In the event that a Repair Head receives a retransmission and knows that its children need this repair, it re-multicasts the retransmission to its children.

The scope of retransmission (the multicast TTL) is considered part of the Control Channels multicast address, and is derived during tree configuration.

A Repair Head maintains the following state for each of its children, for the purpose of providing repair service to the local group:

- HighestConsecutivelyReceived. A Sequence Number indicating all Data messages up to this number (inclusive) that have been received by a given child.

- MissingMessages. A data structure to keep track of the reception status of the Data messages with Sequence Number higher than HighestConsecutivelyReceived.

The minimum HighestConsecutivelyReceived value of all its children is kept as the variable LocalStable.

A Repair Head also maintains a retransmission buffer. The size of the retransmission buffer MUST be greater than the maximum value of a Senders transmission window. The retransmission buffer MUST keep all the Data messages received by the Repair Head with Sequence Number higher than LocalStable, optionally some messages with Sequence Number lower than LocalStable if there is room (beyond the maximum value of Senders transmission window). The latter messages are kept in the retransmission buffer in case a Receiver from another group losses its parent and needs to join this group.

As TRACK messages are received, the Repair Head updates the above state variables.

To perform local repair, a Repair Head implements a retransmission queue with memory. Each lost message is entered into the retransmission queue in increasing order according to its Sequence Number. If the same Data message has already been retransmitted recently (recognized due to the queues memory) it is delayed by the local group RTT (see roundtrip time measurement) before retransmission.

Retransmissions MAY NOT be sent at a faster rate than the current TransmissionRate advertised by the Sender.

4.2.3 Flow and Rate Control

TRACK offers the ability to limit the rate of Data traffic, through both flow control and rate limits.

When a Receiver sends a TRACK to its parent, the HighestAllowed field provides information on the status of the Receivers flow control window. The value of HighestAllowed is computed as follows:

HighestAllowed = seqnum + ReceiverWindow

Where seqnum is the highest Sequence Number of consecutively received data messages at the Receiver. The size of the ReceiverWindow may either be based on a parameter local to the Receiver or be a global parameter.

If flow control is enabled for a given Data Session, then a Sender MUST NOT send any Data messages to the Data Channel Protocol that are higher than the current value for HighestAllowed that it has. On startup, HighestAllowed is initialized to ReceiverWindow.

In addition, the Sender application MAY provide minimum and maximum rate limits. Unless overridden by the Data Channel Protocol, a Sender will not offer Data messages to the Data Channel Protocol at lower than MinimumDataRate (except possibly during short periods of

time when certain slow Receivers are being ejected), or higher than MaximumDataRate. If a Receiver is not able to keep up with the minimum rate for a period of time, it SHOULD leave the group promptly. Receivers that leave the group MAY attempt to rejoin the group at a later time, but SHOULD NOT attempt an immediate reconnection.

4.2.4 Reliability Window

The Sender and each Repair Head maintain a window of messages for possible retransmission. As messages are acknowledged by all of its children, they are released from the parents retransmission buffer, as described in 4.2.2. In addition, there are two global parameters that can affect when a parent releases a data message from the retransmission buffer -- MinHoldTime, and MaxHoldTime.

MinHoldTime specifies a minimum length of time a message must be held for retransmission from when it was received. This parameter is useful to handle scenarios where one or more children have been disconnected from their parent, and have to reconnect to another. If, for example, MinHoldTime is set to FAILURE_DETECTION_REDUNDANCY * 2 * ConstantHeartbeatPeriod, then there is a high likelihood that any child will be able to recover any lost messages after reconnecting to another parent.

The Sender continually advertises to the members of the Data Session both edges of its retransmission window. The higher value is the SeqNum field in each Data or NullData message, which specifies the highest Sequence Number of any data message sent. The trailing edge of the window is advertised in the HighestReleased field. This specifies the largest Sequence Number of any message sent that has subsequently been released from the Senders retransmission window. If both values are the same then the window is presently empty. Zero is not a legitimate value for a data Sequence Number, so if either field has a value of zero, then no messages have yet reached that state. All Sequence Number fields use Sequence Number arithmetic so that a Data Session can continue after exhausting the Sequence Number space.

When a member of a Data Session receives an advertisement of a new HighestReleased value, it stores this, and is no longer allowed to ask for retransmission for any messages up to and including the HighestReleased value. If it has any outstanding missing messages that are less than or equal to HighestReleased, it MAY move forward and continue delivering the next data messages in the stream. It also SHOULD report an error for the messages that are no longer recoverable.

MaxHoldTime specifies the maximum length of time a message may be held for retransmission. This parameter is set at the Sender which uses it to set the HighestReleased field in data message headers. This is particularly useful for real-time, semi-reliable streams such as live video, where retransmissions are only useful for up to a few seconds. When combined with Unordered delivery semantics, and application-level jitter control at the Receivers, this provides Time Bounded Reliability. MaxHoldTime MUST always be larger than MinHoldTime.

4.2.5 Ordering Semantics

TRACK offers two flavors of ordering semantics: Ordered or Unordered. One of these is selected on a per session basis as part of the Session Configuration Parameters.

Unordered service provides a reliable stream of messages, without duplicates, and delivers them to the application in the order received. This allows the lowest latency delivery for time sensitive applications. It may also be used by applications that wish to provide its own jitter control.

Ordered service provides TCP semantics on delivery. All messages are delivered in the order sent, without duplicates.

4.2.6 Retransmission Requests.

A Receiver detects that it has missed one or more Data messages by gaps in the sequence numbers of received messages. Each Receiver keeps track of HighestSequenceNumber, the highest sequence number known of for a Data Session, as observed from Data, RData, and NullData messages. Any sequence numbers between HighestReleased and HighestSequenceNumber that have not been received are assumed to be missing.

When a Receiver detects missing messages it MAY send off a request for retransmission, if local retransmission is enabled. It does this by sending a Retransmission Request message. The timing of this request is described below.

4.2.7 End Of Stream.

When an application signals that a Data Session is complete, the Sender advertises this to its children by setting the End of Session option on the last Data Message in the Data Session, as well as all subsequent retransmissions of that Data Message, and all subsequent Null Data messages.

The Sender SHOULD NOT leave the Data Session until it has a report from the TRACK reports that all group members have left the Data Session, or it has waited a period of at least FAILURE_DETECTION_REDUNDANCY * TrackTimeout seconds.

4.3 Control Traffic Generation and Aggregation.

One of the largest challenges for scaleable reliable multicast protocols has been that of controlling the potential explosion of control traffic. There is a fundamental tradeoff between the latency with which losses can be detected and repaired, and the amount of control traffic generated by the protocol.

TRACK messages are the primary form of control traffic in this BB. They are sent from Receivers and Repair Heads to their parents. TRACK messages may be sent for the following purposes:
- to request retransmission of messages
- to advance the Senders transmission window for flow control purposes
- to deliver application level confirmation of data reception
- to propagate other relevant feedback information up through the session (such as RTT and loss reports, for congestion control)

4.3.1 TRACK Generation with the Rotating TRACK Algorithm

Each Receiver sends a TRACK message to its parent once per AckWindow of data messages received. A Receiver uses an offset from the boundary of each AckWindow to send its TRACK, in order to reduce burstiness of control traffic at the parents. Each parent has a maximum number of children, MaxChildren. When a child binds to the parent, the parent assigns a locally unique ChildID to that child, between 0 and MaxChildren-1.

Each child in a tree generates a TRACK message at least once every AckWindow of data messages, when the most recent data messages Sequence Number, modulo AckWindow, is equal to MemberID. If the message that would have triggered a given TRACK for a given node is missed, the node will generate the TRACK as soon as it learns that it has missed the message, typically through receipt of a higher numbered data message.

Together, AckWindow and MaxChildren determine the maximum ratio of control messages to data messages seen by each parent, given a constant load of data messages.

In each data message, the Sender advertises the current MessageRate (measured in messages per second) it is sending data at. This rate is generated by the congestion control algorithms in use at the Sender.

At the time a node sends a regular TRACK, it also computes a TRACKTimeout value:

interval = AckWindow / MessageRate

TRACKTimeout = 2 * interval

If no TRACKs are sent within TRACKTimeout interval, a TRACK is generated, and TRACKTimeout is increased by a factor of 2, up to a value of MAX_TRACK_TIMEOUT.

This timer mechanism is used by a Receiver to ensure timely repair of lost messages and regular feedback propagation up the tree even when the Sender is not sending data continuously. This mechanism complements the AckWindow-based regular TRACK generation mechanism.

4.3.2 TRACK Aggregation.

There are many reasons for providing feedback from all the Receivers to the Sender in an aggregated form. The major ones are listed below:

1) End-to-end delivery confirmation. This confirmation tells the Sender that all the Receivers (in the entire tree) have received data messages up to a certain Sequence Number. This is carried in an Application Level Confirmation message.

2) Flow control. The aggregated information is carried in the field HighestAllowed. It tells the Sender the highest Sequence Number that all the Receivers (in the entire tree) are prepared to receive.

3) Congestion control feedback. Information about the state of the tree can be passed up to help control the congestion control algorithms for the group.

4) Counting current membership in the group. This information is carried in the field SubTreeCount. This lets the Sender know the number of Receivers currently connected to the repair tree.

5) Measuring the round-trip time from the Sender to the "worst" Receiver.

A Repair Head maintains state for each child. Each time a TRACK (from a child) is received, the corresponding states for that child are updated based on the information in the TRACK message. When a Repair Head sends a TRACK message to its parent, the following fields of its TRACK message are derived from the aggregation of the

corresponding states for its children. The following rules describe how the aggregation is performed:

- WorstLossRate. Take the maximum value of the WorstLossRate from all Children.
- SubTreeCount. Take the sum of the SubTreeCount from all Children.
- HighestAllowed. Take the minimum of the HighestAllowed value from all children.
- WorstEdgeThroughput. Take the minimum value of the WorstEdgeThroughput field from all Children.
- UnicastCost. Take the sum of the UnicastCost from all Children.
- MulticastCost. Take the sum of the MulticastCost from all Children.
- SenderDallyTime: take the minimum value, for all of the children, of (childs reported SenderDallyTime + childs local dally time).
- FailureCount: take the sum of the FailureCount for all Children.
- FailureList: concatenate the FailureList fields for all Children, up to a maximum list size of MaxFailureListSize.

Note, the SenderTimeStamp, ParentTimestamp, and ParentDallyTime fields are not aggregated. The Sender will derive the roundtrip time to the worst Receiver by doing its local aggregation for SenderDallyTime and then compute:

RTT = currentTime û SenderTimeStamp û SenderDallyTime.

Application level confirmations (ALCs) are handled as follows. For a set of ALC requests from receivers, the ones with the highest value for HighConfirmationSequenceNumber are considered, and all others are discarded.

For the ConfirmationStatus field, the following rules apply. Note that ConfirmationStatus of SomeReceiversAcknowledge can correspond to a ConfirmationCount of zero.

```
If all children report AllReceiversAcknowledge Then
        ConfirmationStatus = AllReceiversAcknowlege
Else If at least one child reports (ListOfFailures OR
  FailuresExceedMaximumListSize) Then
        If the count of all reported failures >
          MaximumFailureListSize Then
             ConfimationStatus = FailuresExceedMaximumListSize
        Else
             ConfirmationStatus = ListOfFailures
Else
        ConfirmationStatus = SomeReceiversAcknowledge
```

The ConfirmationCount field is equal to the sum of the ConfimationCount for the aggregated ALC reports of all Children. The PendingCount field is equal to the sum of the PendingCount fields of

all Children. The FailureList field is the concatenation of the

FailureList fields of all aggregated ALC reports of all children, up to a maximum length of MaximumFailureListSize.

In addition to these fields with fixed aggregation rules, TRACK supports a set of user defined aggregation statistics. These statistics are self describing in terms of their data type and aggregation method. Statistics reports are numbered, and only the most recent statistics report request is aggregated to the Sender. Statistics are aggregated over the set of Child statistics reports that have been received with that number. Aggregation methods include minimum, maximum, sum, product, and concatenation.

4.3.3 Statistics Reporting.

A Sender can request a list of aggregated statistics from all Receivers in the group. There are a set of predefined statistics, such as loss rate and average throughput. There is also the capacity to request a set of other TRACK statistics, as well as application defined statistics.

The format of each statistic is self-describing, both in terms of data type, size, and aggregation method. A Sender reliably sends out a statistics request by attaching it as an option to a Data message. When a Receiver gets a request for a statistic, it fills in the data fields, and forwards it up the tree in the next TRACK message. Since TRACKs are not reliable, multiple copies are sent in a total of NumReplies consecutive TRACK messages from each Receiver. Each statistics report is aggregated according to the method described in the statistic, and the result is delivered to the Sender.

Most aggregation options have fixed length no matter how many Receivers there are. The one exception is concatenation, which creates a list of values from some or all Receivers, up to a length of MaximumStatisticsListSize entries. It is NOT RECOMMENDED to use this to create group-wide lists, unless the groups size is carefully controlled.

4.4 Application Level Confirmed Delivery.

Flow control and the reliability window are concerned with goodput, of delivering data with a high probability that it is delivered at all Receivers. However, neither mechanism provides explicit confirmation to the Sender as to the list of recipients for each message. Application level confirmed delivery allows applications to determine the set of applications that have received a set of data messages.

There are three primary factors that determine the reliability semantics of a message: the senders knowledge of the Receiver list,

the application level actions that must be performed in order to consider a message delivered, and the response to persistent failure conditions at Receivers. For example, an extremely strong distributed guarantee would consist of the following. First, the full Receiver membership list is known at the Sender, and verified to make sure no Receivers have left the group. Second, the application at each Receiver must write the Data to persistent store before it can be acknowledged. Third, Receivers are given a very long period of time - say one hour û to recover all lost Data messages, before they are ejected from the Data Session. In the meantime, transmission of Data messages is flow controlled by the slowest receivers.

A weaker form of reliability would include the following. First, that the Sender gets a count of Receivers, and otherwise depends on the distributed group membership algorithms to maintain the membership list. Second, that Data messages are considered reliably delivered as soon as the application receives the Data from TRACK. Third, that Retransmissions are limited to only 30 seconds, and Receivers must choose to leave the Data Session or continue with missing Data messages, if a failure takes longer than this period to recover from.

TRACK provides the functionality to easily implement a wide range of application level confirmation semantics, based on how these three items are configured. It is the applications responsibility to then select the configurations it desires for a given Data Session.

4.4.1 Application Level Confirmation Mechanisms

The primary mechanism for application level confirmation (ALC) of delivery is the ALC report. To check for ALC of delivery, a Sender issues a Application Level Confirmation Request, by attaching this message as an option to a Data message, and reliably transmitting it to all Receivers. Each ALC Request includes a specified level of reliability, a reply redundancy factor, and the range of Data message sequence numbers that the ALC Confirmation covers.

When a Receiver gets an ALC Request, it checks to see if the application has delivered the specified range of Data Messages, including both the Low Confirmation Sequence Number and the High Confirmation Sequence Number. When it sends the next TRACK out, it sets the ConfirmationStatus field to either SomeReceiversAcknowledge if it is still pending confirmation, AllReceiversAcknowledge if it has application level confirmation, ListOfFailures if it has a failure and MaximumFailureListSize > 0, or FailuresExceedsMaximumListSize otherwise. It also sets the ConfirmCount to 1 if it has a confirmation, and PendingCount to 1 if

it is still pending. If the Immediate ACK bit is set in the ALC Request, the Receiver generates an ACK immediately.

One example of how an application can implicitly signal confirmation of delivery is through the freeing of buffers passed to it by the transport. The API could specify that whenever an application has freed up a buffer containing one or more data messages, then these messages are considered acknowledged by the application. Alternatively, the application could be required to explicitly acknowledge each message.

4.5 Distributed RTT Calculations.

This TRACK BB provides two algorithms for distributed RTT calculations ù LocalRTT measurements and SenderRTT measurements. LocalRTT measurements are only between a parent and its children. SenderRTT measurements are end-to-end RTT measurements, measuring the RTT to the worst Receiver as selected by the congestion control algorithms.

The SenderRTT is useful for congestion control. It can be used to set the data rate based on the TCP response function, which is being proposed for the congestion control building blocks.

The LocalRTT can be used to (a) quickly detect faulty children (as described under fault detection) or (b) avoid sending unnecessary retransmissions (as described in the local repair algorithm).

In the case of LocalRTT measurements, a parent initiates measurement by including a ParentTimestamp field in a Heartbeat message sent to its children. When a child receives a Heartbeat message with this field set, it notes the time of receipt using its local system clock, and stores this with the message as HeartbeatReceiveTime. When the child next generates a TRACK, just before sending it, it measures its system clock again as TRACKSendTime, and calculates the LocalDallyTime.

LocalDallyTime = TRACKSendTime û HeartbeatReceiveTime.

The child includes this value, along with the ParentTimestamp field, as fields in the next TRACK message sent. Every heartbeat message that is multicast to all children SHOULD include a ParentTimestamp field.

The SenderRTT algorithm is similar. A Sender initiates the process by including a SenderTimestamp field in a data message. When a Receiver gets a message with this field set, it keeps track of the DataReceiveTime for that message, and when it generates the next TRACK message, includes the SenderTimestamp and SenderDallyTime

value. These values are aggregated by Repair Heads, as described above.

Each node only keeps track of the most recent value for {SenderTimestamp, DataReceiveTime} and {ParentTimestamp, HeartbeatReceiveTime}, replacing any older values any time that a new message is received with these values set. As long as it has non-zero values to report, each node sends up both a {SenderTimestamp, SenderDallyTime} and a {ParentTimestamp, LocalDallyTime} set of fields in each TRACK message generated.

Unless redefined by the TRACK PI, these RTT measurements are averaged using an exponentially weighted moving average, where the first RTT measurement, RTT_measurement, initializes the average RTT_average, and then each successive measurement is averaged in according to the following formula. The RECOMMENDED value for alpha is 1/8.

RTT_average = RTT_measurement * alpha + RTT_average (1-alpha)

4.6 SNMP Support

The Repair Heads and the Sender are designed to interact with SNMP management tools. This allows network managers to easily monitor and control the sessions being transmitted. All TRACK nodes MAY have SNMP MIBs defined in a separate document. SNMP support is OPTIONAL for Receiver nodes, but is RECOMMENDED for all other nodes.

4.7 Late Join Semantics

TRACK offers three flavors of late join support:

a) No Recovery
   A Receiver binds to a Repair Head after the session has started and agrees to the reliability service starting from the Sequence Number in the current data message received from the Sender.

b) Continuation
   This semantic is used when a Receiver has lost its Repair Head and needs to re-affiliate. In this case, the Receiver must indicate the oldest Sequence Number it needs to repair in order to continue the reliability service it had from the previous Repair Head. The binding occurs if this is possible.

c) No Late Join
   For some applications, it is important that a Receiver receives either all data or no data (e.g. software distribution). In this case option (c) is used.

These are specified by the LateJoinSemantics session parameter, and enforced by a Parent when a Child attempts to bind to it.

5. Message Types

The following table summarizes the messages and their fields used by the TRACK BB. All messages contain the session identifier. For more details, please see the sample TRACK PI [17].

Message	From	To	Mcast?	Fields
BindRequest	Child	Parent	no	Scope, Level, Role, Rejoin BindSequenceNumber, SubTreeCount
BindConfirm	Parent	Child	no	RepairAddr, BindSequenceNumber LowestAvailableRepair Level, ChildIndex, Role
BindReject	Parent	Child	no	Reason, BindSequenceNumber
UnbindRequest	Child	Parent	no	Reason, ChildIndex
UnbindConfirm	Parent	Child	no	
EjectRequest	Parent	Child	either	Reason, AlternateParent
EjectConfirm	Child	Parent	no	
Heartbeat	Parent	Child	either	Level, ParentTimestamp ChildrenList, SeqNum HighestReleased
NullData, OData	Sender	all	yes	SenderTimeStamp, DataLength HighestReleased, SeqNum EndOfStream, TransmissionRate
Rdata	Parent	Child	yes	SenderTimeStamp, DataLength HighestReleased, SeqNum EndOfStream, TransmissionRate
Track	Child	Parent	no	BitMask, SubTreeCount Slowest, HighestAllowed

				ParentThere, ParentTimeStamp ParentDallyTime, SenderTimeStamp SenderDallyTime, CongestionControl FailureList
ALCRequest	Sender	Receiver	yes	Immediate, Reliability NumReplies, SeqNumRange
ALCReply	Child	Parent	yes	SeqNumRange, ConfirmStatus ConfirmCount, PendingCount FailedChildren
StatsRequest	Sender	Receiver	yes	Immediate, StatsSeqNum NumReplies, StatsList
StatsReply	Child	Parent	yes	StatsSeqNum, StatsList

The various fields of the messages are described as follows:

- BindSequenceNumber: This is a monotonically increasing sequence number for each bind request from a given Receiver for a given Data Session.

- Scope: an integer to indicate how far a repair message travels. This is optional.

- Rejoin: a flag as to whether this Receiver was previously a member of this Data Session.

- Level: an integer that indicates the level in the repair tree. This value is used to keep loops in the tree from forming, in addition to indicating the distance from the Sender. Any changes in a nodes level are passed down to the Tree BB using the treeLevelUpdate interface.

- Role: This indicates if the bind requestor is a Receiver or Repair Head.

- SubTreeCount: This is an integer indicating the current number of Receivers below the node.

- RepairAddr: This field in the BindConfirm message is used to tell the Receiver which multicast address the Repair Head will be sending retransmissions on. If this field is null, then the Receiver should expect retransmissions to be sent on the Senders data multicast address.

- AlternateParent: This is an optional field that specifies another parent a Child may attempt to bind to.

- SeqNum: an integer indicating the Sequence Number of a data message within a given Data Session. For a Heartbeat, it is the highest sequence number the parent knows about.

- ChildIndex: This is an integer the Repair Head assigns to a particular child. The child Receiver uses this value to implement the rotating TRACK Generation algorithm.

- LowestRepairAvailable: This is the lowest sequence number that a Repair Head will provide repairs for.

- Reason: a code indicating the reason for the BindReject, UnbindRequest, or EjectRequest message.

- ParentTimestamp: This field is included in Heartbeat messages to signal the need to do a local RTT measurement from a parent. It is the time when the parent sent the message.

- ChildrenList: This field contains the identifiers for a list of children. As part of the keepalive message, this field together with the SeqNum field is used to urge those listed Receivers to send a TRACK (for the provided SeqNum). The Repair Head sending this must have been missing the regular TRACKs from these children for an extended period of time.

- SenderTimestamp: This field is included in Data messages to signal the need to do a roundtrip time measurement from the Sender, through the tree, and back to the Sender. It is the time (measured by the Senders local clock) when it sent the message.

- ApplicationSynch: a Sequence Number signaling a request for confirmed delivery by the application.

- EndOfStream: indicates that this message is the end of the data for this session.

- TransmissionRate: This field is used by the Sender to tell the Receivers its sending rate, in messages per second. It is part of the data or nulldata messages.

- HighestReleased: This field contains a Sequence Number, corresponding to the trailing edge of the Senders retransmission window. It is used (as part of the data, nulldata or retransmission headers) to inform the Receivers that they should no longer attempt to recover those messages with a smaller (or same) Sequence Number.

- HighestAllowed: a Sequence Number, used for flow control from the Receivers. It signals the highest
Sequence Number the Sender is allowed to send that will not overrun the Receivers buffer pools.

- BitMask: an array of 1s and 0s. Together with a Sequence Number it is used to indicate lost data messages. If the ith element is a 1, it indicates the message SeqNum+i is lost.

- Slowest: This field contains a field that characterizes the slowest Receiver in the subtree beneath (and including) the node sending the TRACK. This is used to provide information for the congestion control BB.

- SenderDallyTime: This field is associated with a SenderTimestamp field. It contains the sum of the waiting time that should be subtracted from the RTT measurement at the Sender.

- ParentDallyTime: This is the same as the SenderDallyTime, but is associated with a ParentTimestamp instead of a SenderTimestamp.

- DataLength: This is the length of the Data payload.

- CongestionControl: This includes any additional congestion control variables for aggregation, such as WorstLossRate, WorstEdgeThroughput, UnicastCost, and MulticastCost.

- ApplicationConfirms: This is the SeqNum value for which delivery has been confirmed by all children at or below this parent.

- FailedChildren: This is a list of all children that have recently been dropped from the repair tree.

- Immediate: If set to 1, a Receiver should immediately send a TRACK on receipt of this packet.

- Reliability: The level of reliability required in order to consider the set of data packets reliably delivered.

- NumReplies: The number of consecutive TRACK messages that should be sent with this message attached

- SeqNumRange: The set of data messages that the ALC request applies to.

- ConfirmStatus: The acknowledgement status of the Receivers in the subtree up to the node that sends this message.

- ConfirmCount: The number of Receivers in the subtree up to the node that sends this message, that have acknowledged the ALC request.

- PendingCount: The number of Receivers in this subtree that are still pending in their decision as to acknowledging this ALC request.

- StatsSeqNum: The number of this request for statistics.

- StatsList: The list of statistics to be filled in by Receivers, and aggregated by the control tree.

6. Global Configuration Variables, Constants, and Reason Codes

6.1 Global Configuration Variables

These are variables that control the Data Session and are advertised to all participants. Some of them MAY instead be configured as constants.

- TimeMaxBindResponse: the time, in seconds, to wait for a response to a BindRequest. Initial value is TIMEOUT_PARENT_RESPONSE (recommended value is 3). Maximum value is MAX_TIMEOUT_PARENT_RESPONSE.

- MaxChildren: The maximum number of children a Repair Head is allowed to handle. Recommended value: 32.

- ConstantHeartbeatPeriod: Instead of dynamically calculating the HeartbeatPeriod, a constant period may be used instead. Recommended value: 3 seconds.

- MinimumHeartbeatPeriod: The minimum value for the dynamically calculated HeartbeatPeriod. Recommended value: 1 second.

- MinHoldTime: The minimum amount of time a Repair Head holds on to data messages.

- MaxHoldTime: The maximum amount of time a Repair Head holds on to data messages.

- AckWindow: The number of messages seen before a Receiver issues an acknowledgement. Recommended value: 32.

- LateJoinSemantics: The options available to a Receiver who wishes to join a Data Session that is already in progress.

- MaximumFailureListSize: The maximum number of entries that can be in a failure list. This MUST be small enough that the FailureList does not ever cause a TRACK to exceed the size of a maximum UDP packet. Recommended value: 800.

- MaximumStatisticsListSize: The maximum number of entries that can be in a statistics list. This MUST be small enough that the FailureList does not ever cause a TRACK to exceed the size of a maximum UDP packet. Recommended value: 100.

- MaximumDataRate: The maximum admission rate for data messages from the application to the Data Channel Protocol.

- MinimumDataRate: The minimum admission rate for data messages from the application to the Data Channel Protocol.

6.2 Constants

- NUM_MAX_PARENT_ATTEMPTS: The number of times to try to bind to a Repair Head before declaring a PARENT_UNREACHABLE error. Recommended value is 5.

- TIMEOUT_PARENT_RESPONSE: The minimum value, in seconds, between attempts to contact a parent. Recommended value is 1 second.

- MAX_TIMEOUT_PARENT_RESPONSE: The maximum value, in seconds, between attempts to contact a parent. Recommended value is 16.

- NULL_DATA_PERIOD: The time between transmission of NullData Messages. Recommended value is 1.

- FAILURE_DETECTION_REDUNDANCY: The number of times a message is sent without receiving a response before declaring an error. Recommended value is 3.

- MAX_TRACK_TIMEOUT: The maximum value for TRACKTimeout. Recommended value is 5 seconds.

- TRANSMISSION_REDUNDANCY: The number of times a failure notification is redundantly sent up the tree in a TRACK message. Recommended value is 3.

6.3 Reason Codes

- BindReject reason codes
  - LOOP_DETECTED
  - MAX_CHILDREN_EXCEEDED

- UnbindRequest reason codes
  - SESSION_DONE
  - APPLICATION_REQUEST
  - RECEIVER_TOO_SLOW

- EjectRequest reason codes
  - PARENT_LEAVING
  - PARENT_FAILURE
  - CHILD_TOO_SLOW
  - PARENT_OVERLOADED

7. Security

As specified in [12], the primary security requirement for a TRACK protocol is protection of the transport infrastructure. This is accomplished through the use of lightweight group authentication of the control and, optionally, the data messages sent to the group. These algorithms use IPsec and shared symmetric keys. For TRACK, [12] recommends that there be one shared key for the Data Session and one for each Local Control Channel. These keys are distributed through a separate key manager component, which may be either centralized or distributed. Each member of the group is responsible

for contacting the key manager, establishing a pair-wise security association with the key manager, and obtaining the appropriate keys.

The exact algorithms for this BB are presently the subject of research within the IRTF Secure Multicast Group (SMuG) and standardization within the Multicast Security working group.

8. References

[1] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

[2] Whetten, B., et. al. "Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer." RFC 3048, January 2001.

[3] Handley, M., et. al. "The Reliable Multicast Design Space for Bulk Data Transfer." RFC 2887, August 2000.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

[5] Whetten, B., Taskale, G. "Overview of the Reliable Multicast Transport Protocol II (RMTP-II)." IEEE Networking, Special Issue on Multicast, February 2000.

[6] Nonnenmacher, J., Biersack, E. "Reliable Multicast: Where to use Forward Error Correction", Proc. 5th. Workshop on Protocols for High Speed Networks, Sophia Antipolis, France, Oct. 1996.

[7] Nonnenmacher, J., et. al. "Parity-Based Loss Recovery for Reliable Multicast Transmission", In Proc. of ACM SIGCOMM 97, Cannes, France, September 1997.

[8] Rizzo, L. "Effective erasure codes for reliable computer communications protocols", DEIT Technical Report LR-970115.

[9] Nonnenmacher, J., Biersack, E. "Optimal Multicast Feedback", Proc. IEEE INFOCOM 1998, March 1998.

[10] Whetten, B., Conlan, J. "A Rate Based Congestion Control Scheme for Reliable Multicast", GlobalCast Communications Technical White Paper, November 1998. http://www.talarian.com/rmtp-ii

[11] Padhye, J., et. al. "Modeling TCP Throughput: A Simple Model and its Empirical Validation". University of Massachusetts Technical Report CMPSCI TR 98-008.

[12] Hardjorno, T., Whetten, B. "Security Requirements for TRACK" draft-ietf-rmt-pi-track-security-00.txt, June 2000. Work in Progress.

[13] Golestani, J., "Fundamental Observations on Multicast Congestion Control in the Internet", Bell Labs, Lucent Technology, paper presented at the July 1998 RMRG meeting.

[14] Kadansky, M., D. Chiu, J. Wesley, J. Provino, "Tree-based Reliable Multicast (TRAM)", draft-kadansky-tram-02.txt, Work in Progress.

[15] Whetten, B., M. Basavaiah, S. Paul, T. Montgomery, "RMTP-II Specification", draft-whetten-rmtp-ii-00.txt, April 8, 1998. Work in Progress.

[16] Kadansky, M., Chiu, D. M., Whetten, B., Levine, B. N., Taskale, G., Cain, B., Thaler, D., Koh, s. J., "Reliable Multicast Transport Building Block: Tree Auto-Configuration", draft-ietf-rmt-bb-tree-config-02.txt, March 2, 2001. Work in Progress.

[17] Whetten, B. et. al., "TRACK Protocol Instantiation Over UDP", draft-ietf-rmt-track-pi-udp-00.txt, November 2002.

[18] Adamson, B., et. al., "NACK Oriented Reliable Multicast Protocol (NORM), draft-ietf-rmt-pi-norm-02.txt, July 2001. Work in Progress.

[19] Vicisano, L., et. al., "Asynchronous Layered Coding - A scalable reliable multicast protocol", draft-ietf-rmt-pi-alc-02.txt, July 2001. Work in Progress.

[20] Speakman, T., et. al., "Pragmatic General Multicast (PGM)", draft-speakman-pgm-spec-06.txt, Feb 2001. Work in Progress.

[21] Kermode, R., Vicisano, L., "Author Guidelines for RMT Building Blocks and Protocol Instantiation Documents", RFC 3269.

10. Acknowledgements

We would like to thank the follow people: Sanjoy Paul, Seok Joo Koh, Supratik Bhattacharyya, Joe Wesley, and Joe Provino.

11. Authors Addresses

Brian Whetten
890 Sea Island Lane
Foster City, CA 94404
b2@whetten.net

Dah Ming Chiu
Sun Microsystems Laboratories
1 Network Drive
Burlington, MA 01803
dahming.chiu@sun.com

Miriam Kadansky
Sun Microsystems Laboratories
1 Network Drive
Burlington, MA 01803
miriam.kadansky@sun.com

Seok Joo Koh
sjkoh@pec.etri.re.kr

Gursel Taskale
TIBCO Corporation
gursel@tibco.com