Reliable Multicast Transport (RMT) WG
Internet Draft
Document: draft-ietf-rmt-track-arch-00.txt

B. Whetten
Talarian
D.Chiu
Sun Microsystems
S.Paul
Edgix
Miriam Kadansky
Sun Microsystems
Gursel Taskale
Talarian
July 14, 2000

TRACK ARCHITECTURE
A SCALEABLE REAL-TIME RELIABLE MULTICAST PROTOCOL

Status of this Memo

1. Abstract

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 in to 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].

The TRACK protocol instantiation (TRACK for short) is designed to reliably and efficiently send data from a single sender to large groups of

simultaneous recipients in real time. The term real-time is understood in the industry as minimal latency including network propagation and processing delays. TRACK PI provides functions similar to the NACK PI, and adds support for a tree-based hierarchy (in its simplest form may consist of only the sender as the Repair Head) of Repair Heads (RH), which increases scalability by providing aggregation of control traffic and local retransmission of lost packets. In addition to using negative acknowledgements (NACKs) and forward error correction (FEC) for efficient reporting and retransmission of lost packets, it also provides tree-based ACKnowledgements (ACKs). ACKs provide the Sender with confirmation of delivery of data packets to the Receivers. Like the NACK PI, it may also take advantage of Generic Router Assist where available.

This document proposes a design rationale for the TRACK PI, an architecture for TRACK, and a set of functional requirements TRACK has of other Building Blocks. This document is not a protocol instantiation specification.

2. 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 [4].

3. Design Rationale and Protocol Requirements

This section discusses many of the requirements imposed on the design of the TRACK PI, as well as a design rationale which guides the aspects where there is flexibility in selecting from different potential design decisions.

3.1 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. The IETF is specifically chartered with producing standards for the Internet, so this must be the primary target network type. However, robust transport protocols are grown, not created, and most of the short term deployment experience will likely come from controlled networks. Therefore, TRACK is designed to support both.

3.2 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 AS's where policies are

inconsistent. Hence, it is preferable to design tools that are fully distributed and automatic. To address these requirements, TRACK supports both manual and automatic algorithms for monitoring, management, and configuration.

3.3 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. In addition, the congestion control component used in TRACK will specifically deal with the high bandwidth-delay product faced in many satellite networks and the high link level loss rate faced by some wireless networks. Finally, TRACK does not assume that the topology used for sending control packets has any congruence to the topology of the multicast address used for sending data packets.

3.4 Use of Network Infrastructure

There is wide consensus that in order to scale a real-time reliable multicast protocol, there must be some use made of the network infrastructure (the routers and servers inside the network). New software that supports the transport layer typically would run in either the routers or the servers in the network, or both. Deployment of router software (such as that in the Generic Router Assist BB) is a powerful solution, but typically requires very long time cycles, is of necessity limited in functionality, and requires a graceful upgrade path. Server software (such as the Repair Head control tree) is much easier to deploy, but may require new hardware to be added to the network.

In controlled networks, particularly during the first deployment phases of reliable multicast, it is reasonable to deploy servers that only support a single application, or even to use selected end clients themselves to perform the functions necessary for scalability. For widely deployed Internet infrastructure components, the server infrastructure is usually dedicated to just the single protocol, but supports all instances of that protocol running across that piece of the network. Examples of this usage model include DNS, DHCP, NNTP, and HTTP. Therefore, the control nodes used in TRACK are designed to be run both on dedicated network servers able to support hundreds or thousands of simultaneous data sessions, as well as on an end user computer.

A number of extensions to IP multicast, such as subtree multicast, NACK suppression, ACK aggregation, tree configuration discovery, and higher fidelity congestion control reports, have been proposed which can run in the routers. If deployed widely, these would make reliable multicast protocols easier to configure and to scale more readily. Some or all of these features are being standardized as part of the Generic Router Assist

(GRA) component. TRACK is designed to take advantage of GRA as it becomes available, but not to require it. Ubiquitous deployment of GRA would likely reduce the number of dedicated TRACK servers needed for large scale (i.e. more than 1000 Receiver) deployments, and improve the performance of the protocol.

3.5 Targeted Application Types

Multicast applications can be divided into two classes, few-to-many and many-to-many. Many-to-many applications include multi-user games, small group conferencing, and computer supported collaborative work. These applications typically treat all members in a group as peers, require special semantics such as total ordering of messages from multiple Senders, and often have moderate scalability requirements. Other protocols, such as RMP, have been designed to support these many-many applications.

In line with the charter for RMT, TRACK focuses on one to many bulk data distribution applications, such as multicast file transfer, electronic software distribution, real time news and financial market data distribution, "push" applications, audio/video/data streaming, distance learning, and some types of server replication.

In order to meet these requirements, TRACK treats each Sender as an independent entity, and provides no ordering or other shared state across data sessions, although multiple data sessions can share the same control infrastructure. The protocol is designed to scale to at least many thousands of simultaneous Receivers. TRACK provides a strong, but fully distributed membership protocol, which supports scaling to many thousands of simultaneous Receivers while providing confirmed delivery on messages. Similar to TCP, TRACK continuously streams data to receivers, performing acknowledgement and retransmission of older data packets at the same time that new data packets are being sent. It also provides some special support for real-time applications such as audio/video/data streaming and live financial market data distribution.

Some real-time applications require jitter control for smooth playback. This can be accomplished by using the unordered delivery option of TRACK and performing jitter control in the application. Typically, this requires the application to maintain a separate buffer to smooth out the per packet delay variations.

TRACK also supports sender-controlled recovery window. In each data packet, the sender may indicate to all receivers that data older than certain sequence number are no longer worthy of recovering. (See section on "Delivery Semantics" for more details). This mechanism helps the transport better support applications that distribute content that ages quickly, such as stock quotes.

3.6 IETF Mandated Criteria

In addition to the requirements imposed by the targeted network and

application types, TRACK is designed to meet all of the requirements proposed by the IETF in RFC2357.

- Congestion Control. TRACK includes provably safe and TCP-friendly congestion control algorithms that also scale to large groups.

- Well-controlled, Scaleable Behavior. TRACK includes carefully analyzed algorithms that manage and smooth the control traffic and retransmissions. These are key to avoiding NACK implosion, ACK implosion, and retransmission implosion (the local recovery pathology).

- Security. TRACK supports protection of the transport infrastructure, through the use of lightweight authentication of control and data packets.

3.7 Graceful Evolution

Creating robust, universally applicable standard protocols takes a great deal of time and protocol evolution. While TRACK is being written as a standard, it will have to continue to evolve as real world experience is gained with the protocol, similar to how TCP has been tuned over almost 20 years of research and development. TRACK addresses this through its use of Building Blocks, which allow particular algorithms to be broken out in to separate components with well defined interfaces. This allows evolution of these components, hopefully with little or no changes required to the rest of the protocol.

TRACK also addresses evolution through its use of session parameters. TRACK is presently dependent on a number of parameters which MUST be configured throughout the tree for optimal operation. TRACK provides mechanisms to automatically distribute these parameters to all members of the group, and OPTIONALLY provides mechanisms to dynamically change some of these parameters during group operation.

TRACK also provides SNMP management and monitoring tools. Over time, deployment experiences will provide input on which values work best for most deployments, leading to further refinements of the standard.

3.8 Algorithm Selection

The above design criteria applies to the general architecture of the protocol. Additional criteria were used for selecting the optimal algorithms for different sets of functions. These rationales are described below, along with relevant functions.

4. Architectural Overview

4.1 TRACK Entities

4.1.1 Node Types

TRACK divides the operation of the protocol into three major entities: Sender, Receiver, and Repair Head. TRACK's 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, server level multicast, and multicast key management.

4.1.2 Multicast Group Address

A multicast group address is a pair consisting of an IP multicast address and a UDP port number. 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. Data multicast address and control multicast address are both multicast group addresses.

4.1.3 Data Session

A Data Session is the unit of reliable delivery of TRACK. It consists of a sequence of sequentially numbered Data packets, 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. It is uniquely identified by a combination of a Session ID, sender's address and port, and the multicast address and port.

A given Data Session is received by a set of zero or more Receivers, and a set of zero or more of 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 Streams it wishes to receive.

4.1.4 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 are aggregated as the information is relayed to higher nodes closer to the sender. Each Data Session uses a Control Tree.

Each RH in the control tree uses a separate multicast address for communicating with its children. Optionally, these RH multicast addresses

may be the same as the multicast address of the Data Channel.

4.1.5 Session ID

A Session ID is a 32-bit number (to be formally defined in the Common Packet Header BB) chosen either by the application that creates the session or selected by TRACK. Senders and Receivers use the Session ID to distinguish Data Streams. A Sender may specify a Session ID in the range from (2^31) to (2^32)-1. Numbers in the range from 0 to (2^31)-1 are reserved. If a sender specifies 0 as the Stream ID, then TRACK randomly assigns a Stream ID in the range from 1 to (2^31)-1. If a Session ID is selected that is already in use on a Control Tree, the new stream will fail, and will need to select a new Session ID.

A session is uniquely identified by its Session ID, its sender's address/port, and its Data Multicast Address and port.

4.1.6 Packet Sequence Numbers

A packet 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 packet in a Data Stream. A sender node assigns consecutive sequence numbers to the Data packets provided by the Sender application. Zero is reserved to indicate that the data session has not yet started.

4.1.7 Data Queue

A Data Queue is a buffer, maintained by a Sender or a Repair Head, for transmission and retransmission of the Data packets provided by the Sender application. New Data packets are added to the data queue as they arrive from the sending application, up to a specified buffer limit. The admission rate of packets to the network is controlled by flow and congestion control algorithms. Once a packet has been received by the Receivers of a Data Stream, it may be deleted from the buffer.

4.1.8 Packet Types

TRACK defines a set of packets, which can be implemented either on top of UDP or directly on top of IP. All TRACK packets will conform to the Common Packet Headers BB. Each TRACK packet definition consists of a fixed header, zero or more option headers, followed by data or control information.

Data is carried in Data packets. The same packet type is used both to transmit Data the first time and for retransmissions of lost packets. A bit in the packet header is set when the packet is a retransmission. Each Data packet has a Session ID and a sequence number, which identify the packet and allow a receiver application to reconstruct the data stream from the Data packets.
Receivers and Repair Heads unicast periodic status packets to their parents. An ACK is sent regularly to indicate the status of the Data

packets which have arrived and to furnish congestion control statistics about the state of data reception at the node. An ACK requests retransmission of Data packets that have not been received. An ACK also acknowledges packets that have become stable. A NACK is an ACK that is used to request immediate recovery of lost Data packets. ACKs and NACKs have the same format, but ACKs are passed all the way up the tree, while NACKs are only sent as far as needed to find a node which can provide all the requested retransmissions. A child will also send an ACK in response to a NullData or Heartbeat packet if it has not sent an ACK within a certain time interval.

TRACK uses the Tree-Building draft as a reference for building its repair tree. The following is a description of TRACK's implementation of tree building that is consistent with that draft.

When a Receiver or Repair Head wishes to establish a repair service relationship, it uses a Bind packet to bind to a parent Repair Head. A parent sends an Accept or Reject after it processes a Bind packet. The Reject message comes with a reason code that explains the reason for rejection. The reason may indicate that the parent is not connected into the tree yet, so that the receiver can try again later (see open issue). If the parent sends an Accept, this constitutes Joining a session.

When a Receiver or Repair Head wishes to leave a session, it sends a Leave request to its parent. The parent replies with a LeaveConfirm packet, at which time the child is allowed to leave.

A Repair Head or Sender periodically sends Heartbeat packets to notify its child nodes that it is alive.

If a Sender has no data to send for a session, it periodically multicasts a NullData packet on the Data Multicast Address. NullData packets inform receivers about the state of the Data Stream and the Sender.

If a child node is not operating normally, or a parent node restarts after a failure and receives a packet from a child not in its child list, then the parent node sends an Eject packet to the child node, causing the child node to terminate its connection to the control tree.

4.2 Basic Operation of the Protocol

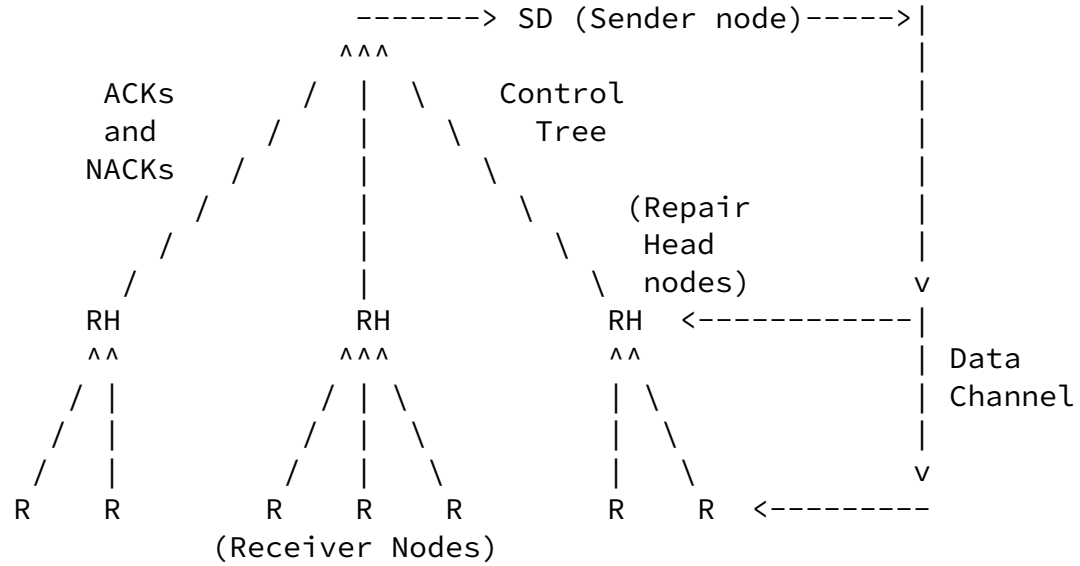
For each Data Session, TRACK provides sequenced, reliable delivery of data from a single Sender to up to tens of thousands of Receivers. A TRACK Data Session consists of a network that has exactly one Sender node, zero or more Receiver nodes and zero or more Repair Heads.

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

A Sender joins the TRACK tree and multicasts data packets on the Data Multicast Address. All of the nodes in the session subscribe to the class

D IP multicast address and UDP port associated with the Data Multicast Address.

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



A Receiver joins a Data Multicast Address to receive data. A Receiver periodically informs its parent about the packets that it has or has not received by unicasting an ACK packet to the parent. Each parent node aggregates the ACKs from its child nodes and (if it is not the Sender) unicasts a single aggregated ACK to its parent. For lower latency recovery in low loss networks, Receivers can also generate NACKs upon detection of losses. These have the same format as a ACK, but are only passed up the tree as far as necessary in order to find a Repair Head that can retransmit the packet. The Repair Heads provide NACK suppression, which provides traffic minimization benefits similar to ACK aggregation.

The Sender and each Repair Head have a multicast Local Control Channel to their children. This is used for transmitting Heartbeat packets that inform their child nodes that the parent node is still functioning. This channel is also used to perform local retransmission of lost data packets to just these children. TRACK will 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. Data and NullData packets regularly exercise the downward data direction. Heartbeat packets exercise the downward control direction. ACKs, NACKs, and HeartbeatResponse packets 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.

In addition to using ACKs, NACKs, and Repair Heads for scaleable loss

notification and retransmission, TRACK also supports the optional use of Generic Router Assist (GRA) and integrated Forward Error Correction (FEC). Two of the major functions of GRA are NACK suppression and dynamically scoped local retransmission. These functions, if enabled, are independently deployed between each parent and its children. For the purpose of GRA NACK functions, each parent is considered to be a Sender and the children of that parent are considered as the Receivers.

Retransmission requests, both NACKs and ACKs, contain selective bitmaps indicating which packets need to be retransmitted. If FEC is enabled, these bitmaps provide enough information to determine the number of parity packets to be sent rather than sending individual retransmissions.

4.3 Session Creation

Before a data session starts 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 in to 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 actual algorithms for tree configuration will be part of the Automatic Tree Configuration BB, and are discussed in the next section.

To start a data session, a Sender communicates to the Receivers, via either an external service or through the application itself, the Data Multicast Address that will be used for the Data Session. It may advertise 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 constants. It may also advertise certain hints for the tree configuration algorithms and metrics. In this way, the Sender enforces a set of uniform Session Configuration Parameters on all members of the session.

After receiving this out of band communication, the Receivers join the Data Multicast Address, and attempt to bind to either the Sender or a local Repair Head. The tree configuration algorithms are responsible for providing the Receiver with a list of one or more nodes which it will attempt to bind to. It will attempt to bind to the first node in the list, 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.

When a Repair Head has a Receiver bind to it for a given Data Session, it then also binds to another Repair Head or to the Sender, depending on the list given to it by the tree configuration algorithms. The tree configuration algorithms are responsible for ensuring that the tree is formed without loops.

Once the Sender initiates tree building, it is also free to start sending Data packets on the Data Multicast Address. Repair Heads and Receivers may start receiving these packets, but may not request retransmission or

deliver data to the application until they receive confirmation that they have successfully bound to the group.

Some of the Session Configuration Parameters MAY be changed dynamically by the Sender by advertising the changed values as part of the NullData packets periodically sent through the tree. If a given Session Configuration Parameter must be the same at all nodes in order to provide safe operation, it MUST NOT be dynamically changed once the Data Session has started.

4.4 Tree Configuration

TRACK is designed to work either with manual configuration of the tree, or with optional automatic tree configuration. Tree configuration is responsible for providing each Receiver and Repair Head with a list of one or more appropriate parents to attempt to bind to.

The goals of automatic tree configuration are:

- allow Receivers to automatically locate their best Repair Head(s), and obtain the local control channel multicast address.
- provide automatic configuration of the Repair Head with either Repair Heads that are servers operating in the network, or with dynamically selected receivers.

These algorithms are specified in the Tree Configuration BB [16]. In order to make sure that TRACK can be standardized in a timely fashion, the automatic tree configuration algorithms need to be separate from the rest of the TRACK protocol, so that TRACK can be deployed even without these algorithms. When these algorithms from the Tree Configuration BB are not available, TRACK will use static configuration.

4.5 Data Transmission and Retransmission

Data is multicast by a Sender on the Data Multicast Address. Retransmissions of data packets may be multicast by the Sender on the Data Multicast Address or be multicast on a Local Control Channel by a Repair Head. In order to provide NACK suppression and to work with proactive FEC, retransmissions are always multicast. If Generic Router Assist is enabled, the routers may provide NACK suppression and allow dynamically scoped retransmission to just the subset of Receivers and Repair Heads that have missed a packet.

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 packets 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 packets 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 packets sent to this address. If the Receiver's Repair Head fails and it transfers to another Repair Head, this minimizes the number of data packets it needs to recover after binding to the new Repair Head.

There are two types of retransmissions: local retransmission and dynamically scoped retransmission.

4.5.1 Local Retransmission

If a Repair Head or Sender determines from its child node's ACKs or NACKs that a Data packet was missed, the Repair Head retransmits the Data packet or, if FEC is enabled, an FEC parity packet. The Repair Head or Sender multicasts the Retransmission packet 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 is considered part of the Control Channel's multicast address, and is derived during tree configuration.

4.5.2 Dynamically Scoped Retransmission

Dynamically Scoped Retransmission may be used on a network whose routers support dynamically scoped retransmissions through Generic Router Assist. Dynamically Scoped Retransmissions use soft state kept in the routers to constrain the Retransmission to only the children that have requested them through a NACK. Dynamically Scoped Retransmissions are known to be susceptible to router topology changes. Therefore, only the first retransmission of a packet is sent via this mechanism. Thereafter, only the above two mechanisms should be used. This will allow the protocol to provide connectivity even during router topology changes, albeit with less efficiency.

4.6 Control Traffic Management

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. In conjunction with the dynamic global tree parameters, TRACK provides a set of algorithms that carefully control and manage this traffic, preventing control traffic explosion.

Despite their different names, ACKs and NACKs both function as selective acknowledgements of the window of contiguous sequence numbers that have not yet been fully acknowledged. The only difference between the packet headers is a single flag.

ACK packet frequency is controlled by setting a number of tree wide

parameters controlling their maximum rate of generation. The primary parameter is the ratio parameter, R, for the maximum number of ACK packets to be generated per data packet sent. The higher R is, the faster positive acknowledgements will be generated all the way back to the sender. This induces more back-channel traffic.

ACKs MUST be enabled for any Data Session. NACKs SHOULD be implemented as part of any implementation, and MAY be enabled for any given Data Session. If enabled, then on detection of a lost packet, a Receiver waits a random interval before sending a NACK. If the Receiver receives the retransmitted data before the NACK timer expires, the Receiver cancels the NACK. This reduces the chance that multiple Receivers generate a NACK for the same packet.

A Repair Head node multicasts a Data packet to its children as soon as it gets a NACK request for that packet, unless it retransmitted that packet previously in a configurable time period. If it does not have the missing packet, it forwards the NACK to its parent, and multicasts a control packet to its children to suppress any further NACKs for that packet from them. The Repair Head forwards only one NACK for a missing Data packet within a specified period of time. If more than one packet has been detected as missing before the NACK is sent, the NACK will request all of the missing packets.

NACKs are particularly good for providing real-time data distribution in networks with low loss rates and short to moderate RTT times. See [5] for comparisons on the tradeoffs between ACKs and NACKs for low latency recovery of lost packets.

4.7 Integrated Forward Error Correction

Work [6][7][8] has shown the benefits of incorporating reactive forward error correction (FEC) into reliable multicast protocols. This feature encodes data packets with FEC algorithms, but does not transmit the parity packets until a loss is detected. The parity packets are then transmitted and are able to repair different lost packets at different Receivers. This is a powerful tool for providing scalability in the face of independent loss. When implemented, it is a simple matter to also provide proactive FEC which automatically transmits a certain percentage of parity packets along with the data. This is particularly useful when a high minimum error rate is expected, or when low latency is particularly important. Both of these are optionally supported in TRACK.

FEC is organized around windows of packets. TRACK Data packets include an FEC offset window field, which identifies the offset of a given packet within an FEC window. Combined with the FEC session configuration parameters, this allows receivers to decode a combination of Data and parity packets, to generate each window of Data packets. Proactive FEC packets are parity packets sent as global retransmissions at the same time a window of Data packets are sent. Reactive FEC packets are sent either from a Repair Head or a Sender, in response to requests for

retransmissions. If using reactive FEC, a Repair Head must first have all the packets in a window before it can respond to any request for retransmission. The ACK and NACK bitmaps, combined with the information in the headers of the Data packets, provides each Repair Head with enough information to determine which parity packets the RH must compute and send in response to requests for retransmission.

4.8 Flow and Congestion Control

Flow and congestion control algorithms act to prevent the Senders from overflowing the Receivers' buffers and to force them to share the network fairly and safely with other TCP and RM connections. TRACK uses a combination of a transmission window for flow control, and the dynamic rate control algorithms specified in the Congestion Control (CC) BB for congestion control. These algorithms have been proven to meet all the requirements for flow and congestion control, including being safe for use in a general Internet environment, and provably fair with TCP.

The Sender application provides the minimum and maximum rate limits as part of the global parameters. A Sender will not transmit at lower than the minimum rate (except possibly during short periods of time when certain slow receivers are being ejected), or higher than the maximum rate. If a Receiver is not able to keep up with the minimum rate for a period of time, the CC BB algorithms will cause it to leave the group. Receivers that leave the group MAY attempt to rejoin the group at a later time, but SHOULD NOT attempt an immediate reconnection.

4.9 Notification of Confirmed Delivery

TRACK provides a simple membership count for each session. This is done by each repair head counting/aggregating its (subtree) membership count and propagating it up the tree to the sender. The propagation up the tree is piggybacked on the regular TRACK (ACK and NACK) packets.

Depending on whether there are late joiners, and receiver and repair head failures, this count may fluctuate over the duration of the session.

Whether this counting is done or not can be controlled by a session-wide configuration parameter.

A complete list of receiver membership can only be obtained if each repair head (including the sender) supports an SNMP interface that supports getting membership ids. Such SNMP support is optionally required for dedicated repair servers (but not required of regular receivers).

4.10 Fault Detection and Recovery

4.10.1 Sender node failure detection

A Sender node that has no data to send will periodically send NullData packets on the Data Multicast Address. If a Receiver or a Repair Head

fails to receive Data packets or NullData packets for a session sent by the Sender, the Receiver detects a Sender failure.

4.10.2 Repair Head failure detection

Each Repair Head node sends Heartbeat packets to its child nodes on its multicast Control Tree. If the child nodes do not receive any Heartbeats from their parent Repair Head, they detect failure of the parent.

4.10.3 Receiver node failure detection

A Receiver node sends ACKs and (optionally) NACKs for each of the active sessions that it has joined. If none of the sessions are active, then the Receiver sends HeartbeatResponse packets to its parent.

If a Receiver's parent node does not receive a ACK, NACK or a HeartbeatResponse packet within a specified time interval, the parent detects the failure of the Receiver and removes the child from its child list.

4.10.4 Repair Head discovery

TRACK supports an option which allows the nodes in the TRACK tree to acquire the addresses and location of its ancestors in the control tree and the addresses of its parent's siblings. If a TRACK node's parent fails, then the node can use the acquired information to join an alternate control node.

4.10.6 Recovery

When a child node detects failure of its parent node, it can try to reconnect to an alternate Repair Head of the TRACK tree, or it can try to reconnect directly to the Sender.

4.11 Reliability Semantics

The reliability semantics TRACK provides are defined by the binding between a receiver and its repair head. When this binding is established, the repair head agrees to provide retransmission of missed packets for the receiver starting from a specific (receiver requested) sequence number. At this time, the repair head MUST not have discarded any data packet starting from this sequence number.

Subsequently, a repair head needs to discard older packets from its buffer from time to time. The following two factors influence when to discard an old packet:
a) Stability - When all receivers immediately subordinate to the repair head have acknowledged receipt of a packet, that packet is considered stable. When the whole sub-tree of receivers below a repair head have received a packet, it is considered as "strictly stable". TRACK provides no explicit support for this strict sense of stability (note

this form of reliability is also referred to as "pessimistic reliability").
b) Sender recovery window - Each data packet carries two sequence numbers: one is the sequence number of the current data packet, and the other is the sender recommended sequence number where recovery should start from (smaller than the current sequence number). This pair of sequence numbers forms a sender-suggested recovery window.
A repair head MUST not discard any packet before it becomes stable. Per binding agreement or session wide configuration, a repair head MAY be allowed to discard a packet when it moves outside of the sender recovery window.

When a repair head's buffer is filled up and none of the packets can be discarded (due to stability or recovery window requirements), newly arrived packets must be discarded and recovered later.

A receiver SHOULD NOT try to recover packets outside of the sender recovery window.

When a receiver loses its repair head due to network partition or repair head crashing, the receiver MAY continue with the same reliability service if it manages to find and re-affiliate with another repair head. If the receiver fails to find an alternative repair head that can continue to provide reliability service where the previous repair head left off, this receiver MUST indicate failure to its application.

4.12 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 packets, 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 packets are delivered in the order sent, without duplicates.

4.13 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 have SNMP MIBs defined. SNMP support is optional for Receiver nodes, but is required for all other nodes.

4.14 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 packet 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.

4.15 Application Signaling for Notification

TRACK provides two forms of application signaling for speedy acknowledgement:
a) End of stream - this is done when the application has finished sending all its data, and wants to finish the session.
b) Synch - this is done when the application comes to a point in its data distribution that it wants to make sure all packets have been received before proceeding further. In this case the session is not ending.

In both cases, the application SHOULD be able to signal this through its transport API. In turn, TRACK will carry the signal as a flag in its data (or NullData) packets. For case (a), the flag is set in the last data packet of the session, and in additional NullData packets carrying the last sequence number. For case (b), the flag is set in the data packet the application requires synch, and in additional NullData packets sent prior to new data packets following the synch sequence number.

Upon receiving "end of stream", a receiver must try to recover data packets up to the indicated last sequence number and send its final ACK to its repair head. The receiver can then leave the repair head. When all the packets up to the last packet become stable, the repair head can leave.

Upon receiving "synch", the receivers and repair heads perform the same operations as in "end of stream" except they keep their binding.

5. Functional Specification for TRACK Requirements of Building Blocks

Work [2] provides a rationale for decomposing the RMT protocols in to Building Blocks and Protocol Instantiations. This section provides a simple specification of the functions that TRACK requires from each of the Building Blocks. It also provides some basic description of the interfaces

between these components.

Since the following overlaps with what is done in the BBs, all of section 5 is for discussion purposes only, and is not meant to replace what is specified in the supporting BBs. The BBs will define the actual algorithms.

5.1 NACK-based Reliability

This building block defines NACK-based loss detection/notification and recovery. The major issues it addresses are implosion prevention (suppression) and NACK semantics (i.e. how packets to be retransmitted should be specified, both in the case of selective and FEC loss repair).

The NACK suppression mechanisms used by TRACK are unicast NACKs with multicast confirmation and exponentially distributed timers. These suppression mechanisms primarily need to both minimize delay while also minimizing redundant messages. They may also need to have special weighting to work with Congestion Feedback.

5.1.1 NACK BB Algorithms

Exponential Back Off. When a packet is detected as lost, an exponentially distributed timer is set, based on the algorithms in [9]. This timer is biased based on the input congestion weighting factor. If either a packet or an explicit suppression message with the same sequence number is detected before the timer goes off, the timer is cancelled.

NACK Generation. When a timer goes off, the protocol instantiation is notified to generate a NACK for that sequence number. The protocol instantiation may, at its discretion, group multiple NACK notifications in to a single NACK packet. For TRACK, NACKs are implemented as a unicast packet with a multicast confirmation response.

Response to a Retransmission Request. When a Repair Head or other possible retransmission agent receives the first NACK from another group member for a given packet, it notifies the protocol instantiation to send either a data retransmission or, if it doesn't have the packet for retransmission, an optional suppression message. It then sets an embargo timeout, tied to the RTT to the furthest Receiver, during which other requests for the same packet will be ignored. The length of this embargo doubles each time that a retransmission is sent. This algorithm should also work with requests for retransmissions that come in the form of ACKs, as the algorithms and packet formats for both are identical, with the exception of the suppression mechanisms used.

GRA Signaling. A primary function of GRA is to do NACK elimination/suppression and subcasting of repairs. In order to do this, the transport need to signal the GRA-enabled routers to turn on the appropriate algorithms. This algorithm has to deal with issues such as router topology changes. While not dealt with in detail here, this is a

very subtle issue, which will have to be dealt with carefully.

5.1.2 NACK BB Parameters

Congestion Weighting. From Congestion Control BB. This is a weighting parameter for NACK suppression timers. The exact algorithms for this are still to be determined.

Loss Notification. From TRACK protocol instantiation. Notification at a Sender that a packet has been detected as lost, and the sequence number of that packet.

Retransmission Request. From TRACK protocol instantiation. When a NACK or ACK with a request for retransmission is received, this needs to be passed to the BB for handling retransmission requests.

GRA Enabled. From PI. Is GRA enabled in the network?

5.2 FEC Repair BB

This building block is concerned with packet level FEC repair. It specifies the FEC codec selection and the FEC packet naming (indexing) for both reactive FEC and proactive FEC.

5.2.1 FEC BB Algorithms

FEC Input. Receive a window of packets (not necessarily all at once), and store pointers to them for use in the FEC Create Parity algorithm.

FEC Create Parity. Given a window of packets, create and return a parity packet. If a window is not yet full, first call the FEC Flush function. If there are no more parity packets that can be generated for this window, then return an error or else return a parity packet that has already been generated. This uses one of a set of codecs, specified through the use of codepoints. For TRACK, it is expected that the codecs will operate over relatively small windows, to work with real-time applications and congestion control.

FEC Flush. For a window that needs a parity packet, but is not yet full, FEC flush creates all-zero packets for the rest of the packets in the window. No more calls to FEC input can be made for this window after FEC flush has been called.

FEC Decode. Given a set of received data and/or parity packets, decode the window using the specified FEC codec.

5.2.2 FEC BB Parameters

Codec Code Point Index. What is the codec being used for encoding and decoding? This is a fixed parameter per data stream.

FEC Window Size. What is the number of packets in an FEC window? This is a fixed parameter per data stream.

FEC Maximum Parity. This is the maximum number of parity packets that can be generated over a given window size. This is a fixed parameter per data stream.

Data Packet Sequence Number. This is the sequence number of a data packet. This is input to FEC Input from the PI.

FEC Window Offset. For a given packet, what is the offset in to an FEC window? This is associated with each Data packet that uses FEC. It is a header field on each Data packet sent. It is sequential over each packet in a window, unless a Flush occurs on a partially full window. In that case, the window offset of this last packet is set to FEC Window Sizeû1. For parity packets, the FEC Window Offset starts at FEC Window Size, and goes up to FEC Window Size + FEC Maximum Parityû1. This is returned from FEC Input and from FEC Create Parity.

5.3 Congestion Control BB

TRACK uses a source-based rate regulation algorithm, with a single rate provided to all the Receivers in the session.

The following set of algorithms and parameters is a subset of those needed for a full implementation, but give an idea of what is required.

5.3.1 Congestion Control BB Algorithms

Initialization. A number of transport-wide parameters must be fed to each of the nodes in the group, such as minimum rate, maximum rate, data segment size, etc.

Receiver Measurements. The Receiver must keep track of its average loss rate, and RTT to the Sender. We will call these measures "congestion reports".

Receiver Feedback. The Receivers must feed these congestion back to the Sender, piggybacked on both NACKs and ACKs.

Hierarchical Aggregation. Restricted worst edge aggregation should be used to aggregate the congestion reports in the ACKs and/or NACKs being fed up the tree [10]. Every time that an ACK or NACK is generated, this algorithm should be called to fill in the appropriate fields. Every time an ACK or NACK is received, this algorithm should be called to process the congestion control fields in the packet. This algorithm must also be notified every time a new child joins or leaves at a Repair Head or Sender.

Sender Rate Control. Based on the congestion reports received, the Sender must change its sending rate.

TCP Friendly Equation. Given values for RTT, DataSize, and LossRate, this generates a target throughput rate according to a modified version of the complex TCP model given in [11].

5.3.2 Congestion Control BB Parameters

Initialization Parameters. A set of different options, some of which can be permanent constants, but others are selected by either the Sender or a network manager.

Lost Packet. Every time a packet is detected as lost, the Senders must be notified of this.

RTT Measurement. Every time a RTT measurement is generated, either between Sender and Receiver(s), or between one level of the tree and another, the CC BB must be notified.

Highest Allowed Sequence Number (HASN). This is used to implement "receiver-driven" window control [13]. Each receiver can keep track of a congestion window and compute the HASN to be included in each ACK. A Repair Head aggregates the HASNs by computing the minimum value from all its children and forwards that as its own HASN up the tree.

5.4 Generic Router Assist BB

The task of designing scaleable RM protocols can be made easier by the presence of some specific support in routers. In some application-specific cases, the increased benefits afforded by the addition of special router support can justify the resulting additional complexity and expense.

Functional components which can take advantage of router support include feedback aggregation/suppression (both for loss notification and congestion control) and constrained retransmission of repair packets.

The process of designing and deploying these mechanisms inside routers can be much slower than the one required for end-host protocol mechanisms. Therefore, it would be highly advantageous to define these mechanisms in a generic way that multiple protocols can use if it is available, but do not necessarily need to depend on.

This component has two halves, a signaling protocol and actual router algorithms. The signaling protocol allows the transport protocol to request from the router the functions that it wishes to perform, and the router algorithms actually perform these functions.

An important component of the signaling protocol is some level of commonality between the packet headers of multiple protocols, which allows the router to recognize and interpret the headers. This is covered in the section on common packet headers, below.

5.4.1 GRA BB Algorithms

NACK Suppression. NACKs are sent towards the parent Repair Head or Sender, with a Router Alert option on. GRA enabled routers detect these packets and suppress redundant NACKs. It then updates a soft state table so that it knows to retransmit the requested packet to the requesting children, using Dynamic Selective Retransmission. The NACK suppression algorithm needs to work with both ACKs and NACKs, in order for Dynamic Selective Retransmission to work with TRACK. This means that GRA can not suppress ACKs but must still use them to update its state for retransmissions. It also means that GRA must work with ACK and NACK selective bitmaps, not just NACKs that request a single packet.

Dynamic Selective Retransmission. When a retransmission occurs, it is only forwarded to the interfaces of each router that have signaled through the use of NACKs that they need to see that packet.

Nearest Repair Head Hint. The router is made aware of the nearest Repair Heads, and is able to tell a child which is the best candidate for it to use. This must only be used as a hint to children.

Fine Grained Loss Reports. A major limitation of TFMCC is its limitation of only getting 1-bit loss reports (i.e. a packet is lost, or it is not) from the routers. A 8 or 16 bit report, piggybacked on to data packets, with the cumulative loss detected across all interfaces of GRA enabled routers the data packet crossed, would allow TRACK to become much more responsive to changes in network conditions. These reports can only be used as hints.

Signaling Protocol. The functions for GRA need to be requested by the protocol ahead of time, and then the run time packet headers need to be decipherable by the router.

5.4.2 GRA BB Parameters

GRA Enabled. Is GRA enabled in any of the routers in the network? Which functions do the deployed version of GRA support?

Packet Format. Which type of packet format is GRA to operate over? It is likely that different protocol instantiations will require differences in the packet headers they send to the router. This is tied to the common packet header BB, below.

5.5 Automatic Tree Configuration BB

TRACK takes advantage of hierarchical Repair Heads, to greatly increase the theoretical scalability of the protocol. These Repair Heads are used to form a tree with the source at the root, the Receivers at the leaves of the tree, and the Repair Heads in the middle. The Repair Heads can either be dedicated server software for this task, or they may be application nodes that are performing dual duty.

The effectiveness of these agents to assist in the delivery of data is highly dependent upon how well the logical tree they use to communicate matches the underlying routing topology. The purpose of this building block is to construct and manage the logical tree connecting the agents. Ideally, this building block will perform these functions in a manner that adapts to changes in session membership, routing topology, and network availability.

5.5.1 Auto Tree BB Algorithms

These are discussed in section 3.3. They are not yet mature enough to break down in to component parts.

5.5.2 Auto Tree BB Parameters

These are discussed in section 3.3. The algorithms are not yet mature enough to break down in to the parameters needed.

5.6 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 packets 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 TRACK protocol then provides options for piggy-backing key update messages on the Data Session and each Local Control Channel of the protocol. These can either include a new shared group key (encrypted with the old group key) or a notification that the group key(s) are being changed and that the group members should contact the key manager to get the new key(s). The former typically occurs on a periodic basis, while the latter may occur when a group member leaves.

The exact algorithms for this BB is presently the subject of research within the IRTF Secure Multicast Group (SMuG). Solutions for these requirements will be standardized within the IETF when ready.

5.7 Common Headers BB

As pointed out in the generic router support section, it is important to have some level of commonality across packet headers. It may also be useful to have common data header formats for other reasons. This building block consists of recommendations on fields in their packet headers that protocols should make common across themselves. TRACK needs to implement these recommendations in the TRACK PI.

5.7.1 Common Header BB Fields

GRA Signaling. The Retransmission, NACK and ACK packet headers need to provide a means for signaling their existence to GRA. For NACK and ACK headers, the selective bitmap needs to be specified in a common way across all protocols so that the GRA component can interpret these fields and determine the sequence numbers of the packets that are being requested. For the Retransmission packets, the sequence number of the packet needs to be in a standard position so that GRA can interpret it. For both NACK, ACK and Retransmission packets, the Session ID needs to be specified in a standard way across protocols.

Data Packets. The identification of data packets within a stream should be common across all protocols, both to aid in commonality of application semantics across protocols and to aid in GRA signaling. A Data Packet is identified by three fields: the Session ID, the Sequence Number, and the FEC Window Offset. The Session ID may include the multicast address and/or a unique ID. The sequence number starts at 1 and increments with each Data packet sent in the Session. Sequence numbers are always sequentially generated, without gaps. The FEC Window Offset specifies the offset of a Data packet in to an FEC window. For a window of W generated packets, a maximum window size of M, and a maximum parity size of P, the packets are numbered as follows. The first W-1 packets are numbered as 0 through W-2, with W never to exceed M. Packet W is always numbered as M-1, so that Receivers and Repair Heads can detect a partially filled window. The P parity packets are numbered M through M+P-1.

IP and UDP. It is easiest to implement protocols in the application space using UDP packets, but eventual kernel implementations will have TRACK implemented directly on top of IP. Other protocols share this requirement, and the way that this transition is done should be specified across all protocols.

6. Security Considerations

7. 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." Internet Draft, draft-ietf-rmt-buildingblocks-02.txt, Work in Progress.

3) Handley, M., et. al. "The Reliable Multicast Design Space for Bulk Data Transfer." Internet Draft, draft-ietf-rmt-design-space-01.txt, Work in Progress.

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 Protocols." 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) draft-ietf-rmt-bb-tree-config-00.txt

8. Acknowledgments

Special thanks goes to the following individuals, who have

contributed to the design and review of this document.

Supratik Bhattacharyya, Sprint Labs

Seok Koh, ETRI Korea
Joseph Wesley, Sun Microsystems

9. Author's Addresses

Brian Whetten
Talarian Corporation
333 Distel Circle
Los Altos CA 94022
whetten@talarian.com

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

Sanjoy Paul
Edgix Corporation
130 W. 42nd Street, Suite 850
New York, NY 10036
sanjoy@edgix.com

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

Gursel Taskale
Talarian Corporation
333 Distel Circle
Los Altos CA 94022
whetten@talarian.com

Full Copyright Statement