RMT Working Group Brian Whetten/Consultant
Internet Engineering Task Force Dah Ming Chiu/Sun Microsystems
Internet Draft Miriam Kadansky/Sun Microsystems
draft-ietf-rmt-bb-tree-config-03.txt Seok Joo Koh/ETRI/KOREA
18 November, 2002 Gursel Taskale/Tibco
Expires 18 May, 2003 Brian Neil Levine/Umass

Reliable Multicast Transport Building Block: Tree Auto-Configuration

<draft-ietf-rmt-bb-tree-config-03.txt>

Status of this Memo

Abstract

The Reliable Multicast Transport Working Group has been chartered to standardize multicast transport services. This working group expects to initially standardize three protocol instantiations. This draft is concerned with the requirements of the tree-based

ACK protocol. In particular, it is concerned with defining the building block for auto-configuration of the logical ACK-tree. According to the charter, a building block is "a coarse-grained modular component that is common to multiple protocols along with abstract APIs that define a building block's access methods and their arguments." For more information, see the Reliable Multicast Transport Building Blocks and Reliable Multicast Design Space documents [WLKHFL01][HWKFV00].

Table of Contents

1. Introduction

The Reliable Multicast Transport (RMT) working group has been chartered to standardize IP multicast transport services. This draft is concerned with the requirements of the tree-based ACK protocol [WCPKT00]. In particular, this draft defines a building block for auto-configuration of a tree comprised of a single Sender, Service Nodes, and Receivers into a tree (called a Session

Tree in this document). The design goals of this draft are motivated by the needs of TRACK-based protocols, however the trees it constructs are useful for other services.

This building block can be interfaced to any other BB or PI wishing to use a tree structure. To actually use this BB's features, the PI needs to includes the messages described in this BB in its packets. An example of how to use this BB can be found in the TRACK PI[WCPKT01].

The process of session tree construction is difficult for IP multicast. The best session trees match the underlying multicast routing tree topology [LPG98], however the multicast service model [DEE89] does not provide explicit support for discovering routing tree topology.

Furthermore, deployed multicast architectures can vary; for example, hosts may be restricted to multicast reception and not transmission with Source-Specific multicast routing [HC00]; and routers may provide special extended routing services with Generic Router Assist [CST00]. The RMT charter does not restrict the use of any particular network service in constructing the tree. It only suggests preferred scenarios. Accordingly, there are several viable solutions for constructing a tree, depending on network conditions.

The optimality of a tree may also depend on other factors, such as the need for load balancing, and the need to minimize the depth when used for collecting feedback information. The goal of this building block is to specify a distributed procedure for automatically constructing a tree that is loop-free and as efficient as possible given the information available.

This draft describes a unified solution for tree construction in the presence of different multicast service models and routing protocols. In particular, it specifies a single procedure which may be used with various techniques for service node discovery and distance measurements, several of which are specified within this document. The difference in these techniques primarily affects the optimality of the tree. The unified algorithm ensures that different implementations can interoperate and construct a loop-free tree.

In order to accommodate various multicast deployments, this document divides the tree building process into the following major components:

1. Several techniques for discovering neighboring service nodes.

   In particular: Static, Expanding Ring Search, and Mesh.

   Discovering neighboring service nodes is a necessary condition for getting connected, so each node in the session

   must implement at least one of the above techniques.

2. Several algorithms for selecting neighboring service nodes.

   In particular, the measurement and use of neighbor distance and sender distance are described. These service node selection algorithms help produce a good tree.

3. A single distributed procedure for construction and maintenance of loop-free Session Trees.

1.1 Terminology

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.

Session

A session is used to distribute data over a multicast address. A Session Tree is used to provide reliability and feedback services for a session.

Sender

The single sender of data on a multicast session. The Sender is the root of the Session Tree.

Receiver

A receiver receives data from the sender via the multicast session.

Session Identifier

A fixed-size number, chosen either by the application that creates the session or by the transport. Senders and Receivers use the Session Identifier to distinguish sessions. The size of this number is specified by the Protocol Instantiation (PI).

Service Node (SN)

A node within the tree which receives and retransmits data, and aggregates and forwards control information toward the Sender. The Sender operates as the root Service Node in any session tree. Service Nodes MAY be dedicated servers within a network designed to participate in multiple Sessions and support multiple trees, or they MAY be Receivers participating in an individual session. SNs MAY limit the number of Children they choose to service, and MAY also make other restrictions on the characteristics of each Child (distance, location, etc.). An SN that has accepted Children for a session is called a Parent.
In other documents, Service Node is sometimes referred to as Repair

Head (RH).

Session Tree (ST)

The Session Tree is a tree spanning all receivers of a multicast session. It is rooted at the Sender, consisting of zero of more Service Nodes as interior nodes, and zero or more receivers as leaf nodes. An ST is constructed for the forwarding of control information back to the Sender as well as for the resending of missed data to the Receivers. The ST for a particular session may change over the course of the session.

Parent

A Parent is an SN or Receiver's predecessor in the ST on the path toward the Sender. Every SN or Receiver on the tree except the Sender itself has a parent. Each Parent communicates with its children using either an assigned multicast address or through unicast. If a multicast address is used, this may be the same address used by the session, or one specifically assigned to the Parent.

Children

The set of Receivers and SNs for which an SN or the Sender is providing repair and feedback services.

Tree Level

A number indicating the number of "generations" a node is from the root. The sender is at TL=0. Those that use the sender as their parent are at TL=1 and so on. When a receiver is not connected to the tree yet, it has a tree level value greater or equal to 128. The reason for reserving part of the space (of tree levels) for indicating "off-tree" is so that special measures can be used to prevent forming loops. The largest value is 255, so the range of off-tree levels are in the range 128 - 255. Initially, all receivers have a TL value of 128. Once a Node joins the tree, its Tree Level is updated to be one more than its Parent's level.

Distance Metric

There are several techniques to quantify distances between nodes (Receivers, SNs, and the Sender) in a session. Each type of quantification is called a distance metric. Several Distance Metrics are described in this draft.

Sender Distance

The distance from a node to the Sender.

Neighbor Distance

The distance from a node to a Neighbor.

Neighbor

A node's (Receiver or SN) Neighbors are SNs that are close to the node, according to the Distance Metric(s) used by the node.

1.2 Assumptions

This document describes how to build trees under the following conditions:

a. The multicast group has only a single sender.
b. A single SN can serve multiple sessions.
c. Sessions can take advantage of a pre-deployed infrastructure of SNs (ones that are not necessarily aware of a session before the receivers), or recruit Receivers to be SNs.

d. Generic Router Assist[CST00] and Expanding Ring Search[YGS95] are not required of the network infrastructure, but if available they should be able to be utilized.

1.3 Requirements

The following are specifically required:

a. While tree-building may take advantage of information from the routing layer, the mechanisms described are independent of the routing protocol(s) used by the underlying multicast tree.
b. All trees constructed must be loop-free
c. These mechanisms must support late joiners and tree optimization

1.4 Applicability Statement

The authors recognize that automatic tree construction is a very difficult problem. Nonetheless, complete reliance on manual configuration is very user unfriendly and error prone as well.

This building block describes a procedure for constructing loop-free trees when there is minimal manual configured information available.

This is analogous to providing a system with default configurations that allow the system to work correctly, but not necessarily optimally.

There are many possible criteria for tree optimality. This BB does

not attempt to define a single optimality criterion, nor does it try to produce an optimal tree. It is, however, the goal of the BB to construct better trees as more configuration and measurement data are introduced to the procedure.

This BB describes only a subset of the possible parameters for constructing optimal trees, in particular sender distance and neighbor distance. There are many techniques for measuring these distances. Some of the techniques may not be applicable globally.

Expanding ring search (ERS) is an effective technique in a local subnet or intranet (especially when the IP multicast routing protocol is dense-mode based). On the other hand, it is not practical in a multi-domain network; it is not effective when the routing protocol is sparse-mode based; and it can add significant control traffic overhead.

Generic Router Assist (GRA) can provide measurement hooks to determine SNs that are located along the path for multicast data distribution. However, such facilities may not be available in all networks.

The tree construction procedure does allow manual configuration and various distance measurement techniques to be selectively and independently applied for different subgroups of receivers and SNs, to achieve incremental improvement to the quality of the tree.

There are many other criteria for tree-building than what is described in this document, for instance, methods based on load balancing and minimizing feedback latency.

2. Overview

The tree building process described within this document builds logical trees which consist of:

1. A root node (the Sender)
2. Intermediate nodes (Service Nodes or SNs) which may be either Receivers or nodes specifically allocated to the task of repair and aggregation
3. Leaf nodes which are Receivers only

Session trees are spanning trees rooted at the Sender. Session trees can be used for the forwarding of control information (i.e. ACKs) towards the root, or for forwarding of data (i.e. repairs) towards the leaf nodes.

Session trees are constructed per Sender; each node wishing to join the tree discovers its neighboring SNs and then selects its best parent based on locally available information, such as the relative

sender distances and neighbor distances. This document specifies several techniques for measuring distances.

It is important to note that SNs may be actual Receivers (e.g. Receivers willing and able to also function as SNs) or pre-deployed "specialized" servers that are signaled to join the tree by Receivers. We use the term Service Node to refer to either a Receiver or "server" which is participating as part of the logical tree formation.

Tree construction, regardless of SN discovery and selection algorithm, proceeds generically as follows.

1. Session Announcement

   Receivers of a session use standard out-of-band mechanisms for discovery of a session's existence (e.g. Session Advertisement ([HPW00], URL, etc). In this way, a Receiver discovers the multicast group address, the Sender's address, and other information necessary for logical tree construction. Sessions may be announced in two parts, the first part containing generic information about the session, such as the multicast address, and the second part, announced on the multicast address, containing additional information.

2. Measurements to the Sender (optional)

   All SNs and Receivers that know about the session optionally determine their distance to the Sender.

3. Neighbor Discovery

   Meanwhile, each Receiver discovers nearby SNs (candidate parents) for the Session using the neighbor discovery algorithm(s).

4. Service Node Selection

   Once a Receiver (or SN needing to join the tree) discovers a nearby SN, it obtains the SN's distance to the Sender as well as the SN's distance to the Receiver, tree level, and other suitability values, if available. After discovery is complete, the best SN is selected.

5. Binding to Service Node

   The Receiver or SN then binds to the chosen SN. If a bind is unsuccessful, the Receiver or SN retries with another nearby SN, or starts the discovery process all over again. Once an

SN receives a bind from a child, that SN must then also join

the tree if it has not already, discovering an SN of its own,
possibly using a different method than leaf Receivers.

6. Optimization (optional) and Fault Recovery

During a session, a Receiver or SN may change to a different
SN for a number of reasons described below, including fault
tolerance. The Session Tree is maintained and optimized over
time.

This building block provides mechanisms for maintaining and optimizing the tree, as well as tearing it down when the Sender is done with it. In the rest of this document, the term 'Node' denotes a Receiver or SN.

```
                    +--------------------+
                    |    1. Session      |
                    |   Advertisement    |
                    +--------------------+
                              |
                              | Node receives tree-building
parameters
                              V
                    +--------------------+
                    |  2. Measurements   |
<-------------------------|
                    |   to the Sender
|                          |
                    |     (optional)
|                          |
                    +-------------------
+                          |
|                                   |
|                                   |
```

```
V                                       |
                                     +--------------------
+                          |
                                     |    3. Neighbor
|                          |
                                     |      Discovery
|                          |
                                     +--------------------
+                          |

|                                       |

|                                       |

V                                       |
                                     +--------------------
+                          |
                                     |  4. Service Node
|                          |
                                     |     Selection
|                          |
                                     +--------------------
+                          |

|                                       |
                                              | Node picks best
Neighbor             |

V                                       |
                                     +--------------------
+                          |
                                     |   5. Binding to
|---------------------------
                                     |    Service Node     |   6. Optimization
(optional)
                                     |                     |      and Fault
Recovery
                                     +--------------------+
```


3. Session Announcement

The first step in the tree-building process is for a node to be informed of the session's existence. This can be done using some out-of-band method, such as Session Advertisement [HPW00], URL, e-mail, etc.

SNs do not necessarily receive these advertisements. If an SN is not a Receiver, it obtains the advertisement information once it is contacted by a Receiver.

The advertisement includes the multicast address being used, the Sender's address, the Session Identifier, any specific port numbers to be used, and any global information useful for tree construction. The advertisement may also contain information about one or more Service Node Discovery options (such as Static, ERS, and Mesh) that can possibly be used by Receivers in the session.

4. Service Node Discovery and Selection

Discovery is the process by which a node determines a suitable Service Node. During the discovery process, suitable neighbors are found, Sender distances are optionally exchanged, and the best SN is selected.

4.1 Service Node Discovery Algorithms

This draft describes three algorithms for discovering SNs: Static, Expanding Ring Search (ERS), and Mesh.

Multiple algorithms may be used within a single session. For example, SNs may use the Mesh algorithm, while the receivers use static configuration to discover the SNs; alternatively, some Receivers may use static configuration while other Receivers depend on ERS (in an intranet where ERS is available). Each Receiver may pre-configure which algorithm to use before it starts.

The transport protocols request the following information from this BB using the getSNs interface.

Service Nodes:

1. ParentAddress: the address and port of the parent node to which the node should connect
2. UDPListenPort: the number of the port on which the node will listen for its children's control messages
3. RepairAddr: the multicast address, UDP port, and TTL on which this node sends control messages to its children.

Receivers:

1. ParentAddress.

Senders:

1. UDPListenPort
2. RepairAddr

After the above information is obtained from auto-tree-config, the transport protocol may perform the necessary Bind operations for participating in the Session Tree.

4.1.1 Static

Static algotirhm relies on a functional entity, named Tree Configurator (TC), which is pre-configured by Sender for tree configuration in a static manner. TC may be simply implemented by a program and thus installed at Sender or any other host. It is not necessarily specialized infrastructure.

Static scheme is a typical top-down tree configuration approach. TC is used to govern the tree building based on its own (session- specific) tree configuration and SN(s) selection rules for the new joiners.

If a TC is used for tree building, its address and port MUST be included with the session advertisement. Receivers and SNs will realize there is a TC for the session via Session Announcement, and they can contact with the TC to get a list of candidate SNs by sending a unicast Query message.

In response to a Query message, the TC replies with a Advertise message that contains a list of candidate SNs available to the new joiner. The rule of determining such candidate SNs may depend on the pre-configured mechanism taken by TC. For example, TC may determine the candidate SN list for a Node among the possible SNs (it contains at that time) by considering which SNs are in the same network domain with the Node (i.e., via comparing their network prefix), or by considering the load balancing for the tree topology it has configured until then. For this purpose, TC maintains a pool of active SNs for the session. The list of candidate SNs carried by Advertise message is ordered in decreasing levels of preference, in which a lower number represents a higher preference.

When a Receiver Node receives the responding Advertise message from TC, the Node MAY proceed to try to bind to a candidate SN following the given order by sending a BindRequestmessage, and then waits for the responding message such as BindConfirm or BindReject from TC. These binding steps will be done according to the TRACK mechanism, as described in the TRACK PI [WCMKT02]

In the Static algorithm, SN discovery with a TC proceeds as follows:

1. The node joins the multicast session and learns of the location of TC (from the session announcement).

2. The node optionally discovers its distance from the Sender. Any metric described in Section 4.2 may be used.

3. The node sends a Query message to the TC for a parent. The request includes (optionally) the node's distance to the sender and whether the node functions as an SN or not.

4. The TC chooses one or more candidate parents (SNs) for the node from the active SNs by its own tree configuration rule. The selection of candidate parents may be done by comparing the network prefix or by referring to any other information such as the number of currently attached children, etc.

5. The TC MUST responds to the Query message with an Advertise message, which include the candidate parent list. In the list, each entry contains the corresponding IP address and port of an SN. All the entries in the list SHOULD be arranged in the decreasing order of preference levels. In the rejection case, the Advertise message does not include any candidate parent. In this case, the Node

may
resort to the other mechanism such as ERS and Mesh.

In the success case, the node will be enrolled as an
active
SN by the TC, if it functions as an SN in the
session. Each
active SN (functioning as a parent for the Session
Tree)
SHOULD send the TC the periodic Query messages with
a flag
indicating that it is active over a specific time
interval.
Based on the Query messages, the TC updates a pool
of active
SNs in the session. In response to the Query
message, the TC
sends a Advertise with a flag simply indicating that
the
Query message is received.

6. After receiving a successful Advertise message from
the TC,
the node will try to connect to its parent by
sending
BindRequest messages based on the candidate parent
list, as
described in Section 5.

4.1.2 ERS

ERS is a typical bottom-up tree configuration approach,
which can
be used only in the network environments where IP multicast
transmissions are allowed by Receivers and SNs.

In ERS algorithm, SN discovery with a TC proceeds as
follows:

1. The Nodes first look for Neighbors using a multicast
Query
message. The initial TTL value in the Query message,
TTLNeighborInit, is specified in the session
announcement

and may be as large as the session TTL (TTLSession).

An SN that is able to handle additional Children

MUST

respond to a Query message by multicasting an

Advertise

message.

If the SN is not yet a Parent, the TTL used in this

response

is the same TTL used in the Query message. If the SN

is a

Parent, the TTL used is the greater of the Query TTL

and the

Parent's current Advertise TTL.

2. The Node listens for Advertise messages after

sending the

Query message. If one or more Advertise messages are
received during a SolicitPeriod, the best SN among

them is

selected as described in section 4.3.

3. If no Advertise messages are received, the Node

sends

another multicast Query message with a TTL that is
incremented by TTLIncrement. The process of sending

the

multicast Query message with an increasing TTL value
continues until a response is received.

The TTL value is limited by a value, TTLMax, also

specified

in the session announcement. TTLMax defaults to the

value

of TTLSession.

If the TTL value required to reach the soliciting

Node is

greater than the TTL used to reach the SN, an

Advertise

message may not reach the Node. However, if future

Query

messages have increased TTL values, the TTL may

eventually

be large enough for the Advertise message to reach

the Node.

However, it is possible that the Node will not

locate any

SNs using Expanding Ring Search. It is advisable

that a

backup method, such as static, be available.

4. SNs MUST suppress sending Advertise messages in
response to
Query messages if one was sent with at least the
Query's TTL
within the last SolicitPeriod.

After multicasting a Query message, a node MUST wait
for an
interval, BetweenQuery, before sending another Query
message.
Nodes SHOULD suppress sending Query messages for
BetweenQuery seconds when they first start in order
to
collect information from Advertise messages already
solicited by other nodes.

5. Getting a successful Advertise message via ERS, the
Node
will try to connect to the parent by sending
BindRequest
messages, which are described in Section 5.

The following variables are used in the Expanding Ring
Search
algorithm.

- TTLNeighborInit: This is the initial TTL value to be used
by the
ERS if no other TTL value is specified by the algorithm.

- TTLIncrement: This is the periodic increment for the TTL
used in
ERS.

- TTLMax: This is a configured maximum TTL value to be used
by
either Query or Advertise messages.

- TTLSession: This is the session TTL value for the
multicast
session.

- SolicitPeriod: Each receiver MUST not send more than one

QUERY

message per SolicitPeriod. When SN's responds to QUERY
messages,

it also suppresses its ADVERTISE message if one has been
sent

less than SolicitPeriod ago. This parameter is used to
control

the amount of control traffic during tree construction if
ERS is

used.

- BetweenQuery: This is the time interval a node must wait
before

sending successive Query messages.

- MaxBindAttempts: This variable is an integer used to
control how

many times a receiver tries to bind to a SN before giving
up and

try another one.

- BindPeriod: In order to prevent loops, sometimes a SN
must reject

a BindRequest (for example, when the SN is not on the
tree yet

and has a BindRequest outstanding itself) from a
receiver. In

this case, if the receiver needs to retry binding to the
same SN

again (perhaps because the receiver does not discover any
alternative SN's), then it must wait for BindPeriod
seconds.

4.1.3 Mesh

The mesh approach relies on a set of SN's already deployed
as

infrastructure servers. These SN's are on-line, but are
not

necessarily aware of any particular session unless informed
by the

following mechanisms.

SN's in the mesh are configured to know who their neighbor

SN's are, and exchange reachability information with their neighbors in a way analogous to routers in a network. The actually protocol used by SN's to exchange such reachability information is outside the scope of this BB. (In principle, a routing protocol such as shortest-path-first, or a link-state-protocol can be adapted for this purpose). Instead, this BB specifies the following properties that the mesh of SN's MUST satisfy:

a) Each SN knows a subset of SN's in the mesh as its immediate neighbors.

b) Each SN has a "forwarding table", such that given any other (destination) SN in the mesh, the forwarding table gives a "next-hop" SN that can be used to reach the destination SN, plus the distance to the destination SN from the local SN.

c) A given SN in the mesh can "broadcast" information to all other SN's in the mesh (in the sense of having a means of sending the same information to all other SN's, but not necessarily simultaneously).

d) All potential sender and receivers of a multicast session can discover a "neighboring" SN in the mesh, using the neighbor discovery mechanisms described in Section 4.1.

The reason for running a routing-like algorithm to maintain the forwarding tables in each SN is to provide fault tolerance when some SN's in the mesh fail. When that happens, the remaining SN's exchange information with each other to update the forwarding

tables. In steady state, the mesh of SN's must still
satisfy the
above 4 properties.

In the simplest form, each SN in the mesh has a forwarding
table
that contains all other SN's in the mesh. This is called a
fully-
connected mesh.

As mentioned earlier, the Mesh scheme assumes that there is
a pre-
deployed infrastructures of SN servers. That is, the SN-SN
bindings
and Sender-SN (Sender's SNs) will be performed internally
by the
provider's own policy. The only thing done by Sender is to
inform
its SN's that a session starts. The relationships between
Sender
and its SN's (i.e., Sender-SN bindings) MUST also be pre-configured.

For example, the internal bindings between Sender and SN's
MAY be
done as follows:

1. The sender locates a neighbor SN in the mesh by a
pre-
configured mechanism. This SN is referred to as the
sender's
SN.

2. The sender sends the multicast session id, address
and port
(all these can be set as a abbreviated session
announcement
message) to the sender's SN.

3. The sender's SN in turn "broadcasts" the session
information
to all SN's in the mesh; since SN's can support
multiple
sessions simultaneously, they keep the information
about

each session in an entry in a session table.

4. After the Sender-SN bindings, Sender will multicasts
its
session announcement to the multicast receivers.
Then, the
sender's SN binds to the sender by sending a
BindRequest
message.

During the internal processings of Sender-SN binding
described
until now, any Query and Advertise messages are not used.

When a session starts, the bindings between SN and
Receivers will
be done. The tree binding of SN-receiver is done with the
Tree
Configurator (TC), which was used in the Static algorithm.
The
main difference between Static algorithm with TC and Mesh
algorithm with TC is that the active SNs are considered as
candidate SNs in the Static scheme, while the pre-deployed
SNs are considered as candidates in the Mesh scheme. That
is, in
the Mesh scheme, the TC is required to already get the
information
on the locations of the pre-deployed SNs.

Then the tree buildings between receivers and SNs are done
as
follows:

1. When a session starts, a receiver sends a Query
message to
the TC. The receiver may optionally discovers its
distance
from the Sender. Any metric described in Section 4.2
may be
used.

2. The TC chooses one or more candidate parents SNs for
the
receiver from the pre-deployed SNs by its own tree
configuration rule, as described in Section 4.1.1.

3. TC MUST respond to the Query message with an
Advertise
message, which include the candidate parent list. In
the list,
each entry contains the corresponding IP address and

port of
the parent.

4. Receiving a successful Advertise message from the TC,
the
Node will try to connect to its parent by sending
BindRequest

messages based on the candidate parent list, as
described in
Section 5.

Once a receiver is connected to a parent SN, the SN-SN
bindings
will also be done internally by the pre-configured
provider's
policy. For example, each SN in the mesh tries to bind to
its
"next-hop" SN. If the "next-hop" SN is not reachable for
some
reason, an SN may also try to bind to any neighbor SN as a
back-up
alternative. These procedures will be devised to ensure
that the
loop freedom is guaranteed in section 5.

4.2. Distance Measurement

Different techniques can be used to determine distances
between
nodes in a session. The distances of interest are:

Sender distance - the distance from a SN to sender

Neighbor distance - the distance from a receiver to a
neighboring SN

These distances can be used in selecting a Service Node
ifseveral
are discovered.

These techniques quantify distance differently. Each
specific way
of quantifying distance is called a metric. Different
Metrics are
not necessarily comparable. For example, if distance

between A and B is X using metric m1, and distance between A and C is Y using metric m2, then X > Y does not necessarily imply B is farther from A than C.

Only distances of the same metric should be compared and ranked. Therefore, a receiver SHOULD only rank two SN's based on their respective sender distance if those distances are based on the same metric.

On the other hand, it is not necessary for all receivers to use the same metric to select theirneighboring SN to connect to. Suppose receivers use neighbor distance as a selection criterion. One receiver may determine neighbor distances to SN's based on hop count, whereas another receiver may determine neighbor distances to its neighboring SN's based on delay.

4.2.1 TTL Hop-Count

If this metric is used, a node periodically sends a Beacon message on the Session's multicast address. The Beacon message includes the original time-to-live value set by the node. The distance to the node is then calculated as

Beacon's original TTL - Beacon's current TTL

One node is closer than another if its distance is a lower number. Note that the TTL value may not be available in some implementation environments.

4.2.2 Number of Levels

One metric a receiver can use for SN selection is the number of levels the SN is from the sender. For example, given two SN's in close proximity to a receiver, if one SN is n levels from the sender and the other is m levels, where m<n, the receiver SHOULD select the SN with m levels. This is because a shallower tree allows faster propagation of feedback information to the sender. (Note, we assumed the choice is between two SN's equally close to the receiver. The receiver to SN distance is another consideration).

The number of levels metric is not generally available, as the tree may be constructed bottom up. If the mesh approach is used, however, the distance in the SN's forwarding tables can be implemented as an estimate of the number of levels from the sender.

4.2.3 Delay

Another metric is the delay from an SN to the sender. If the SN is directly connected to the sender, then the delay would simply be the time to send feedback from the SN to the sender. If the SN is several levels down from the sender, then the delay would be the sum of the delays for each level (with some jitter time added in each level). For example, given two SN's in close proximity to a receiver, the receiver SHOULD select the SN with a smaller delay to the sender. This is again for the purpose of minimizing the feedback time.

If the tree is built bottom up, this metric cannot be used. If the mesh approach is used, this metric can be implemented, although it requires the SN's in the mesh to exchange distance information based on the delay metric.

4.2.4 Address

For IPv6 addresses[HOD98], distance can be approximately determined by the number of aggregation levels one address has in common with another. For this metric, one node is closer than another if its address has more aggregation levels in common with the querying node's address.

4.2.5 Static

The node's distance to other nodes may be made available in some well-known location. One node's is closer than another if its distance is a lower number.

4.2.6 GRA

The node's distance to the sender may be determined with help of a GRA message that lists the set of GRA routers on the path from the source.

4.3 Service Node Selection

Once Neighbors have been discovered, a node selects the best one using whatever distance information is available.

If there is no sender distance information to compare, the best SN is simply the one that is closest to the node, without a loop being formed by binding the node to the SN.

If sender distances are available, there are two cases:

Leaf nodes: For leaf nodes, the goal is to use the closest SN possible.

SNs: For SNs joining the tree, it is important to pick an SN that is closer to the sender; neighbor distance is a secondary factor.

Once an SN has been selected, the node tries to bind to it as described in Section 5. Loop prevention is done during the bind process using only Tree Level information.

This algorithm is recommended because it assigns each node the closest SN, and does not require all nodes to measure their sender distance at the start of the session. Depending on the selected metric, multiple nodes measuring sender distance could cause message implosion, and delay tree construction. On the other hand, the SNs selected may actually be further from the Sender than their children are. However, it may be necessary to assign nodes to non-optimal SNs in order to get them on the tree, since it is possible that no SN closer to the Sender can accept any more children.

Alternatively, nodes may be required to measure sender distance before selecting an SN in order to ensure that each parent is closer to the Sender than its children. Presumably, this results in a tree in which parents detect message loss before their children, minimizing repair requests.

5. Tree Formation

The following is a detailed description of the tree formation process. All tree construction follows this pattern. The ONLY differences between instantiations of this building block lie in how nodes discover and select neighbors.

Once an SN has been selected, the Node sends a BindRequest message to the SN. If the SN has an outstanding request to bind to another SN, it must refuse the incoming bind request in case it would form a loop.

Otherwise, it MAY accept the Node as a child as long as selecting it would not cause a loop in the tree. Loop freedom is guaranteed by these two rules:

1. If the requesting node does not have children, the SN can accept it as a child as long as the SN has no outstanding bind requests. If it does have an outstanding bind request, the SN can accept the node as a child if its address is less than the child's address.

2. If the requesting node has children, the SN can accept it as a child if

a. the SN's level is 128, i.e., it is the top of a sub-tree not yet connected to the Sender, or

b. the SN's level is less than 128, i.e., it is connected to the Sender.

The second rule prevents a node from selecting one of its own children as its parent. Two nodes at level 128 are prevented from selecting each other using tie-breaking criteria described

step 1
above.

If the SN accepts the Node as a Child, it returns a
BindConfirm
message. If it does not accept the Node, it sends a
BindReject
message. If the Node does not receive a response after
MaxBindAttempts tries every BindPeriod seconds, it MAY
select the
next best Neighbor from its cached list, or else run the
Service
Node Discovery process again to determine an alternate SN
to try.

The BindReject message contains a reason code. If the code
indicates that the node was rejected because the SN was not
yet on
the tree, the node MAY choose to retry that SN after
BindPeriod
seconds, or select a different available SN.

The BindReject message may also include a list of
alternative SNs
for the node to try.

The BindConfirm message MUST include the Parent's current
Tree
Level. The Node MUST set its Tree Level to one more than
the
Parent's level.

The BindConfirm message also MUST also indicate the
starting
sequence number of the message from which data reliability
is
assured. This information is included in the BindConfirm
message
to enable receivers to meet the PI's late join
requirements. If
nodes join the tree after the Sender has started to send
data, it
is possible that some of the data is no longer available
within

the tree. Nodes may need to have specific information about repair availability before selecting a Parent.

Service Nodes MAY limit the number of children they support depending on their capacity. Once an SN has accepted its maximum number of children, it stops accepting new children until a change in membership causes its count of children to go below this limit.

If an SN limits the number of children it supports, it MUST reserve at least one child slot for other SNs. This guarantees the growth of the repair tree.

6. Tree Maintenance

Tree maintenance is an ongoing process active in every Node. Because the tree is based on the operation of SNs, as well as the various underlying metrics that may change over time, it is important that these dependencies be monitored for changes. Nodes MUST monitor Parents for liveness and changes in tree level, and SHOULD continue to run the Neighbor Discovery and Selection process in order to optimize their choice of SN. Parents must also monitor Children for liveness.

6.1 Monitoring Parents and Children

The upper Building Block or Protocol Instantiation is responsible for monitoring parents and children. Monitoring messages from parents to children MUST contain the Parent's current Tree Level. Children MUST set their Tree Level to one more than their Parent's level.

If a Child loses contact with its Parent for a period of time, it MUST report it using the lostSN interface, and attempt to bind with an alternate SN.

A Child that is leaving a session MUST send a unicast UnbindRequest message to its Parent. The Parent MUST respond with an UnbindConfirm message.

A Parent that is leaving the session MUST send an EjectRequest

message to its Children indicating that they need to bind with an alternate SN. If possible the EjectRequest message is multicasted, but the EjectRequest message can also be sent via unicast to each child individually. Upon receiving an EjectRequest message from its parent, a receiver sets its Tree Level to 128 again. Using the heartbeat mechanism, the Tree Level for all receivers in the affected subtree will be updated (to a value higher than 128).

If a Parent does not hear from a Child for a period of time, or it receives a UnbindRequest message from a Child, it removes that Child from its list of Children, and reports the loss using the removeChild interface.

6.2 Optimizing the Tree

Implementations of this building block SHOULD continue to run the Neighbor Discovery and Selection process in order to optimize the choice of SN. This continuous process also keeps the distance information for the current Parent up-to-date. Whenever the process returns a better SN than the current one, the Node MAY bind to the new SN. Once the new SN is bound to, the Node

MUST send a UnbindRequest message to the original Parent. A Parent with no Children MAY leave the session.

7. Messages

These messages are required for implementations of this building block. The list below indicates which message contents are required by implementations. Implementations may also include other protocol-specific information in these messages. Note that these messages are parts of packets specified in PI's that use this BB.

Message Name m- or u-cast	Description	Contents
Query both	A message used to discover neighbors. The unicast is sent to TC; the multicast is used in ERS.	Sender distance (optional), TTL (m-cast only)
Advertise both	A message used to advertise an SN. The unicast is sent from the TC; the multicast is sent by SNs themselves	IP address, port, Sender distance (optional)
BindRequest unicast	Request to SN to join tree	Current Tree Level

```
+---------------------+
```



```
                 |  BindConfirm   | SN accepts BindRequest      | Current Tree
Level  |
                 |   unicast      |
|                      |
                 +----------------+----------------------------
+---------------------+
                 |  BindReject    | SN rejects BindRequest      | Reject
reason,      |
                 |   unicast      |                             | alternate SN
list   |
                 +----------------+----------------------------
+---------------------+
                 | UnbindRequest  | Child leaving Parent(u-cast)| Unbind
reason       |
                 |    unicast     | Parent leaving tree (m-
cast)|                      |
                 +----------------+----------------------------
+---------------------+
                 | UnbindConfirm  |  Acknowledgement of
|                      |
                 |   unicast      |  UnbindRequest message
|                      |
                 +----------------+----------------------------
+---------------------+
                 |  EjectRequest  | Parent refusing or leaving  | Eject
reason        |
                 |    both        | service to Children
|                      |
                 +----------------+----------------------------
+---------------------+
                 |  EjectConfirm  |  Acknowledgement of
|                      |
                 |   unicast      |  EjectRequest message
|                      |
                 +----------------+----------------------------
+---------------------+
```

8. External Interfaces

This section describes external interfaces for the building block.

8.1 Interfaces to this BB.

These may be used by a PI, or by a higher-level BB.

8.1.1 start(boolean SN, advertisement)

start notifies the BB to begin operation. If the SN parameter is set to TRUE, the BB also starts SN operation.

8.1.2 end()

end notifies the BB to end operation.

8.1.3 incomingMessage(message)

This interface is used to pass an incoming message down from the PI.

8.1.4 getStatistics

getStatistics returns current BB statistics to the upper BB or PI.

8.1.5 getSNs

getSNs instructs the BB to start the process of finding SN candidates for this node. getSNs may return immediately with a list of

candidate SN, or may use the SNlist interface (see section 8.2.1) to return the list at a later time.

8.1.6 setSN(SN)

setSN informs the BB that the PI or upper BB has successfully bound to an SN.

8.1.7 acceptChild(node)

acceptChild asks the BB to accept or reject the node as a member. The BB returns a boolean value in response.

8.1.8 removeChild(node)

removeChild is called to inform the BB that the child is no longer a member.

8.1.9 treeLevelUpdate(newLevel)

This interface is used to pass down any update to the node's tree level that the upper BB or the PI has learned. newLevel replaces the BB's current tree level.

8.1.10 lostSN

lostSN notifies the BB that the connection to the SN was lost.

8.1.11 setOptimization(boolean)

setOptimization tells the BB to start or stop the tree optimization process. The upper BB or PI may want to control when tree optimization takes place.

8.1.12 recordSNs(destination)

recordSNs tells the BB to record the current SN, plus the list of alternates, to the indicated destination.

8.2 Interfaces from this BB to the PI or a higher-level BB.

8.2.1 outgoingMessage(message)

outgoingMessage instructs the PI to send the message.

8.2.2 SNlist(list)

SNlist returns to the upper PI or BB a list of SN candidates. The list may contain additional information for each candidate, such as details about the data packets that it has available for repair.

9. References

[DEE89] Deering, S., "Host Extensions for IP Multicasting", RFC 1112.

[ECTP02] ITU-T Q.8/17 and ISO/IEC JTC1/SC6/WG7, "Enhanced Communication Transport Protocol: Specification of Simplex Multicast Transport," ITU-T Recommendation X.606 and ISO/IEC 14476-1, Feb., 2002.

[HC00] H. Holbrook, B. Cain, "Source-Specific Multicast for IP," Internet Draft, Internet Engineering Task Force, November 2000.

[HOD98] R. Hinden, M. O'Dell, S. Deering, "An IPv6 Aggregatable Global Unicast Address Format," Request For Comments 2374, July 1998.

[HPW00] M. Handley, C. Perkins, E. Whelan, "Session Announcement Protocol", Internet Draft, Internet Engineering Task Force, March 2000.

[HWKFV00] M. Handley, B. Whetten, R. Kermode, S. Floyd, L. Vicisano, and M. Luby, "The Reliable Multicast Design Space for Bulk Data Transfer," RFC 2887, August 2000.

[K01] S. Koh, et. al., "Configuration of ACK Tress for Multicast Transport Protocols," ETRI Journal, Vol. 23, No. 3, September 2001.

[KV02] R. Kermode, L. Vicisano, "Author Guidelines for RMT Building Blocks and Protocol Instantiation Documents," RFC 3269, April 2002.

[LPG98] B.N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves, "Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation," Proc. Sixth ACM International Multimedia

Conference (ACM Multimedia 98), Bristol, UK, September
1998.

[PTM93] C. Partridge, T. Mendez, and W. Milliken. "Host
anycasting
service." Request For Comments 1546, November 1993.

[WCMKT02] B. Whetten, D. Chiu, M. Kadansky, S. Koh, G.
Taskale,
"TRACK Protocol Instantiation over UDP: Tree
Acknowledgement based
Reliable Multicast," Internet Draft, November 2002.

[WLKHFL01] B. Whetten, L. Vicisano, R. Kermode, M.
Handley, S.
Floyd, and M. Luby, "Reliable Multicast Transport Building
Blocks
for One-to-Many Bulk-Data Transfer," RFC 3048, January
2001.

[YGS95] R. Yavatkar, J. Griffioen and M. Sudan, "A Reliable
Dissemination Protocol for Interactive Collaborative
Applications",
University of Kentucky, 1995.

[SV02] T. Speakman, and L. Vicisano, "Reliable Multicast
Transport
Building Block: Generic Router Assist - Signalling Protocol
Specification," Internet Draft, Internet Engineering Task
Force,
July 2002.

Authors' Addresses

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

Dah Ming Chiu
dahming.chiu@sun.com
Miriam Kadansky
miriam.kadansky@sun.com
Sun Microsystems Laboratories
1 Network Drive

Burlington, MA 01803

Seok Joo Koh
sjkoh@etri.re.kr
ETRI/Korea
161 Kajong-dong
Yusong-Gu, TAEJON, 305-350, KOREA

Gursel Taskale
gursel@tibco.com

Brian Neil Levine
brian@cs.umass.edu

Full Copyright Statement