A. Ballardie
Consultant

March 1997

Core Based Trees (CBT) Multicast Routing

-- Protocol Specification --

Status of this Memo

Abstract

This document describes the Core Based Tree (CBT) network layer multicast routing protocol. CBT builds a shared multicast distribution tree per group, and is suited to inter- and intra-domain multicast routing.

CBT is protocol independent in that it makes use of unicast routing to establish paths between senders and receivers. The CBT architecture is described in [1].

This document is progressing through the IDMR working group of the IETF. CBT related documents include [1, 5, 6]. For all IDMR-related documents, see http://www.cs.ucl.ac.uk/ietf/idmr.

TABLE OF CONTENTS

1. Changes Since Previous Revision........................... 3

2. Introduction & Terminology................................ 4

3. CBT Functional Overview................................... 5

4. CBT Protocol Specificiation Details....................... 8

4.1 CBT HELLO Protocol.................................... 8

4.1.1 Sending HELLOs.................................. 9

4.1.2 Receiving HELLOs................................ 9

4.2 JOIN_REQUEST Processing............................... 10

4.2.1 Sending JOIN_REQUESTs........................... 10

4.2.2 Receiving JOIN_REQUESTs......................... 10

4.3 JOIN_ACK Processing................................... 11

4.3.1 Sending JOIN_ACKs............................... 11

4.3.2 Receiving JOIN_ACKs............................. 12

4.4 QUIT_NOTIFICATION Processing.......................... 12

4.4.1 Sending QUIT_NOTIFICATIONs...................... 12

4.4.2 Receiving QUIT_NOTIFICATIONs.................... 13

4.5 CBT ECHO_REQUEST Processing........................... 14

4.5.1 Sending ECHO_REQUESTs........................... 14

4.5.2 Receiving ECHO_REQUESTs......................... 14

4.6 ECHO_REPLY Processing................................. 15

4.6.1 Sending ECHO_REPLYs............................. 15

4.6.2 Receiving ECHO_REPLYs........................... 15

4.7 FLUSH_TREE Processing.................................. 16
4.7.1 Sending FLUSH_TREE Messages...................... 16
4.7.2 Receiving FLUSH_TREE Messages.................... 16
5. Timers and Default Values.................................. 16
6. CBT Packet Formats and Message Types...................... 17
6.1 CBT Common Control Packet Header....................... 18
6.2 HELLO Packet Format.................................... 19
6.3 JOIN_REQUEST Packet Format............................. 19
6.4 JOIN_ACK Packet Format................................. 20
6.5 QUIT_NOTIFICATION Packet Format........................ 21
6.6 ECHO_REQUEST Packet Format............................. 21
6.7 ECHO_REPLY Packet Format............................... 22
6.8 FLUSH_TREE Packet Format............................... 23
7. Core Router Discovery...................................... 23
7.1 Bootstrap Message Format............................... 25
7.2 Candidate Core Advertisement Message Format............ 25
8. Interoperability Issues.................................... 25
Acknowledgements.. 26
References.. 26
Author Information.. 27

1. Changes since Previous Revision (05)

This revision of the CBT protocol specification differs significantly from the previously released revision (05). Consequently, this revision represents version 2 of the CBT protocol. CBT version 2 is not, and was not, intended to be backwards compatible with version 1; we do not expect this to cause extensive compatibility problems because we do not believe CBT is at all widely deployed at this stage. However, any future versions of CBT can be expected to be backwards compatible with this version.

The most significant changes to version 2 compared to version 1 include:

+o new LAN mechanisms, including the incorporation of an HELLO protocol.

+o new simplified packet formats, with the definition of a common CBT control packet header.

+o a generic intra-domain core discovery ("bootstrap") mechanism, to be specified separately, and published soon.

This specification revision is a complete re-write of the previous revision.

2. Introduction & Terminology

In CBT, a "core router" (or just "core") is a router which configured to act as a "meeting point" between a sender and group receivers. The term "rendezvous point (RP)" is used equivalently in some contexts [2]. Each core router is configured to know it is a core router.

A router that is part of a CBT distribution tree is known as an "on-tree" router. An on-tree router maintains active state for the group.

We refer to a broadcast interface as any interface that supports multicast transmission.

An "upstream" interface (or router) is one which is on the path towards the group's core router with respect to this router. A "downstream" interface (or router) is one which is on the path away from the group's core router with respect to this router.

Other terminology is introduced in its context throughout the text.

3. CBT Functional Overview

The CBT protocol is designed to build and maintain a shared multicast distribution tree that spans only those networks and links leading to interested receivers.

To achieve this, a host first expresses its interest in joining a group by multicasting an IGMP host membership report [3] across its attached link. On receiving this report, a local CBT aware router invokes the tree joining process (unless it has already) by generating a JOIN_REQUEST message, which is sent to the next hop on the path towards the group's core router (how the local router discovers which core to join is discussed in section 7). This join message must be explicitly acknowledged (JOIN_ACK) either by the core router itself, or by another router that is on the unicast path between the sending router and the core, which itself has already successfully joined the tree.

The join message sets up transient join state in the routers it traverses, and this state consists of <group, incoming interface, outgoing interface>. "Incoming interface" and "outgoing interface" may be "previous hop" and "next hop", respectively, if the corresponding links do not support multicast transmission. "Previous hop" is taken from the incoming control packet's IP source address, and "next hop" is gleaned from the routing table - the next hop to the specified core address. This transient state eventually times out unless it is "confirmed" with a join acknowledgement (JOIN_ACK) from upstream. The JOIN_ACK traverses the reverse path of the corresponding join message, which is possible due to the presence of the transient join state. Once the acknowledgement reaches the router that originated the join message, the new receiver can receive traffic sent to the group.

Loops cannot be created in a CBT tree because a) there is only one active core per group, and b) tree building/maintenance scenarios which may lead to the creation of tree loops are avoided. For example, if a router's upstream neighbour becomes unreachable, the router immediately "flushes" all of its downstream branches, allowing them to individually rejoin if necessary. Transient unicast loops do not pose a threat because a new join message that loops back on itself will never get acknowledged, and thus eventually times out.

The state created in routers by the sending or receiving of a JOIN_ACK is bi-directional - data can flow either way along a tree "branch", and the state is group specific - it consists of the group address and a list of local interfaces over which join messages for the group have previously been acknowledged. There is no concept of "incoming" or "outgoing" interfaces, though it is necessary to be able to distinguish the upstream interface from any downstream interfaces. In CBT, these interfaces are known as the "parent" and "child" interfaces, respectively. We recommend the parent be distinguished as such by a single bit in each multicast forwarding cache entry.

With regards to the information contained in the multicast forwarding cache, on link types not supporting native multicast transmission an on-tree router must store the address of a parent and any children. On links supporting multicast however, parent and any child information is represented with local interface addresses (or similar identifying information, such as an interface "index") over which the parent or child is reachable.

When a multicast data packet arrives at a router, the router uses the group address as an index into the multicast forwarding cache. A copy of the incoming multicast data packet is forwarded over each interface (or to each address) listed in the entry except the incoming interface.

Each router that comprises a CBT multicast tree, except the core router, is responsible for maintaining its upstream link, provided it has interested downstream receivers, i.e. the child interface list is non-NULL. A child interface is one over which a member host is directly attached, or one over which a downstream on-tree router is attached. This "tree maintenance" is achieved by each downstream router periodically sending a CBT "keepalive" message (ECHO_REQUEST) to its upstream neighbour, i.e. its parent router on the tree. One keepalive message is sent to represent entries with the same parent, thereby improving scalability on links which are shared by many groups. On multicast capable links, a keepalive is multicast to the "all-cbt-routers" group (IANA assigned as 224.0.0.15); this has a suppressing effect on any other router for which the link is its parent link. If a parent link does not support multicast transmission, keepalives are unicast.

The receipt of a keepalive message over a valid child interface immediately prompts a response (ECHO_REPLY), which is either unicast or multicast, as appropriate.

The ECHO_REQUEST does not contain any group information; the ECHO_REPLY does, but only periodically. To maintain consistent information between parent and child, the parent periodically reports, in an ECHO_REPLY, all groups for which it has state, over each of its child interfaces for those groups. This group-carrying echo reply is not prompted explicitly by the receipt of an echo request message. A child is notified of the time to expect the next echo reply message containing group information in an echo reply prompted by a child's echo request. The frequency of parent group reporting is at the granularity of minutes.

It cannot be assumed all of the routers on a multi-access link have a uniform view of unicast routing; this is particularly the case when a multi-access link spans two or more unicast routing domains. This could lead to multiple upstream tree branches being formed (an error condition) unless steps are taken to ensure all routers on the link agree which is the upstream router for a particular group. CBT routers attached to a multi-access link participate in an explicit election mechanism that elects a single router, the designated router (DR), as the link's upstream router for all groups. Since the DR might not be the link's best next-hop for a particular core router, this may result in join messages being re-directed back across a multi-access link. If this happens, the re-directed join message is unicast across the link by the DR to the best next-hop, thereby preventing a looping scenario. This re-direction only ever applies to join messages. Whilst this is suboptimal for join messages, which are generated infrequently, multicast data never traverses a link more than once (either natively, or encapsulated).

In all but the exception case described above, all CBT control messages are multicast over multicast supporting links to the "all-cbt-routers" group, with IP TTL 1. The IP source address of CBT control messages is the outgoing interface of the sending router. The IP destination address of CBT control messages is either the "all-cbt-routers" group address, or the IP address of a router reachable over one of the sending router's interfaces, depending on whether the sender's outgoing link supports multicast transmission. All the necessary addressing information is obtained as part of tree set up.

If CBT is implemented over a tunnelled topology, when sending a CBT control packet over a tunnel interface, the sending router uses as the packet's IP source address the local tunnel end point address, and the remote tunnel end point address as the packet's IP destination address.

4. Protocol Specification Details

Details of the CBT protocol are presented in the context of a single router implementation.

4.1. CBT HELLO Protocol

The HELLO protocol is used to elect a designated router (DR) on broadcast-type links. It is also used to elect a designated border router (BR) when interconnecting a CBT domain with other domains (see [5]).

A router represents its status as a link's DR by setting the DR-flag on that interface; a DR flag is associated with each of a router's broadcast interfaces. This flag can only assume one of two values: TRUE or FALSE. By default, this flag is FALSE.

HELLO messages are multicast periodically to the all-cbt-routers group, 224.0.0.15, using IP TTL 1. The advertisement period is [HELLO_TIMER] seconds. [HELLO_TIMER] comprises a configured [HELLO_INTERVAL], to which is added [RND_RSP] seconds - a random response interval. This random response additive is required to avoid the potential problem of synchronisation between HELLO advertisements (or other control messages) from different routers. The HELLO protocol's convergence time is set at [HELLO_CONV] seconds - the time after which no further HELLOs are expected in any one round of the protocol.

Each HELLO advertising router includes the upper bound of its [RND_RSP] timer in its HELLO advertisements. This is necessary so that all routers attached to the link can agree on a common HELLO convergence time [HELLO_CONV]; in any one round of the HELLO protocol, a router assumes the minimum of the upper bound of its configured [RND_RSP] and that of any received advertisement's. The minimum upper bound is then used as this router's [RND_RSP] upper bound in the next round of the protocol. [HELLO_CONV] is set to this minimum upper bound + 2 seconds (the 2 seconds being a response "safety margin") for the next round of the protocol.

A network manager can preference a router's DR eligibility by optionally configuring a HELLO preference. Valid configuration values range from 1 to 254 (decimal), 1 representing the "most eligible" value. In the absence of explicit configuration, a router assumes the default HELLO preference value of 255. The elected DR uses HELLO preference zero (0) in HELLO advertisements, irrespective of any configured preference. The DR continues to use preference zero for as long as it is running.

The DR election winner is that which advertises the lowest HELLO preference, or the lowest-addressed in the event of a tie.

The situation where two or more routers attached to the same broadcast link are advertising HELLO preference 0 should never arise. However, should this situation arise, all but the lowest addressed zero-advertising router relinquishes its claim as DR immediately by unsetting the DR flag on the corresponding interface. The relinquishing router(s) subsequently advertise their previously used preference value in HELLO advertisements.

4.1.1. Sending HELLOs

When a router starts up, it multicasts two HELLO messages over each of its broadcast interfaces in successsion. The DR flag is initially unset (FALSE) on each broadcast interface.

A router sends a HELLO message whenever its [HELLO_TIMER] expires.

Whenever a router sends a HELLO message, it resets its [HELLO_TIMER].

4.1.2. Receiving HELLOs

On receipt of any HELLO message, a router adjusts its [RND_RSP] upper bound to the minimum of this router's configured [RND_RSP] upper bound and that received in the received HELLO. The router also adjusts its [HELLO_CONV] as described above.

A router need not respond to a HELLO message if the received HELLO is "better" than its own. Thus, in steady state, the HELLO protocol incurs very little traffic overhead.

If the received HELLO message is "better" (lower preferenced, or equally preferenced but lower addressed) than it would send itself, it immediately unsets its DR flag on the arriving interface if the DR flag is set on that interface. It also resets its [HELLO_TIMER].

If the received HELLO message is not "better" than this router would send itself, it sets its [RND_RSP] random response timer; on expiry, the router responds with its own HELLO message . If no "better" HELLO message is received within the current [HELLO_CONV], the router sets the DR flag on the corresponding interface.

4.2. JOIN_REQUEST Processing

A JOIN_REQUEST is the CBT control message used to register a member host's interest in joining the distribution tree for the group.

4.2.1. Sending JOIN_REQUESTs

A JOIN_REQUEST can only ever be originated by a leaf router, i.e. a router with directly attached member hosts. This join message is sent hop-by-hop towards the core router for the group (see section 7). The originating router caches <group, NULL, upstream interface> state for each join it originates. This state is known as "transient join state". The absence of a "downstream interface" (NULL) indicates that this router is the join message originator, and is therefore responsible for any retransmissions of this message if a response is not received within [JOIN_RTX_INTERVAL]. It is an error if no response is received after [JOIN_TIMEOUT] seconds. If this error condition occurs, the joining process may be re-invoked by the receipt of the next IGMP host membership report from a locally attached member host.

Note that if the interface over which a JOIN_REQUEST is to be sent supports multicast, the JOIN_REQUEST is multicast to the all-cbt-routers group, using IP TTL 1. If the link does not support multi-cast, the JOIN_REQUEST is unicast to the next hop on the unicast path to the group's core.

4.2.2. Receiving JOIN_REQUESTs

On broadcast links, JOIN_REQUESTs which are multicast may only be forwarded by the link's DR. Other routers attached to the link may process the join (see below). JOIN_REQUESTs which are multicast over a point-to-point link are only processed by the router on the link which does not have a local interface corresponding to the join's network layer (IP) source address. Unicast JOIN_REQUESTs may only be processed by the router which has a local interface corresponding to the join's network layer (IP) destination address.

With regard to forwarding a received JOIN_REQUEST, if the receiving router is not on-tree for the group, and is not the group's core router, the join is forwarded to the next hop on the path towards the core. The join is multicast, or unicast, according to whether the outgoing interface supports multicast. The router caches the following information with respect to the forwarded join: <group, downstream interface, upstream interface>.

If this transient join state is not "confirmed" with a join acknowledgement (JOIN_ACK) message from upstream, the state is timed out after 1.5 times [JOIN_RTX_INTERVAL].

If the receiving router is the group's core router, the join is "terminated" and acknowledged by means of a JOIN_ACK. Similarly, if the router is on-tree and the JOIN_REQUEST arrives over an interface that is not the upstream interface for the group, the join is acknowledged.

If [RND_RSP] pertaining to a JOIN_REQUEST is active (i.e. running), if a JOIN_REQUEST is received for the same group over that group's parent interface, cancel [RND_RSP] for the impending JOIN_REQUEST.

If this router has a cache-deletion-timer [CACHE_DEL_TIMER] running on the arrival interface for the group specified in a multicast join, the timer is cancelled.

If a multicast JOIN_REQUEST is received and the QUIT_TIME bit (see section 4.4.1) is set on the arrival interface for the specified group, unset the QUIT_TIME bit.

4.3. JOIN_ACK Processing

A JOIN_ACK is the mechanism by which an interface is added to a router's multicast forwarding cache; thus, the interface becomes part of the group distribution tree.

4.3.1. Sending JOIN_ACKs

The JOIN_ACK is sent over the same interface as the corresponding JOIN_REQUEST was received. The sending of the acknowledgement causes the router to add the interface to its child interface list in its forwarding cache for the group, if it is not already. If the router does not yet have active state for this group, this router must be the core router for the group; the core creates a forwarding cache entry and includes the interface in its child interface list, and sends the JOIN_ACK downstream.

A JOIN_ACK is multicast or unicast, according to whether the outgoing interface supports multicast transmission or not.

4.3.2. Receiving JOIN_ACKs

The group and arrival interface must be matched to a <group, ..., upstream interface> from the router's cached transient state. If no match is found, the JOIN_ACK is discarded. If a match is found, a CBT forwarding cache entry for the group is created, with "upstream interface" marked as the group's parent interface.

If "downstream interface" in the cached transient state is NULL, the JOIN_ACK has reached the originator of the corresponding JOIN_REQUEST; the JOIN_ACK is not forwarded downstream. If "downstream interface" is non-NULL, a JOIN_ACK for the group is sent over the "downstream interface" (multicast or unicast, accordingly). This interface is installed in the child interface list of the group's forwarding cache entry.

Once transient state has been confirmed by transferring it to the forwarding cache, the transient state is deleted.

4.4. QUIT_NOTIFICATION Processing

A CBT tree is "pruned" in the direction downstream-to-upstream whenever a CBT router's child interface list for a group becomes NULL.

4.4.1. Sending QUIT_NOTIFICATIONs

A QUIT_NOTIFICATION is sent to a router's parent router on the tree whenever the router's child interface list becomes NULL.

A QUIT_NOTIFICATION is not acknowledged; once sent, all information pertaining to the group it represents is deleted from the forwarding cache after a short interval.

To ensure consistency between a child and parent router given the potential for loss of a QUIT_NOTIFICATION, there is a QUIT_TIME bit associated with the parent of each group entry; whenever a QUIT_NOTIFICATION is sent for a group, the QUIT_TIME bit for that group entry is set for a maximum of [QUIT_TIME] seconds before the entry is deleted and the QUIT_TIME bit unset. By default, this bit is unset.

When the QUIT_TIME bit is set, if the router detects multicast traffic for the group arriving over a to-be-deleted parent interface (one over which a quit has recently been sent), the router sends another QUIT_NOTIFICATION over that interface. This is multicast, or unicast, as appropriate for the outgoing link. It continues to do so at [QUIT_RATE] second intervals so long as data continues to arrive, and provided [QUIT_TIME] has not yet expired.

If, after sending a QUIT_NOTIFICATION a multicast JOIN_REQUEST for the specified group arrives over the interface the quit was sent, the QUIT_TIME bit is immediately unset if it is set (any traffic arriving over the interface will be for/from another child router attached to the same link).

4.4.2. Receiving QUIT_NOTIFICATIONs

The group reported in the QUIT_NOTIFICATION must be matched with a forwarding cache entry. If no match is found, the QUIT_NOTIFICATION is ignored and discarded. If a match is found, if the arrival interface is a valid child interface in the group entry, how the router proceeds depends on whether the QUIT_NOTIFICATION was multicast or unicast.

If the QUIT_NOTIFICATION was unicast, the corresponding child interface is deleted from the group's forwarding cache entry, and no further processing is required.

If the QUIT_NOTIFICATION was multicast, and the arrival interface is a valid child interface for the specified group, the router sets a cache-deletion-timer [CACHE_DEL_TIMER].

Because this router might be acting as a parent router for multiple downstream routers attached to the arrival link, [CACHE_DEL_TIMER] interval gives those routers that did not send the QUIT_NOTIFICATION, but received it over their parent interface, the opportunity to ensure that the parent router does not remove the link from its child interface list.

Therefore, on receipt of a multicast QUIT_NOTIFICATION over a parent interface, a receiving router starts a random response interval timer which is set to [RND_RSP] seconds.

If a multicast JOIN_REQUEST is received over the same interface (parent) for the same group before this router's [RND_RSP] timer expires, it suppresses the multicasting of its own similar JOIN_REQUEST.

If a multicast JOIN_REQUEST is not received via the router's parent link before [RND_RSP] expires, a JOIN_REQUEST is multicast over the link for the previously quit group, with IP TTL 1.

4.5. ECHO_REQUEST Processing

The ECHO_REQUEST message allows a child to monitor reachability to its parent router for a group (or range of groups if the parent router is the parent for multiple groups). Group information is not carried in ECHO_REQUEST messages.

4.5.1. Sending ECHO_REQUESTs

Whenever a router creates a forwarding cache entry due to the receipt of a JOIN_ACK, the router begins the periodic sending of ECHO_REQUEST messages over its parent interface. The ECHO_REQUEST is multicast to the "all-cbt-routers" group over multicast-capable interfaces, and unicast to the parent router otherwise.

ECHO_REQUEST messages are sent at [ECHO_INTERVAL] second intervals. Whenever an ECHO_REQUEST is sent, [ECHO_INTERVAL] is reset.

If, for any echo-request sent to a parent, the expected response (ECHO_REPLY) is not forthcoming within [ECHO_RTX_INTERVAL], the echo request message is retransmitted. If no response is forthcoming within [ECHO_TIMEOUT] seconds, the router sends a FLUSH_TREE message over each of its child interfaces for the group, then removes all forwarding cache state for the group.

4.5.2. Receiving ECHO_REQUESTs

If a ECHO_REQUEST is received over any valid child interface, the receiving router responds with an ECHO_REPLY message over the same interface. This message is multicast to the "all-cbt-routers" group over multicast-capable interfaces, and unicast otherwise.

If a multicast ECHO_REQUEST message arrives via any valid parent interface, the router resets its [ECHO_INTERVAL] timer for that upstream interface, thereby suppressing the sending of its own ECHO_REQUEST over that upstream interface.

4.6. ECHO_REPLY Processing

ECHO_REPLY messages allow a child to monitor the reachability of its parent, and ensure the group state information is consistent between them.

4.6.1. Sending ECHO_REPLY messages

An ECHO_REPLY message is sent in direct response to receiving an ECHO_REQUEST message, provided the ECHO_REQUEST is received over any one of this router's valid child interfaces. Additionally, an ECHO_REPLY is sent periodically by a parent router over each of its child links, reporting all groups for which the link is its child.

ECHO_REPLY messages are unicast or multicast, as appropriate.

4.6.2. Receiving ECHO_REPLY messages

An ECHO_REPLY message must be received via a valid parent interface. When received, the child router resets its [ECHO_INTERVAL] timer for this upstream interface. The child router also caches the reported "group report interval" (seconds) - the time at which the next group carrying ECHO_REPLY will be sent by the parent router. Like [ECHO_INTERVAL], this is cached per upstream interface. If the group carrying ECHO_REPLY does not arrive shortly after "group report interval" has expired, a QUIT_NOTIFICATION is sent for each group for which the non-reporting router is the parent.

If this echo reply carries a list of groups, the child router must match all those of its forwarding cache entries for which the arrival interface is the upstream interface. If the parent router does not consider itself the parent router for group(s) which the child thinks is its parent, the child sends a FLUSH_TREE message downstream for each such group. If this router has directly attached members for any of the flushed groups, the receipt of an IGMP host membership report for any of those groups will prompt this router to rejoin the corresponding tree(s).

If the upstream router considers itself the parent for more groups than does the receiving router, this router sends a QUIT_NOTIFICATION for each of those groups for which the QUIT_TIME bit is set in the forwarding cache. Otherwise, the router takes no action.

4.7. FLUSH_TREE Processing

The FLUSH_TREE (flush) message is the mechanism by which a router invokes the tearing down of all its downstream branches for a particular group. The flush message is multicast to the "all-cbt-routers" group when sent over multicast-capable interfaces, and unicast otherwise.

4.7.1. Sending FLUSH_TREE messages

A FLUSH_TREE message is sent over each downstream (child) interface when a router has lost reachability with its parent router for the group (detected via ECHO_REQUEST and ECHO_REPLY messages). All group state is removed from an interface over which a flush message is sent.

4.7.2. Receiving FLUSH_TREE messages

A FLUSH_TREE message must be received over the parent interface for the specified group, otherwise the message is discarded.

The flush message must be forwarded over each child interface for the specified group.

Once the flush message has been forwarded, all state for the group is removed from the router's forwarding cache.

5. Timers and Default Values

This section provides a summary of the timers described above, together with their default values.

+o [HELLO_INTERVAL]: a base value making up the bulk of the interval between sending a HELLO message. Default: 60 seconds.

+o [RND_RSP]: router's random response interval. Default: 2 seconds.

+o [HELLO_TIMER]: (variable) interval between sending HELLO messages. [HELLO_TIMER] = [HELLO_INTERVAL + RND_RSP]

+o [HELLO_CONV]: convergence time of one round of the HELLO protocol. [HELLO_CONV] = [min(RND_RSP) + 2 seconds].

+o [JOIN_RTX_INTERVAL]: retransmission time for JOIN_REQUESTs. Default: 5 seconds.

+o [JOIN_TIMEOUT]: time to raise exception due to tree join failure. Default: 3.5 times [JOIN_RTX_INTERVAL].

+o [CACHE_DEL_TIMER]: time to remove child interface from forwarding cache. Default: 2 seconds.

+o [QUIT_TIME]: time to remove parent interface from forwarding cache entry. Unset QUIT_TIME bit. Default: 60 seconds.

+o [QUIT_RATE]: period for sending QUIT_NOTIFICATION if traffic persists. Default: 15 seconds.

+o [ECHO_INTERVAL]: interval between sending ECHO_REQUEST to parent routers. Default: 60 seconds.

+o [ECHO_RTX_INTERVAL]: retransmission time for ECHO_REQUESTs. Default 2 seconds.

+o [ECHO_TIMEOUT]: time to consider parent unreachable. Default: 3.5 times [ECHO_RTX_INTERVAL].

6. CBT Packet Formats and Message Types

CBT control packets are encapsulated in IP. CBT has been assigned IP protocol number 7 by IANA [4].

6.1. CBT Common Control Packet Header

All CBT control messages have a common fixed length header.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  vers | type  |  addr len     |         checksum              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1. CBT Common Control Packet Header

This CBT specification is version 2.

CBT packet types are:

+o   type 0: HELLO

+o   type 1: JOIN_REQUEST

+o   type 2: JOIN_ACK

+o   type 3: QUIT_NOTIFICATION

+o   type 4: ECHO_REQUEST

+o   type 5: ECHO_REPLY

+o   type 6: FLUSH_TREE

+o   type 7: Bootstrap Message

+o   type 8: Candidate Core Advertisement

+o   Addr Length: address length in bytes of unicast or multicast addresses carried in the control packet.

+o   Checksum: the 16-bit one's complement of the one's complement sum of the entire CBT control packet.

6.2. HELLO Packet Format

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                     CBT Control Packet Header                 |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | rnd response  |  Preference    |  reserved  |   option type   |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | option len   |               option value                     |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2. HELLO Packet Format

HELLO Packet Field Definitions:

+o rnd response: random response interval in seconds.

+o preference: sender's HELLO preference.

+o option type: the type of option present in the "option value" field. One option type is currently defined: option type 0 (zero) = BR_HELLO; option value 0 (zero); option length 0 (zero). This option type is used with HELLO messages sent by a border router (BR) as part of designated BR election (see [5]).

+o option len: length of the "option value" field in bytes.

+o option value: variable length field carrying the option value.

6.3. JOIN_REQUEST Packet Format

JOIN_REQUEST Field Definitions

+o group address: multicast group address of the group being joined. For a "wildcard" join (see [5]), this field contains the value of INADDR_ANY.

+o originating router: router that originated this JOIN_REQUEST.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    CBT Control Packet Header                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          group address                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                        originating router                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           target router                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  option type  |  option len   |         option value          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3. JOIN_REQUEST Packet Format

+o target router: target (core) router for the group.

+o option type: allows the specification of a variety of JOIN_REQUEST options. One option is currently defined: option type 0 (zero) = BR_JOIN; option length 0 (zero); option value 0 (zero). This option is used by a CBT domain border router to join an internal core for all groups that map to that core. The state instantiated by a JOIN_REQUEST with this option set is represents (*, core). For further details, see [5].

6.4. JOIN_ACK Packet Format

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                    CBT Control Packet Header                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                          group address                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           target router                       |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  option type  |  option len   |         option value          |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4. JOIN_ACK Packet Format

JOIN_ACK Field Definitions

+o group address: multicast group address of the group being joined.

+o target router: router (DR) that originated the corresponding JOIN_REQUEST.

6.5. QUIT_NOTIFICATION Packet Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    CBT Control Packet Header                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          group address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    originating child router                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5. QUIT_NOTIFICATION Packet Format

QUIT_NOTIFICATION Field Definitions

+o group address: multicast group address of the group being joined.

+o originating child router: address of the router that originates the QUIT_NOTIFICATION.

6.6. ECHO_REQUEST Packet Format

ECHO_REQUEST Field Definitions

+o originating child router: address of the router that originates the ECHO_REQUEST.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    CBT Control Packet Header                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    originating child router                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6. ECHO_REQUEST Packet Format

6.7. ECHO_REPLY Packet Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    CBT Control Packet Header                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    originating parent router                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     group report interval     |         num groups            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       group address #1                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       group address #2                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           ......                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       group address #n                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7. ECHO_REPLY Packet Format

ECHO_REPLY Field Definitions

+o oringinating parent router: address of the router originating this ECHO_REPLY.

+o group report interval: number of seconds until the sending router will send its next ECHO_REPLY containing a list of group addresses.

+o num groups: the number of groups being reported by this ECHO_REPLY.

+o group address: a list of multicast group addresses for which this router considers itself a parent router w.r.t. the link over which this message is sent.

6.8. FLUSH_TREE Packet Format

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   CBT Control Packet Header                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         group address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8. FLUSH_TREE Packet Format

FLUSH_TREE Field Definitions

+o group address: multicast group address of the group being "flushed".

7. Core Router Discovery

For intra-domain core discovery, CBT has decided to adopt the "bootstrap" mechanism currently specified with the PIM sparse mode protocol [2]. This bootstrap mechanism is scalable, robust, and does not rely on underlying multicast routing support to deliver core router information; this information is distributed via traditional unicast hop-by-hop forwarding.

It is expected that the bootstrap mechanism will be specified independently as a "generic" RP/Core discovery mechanism in its own separate document. It is unlikely at this stage that the bootstrap mechanism will be appended to a well-known network layer protocol, such as IGMP [3], though this would facilitate its ubiquitous (intra-domain) deployment. Therefore, each multicast routing protocol requiring the bootstrap mechanism must implement it as part of the multicast routing protocol itself.

A summary of the operation of the bootstrap mechanism follows (details are provided in [7]). It is assumed that all routers within the domain implement the "bootstrap" protocol, or at least forward bootstrap protocol messages.

A subset of the domain's routers are configured to be CBT candidate core routers. Each candidate core router periodically (default every 60 secs) advertises itself to the domain's Bootstrap Router (BSR), using "Core Advertisement" messages. The BSR is itself elected dynamically from all (or participating) routers in the domain. The domain's elected BSR collects "Core Advertisement" messages from candidate core routers and periodically advertises a candidate core set (CC-set) to each other router in the domain, using traditional hop-by-hop unicast forwarding. The BSR uses "Bootstrap Messages" to advertise the CC-set. Together, "Core Advertisements" and "Bootstrap Messages" comprise the "bootstrap" protocol.

When a router receives an IGMP host membership report from one of its directly attached hosts, the local router uses a hash function on the reported group address, the result of which is used as an index into the CC-set. This is how local routers discover which core to use for a particular group.

Note the hash function is specifically tailored such that a small number of consecutive groups always hash to the same core. Furthermore, bootstrap messages can carry a "group mask", potentially limiting a CC-set to a particular range of groups. This can help reduce traffic concentration at the core.

If a BSR detects a particular core as being unreachable (it has not announced its availability within some period), it deletes the relevant core from the CC-set sent in its next bootstrap message. This is how a local router discovers a group's core is unreachable; the router must re-hash for each affected group and join the new core after removing the old state. The removal of the "old" state follows the sending of a QUIT_NOTIFICATION upstream, and a FLUSH_TREE message downstream.

7.1. Bootstrap Message Format

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |             CBT common control packet header                  |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |      For full Bootstrap Message specification, see [7]        |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 9. Bootstrap Message Format

7.2. Candidate Core Advertisement Message Format

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |              CBT common control packet header                 |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |   For full Candidate Core Adv. Message specification, see [7] |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 10. Candidate Core Advertisement Message Format

8. Interoperability Issues

Interoperability between CBT and DVMRP is specified in [5].

Interoperability with other multicast protocols will be fully specified as the need arises.

Acknowledgements

Special thanks goes to Paul Francis, NTT Japan, for the original brainstorming sessions that brought about this work.

Others that have contributed to the progress of CBT include Ken Carlberg, Eric Crawley, Nitin Jain, Steven Ostrowsksi, Radia Perlman, Scott Reeve, Clay Shields, Sue Thompson, Paul White.

The participants of the IETF IDMR working group have provided useful feedback since the inception of CBT.

References

[1] Core Based Trees (CBT) Multicast Routing Architecture; A. Ballardie; ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-cbt-arch-**.txt. Working draft, 1997.

[2] Protocol Independent Multicast (PIM) Sparse Mode/Dense Mode; D. Estrin et al; ftp://netweb.usc.edu/pim Working drafts, 1996.

[3] Internet Group Management Protocol, version 2 (IGMPv2); W. Fenner; ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-igmp-v2-**.txt. Working draft, 1996.

[4] Assigned Numbers; J. Reynolds and J. Postel; RFC 1700, October 1994.

[5] CBT Border Router Specification for Interconnecting a CBT Stub Region to a DVMRP Backbone; A. Ballardie; ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-cbt-dvmrp-**.txt. Working draft, March 1997.

[6] Scalable Multicast Key Distribution; A. Ballardie; RFC 1949, July 1996.

[7] A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing; D. Estrin et al.; Technical Report; ftp://catarina.usc.edu/pim

Author Information:

Tony Ballardie,
Research Consultant,

e-mail: ABallardie@acm.org