Inter-Domain Multicast Routing (IDMR)                          A. Ballardie
INTERNET-DRAFT                                                    Consultant
                                                                     B. Cain
                                                                Bay Networks
                                                                    Z. Zhang
                                                                Bay Networks

August 1998

# Core Based Trees (CBT version 3) Multicast Routing

-- Protocol Specification --

## Status of this Memo

## Abstract

This document describes the Core Based Tree (CBT version 3) 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 may use a separate multicast routing table, or it may use that of underlying unicast routing, to establish paths between senders and receivers. The CBT architecture is described in [1].

This specification supercedes and obsoletes RFC 2189. Changes from

RFC 2189 include support for source specific joining and pruning to provide better CBT transit domain capability, new packet formats, and new robustness features. Section 1 documents the primary changes to RFC 2189.

This document is progressing through the IDMR working group of the IETF.  CBT related documents include [1, 2, 3, 5, 8]. For all IDMR-related documents, see http://www.cs.ucl.ac.uk/ietf/idmr.

TABLE OF CONTENTS

## 1. Changes from RFC 2189

+o forwarding cache support for entries of different granularities, i.e. (*, G), (*, Core), or (S, G), and support for S and/or G masks for representing S and/or G aggregates

+o included support for joins, quits (prunes), and flushes of different granularities, i.e. (*, G), (*, Core), or (S, G), where S and/or G can be aggregates

+o optional one-way join capability

+o improved the LAN HELLO protocol and included a state diagram

+o revised packet format, and provided option support for all control packets

+o  added downstream state timeout to CBT router

+o  revised the CBT "keepalive" mechanism between adjacent on-tree CBT routers

+o  overall provided added clarification of protocol events and mechanisms

Unfortunately, most of these changes are not backwards compatible with RFC 2189, but at the time of writing, these changes remain in advent of any widespread implementation or deployment.

## 2. Building a CBT Multicast Domain

When building a CBT multicast domain that attaches to other multicast domains, this document should be used in conjunction with draft-ietf-idmr-cbt-br-spec-**.txt, which describes the CBT Border Router Specification and discusses various issues related to CBT domain interconnection.

## 3. Introduction & Terminology

In CBT, a "core router" (or just "core") is a router which acts as a "meeting point" between a sender and group receivers. The term "rendezvous point (RP)" is used equivalently in some documents [2].

A router that is part of a CBT distribution tree is known as an "on-tree" router. An router which is on-tree for a group is one which has forwarding state for the group.

We refer to a broadcast interface as any interface that is multicast capable.

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.

## 4. 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.

### 4.1. The First Step: Joining the Tree

As a first step, a host first expresses its interest in joining a group by multicasting an IGMP host membership report [3] across its attached link. Note that all CBT routers, similar to other multicast protocol routers, are expected to participate in IGMP for the purpose of monitoring directly attached group memberships, and acting as IGMP querier should the need arise.

On receiving an IGMP Host Membership Report, a local CBT 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 8). This join message must be explicitly acknowledged (JOIN_ACK) either by the core router itself, or by another router that is on the path between the sending router and the core, which itself has already successfully joined the tree.

By default, joins/join-acks create bi-directional forwarding state, i.e. data can flow in the direction downstream -> upstream, or upstream -> downstream. In some circumstances a join/join-ack may include an option which instantiates uni-directional forwarding state; an interface over which a uni-directional join-ack is forwarded (not received) is automatically marked as pruned.  Data is permitted to be received via a pruned interface, but must not be forwarded over a pruned interface. Prune state can also be instantiated by the QUIT_NOTIFICATION message (see section 4.8).

A join-request is made uni-directional by the inclusion of the "uni-directional" join option (see section 7.2.1), which is copied to the corresponding join-ack; join-request options are always copied to the corresponding join-ack.

CBT now supports source specific joins/prunes so as to be better equipped when deployed in a transit domain; source specific control messages are only ever generated by CBT Border Routers (BRs). Source specific control messages follow G, not S, i.e. they are routed towards the core (not S) and no further.  Thus, (S, G) state only exists on the "core tree" in a CBT domain - those routers and links between a BR and core router.

## 4.2.  Transient State

The join message sets up transient join state in the router that originates it (a LAN's designated router (DR)) and the routers it traverses (an exception is described in section 4.9), and this state consists of <group, [source], downstream address, upstream address>; "source" is optional, and relevant only to source specific control messages.

On broadcast networks "downstream address" is the local IP address of the interface over which this router received the join (or IGMP Host Membership Report), and "upstream address" is the local IP address of the interface over which this router forwarded the join (according to this router's routing table). On non-broadcast networks "downstream address" is the IP address of the join's previous hop, and "upstream address" is the IP address of the next hop (according to this router's routing table). Transient state eventually times out unless the join is explicitly acknowledged. When a join is acknowledged, the transient join state is transferred to the router's multicast forwarding cache, thus becoming "permanent".

If "downstream address" implies a broadcast LAN, the transient state MUST be able to distinguish between a member host being reachable over that interface, and a downstream router being reachable over that interface.  This is necessary so that, on receipt of a JOIN_ACK, a router with transient state knows whether "downstream address" only leads to a group member, in which case the JOIN_ACK need not be forwarded, or whether "downstream address" leads to a downstream router that either originated or forwarded the join prior to this router receiving it, in which case this router must forward a received JOIN_ACK. Precisely how this distinction is made is implementation dependent. A router must also be able to distinguish these two conditions wrt its forwarding cache.

## 4.3. Getting "On-tree"

A router which terminates a JOIN_REQUEST (see section 4.8) sends a JOIN_ACK in response. A join acknowledgement (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.

A router is not considered "on-tree" until it has received a JOIN_ACK for a previously sent/forwarded JOIN_REQUEST, and has instantiated the relevant forwarding state.

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 parent router for a group becomes unreachable, the router (child) 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.

## 4.4. Pruning and Prune State

Any of a forwarding cache entry's children can be "pruned" by the immediate downstream router (child); in CBT, pruning is implemented by means of the QUIT_NOTIFICATION message, which is sent hop-by-hop in the direction: downstream --> upstream. A pruned child must be distinguishable from a non-pruned child - how is implementation dependent. One possible way would be to associate a "prune bit" with each child in the forwarding cache.

The granularity of a quit (prune) can be (*, G), (*, Core), or (S, G). (*, Core) and (S, G) prunes are only relevant to core tree branches, i.e. those routers between a CBT BR and a core (inclusive). (*, G) prunes are applicable anywhere on a CBT tree.

In previous versions of CBT, a quit was sent by a child router to cause its parent to remove it from the tree. Whilst this capability remains, in this version of CBT a quit can also be sent by a child to make the parent's forwarding state more specific.

Refer to section 4.8 for the procedures relating to receiving and forwarding a quit (prune) message.

Data is permitted to be received via a pruned interface, but must not be forwarded over a pruned interface. Thus, pruning is always uni-directional - it can stop data flowing downstream, but does not prevent data from flowing upstream.

CBT BRs are able to take advantage of this uni-directionality; if the BR does not have any directly attached group members, and is not serving a neighbouring domain with group traffic, it can elect not to receive traffic for the group which is sourced inside, or received via, the CBT domain. At the same time, if the BR is the ingress BR for a particular (*, G), or (S, G), externally sourced traffic for (*, G) or (S, G) need not be encapsulated by the ingress BR and unicast to the relevant core router - the BR can send the traffic using native IP multicast.

## 4.5. The Forwarding Cache

A CBT router MUST implement a multicast forwarding cache which supports source specific (i.e. (S, G)) as well as source independent (i.e. (*, G) and (*, Core)) entries. This forwarding cache is known as the router's private CBT forwarding cache, or PFC.

All implementations SHOULD also implement a shared (i.e. protocol independent) multicast forwarding cache - recommended in [8] to facilitate interoperability - which is only used by Border Routers and shared by all protocols operating on the Border Router (hence "shared"). This forwarding cache is known as the router's shared forwarding cache, or SFC. By having all CBT implementations support an SFC, any CBT router is eligible to become a Border Router.

(*, Core) entries are only relevant to a CBT PFC. This state is represented in the cache by specifying the core's IP unicast address in place of a group address/group address range.

Wrt representing groups (G's) in the forwarding cache, G may be an individual Class D 32-bit group address, or may be a prefix representing a contiguous range of group addresses (a group aggregate).

Similarly, for source specific PFC entries, S can be an aggregate. Therefore, the PFC SHOULD support the inclusion of masks or mask lengths to be associated with each of S and G.

In CBT, all PFC entries require that an entry's "upstream" interface is distinguishable as such - how is implementation dependent. CBT uses the term "parent" interchangeably with "upstream", and "child/children" interchangeably with "downstream". A core router's parent is always NULL.

Whenever the sending/receiving of a CBT join or prune results in the instantiation of more specific state in the router (e.g. (*, Core) state exists, then a (*, G) join arrives), the children of the new entry represent the union of the children from all other less specific forwarding cache entries, as well as the child (interface) over which the message was received (if not already included). This is so that at most a single forwarding cache entry need be matched with an incoming packet.

Note that in CBT, there is no notion of "expected" or "incoming" interface for (S, G) forwarding entries - these are treated just like (*, G) entries. Take the following example:

```
                 core
                   |\
                   | \
                   R1 \
                   |  R2 - s1
                   |   |
                   |   |
                   R4-R3 - g1
                   |
                   |
                   BR
```

Figure 1.

In figure 1 suppose R3 joins (*, g1) via the path R4 --> R1 --> core. BR joins (*, core) via the path R4 --> R1 --> core. BR issues a (s1, g1) QUIT_NOTIFICATION resulting in the instantiation of (s1, g1) between BR and the core. Thus, on R4 (s1, g1) and (*, g1) states exist. Assuming (s1, g1) state was instantiated on R4 AFTER (*, g1) state, R4's (s1, g1) child list comprises two interfaces, one pointing to BR, the other pointing to R3 (the latter copied from R4's (*, g1) entry). R4's (s1, g1) parent points towards R1.

Wrt R4's (s1, g1) entry, it is not possible for R4 to determine which is the correct incoming interface for s1 traffic, since R2 may send s1 traffic towards the core, or towards R3. Thus, R4 may receive (s1, g1) traffic via any of its on-tree interfaces, though R4 will not forward the traffic over a pruned child.

A forwarding cache entry whose children are ALL marked as pruned as a result of receiving quit messages may delete the entry provided there exists no less specific state with at least one non-pruned child.

## 4.6. Packet Forwarding

When a data packet arrives, the forwarding cache is searched for a best matching (according to longest match) entry. If no match is found the packet is discarded. If the packet arrived natively it is accepted if it arrives via an on-tree interface, i.e. any interface listed in a matching entry, otherwise the packet is discarded. Assuming the packet is accepted, a copy of the packet is forwarded over each other (outgoing) non-pruned interface listed in the matching entry.

If the packet arrived IP-in-IP encapsulated and the packet has reached its final destination, the packet is decapsulated and treated as described above, EXCEPT the packet need not have arrived via an on-tree interface according to the matching entry.

## 4.7. The "Keepalive" Protocol

The CBT forwarding state created by join/ack messages is soft state. This soft state is maintained by a separate "keepalive" mechanism rather than by join/ack refreshes.

The CBT "keepalive" mechanism operates between adjacent on-tree routers.  The keepalive mechanism is implemented by means of group specific ECHO_REQUEST and ECHO_REPLY messages, with the child routers responsible for periodically (explicitly) querying the parent router. The parent router (implicitly) monitors its children by expecting to periodically receive queries (ECHO_REQUESTs) from each child (per child router on non-broadcast networks; per child interface on broadcast networks).  The repeated absence of either an expected query

(ECHO_REQUEST) or expected response (ECHO_REPLY) results in the corresponding interface being marked as pruned in the router's forwarding cache. This constitutes a state timeout due to an exception condition. An interface can also be pruned in an explicit and timely fashion by means of either a QUIT_NOTIFICATION (downstream to upstream) or FLUSH_TREE (upstream to downstream) message.

Note that the network path comprising a CBT branch only changes due to connectivity failure.  An implementation could, however, invoke the tearing down and rebuilding of a tree branch whenever an underlying routing change occurs, irrespective of whether that change is due to connectivity failure. This is not CBT's default behaviour.

## 4.8. Control Message Precedence & Forwarding Criteria

When a router receives a CBT join or (quit) prune message, if the message contains state for which the receiving router has no matching (according to longest match) state in its forwarding cache, the receiving router creates a forwarding cache entry for the corresponding state and forwards the control message upstream.

CBT join and quit (prune) messages are forwarded as far upstream as the corresponding core router, or first router encountered with equally- or less specific state AND at least one other non-pruned child for that state.  Forwarding state corresponding exactly to the granularity of the join/quit is instantiated in all routers between the join/quit originator and join/quit terminator, inclusive. Take the following examples:

```
            core                        core
             |                           |
             |                           |
             R                           R
             |                          / \
             |                         /   \
            BR                       BR1   BR2

         Figure 2.                   Figure 3.
```

Assume in figure 2 BR has instantiated a priori (*, Core) state between itself and the core. It subsequently wishes to prune (*, G) from (*, Core) so sends a (*, G) QUIT_NOTIFICATION upstream. When the quit (prune) reaches router R, R already has less specific state (i.e. (*, Core)), but this quit message results in its only child interface (leading to BR) being marked as pruned under newly instantiated (*, G) state.  Since router R has no other child under either (*, Core) or (*, G) states, R can forward the received (*, G) quit.

Assume in figure 3 neither BR1 nor BR2 has a priori state between itself and the core. Assume that BR1 is explicitly notified by its neighbouring domain of group membership for G, causing BR1 to send a (*, G) (bi-directional) JOIN_REQUEST towards the core, via router R. R receives the join, instantiates (*, G) state, and forwards the join since it has no pre-existing equal- or less specific state.

Assume subsequently that BR2 is explicitly notified by its neighbouring domain of interest in (S, G) traffic. BR2 sends a (S, G) (bi-directional) JOIN_REQUEST towards the core, via router R. R receives the join, and already has less specific state (i.e. (*, G)) with one other child (leading to BR1). Thus, R instantiates (S, G) state (copying children from (*, G) state) and includes the child (pointing to BR2), but does not forward the (S, G) join due to the pre-existence of less specific state with one other non-pruned child.

A router with a forwarding cache entry whose children are ALL pruned can remove (delete) the corresponding entry UNLESS there exists less specific state with at least one non-pruned child.  If an entry is eligible for deletion, a quit representing the same granularity as the forwarding cache entry is sent upstream.

Returning to the example used with figure 2 above, when router R receives the (*, G) quit sent by BR and instantiates the corresponding state, R can send a (*, G) quit upstream since there are no less specific entries with _other_ non-pruned children. However, the (*, G) state in R cannot be removed despite all (*, G) children being pruned (in this e.g. there is only one child) because a less specific (i.e. (*, Core)) cache entry exists with a non-pruned child.

CBT flush messages are forwarded downstream removing all equally- and more specific state. A flush messsage is terminated by a leaf router, or a router with less specific state; the flush message does not affect the terminating router's less specific state.

## 4.9. Broadcast LANs

It cannot be assumed all of the routers on a broadcast link have a uniform view of unicast routing; this is particularly the case when a broadcast link spans two or more unicast routing domains. This could lead to multiple upstream tree branches being formed for any one group (an error condition) unless steps are taken to ensure all routers on the link agree on a single LAN upstream forwarding router. CBT routers attached to a broadcast link participate in an explicit election mechanism that elects a single router, the designated router (DR); the DR is a "join broker" for all LAN routers in so far as joins are routed according to the DR's view of routing - without a DR there could be conflicts potentially resulting in tree loops. The router that actually forwards a join off-LAN for a group (towards the group's core) is known as the LAN "upstream router" for that group. A group's LAN upstream router may or may not be the LAN DR.

With regards to a JOIN_REQUEST being multicast onto a broadcast LAN, the LAN DR decides over which interface to forward it. Depending on the group's core location, the DR may re-direct (unicast) the join back across the same link as it arrived to what it considers is the best next hop towards the core. In this case, the LAN DR does not keep any transient state for the JOIN_REQUEST it passed on. This best next hop router is then the LAN upstream forwarder for the corresponding group. This re-direction only applies to joins, which are relatively infrequent - native multicast data never traverses a link more than once.

For the case where a DR *originates* a join, and has to unicast it to a LAN neighbour, the DR MUST keep transient state for the join.

On broadcast LANs it is necessary for a router to be able distinguish between a directly attached (downstream) group member, and any (at least one) downstream on-tree router(s). For a router to be able to send a QUIT_NOTIFICATION (prune) upstream it must be sure it neither has any (downstream) directly attached group members or on-tree routers reachable via a downstream interface. How this is achieved is implementation-dependent. One possible way would be for a CBT forwarding cache to maintain 2 extra bits for each child entry - one bit to indicate the presence of a group member on that interface, the other bit indicating the presence of an on-tree router on that interface. Both these bits must be clear (i.e. unset) before this router can send a QUIT_NOTIFICATION for the corresponding state upstream.

## 4.10. The "all-cbt-routers" Group

The IP destination address of CBT control messages is either the "all-cbt-routers" group address, or a unicast address, as appropriate.

All CBT control messages are multicast over broadcast links to the "all-cbt-routers" group (IANA assigned as 224.0.0.15), with IP TTL 1. The exception to this is if a DR decides to forward a control packet back over the interface on which it arrived, in which the DR unicasts the control packet. The IP source address of CBT control messages is the sending router's outgoing interface.

CBT control messages are unicast over non-broadcast media.

A CBT control message originated or forwarded by a router is never processed by itself.

## 4.11. Non-Member Sending

This section is relevant to non-member sending where the data is sourced inside the CBT domain.

A host always originates native multicast data. All multicast traffic is received promiscuously by CBT routers. All but the LAN's designated router (DR) discard the packet. The DR looks up the relevant <core, group> mapping, encapsulates (IP-in-IP) the data, and unicasts it to the group's core router. Consequently, no group state is required in the network between the first hop router and the group's core.

On arriving at the core router, the data packet is decapsulated and disemminated over the group tree in the manner already described.

## 5. Protocol Specification Details

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

## 5.1. CBT HELLO Protocol

The HELLO protocol is used to elect a designated router (DR) on broadcast-type links.

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.

A network manager can preference a router's DR eligibility by optionally configuring an HELLO preference, which is included in the router's HELLO messages. 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.

HELLO messages are multicast periodically to the all-cbt-routers group, 224.0.0.15, using IP TTL 1. The advertisement period is specified by an hello timer, which is [HELLO_INTERVAL] seconds.

HELLO messages have a suppressing effect on those routers which would advertise a "lesser preference" in their HELLO messages; a router resets its hello timer if the received HELLO is "better" than its own. Thus, in steady state, the HELLO protocol incurs very little traffic overhead.

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.

### 5.1.1. 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.  This avoids the situation in which each router on a broadcast subnet believes it is the DR, thus preventing the multiple forwarding of join-requests should they arrive during this start up period.

If, after sending an HELLO message, no "better" HELLO message is received after HOLDTIME seconds, the router assumes the role of DR on the corresponding interface.  Whenever a router's status goes from non-DR to DR it immediately sends a zero preferenced HELLO message.

Once a router becomes DR on an interface, it should remain DR for as long as it is running (assuming a lower-addressed router on the same subnet does not advertise a zero-preferenced HELLO message).

A router sends an HELLO message whenever its hello timer expires, or its transition timer [DR_TRANS_TIMER] (if running) expires.  Whenever a router sends an HELLO message, it resets its hello timer.  The hello timer of the DR is [HELLO_INTERVAL] seconds. The hello timer of all other (non-DR) routers is [HELLO_INTERVAL] + rnd seconds, where "rnd" is a random interval between 1 and [HOLDTIME] seconds.

### 5.1.2. Receiving HELLOs

A router does not respond to an HELLO message if the received HELLO is "better" than its own, or equally preferenced but lower addressed. In this case, if the router has a transition timer [DR_TRANS_TIMER] running on the same interface, the timer is cancelled.

A router must respond to an HELLO message if that received is lesser preferenced (or equally preferenced but higher addressed) than would be sent by this router over the same interface. This response HELLO is sent immediately by the DR, or on expiry of an interval timer which is set between one and [HOLDTIME] seconds by non-DRs - this interval is known as the [DR_TRANS_TIMER] interval. Non-DRs cancel this transition timer if a better hello is received whilst this timer is running.

Figure 4 shows the state diagram for the HELLO protocol.

The following apply to the state diagram:

+o    for the DR, hello timer = HELLO_INTERVAL

+o    for non-DR(s), hello timer = HELLO_INTERVAL + rnd

+o    rnd = random delay timer between 1 and HOLDTIME seconds

+o    the DR always sends HELLO message with Preference zero

+o    trans timer ([DR_TRANS_TIMER]) is a transition timer, set to rnd

```
                                     Start O
                                           | A: Send 2 HELLO's
                                           | A: Start HOLDTIME
                                           V
             E: Recv better HELLO     **********----
             A: Reset hello timer     *  Init  *   | E: Recv worse HELLO
       -------------------------------*        *<---
       |                              **********
       |                                  |
       |                                  | E: HOLDTIME expires
       |                                  | A: Send HELLO
       |                                  | A: Reset hello timer
       |                                  V
       |                              **********
       |                          ----*        *----
       |     E: Rec worse HELLO   |   *   DR   *   | E: hello timer expires
       |     A: Send HELLO        |   *        *   | A: Send HELLO
       |     A: Reset hello timer --->**********<--- A: Reset hello timer
       |                                   |  ^
       |                                   |  | E: HOLDTIME expires
       |                                   |  | A: Send HELLO
       |                                   |  | A: Reset hello timer
       |              E: Recv better HELLO |  -------------------------------
       |              A: Reset hello timer |                              |
       |________________________________   |                              |
                                        |  |                              |
                                        |  |                              |
                 E: Recv better HELLO   |  |                              |
                 A: Cancel trans timer  V  V         E: Recv better HELLO   |
      ********** A: Reset hello timer *************  A: Reset hello timer
*******
      * Not DR *--------------------->*            *<-----------------------
*     *
      * & recd *                     *    Not DR  *                        *
DR  *
      * worse  *                     *            *
*wait *
      * hello  *<--------------------*            *-----------------------
>*     *
      ********** E: Recv worse HELLO  ************* E: hello timer expires
*******
           |     A: Start trans timer |       ^     A: Send HELLO            ^
           |                          |       |     A: Start HOLDTIME        |
           |                          ---------     A: Reset hello timer     |
           |                  E: Rec better HELLO                          |
           |                  A: Reset hello time                          |
           |                                                               |
           -----------------------------------------------------------------
       E: trans timer expires A: Send HELLO A: Start HOLDTIME A: Reset hello
timer
```

Figure 4: HELLO Protocol State Diagram

## 5.2. 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.

A JOIN_REQUEST can be of (*, G), (*, Core), or (S, G) granularity.

### 5.2.1. Sending JOIN_REQUESTs

A JOIN_REQUEST can be only be originated by a LAN designated router (DR), or by a CBT Border Router (BR). A join message cannot be sent by a router that is the core router for the group.

A join message is sent hop-by-hop towards the core router for the group (see section 8 - Core Router Discovery).

Refer to section 4.8 for the procedures relating to forward-ing/receiving a join message.

A router sending a join message caches <group, [source], downstream address, upstream address> state for each join sent/forwarded. This state is known as "transient join state". The router MUST be able to distinguish between reaching a group member host, or a router, or both, via its "Downstream address". How this is achieved is implemen-tation dependent (see section 4.9). A join originator is responsible for any retransmissions of this message if a response is not received within [RTX_INTERVAL]. Retransmissions are not generated by any router other than the join originator.

It is an error if no response is received after [JOIN_TIMEOUT] sec-onds. 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. IGMP host membership reporting may not be applicable to a CBT BR, and so it is recommended [JOIN_TIMEOUT] be extended to, for example, 3 times the default value (see section 6).

### 5.2.2. Receiving JOIN_REQUESTs

If a JOIN_REQUEST is eligible for forwarding upstream (see section 4.8), transient join state is created for this join (unless it already exists) and the join is forwarded upstream.

If this transient join state is not "confirmed" with a join acknowledgement (JOIN_ACK), the state is timed out after [TRANSIENT_TIMEOUT] seconds.

A join cannot be acknowledged by an on-tree router if the join arrives via the router's parent interface for the group. A router which originates an acknowledgment for a join never forwards the join further.

### 5.2.3. Additional Aspects Related to Receiving Multicast JOIN_REQUESTs

Some aspects related to receiving multicast joins have already been discussed in section 4.9.

In addition to that section, if a router receives a multicast join and the router has a  child interface deletion timer [CHILD_DEL_TIMER] running on the same interface that is equally- or less-specific than the received join, the timer is cancelled (see section 5.4.2).  This router acknowledges the received join.

## 5.3. JOIN_ACK Processing

A JOIN_ACK is the mechanism used by a router to confirm to a downstream router that the upstream router has instantiated the desired forwarding state.

A JOIN_ACK must be of the same granularity as the corresponding JOIN_REQUEST, and any JOIN_REQUEST options must be copied to the JOIN_ACK.  The downstream router receiving the join-ack converts its corresponding transient state to its forwarding cache, then removes the relevant transient state.

### 5.3.1. Sending JOIN_ACKs

A router which terminates a JOIN_REQUEST (see section 4.8) sends a JOIN_ACK in response.  A JOIN_ACK is sent over the same interface as the corresponding JOIN_REQUEST was received. Any options present in the join must be copied to the join-ack.

The sending of a JOIN_ACK - which inlcudes the "uni-directional" option - over a child results in the child being pruned.  The sending of a JOIN_ACK over a child that is marked as pruned results in that child being "un-pruned", unless the join-ack contains the uni-directional option.

### 5.3.2. Receiving JOIN_ACKs

An arriving JOIN_ACK must be matched to the corresponding <group, [source], downstream address, upstream address> 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 is created (or updated) by transferring the necessary transient join state to the router's forwarding cache. The interface over which the join-ack arrives becomes the entry's parent.

If the router's transient join state indicates that a router is present downstream, it forwards the join-ack accordingly. A join-ack is not forwarded downstream if this router's transient state indicates ONLY group member hosts reside downstream (as opposed to router(s)). An implementation SHOULD be able to distinguish these two conditions.

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

## 5.4. QUIT_NOTIFICATION Processing

A QUIT_NOTIFICATION (quit or prune) is both a means of improving, i.e. speeding up, group leave latency for CBT leaf routers, and a means for CBT Border Routers to elect not to receive traffic either from sources within, or via, the CBT domain.

A quit (prune) can be of (*, G), (*, Core), or (S, G) granularity.  A single quit message can carry information representing multiple different states.

### 5.4.1. Sending QUIT_NOTIFICATIONs

A CBT router *originates* a QUIT_NOTIFICATION of the relevant granularity when all children of a forwarding cache entry become pruned, AND there exists no less specific state with at least one _other_ non-pruned child.

Forwarding rules for a quit are explained in section 4.8.

A QUIT_NOTIFICATION is not acknowledged.

To help ensure consistency between a child and parent router given the potential for loss of a QUIT_NOTIFICATION, a total of [MAX_RTX] QUIT_NOTIFICATIONs are sent, each HOLDTIME seconds after the previous one.

### 5.4.2. Receiving QUIT_NOTIFICATIONs

The receipt of a valid QUIT_NOTIFICATION results in the arrival interface being marked as pruned. Rules regarding the forwarding of a received quit (prune) are explained in section 4.8.

If a quit is accepted and was unicast, the child via which the quit was received is added to the entry's child list (if not already), and immediately marked as pruned.

If the quit is accepted and was multicast, and the receiving router has pre-existing forwarding cache state of equal granularity, the router sets a child interface deletion timer [CHILD_DEL_TIMER] on the arrival interface with the same granularity.

Because this router might be acting as a parent router for multiple downstream routers attached to the arrival link, [CHILD_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 schedules an ECHO_REQUEST for the group for sending at a random interval between 0 (zero) and HOLDTIME seconds. The granularity of the echo MUST be equal or less specific than the received quit.

The receipt of an ECHO_REQUEST for the group by the parent router over a child interface on which [CHILD_DEL_TIMER] is running for the group, results in the timer being cancelled, provided the echo is equal or less specific than the granularity of the timer.

If the [CHILD_DEL_TIMER] expires, it implies no downstream on-tree router is present on that interface. If no group member is present on the same interface, the child can be marked as pruned in the relevant forwarding cache entry.

## 5.5. ECHO_REQUEST Processing

The ECHO_REQUEST/ECHO_REPLY messages constitute a "keepalive" mechanism which allows a group's child and parent routers to monitor each other's liveness.

ECHO_REQUESTs can be of (*, G), (*, Core), or (S, G) granularity. A single echo can carry information representing multiple different states.

The following timers are specifically relevant to the "keepalive" mechanism. The granularity of the timers corresponds the granularity of the state that is to be "kept alive", i.e. it can be (*, G), (*, Core), or (S, G), and is per interface: [ECHO_INTERVAL], [UPSTREAM_EXPIRE_TIME] (monitors parent interface), and [DOWNSTREAM_EXPIRE_TIME] (monitors child interface).

### 5.5.1. 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 granularity of the echo is equal to that of the sending router's forwarding cache entry, i.e. (*, G), (*, Core), or (S, G). An ECHO_REQUEST is multicast (224.0.0.15, TTL 1) or unicast, as appropriate.

ECHO_REQUEST messages are sent at [ECHO_INTERVAL] second intervals. To avoid undesirable synchronisation effects each of a host's interface's [ECHO_INTERVAL] timers includes a random response interval. Whenever an ECHO_REQUEST is sent, [ECHO_INTERVAL] is reset for each (*, G), or (*, Core), or (S, G), reported in the ECHO_REQUEST.

If no response is forthcoming, the upstream interface timer [UPSTREAM_EXPIRE_TIME] running on the upstream interface for the state reported in the ECHO_REQUEST will eventually expire. A FLUSH_TREE message is sent over all pruned and non-pruned children. The flush message reports the same state granularity as the echo for which no response was forthcoming.

### 5.5.2. Receiving ECHO_REQUESTs

Whenever an ECHO_REQUEST is received on an interface, if the router's interface is a parent interface for the reported state(s) it resets its [ECHO_INTERVAL] timer on that interface for those state(s), if appropriate. This implies that an ECHO_REQUEST which is multicast on a LAN suppresses the ECHO_REQUEST that is about to be sent by another router(s) for the same state(s) over the same interface.

If the router's receiving interface is a child interface for the reported state(s), it resets its [DOWNSTREAM_EXPIRE_TIME] timer on that interface for those state(s), if appropriate, and sends (multicast) an ECHO_REPLY reporting all states for which this router considers itself the parent wrt the child (interface).

Failure to receive an ECHO_REQUEST for a state(s) from a child after [DOWNSTREAM_EXPIRE_TIME] results in the immediate removal of the child from the relevant forwarding cache entry if the child is reachable via a non-broadcast network. If the child is reachable via a broadcast network, the expiry of [DOWNSTREAM_EXPIRE_TIME] results in the removal of the child from the router's relevant forwarding cache entry provided no group members are present on that interface.

## 5.6. ECHO_REPLY Processing

ECHO_REPLY messages are sent in immediate response to ECHO_REQUEST messages received over a valid child interface for the reported state(s). The ECHO_REPLY reports all state(s) for which this router considers itself the parent to the echo-requesting child.

If multiple states need reporting, one or more ECHO_REPLYs may be sent in response to a single ECHO_REQUEST, as necessary.

### 5.6.1. Sending ECHO_REPLY messages

An ECHO_REPLY message is sent in immediate response to receiving an ECHO_REQUEST message via one of this router's valid children for the reported state(s).  The ECHO_REPLY(s) contains a list of all states for which this router considers itself the parent to the child.

### 5.6.2. Receiving ECHO_REPLY messages

For each state reported in an ECHO_REPLY message received from a valid parent, the timers [UPSTREAM_EXPIRE_TIME] and [ECHO_INTERVAL] are refreshed for the reported states.

Failure to receive the relevant ECHO_REPLY [HOLDTIME] seconds after sending an ECHO_REQUEST results in the corresponding ECHO_REQUEST being resent. An ECHO_REQUEST can be resent a maximum of [MAX_RTX] times. If no response is forthcoming, the corresponding state(s) is removed from the parent after [UPSTREAM_EXPIRE_TIME] seconds, and a FLUSH_TREE message is sent over each of the children represented by the state(s).

[Note: 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).]

## 5.7. 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.

A flush can be of (*, G), (*, Core), or (S, G) granularity.  A single flush message can carry information representing multiple different states.

### 5.7.1. Sending FLUSH_TREE messages

A FLUSH_TREE message is sent over all pruned and non-pruned children whenever a router loses connectivity to its parent.

Once a flush message(s) has been sent, the relevant forwarding cache entry/entries are deleted.

### 5.7.2. Receiving FLUSH_TREE messages

CBT flush messages are forwarded downstream removing all equally- and more specific state. A flush messsage is terminated by a leaf router, or a router with less specific state; the flush message does not affect the terminating router's less specific state.

## 6. Timers and Default Values

This section provides a summary of the timers described above, together with their recommended default values. Other values may be configured; if so, the values used should be consistent across all CBT routers attached to the same network.

+o [HELLO_INTERVAL]: the interval between sending an HELLO message. Default: 60 seconds.

+o [HELLO_PREFERENCE]: Default: 255.

+o [HOLDTIME]: generic response interval. Default: 3 seconds.

+o [DR_TRANS_TIMER]: random delay timer used in transition from non-DR to DR. Default: delay set at between 1 and [HOLDTIME] seconds.

+o [MAX_RTX]: default maximum number of retransmissions. Default 3.

+o [RTX_INTERVAL]: message retransmission time. Default: 5 seconds.

+o [JOIN_TIMEOUT]: raise exception due to tree join failure. Default: (3.5*[RTX_INTERVAL]) seconds.

+o [TRANSIENT_TIMEOUT]: delete (unconfirmed) transient state. Default: [JOIN_TIMEOUT] seconds.

+o [CHILD_DEL_TIMER]: remove child interface from forwarding cache. Default: (1.5*HOLDTIME) seconds.

+o [UPSTREAM_EXPIRE_TIME]: time to send a QUIT_NOTIFICATION to our non-responding parent. Default: ([MAX_RTX]*[RTX_INTERVAL] + [HOLD-TIME]) seconds.

+o [DOWNSTREAM_EXPIRE_TIME]: not heard from child, time to remove child interface. Default: ([ECHO_INTERVAL] + [UPSTREAM_EXPIRE_TIME]) seconds.

+o [ECHO_INTERVAL]: interval between sending ECHO_REQUEST to parent routers. Default: 60 + rnd seconds, where "rnd" is between 0 and [HOLDTIME] seconds.

## 7.  CBT Packet Formats and Message Types

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

### 7.1.  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 5. CBT Common Control Packet Header

This CBT specification is version 3.

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 (optional)

+o    type 8: Candidate Core Advertisement (optional)

+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.

## 7.2. **Packet Format for CBT Control Packet Types 0 - 6**

A CBT control packet is divided into 3 parts:

+o    Common Control Packet Header,

+o    Control Packet Payload, and

+o    Control Packet Option(s).

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
   |                    CBT Control Packet Header                  | Header
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
   |Payload Length |  # of options |           reserved            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
   |                           address #1                          | Packet
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
   |                           address #2                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           address #n                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
   |  option type  |  option len   |        option value...        | Option(s)
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6. CBT Control Packet Format for Types 0 - 6.

Control Packet Field Definitions:

+o    # Payload Length: the length of the CBT control packet payload,
     excluding the common control packet header and option(s).

+o    # of options: the number of distinct options (as defined by option
     type) carried in this control packet.

+o    address #n: control packet payload address(es). Different control
     packet types can carry addresses (multicast and/or unicast) as
     their payload (e.g. JOIN_REQUESTs), and some control packet types

carry no addresses in the payload (e.g. HELLOs).

+o option type: unique option identifier.

+o option len: option length. The number of bytes consumed by this option's value.

+o option value: variable length option value.

NOTE: all control messages are padded to a 32-bit boundary.

### 7.2.1. Option Type Definitions

+o type 1: Hello Preference. Applicable only to HELLO packets to denote this HELLO packet's preference value. This option consumes 1 byte of "option value".

+o type 2: Uni-directional. Applicable only to JOIN_REQUESTs to indicate a uni-directional join.

+o type 3: Inclusion List. Enables the reporting of a contiguous set of groups using a group mask, for which this control message should apply.  The mask is represented by an 8-bit "masklen" field which is always included as the first 8 bits of this option's value.  One or more group prefixes follow, each padded out (zeroed) to 32 bits.

+o type 4: Exclusion List. This option allows for the reporting of group(s) to be exempted from the set reported elsewhere in this control packet.  A contiguous range of groups may be specified using a group mask.  The mask is represented by an 8-bit "masklen" field which is always included as the first 8 bits of this option's value.  One or more group prefixes follow, each padded out (zeroed) to 32 bits.

+o type 5: Source Information. This option enables a control message to specify source(s) to be associated with a group(s) carried elsewhere in the control message; if this option is specified as the first option after the control packet payload, the source information applies to the group specified in the payload. If this source information is to apply to a group aggregate (as specified by option type 3), the option specifying the group prefix MUST appear immediately before this option.

A source aggregate (prefix) may be specified using a source mask. The mask is represented by an 8-bit "masklen" field which is always included as the first 8 bits of this option's value. The source (prefix) follows, padded out (zeroed) to 32 bits.

### 7.2.2. Sample Control Packets

This section shows some sample constructions of a selection of different CBT control packet types.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
|   3   |   0   |       4       |            Checksum           | Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
|       4       |       1       |            reserved           | Payload
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
|       1       |       1       |  Preference   |    Padding    | Option
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 7. Sample HELLO packet

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
|   3   |   1   |       4       |            Checksum           | Header
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
|       16      |       0       |            reserved           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
|                         Group Address                         | Packet
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
|                          Core Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Join-Originating DR                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 8. Sample (*, G) JOIN_REQUEST (no options included)

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
    |   3   |   2   |       4       |           Checksum            | Header
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
    |       12      |       0       |           reserved            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
    |                        Group Address                          | Packet
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
    |        Join Originating DR (copied from JOIN_REQUEST)         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 9. Sample (*, G) JOIN_ACK (no options included)

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
    |   3   |   1   |       4       |           Checksum            | Header
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
    |       16      |       1       |           reserved            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
    |                        Group Address                          | Packet
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
    |                         Core Address                          |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Join-Originating DR                      |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
    |       5       |       5       |      24       | Src Addr Pfx..| Option
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |............. Src Addr Prefix .............    |    Padding    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 10. Sample (S, G) 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
   |   3   |   4   |       4       |           Checksum            | Header
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
   |   8 + (n x 4) |       0       |           reserved            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
   |                ECHO_REQUEST Originating Router                | Packet
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
   |                           Address #1                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Address #2                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Address #n                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 11. Sample 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Common
   |   3   |   5   |       4       |           Checksum            | Header
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
   |   8 + (n x 4) |       0       |           reserved            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Control
   |                  ECHO_REPLY Originating Router                | Packet
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ Payload
   |                           Address #1                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Address #2                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           Address #n                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ --------
```

Figure 12. Sample ECHO_REPLY

## 8. **Core Router Discovery**

There are two available options for CBTv2 core discovery; the "boot-strap" mechanism (as currently specified with the PIM sparse mode protocol [2]) is applicable only to intra-domain core discovery, and allows for a "plug & play" type operation with minimal configuration.

The disadvantage of the bootstrap mechanism is that it is much more difficult to affect the shape, and thus optimality, of the resulting distribution tree. Also, to be applicable, all CBT routers within a domain must implement the bootstrap mechanism.

The other option is to manually configure leaf routers with <core, group> mappings (note: leaf routers only); this imposes a degree of administrative burden - the mapping for a particular group must be coordinated across all leaf routers to ensure consistency. Hence, this method does not scale particularly well. However, it is likely that "better" trees will result from this method, and it is also the only available option for inter-domain core discovery currently available.

## 8.1. "Bootstrap" Mechanism Overview

It is unlikely 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 [6]). 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.

## 8.2. 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 [6]        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 13. Bootstrap Message Format

## 8.3. 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 [6] |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14. Candidate Core Advertisement Message Format

Acknowledgements

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

The use of a single core model since CBTv2 owes much to Clay Shields and his work on Ordered CBT (OCBT) [7]. Clay identified and proved several failure modes of CBT(v1) as it was specified with multiple cores, and also suggested using an unreliable quit mechanism, which has appeared since the CBTv2 specification as the QUIT_NOTIFICATION. Clay also provided more general constructive comments on the CBT architecture and specification.

Others that have contributed to the progress of CBT include Ken Carl-berg, Eric Crawley, Jon Crowcroft, Bill Fenner, Mark Handley, Ahmed Helmy, Nitin Jain, Alan O'Neill, Steven Ostrowsksi, Radia Perlman, Scott Reeve, Benny Rodrig, Martin Tatham, Dave Thaler, Sue Thompson, Paul White, and other participants of the IETF IDMR working group.

Thanks also to 3Com Corporation and British Telecom (BT) Plc for assisting with funding this work.

Finally, thanks to Graeme Brown, BT Labs UK, for his ongoing imple-mentation effort porting CBT to FreeBSD. For further information on this implementation contact <graeme.brown@bt-sys.bt.co.uk>, Alan O'Neill <alan.oneill@bt-sys.bt.co.uk>, or Tony Ballardie <ABal-lardie@acm.org>.

## References

[1] Core Based Trees (CBT) Multicast Routing Architecture; A. Ballardie; RFC 2201; ftp://ds.internic.net/rfc/rfc2201.txt.

[2] Protocol Independent Multicast (PIM) Sparse Mode/Dense Mode; D. Estrin et al; http://netweb.usc.edu/pim  RFC XXXX and Working drafts.

[3] Internet Group Management Protocol, version 2 (IGMPv2); W. Fenner; ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-igmp-v2-08.txt. Working draft, 1998.

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

[5] CBT Multicast Border Router Specification; A. Ballardie, B. Cain, Z. Zhang; ftp://ds.internic.net/internet-drafts/draft-ietf-idmr-cbt-br-spec-**.txt.  Working draft, March 1998.

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

[7] The Ordered Core Based Tree Protocol; C. Shields and J.J. Garcia-Luna-Aceves; In Proceedings of IEEE Infocom'97, Kobe, Japan, April 1997; http://www.cse.ucsc.edu/research/ccrg/publications/info-comm97ocbt.ps.gz

[8] Interoperability Rules for Multicast Routing Protocols; D. Thaler; ftp://ds.internic.net/internet-drafts/draft-thaler-multicast-interop-01.txt; March 1997.

Author Information:

Tony Ballardie,
Research Consultant,

e-mail: ABallardie@acm.org

Brad Cain,
Bay Networks Inc.,
3, Federal Street,
Billerica, MA 01821, USA.
e-mail: bcain@baynetworks.com
voice: +1 978 916 1316

Zhaohui "Jeffrey" Zhang,
Argon Networks Inc.,
25, Porter Road,
Littleton, MA 01460, USA.
Phone: +1 (978) 392 4681
e-mail: zzhang@argon.com