A. Ballardie
University College London
S. Reeve & N. Jain
Bay Networks, Inc.

September 1996

Core Based Trees (CBT) Multicast

-- Protocol Specification --

Status of this Memo

Abstract

This document describes the Core Based Tree (CBT) network layer multicast protocol. CBT is a next-generation multicast protocol that makes use of a shared delivery tree rather than separate per-sender trees utilized by most other multicast schemes [1, 2, 3]. The CBT architecture is described in [4a].

This specification includes an optimization whereby unencapsulated (native) IP-style multicasts are forwarded by CBT routers, resulting in very good forwarding performance. This mode of operation is called CBT "native mode". Native mode can only be used in CBT-only domains (footnote 1).

This revision contains two appendices; Appendix A describes simple CBT add-on mechanisms for dynamically migrating a CBT tree to one whose core is directly attached to a source's subnetwork, thereby allowing CBT to emulate shortest-path trees. Appendix B describes a group state aggregation scheme.

This document is progressing through the IDMR working group of the IETF. CBT related documents include [4, 5]. For all IDMR-related documents, see http://www.cs.ucl.ac.uk/ietf/idmr.

NOTE that core placement and management is not discussed in this document.

1. Changes since Previous Revision (05)

This note summarizes the changes to this document since the previous revision (revision 05).

+o inclusion of "first hop router" and "primary core" fields in the CBT mode data packet header.

+o removal of the term "non-core" router, replaced by "on-tree" router.

+o removal of the term "default DR (D-DR)", replaced simply by DR.

+o inclusion of T and S bits in the CBT control and data packet headers (type of service, and security, respectively).

+o CBT control messages are now carried directly over IP rather than UDP (for all implementations).

+o inclusion of an Appendix (A) describing extensions to the CBT protocol to achieve dynamic source-migration of core routers for shortest-path tree emulation.

+o inclusion of an Appendix (B) describing a group state aggregation scheme.

---

1 The term "domain" should be considered synonymous with "routing domain" throughout, as are the terms "region" and "cloud".

+o editorial changes and some re-organisation throughout for extra clarity.

2. Some Terminology

In CBT, the core routers for a particular group are categorised into PRIMARY CORE, and NON-PRIMARY (secondary) CORES.

The "core tree" is the part of a tree linking all core routers of a particular group together.

On-tree routers are those with a forwarding database entry for the corresponding group.

3. Protocol Specification

3.1. Tree Joining Process -- Overview

A CBT router is notified of a local host's desire to join a group via IGMP [6]. We refer to a CBT router with directly attached hosts as a "leaf CBT router", or just "leaf" router.

The following CBT control messages come into play subequent to a subnet's CBT leaf router receiving an IGMP membership report (also termed "IGMP join"):

+o JOIN_REQUEST

+o JOIN_ACK

If the CBT leaf router is the subnet's designated router (see next section), it generates a CBT join-request in response to receiving an IGMP group membership report from a directly connected host. The CBT join is sent to the next-hop on the unicast path to a target core, specified in the join packet; a router elects a "target core" based on a static configuration. If, on receipt of an IGMP-join, the locally-elected DR has already joined the corresponding tree, then it need do nothing more with respect to joining.

The join is processed by each such hop on the path to the core, until either the join reaches the target core itself, or hits a router that is already part of the corresponding distribution tree (as identified by the group address). In both cases, the router concerned terminates the join, and responds with a join-ack (join acknowledgement), which traverses the reverse-path of the corresponding join. This is possible due to the transient path state created by a join traversing a CBT router. The ack fixes that state.

3.2. DR Election

Multiple CBT routers may be connected to a multi-access subnetwork. In such cases it is necessary to elect a subnetwork designated router (DR) that is responsible for generating and sending CBT joins upstream, on behalf of hosts on the subnetwork.

CBT DR election happens "on the back" of IGMP [6]; on a subnet with multiple multicast routers, an IGMP "querier" is elected as part of IGMP. At start-up, a multicast router assumes no other multicast routers are present on its subnetwork, and so begins by believing it is the subnet's IGMP querier. It sends a small number IGMP-HOST-MEMBERSHIP-QUERYs in short succession in order to quickly learn about any group memberships on the subnet. If other multicast routers are present on the same subnet, they will receive these IGMP queries; a multicast router yields querier duty as soon as it hears an IGMP query from a lower-addressed router on the same subnetwork.

The CBT DR is always the subnet's IGMP querier (footnote 2). As a result, there is no protocol overhead whatsoever associated with electing a CBT D-DR.

3.3. Tree Joining Process -- Details

The receipt of an IGMP group membership report by a CBT DR for a CBT group not previously heard from triggers the tree joining process; the DR unicasts a JOIN-REQUEST to the first hop on the (unicast) path to the target core specified in the CBT join packet.

---

2 Or lowest addressed CBT router if the subnet's IGMP querier is non-CBT capable.

Each CBT-capable router traversed on the path between the sending DR and the core processes the join. However, if a join hits a CBT router that is already on-tree, the join is not propogated further, but acknowledged downstream from that point.

JOIN-REQUESTs carry the identity of all the cores associated with the group. Assuming there are no on-tree routers in between, once the join (subcode ACTIVE_JOIN) reaches the target core, if the target core is not the primary core (as indicated in a separate field of the join packet) it first acknowledges the received join by means of a JOIN-ACK, then sends a JOIN-REQUEST, subcode REJOIN-ACTIVE, to the primary core router.

If the rejoin-active reaches the primary core, it responds by sending a JOIN-ACK, subcode PRIMARY-REJOIN-ACK, which traverses the reverse-path of the join (rejoin). The primary-rejoin-ack serves to confirm no loop is present, and so explicit loop detection is not necessary.

If some other on-tree router is encountered before the rejoin-active reaches the primary, that router responds with a JOIN-ACK, subcode NORMAL. On receipt of the ack, subcode normal, the router sends a join, subcode REJOIN-NACTIVE, which acts as a loop detection packet (see section 8.3). Note that loop detection is not necessary subsequent to receiving a join-ack with subcode PRIMARY-REJOIN-ACK.

To facilitate detailed protocol description, we use a sample topology, illustrated in Figure 1 (shown over). Member hosts are shown as individual capital letters, routers are prefixed with R, and subnets are prefixed with S.

```
           A                               B
           |   S1              S4          |
   -------------------      -----------------------------------------------
             |                     |               |               |
           ------                 ------           ------           ------
           | R1 |                 | R2 |           | R5 |           | R6 |
           ------                 ------           ------           ------
      C     |  |                    |                |                 |
      |     |  |                    |    S2          |            S8   |
   ----------  ------------------------------------------        -------------
        S3                 |
                         ------
                         | R3 |
                 |       ------                       D
   | S9          |         |               S5         |
   |             |      ---------------------------------------------
   |  |----|     |                    |
   ---| R7 |-----|                  ------
   |  |----|     |------------------| R4 |
   |          S7 |                  ------            F
   |             |                    |         S6    |
   |-E           |            ---------------------------------
                      |                       |
                      |                     ------
             |---|    |---------------------| R8 |
             |R12 ----|                     ------      G
             |---|    |                       |         |  S10
                      | S14                ----------------------------
                      |                         |
                  I --|                       ------
                      |                       | R9 |
                                              ------
                                                |         S12
                     |             ----------------------------
                 S15 |                        |
                     |                      ------
                     |----------------------|R10 |
                J ---|                      ------      H
                     |                        |         |
                     |             ----------------------------
                     |                           S13
```

Figure 1. Example Network Topology

Taking the example topology in figure 1, host A wishes to join group G. All subnets' routers have been configured to use core routers R4 (primary core) and R9 (secondary core) for a range of group addresses, including G.

Router R1 receives an IGMP host membership report, and proceeds to unicast a JOIN-REQUEST, subcode ACTIVE-JOIN to the next-hop on the path to R4 (R3), the target core. R3 receives the join, caches the necessary group information (transient state), and forwards it to R4 -- the target of the join.

R4, being the target of the join, sends a JOIN_ACK (subcode NORMAL) back out of the receiving interface to the previous-hop sender of the join, R3. A JOIN-ACK, like a JOIN-REQUEST, is processed hop-by-hop by each router on the reverse-path of the corresponding join. The receipt of a join-ack establishes the receiving router on the corresponding CBT tree, i.e. the router becomes part of a branch on the delivery tree. Finally, R3 sends a join-ack to R1. A new CBT branch has been created, attaching subnet S1 to the CBT delivery tree for the corresponding group.

For the period between any CBT-capable router forwarding (or originating) a JOIN_REQUEST and receiving a JOIN_ACK the corresponding router is not permitted to acknowledge any subsequent joins received for the same group; rather, the router caches such joins till such time as it has itself received a JOIN_ACK for the original join. Only then can it acknowledge any cached joins. A router is said to be in a "pending-join" state if it is awaiting a JOIN_ACK itself.

Note that the presence of asymmetric routes in the underlying unicast routing does not affect the tree-building process; CBT tree branches are symmetric by the nature in which they are built. Joins set up transient state (incoming and outgoing interface state) in all routers along a path to a particular core. The corresponding join-ack traverses the reverse-path of the join as dictated by the transient state, and not necessarily the path that underlying routing would dictate. Whilst permanent asymmetric routes could pose a problem for CBT, transient asymmetricity is detected by the CBT protocol.

3.4. Forwarding Joins on Multi-Access Subnets

The DR election mechanism does not guarantee that the DR will be the router that actually forwards a join off a multi-access network; the first hop on the path to a particular core might be via another router on the same subnetwork, which actually forwards off-subnet.

Although very much the same, let's see another example using our example topology of figure 1 of a host joining a CBT tree for the case where more than one CBT router exists on the host subnetwork.

B's subnet, S4, has 3 CBT routers attached. Assume also that R6 has been elected IGMP-querier and CBT DR.

R6 (S4's DR) receives an IGMP group membership report. R6's configured information suggests R4 as the target core for this group. R6 thus generates a join-request for target core R4, subcode ACTIVE_JOIN. R6's routing table says the next-hop on the path to R4 is R2, which is on the same subnet as R6. This is irrelevant to R6, which unicasts it to R2. R2 unicasts it to R3, which happens to be already on-tree for the specified group (from R1's join). R3 therefore can acknowledge the arrived join and unicast the ack back to R2. R2 forwards it to R6, the origin of the join-request.

If an IGMP membership report is received by a DR with a join for the same group already pending, or if the DR is already on-tree for the group, it takes no action.

3.5. On-Demand "Core Tree" Building

The "core tree" - the part of a CBT tree linking all of its cores together, is built on-demand. That is, the core tree is only built subsequent to a non-primary (secondary) core receiving a join-request. This triggers the secondary core to join the primary core; the primary need never join anything.

Join-requests carry an list of core routers (and the identity of the primary core in its own separate field), making it possible for the secondary cores to know where to join when they themselves receive a join. Hence, the primary core must be uniquely identified as such across the whole group. A secondary joins the primary subsequent to sending an ack for the first join it receives.

3.6. Tree Teardown

There are two scenarios whereby a tree branch may be torn down:

+o During a re-configuration. If a router's best next-hop to the specified core is one of its existing children, then before sending the join it must tear down that particular downstream branch. It does so by sending a FLUSH_TREE message which is processed hop-by-hop down the branch. All routers receiving this message must process it and forward it to all their children. Routers that have received a flush message will re-establish themselves on the delivery tree if they have directly connected subnets with group presence.

+o If a CBT router has no children it periodically checks all its directly connected subnets for group member presence. If no member presence is ascertained on any of its subnets it sends a QUIT_REQUEST upstream to remove itself from the tree.

The receipt of a quit-request triggers the receiving parent router to immediately query its forwarding database to establish whether there remains any directly connected group membership, or any children, for the said group. If not, the router itself sends a quit-request upstream.

The following example, using the example topology of figure 1, shows how a tree branch is gracefully torn down using a QUIT_REQUEST.

Assume group member B leaves group G on subnet S4. B issues an IGMP HOST-MEMBERSHIP-LEAVE (relevant only to IGMPv2 and later versions) message which is multicast to the "all-routers" group (224.0.0.2). R6, the subnet's DR and IGMP-querier, responds with a group-specific-QUERY. No hosts respond within the required response interval, so DR assumes group G traffic is no longer wanted on subnet S4.

Since R6 has no CBT children, and no other directly attached subnets with group G presence, it immediately follows on by sending a QUIT_REQUEST to R2, its parent on the tree for group G. R2 responds with a QUIT-ACK, unicast to R6; R2 removes the corresponding child information. R2 in turn sends a QUIT upstream to R3 (since it has no other children or subnet(s) with group presence).

NOTE: immediately subsequent to sending a QUIT-REQUEST, the sender removes the corresponding parent information, i.e. it does not wait for the receipt of a QUIT-ACK.

R3 responds to the QUIT by unicasting a QUIT-ACK to R2. R3 subsequently checks whether it in turn can send a quit by checking group G presence on its directly attached subnets, and any group G children. It has the latter (R1 is its child on the group G tree), and so R3 cannot itself send a quit. However, the branch R3-R2-R6 has been removed from the tree.

4. Tree Maintenance

Once a tree branch has been created, i.e. a CBT router has received a JOIN_ACK for a JOIN_REQUEST previously sent (or forwarded), a child router is required to monitor the status of its parent/parent link at fixed intervals by means of a "keepalive" mechanism operating between them. The "keepalive" protocol is simple, and implemented by means of two CBT control messages: CBT_ECHO_REQUEST and CBT_ECHO_REPLY; a child unicasts a CBT-ECHO-REQUEST to its parent, which unicasts a CBT-ECHO-REPLY in response.

Adjacent CBT routers only need to send one keepalive representing all children having the same parent, reachable over a particular link, regardless of group. This aggregation strategy is expected to conserve considerable bandwidth on "busy" links, such as transit network, or backbone network, links.

For any CBT router, if its parent router, or path to the parent, fails, the child is initially responsible for re-attaching itself, and therefore all routers subordinate to it on the same branch, to the tree.

4.1. Router Failure

An on-tree router can detect a failure from the following two cases:

+o if the child responsible for sending keepalives across a particular link stops receiving CBT_ECHO_REPLY messages. In this case the child realises that its parent has become unreachable and must therefore try and re-connect to the tree for all groups represented on the parent/child link. For all groups sharing a common core set (corelist), provided those groups can be specified as a CIDR-like aggregate, an aggregated join can be sent representing the range of groups. Aggregated joins are made possible by the presence of a "group mask" field in the CBT control packet header (footnote 3).

If a range of groups cannot be represented by a mask, then each group must be re-joined individually.

CBT's re-join strategy is as follows: the rejoining router which is immediately subordinate to the failure sends a JOIN_REQUEST (subcode ACTIVE_JOIN if it has no children attached, and subcode ACTIVE_REJOIN if at least one child is attached) to the best next-hop router on the path to the elected core. If no JOIN-ACK is received after three retransmissions, each transmission being at PEND-JOIN-INTERVAL (5 secs) intervals, the next-highest priority core is elected from the core list, and the process repeated. If all cores have been tried unsuccessfully, the DR has no option but to give up.

+o if a parent stops receiving CBT_ECHO_REQUESTs from a child. In this case, if the parent has not received an expected keepalive after CHILD_ASSERT_EXPIRE_TIME, all children reachable across that link are removed from the parent's forwarding database.

4.2. Router Re-Starts

There are two cases to consider here:

+o Core re-start. All JOIN-REQUESTs (all types) carry the identities (i.e. IP addresses) of each of the cores for a group. If a router is a core for a group, but has only recently re-started, it will not be aware that it is a core for any group(s). In such circumstances, a core only becomes aware that it is such by receiving a JOIN-REQUEST. Subsequent to a core learning its status in this way, if it is not the primary core it acknowledges the received join, then sends a JOIN_REQUEST (subcode ACTIVE_REJOIN) to the primary core. If the re-started router is the primary core, it need take no action, i.e. in all

3 There are situations where it is advantageous to send a single join-request that represents potentially many groups. One such example is provided in [11], whereby a designated border router is required to join all groups inside a CBT domain.

circumstances, the primary core simply waits to be joined by other routers.

+o Non-core re-start. In this case, the router can only join the tree again if a downstream router sends a JOIN_REQUEST through it, or it is elected DR for one of its directly attached subnets, and subsequently receives an IGMP membership report.

4.3. Route Loops

Routing loops are only a concern when a router with at least one child is attempting to re-join a CBT tree. In this case the re-joining router sends a JOIN_REQUEST (subcode ACTIVE REJOIN) to the best next-hop on the path to an elected core. This join is forwarded as normal until it reaches either the specified core, another core, or a on-tree router that is already part of the tree. If the rejoin reaches the primary core, loop detection is not necessary because the primary never has a parent. The primary core acks an active-rejoin by means of a JOIN-ACK, subcode PRIMARY-REJOIN-ACK. This ack must be processed by each router on the reverse-path of the active-rejoin; this ack creates tree state, just like a normal join-ack.

If an active-rejoin is terminated by any router on the tree other than the primary core, loop detection must take place, as we now describe.

If, in response to an active-rejoin, a JOIN-ACK is returned, subcode NORMAL (as opposed to an ack with subcode PRIMARY-REJOIN-ACK), the router receiving the ack subsequently generates a JOIN-REQUEST, subcode NACTIVE-REJOIN (non-active rejoin). This packet serves only to detect loops; it does not create any transient state in the routers it traverses, other than the originating router (in case retransmissions are necessary). Any on-tree router receiving a non-active rejoin is required to forward it over its parent interface for the specified group. In this way, it will either reach the primary core, which unicasts, directly to the sender, a join ack with subcode PRIMARY-NACTIVE-ACK (so the sender knows no loop is present), or the sender receives the non-active rejoin it sent, via one of its child interfaces, in which case the rejoin obviously formed a loop.

If a loop is present, the non-active join originator immediately sends a QUIT_REQUEST to its newly-established parent and the loop is broken.

Using figure 2 (over) to demonstrate this, if R3 is attempting to re-join the tree (R1 is the core in figure 2) and R3 believes its best next-hop to R1 is R6, and R6 believes R5 is its best next-hop to R1, which sees R4 as its best next-hop to R1 -- a loop is formed. R3 begins by sending a JOIN_REQUEST (subcode ACTIVE_REJOIN, since R4 is its child) to R6. R6 forwards the join to R5. R5 is on-tree for the group, so responds to the active-rejoin with a JOIN-ACK, subcode NORMAL (the ack traverses R6 on its way to R3).

R3 now generates a JOIN-REQUEST, subcode NACTIVE-REJOIN, and forwards this to its parent, R6. R6 forwards the non-active rejoin to R5, its parent. R5 does similarly, as does R4. Now, the non-active rejoin has reached R3, which originated it, so R3 concludes a loop is present on the parent interface for the specified group. It immediately sends a QUIT_REQUEST to R6, which in turn sends a quit if it has not received an ACK from R5 already AND has itself a child or subnets with member presence. If so it does not send a quit -- the loop has been broken by R3 sending the first quit.

QUIT_REQUESTs are typically acknowledged by means of a QUIT_ACK. A child removes its parent information immediately subsequent to sending its first QUIT-REQUEST. The ack here serves to notify the (old) child that it (the parent) has in fact removed its child information. However, there might be cases where, due to failure, the parent cannot respond. The child sends a QUIT-REQUEST a maximum of three times, at PEND-QUIT-INTERVAL (5 sec) intervals.

```
                ------
                | R1 |
                ------
                  |
        ---------------------------
                  |
                ------
                | R2 |
                ------
                  |
        ---------------------------
                  |                             |
                ------                          |
                | R3 |--------------------------|
                ------                          |
                  |                             |
        ---------------------------             |
                  |                             |       ------
                ------                          |       |    |
                | R4 |                          |-------| R6 |
                ------                          |       |----|
                  |                             |
        ---------------------------             |
                  |                             |
                ------                          |
                | R5 |--------------------------|
                ------                          |
                                                |
```

Figure 2: Example Loop Topology

In another scenario the rejoin travels over a loop-free path, and the first on-tree router encountered is the primary core, R1. In figure 2, R3 sends a join, subcode REJOIN_ACTIVE to R2, the next-hop on the path to core R1. R2 forwards the re-join to R1, the primary core, which returns a JOIN-ACK, subcode PRIMARY-REJOIN-ACK, over the reverse-path of the rejoin-active. Whenever a router receives a PRIMARY-REJOIN-ACK no loop detection is necessary.

If we assume R2 is on tree for the corresponding group, R3 sends a join, subcode REJOIN_ACTIVE to R2, which replies with a join ack, subcode NORMAL. R3 must then generate a loop detection packet (join request, subcode REJOIN-NACTIVE) which is forwarded to its parent, R2, which does similarly. On receipt of the rejoin-Nactive, the primary core unicasts a join ack back directly to R3, with subcode PRIMARY-NACTIVE-ACK. This confirms to R3 that its rejoin does not form a loop.

5. Data Packet Loops

The CBT protocol builds a loop-free distribution tree. If all routers that comprise a particular tree function correctly, data packets should never traverse a tree branch more than once (footnote 4).

CBT mode data packets from a non-member sender must arrive on a tree via an "off-tree" interface. The CBT mode data packet's header includes an "on-tree" field, which contains the value 0x00 until the data packet reaches an on-tree router. The first on-tree router must convert this value to 0xff. This value remains unchanged, and from here on the packet should traverse only on-tree interfaces. If an encapsulated packet happens to "wander" off-tree and back on again, an on-tree router will receive the CBT encapsulated packet via an off-tree interface. However, this router will recognise that the "on-tree" field of the encapsulating CBT header is set to 0xff, and so immediately discards the packet.

---

4 The exception to this is when CBT mode is operating between CBT routers connected to a multi-access link; a data packet may traverse the link in native mode (if group members are present on the link), as well as CBT mode for sending the data between CBT routers on the tree.

6. Data Packet Forwarding Rules

6.1. Native Mode

In native mode, when a CBT router receives a data packet, the packet may only be forwarded over outgoing tree interfaces (member subnets and interfaces leading to outgoing on-tree neighbours) iff it has been received via a valid on-tree interface (or the packet has arrived encapsulated from a non-member, i.e. off-tree, sender). Otherwise, the packet is discarded.

Before a packet is forwarded by a subnet's DR, provided the packet's TTL is greater than 1, the packet's TTL is decremented.

6.2. CBT Mode

In CBT mode, routers ignore all non-locally originated native mode multicast data packets. Locally-originated multicast data is only processed by a subnet's DR; in this case, the DR forwards the native multicast data packet, TTL 1, over any outgoing member subnets for which that router is DR. Additionally, the DR encapsulates the locally-originated multicast and forwards it, CBT mode, over all tree interfaces, as dictated by the CBT forwarding database.

When a router, operating in CBT mode, receives a CBT-mode encapsulated data packet, it decapsulates one copy to send, native mode and TTL 1, over any directly attached member subnets for which it is DR. Additionally, an encapsulated copy is forwarded over all outgoing tree interfaces, as dictated by its CBT forwarding database.

Like the outer encapsulating IP header, the TTL value of the encapsulating CBT header is decremented each time it is processed by a CBT router.

An example of CBT mode forwarding is provided towards the end of the next section.

7. CBT Mode -- Encapsulation Details

In a multi-protocol environment, whose infrastructure may include non-multicast-capable routers, it is necessary to tunnel data packets between CBT-capable routers. This is called "CBT mode". Data packets are de-capsulated by CBT routers (such that they become native mode data packets) before being forwarded over subnets with member hosts. When multicasting (native mode) to member hosts, the TTL value of the original IP header is set to one. CBT mode encapsulation is as follows:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
| encaps IP hdr | CBT hdr | original IP hdr | data ....|
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 3. Encapsulation for CBT mode

The TTL value of the CBT header is set by the encapsulating CBT router directly attached to the origin of a data packet. This value is decremented each time it is processed by a CBT router. An encapsulated data packet is discarded when the CBT header TTL value reaches zero.

The purpose of the (outer) encapsulating IP header is to "tunnel" data packets between CBT-capable routers (or "islands"). The outer IP header's TTL value is set to the "length" of the corresponding tunnel, or MAX_TTL (255)if this is not known, or subject to change.

It is worth pointing out here the distinction between subnetworks and tree branches (especially apparent in CBT mode), although they can be one and the same. For example, a multi-access subnetwork containing routers and end-systems could potentially be both a CBT tree branch and a subnetwork with group member presence. A tree branch which is not simultaneously a subnetwork is either a "tunnel" or a point-to-point link.

In CBT mode there are three forwarding methods used by CBT routers:

+o IP multicasting. This method sends an unaltered (unencapsulated) data packet across a directly-connected subnetwork with group member presence. Any host originating multicast data, does so in this form.

+o CBT unicasting. This method is used for sending data packets encapsulated (as illustrated above) across a tunnel or point-to-point link; the IP destination address of the encapsulating IP header is a unicast address. En/de-capsulation takes place in CBT routers.

+o CBT multicasting. A CBT router on a multi-access link can take advantage of multicast in the case where multiple on-tree neighbours are reachable across a single physical link; the outer encapsulating IP header contains a multicast address as its destination address. The IP module of end-systems on the same link subscribed to the same group will discard these multicasts since the CBT payload type (protocol id) of the outer IP header is not recognizable by hosts.

CBT routers create forwarding database (db) entries whenever they send or receive a JOIN_ACK. The forwarding database describes the parent-child relationships on a per-group basis. A forwarding database entry dictates over which tree interfaces, and how (unicast or multicast) a data packet is to be sent.

Note that a CBT forwarding db is required for both CBT-mode and native-mode multicasting.

Using our example topology in figure 1, let's assume the CBT routers are operating in CBT mode.

Member G originates an IP multicast (native mode) packet. R8 is the DR for subnet S10. R8 therefore sends a (native mode, TTL 1) copy over any member subnets for which it is DR - S14 and S10 (the copy over S10 is not sent, since the packet was originally received from S10). The multicast packet is CBT mode encapsulated by R8, and unicast to each of its children, R9 and R12; these children are not reachable over the same interface, otherwise R8 could have sent a CBT mode multicast. R9, the DR for S12, need not IP multicast (native mode) onto S12 since there are no members present there. R9 unicasts the packet in CBT mode to R10, which is the DR for S13 and S15. R10 decapsulates the CBT mode packet and IP multicasts (native mode, TTL 1) to each of S13 and S15.

Going upstream from R8, R8 CBT mode unicasts to R4. It is DR for all directly connected subnets and therefore IP multicasts (native mode) the data packet onto S5, S6 and S7, all of which have member presence. R4 unicasts, CBT mode, the packet to all outgoing children, R3 and R7 (NOTE: R4 does not have a parent since it is the primary core router for the group). R7 IP multicasts (native mode) onto S9. R3 CBT mode unicasts to R1 and R2, its children. Finally, R1 IP multicasts (native mode) onto S1 and S3, and R2 IP multicasts (native mode) onto S4.

8. Non-Member Sending

For a multicast data packet to span beyond the scope of the originating subnetwork at least one CBT-capable router must be present on that subnetwork. The DR for the group on the subnetwork must encapsulate the (native) IP-style packet and unicast it to a core for the group (footnote 5). The encapsulation required is shown in figure 3; CBT mode encapsulation is necessary so the receiving CBT router can demultiplex the packet accordingly.

If the encapsulated packet hits the tree at an on-tree router, the packet is forwarded according to the forwarding rules of section 6.1 or 6.2, depending on whether the receiving router is operating in native- or CBT mode. Note that it is possible for the different interfaces of a router to operate in different (and independent) modes.

If the first on-tree router encountered is the target core, various scenarios define what happens next:

+o if the target core is not the primary, and the target core has not yet joined the tree (because it has not yet itself received any join-requests), the target core simply forwards the encapsulated packet to the primary core; the primary core IP address is included in the encapsulating CBT data packet header.

if the target core is not the primary, but has children, the target core forwards the data according to the rules of section 6.

5 It is assumed that CBT-capable routers discover <core, group> mappings by means of some discovery protocol. Such a protocol is outside the scope of this document.

+o if the target core is the primary, the primary forwards the data according to the rules of section 6.2.

9. Eliminating the Topology-Discovery Protocol in the Presence of Tunnels

Traditionally, multicast protocols operating within a virtual topology, i.e. an overlay of the physical topology, have required the assistance of a multicast topology discovery protocol, such as that present in DVMRP [1]. However, it is possible to have a multicast protocol operate within a virtual topology without the need for a multicast topology discovery protocol. One way to achieve this is by having a router configure all its tunnels to its virtual neighbours in advance. A tunnel is identified by a local interface address and a remote interface address. Routing is replaced by "ranking" each such tunnel interface associated with a particular core address; if the highest-ranked route is unavailable (tunnel end-points are required to run an Hello-like protocol between themselves) then the next-highest ranked available route is selected, and so on. The exact specification of the Hello protocol is outside the scope of this document.

CBT trees are built using the same join/join-ack mechanisms as before, only now some branches of a delivery tree run in native mode, whilst others (tunnels) run in CBT mode. Underlying unicast routing dictates which interface a packet should be forwarded over. Each interface is configured as either native mode or CBT mode, so a packet can be encapsulated (decapsulated) accordingly.

As an example, router R's configuration would be as follows:

intf	type	mode	remote addr
#1	phys	native	-
#2	tunnel	cbt	128.16.8.117
#3	phys	native	-
#4	tunnel	cbt	128.16.6.8
#5	tunnel	cbt	128.96.41.1

core	backup-intfs
A	#5, #2
B	#3, #5
C	#2, #4

The CBT forwarding database needs to be slightly modified to accommodate an extra field, "backup-intfs" (backup interfaces). The entry in this field specifies a backup interface whenever a tunnel interface specified in the forwarding db is down. Additional backups (should the first-listed backup be down) are specified for each core in the core backup table. For example, if interface (tunnel) #2 were down, and the target core of a CBT control packet were core A, the core backup table suggests using interface #5 as a replacement. If interface #5 happened to be down also, then the same table recommends interface #2 as a backup for core A.

10. CBT Packet Formats and Message Types

We distinguish between two types of CBT packet: CBT mode data packets, and CBT control packets. CBT control packets carry a CBT control packet header.

CBT control packets are encapsulated in IP, as illustrated below:

```
++++++++++++++++++++++++++++++++
| IP header | CBT control pkt |
++++++++++++++++++++++++++++++++
```

In CBT mode, the original data packet is encapsulated in a CBT header and an IP header, as illustrated below:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
| IP header | CBT header | original IP hdr | data .... |
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The IP protocol field of the inner (original) IP header is used to demultiplex a packet correctly; CBT has been assigned IP protocol number 7. The CBT module then demultiplexes based on the encapsulating CBT header's "type" field, thereby distinguishing between CBT control packets and CBT mode data packets.

The CBT data packet header is illustrated below.

10.1. CBT Header Format (for CBT Mode data)

```
 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 |unused |      type     |   hdr length  | on-tree|unused|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          checksum             |      IP TTL   |     unused    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        group identifier                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        first-hop router                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          primary core                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  reserved     | reserved  |T|S|     Type      |     Length    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        .....Flow-id value.....                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     unused    |       unused      |     Type      |  Length   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        .....Security data......               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4. CBT Header

Each of the fields is described below:

+o Vers: Version number -- this release specifies version 1.

+o type: indicates CBT payload; values are defined for control (0x00), and data (0xff). For the value 0x00 (control), a CBT control header is assumed present rather than a CBT header.

+o hdr length: length of the header, for purpose of checksum calculation.

+o on-tree: indicates whether the packet is on-tree (0xff) or off-tree (0x00).

+o checksum: the 16-bit one's complement of the one's complement of the CBT header, calculated across all fields.

+o IP TTL: TTL value corresponding to the value of the IP TTL value of the original multicast packet, and set in the CBT header by the DR directly attached to the origin host (decremented by CBT routers visited).

+o group identifier: multicast group address.

+o first-hop router: identifies the encapsulating router directly attached to the origin of a multicast packet. This field is relevant to source-migration of a core to the source (see Appendix A). It is set to NULL when core migration is disabled.

+o primary core: the primary core for the group, as identified by "group-id". This field is necessary for the case where non-member senders happen to send to a secondary core, which may not yet be joined to the primary core. This field allows the secondary to know which is the primary for the group, so that the secondary can forward the (encapsulated) data onwards to the primary.

+o T bit: indicates the presence (1) or absence (0) of Type of Service/flow-id value ("type", "length", "type of service/flow-id") .

+o S bit: indicates the presence (1) or absence (0) of a security value ("type", "length", "security data").

10.2. Control Packet Header Format

The individual fields are described below.

```
    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 |unused |     type      |     code      |   # cores     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         hdr length            |            checksum           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         group identifier                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                            group mask                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                           packet origin                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        primary core address                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    target core address (core #1)              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              Core #2                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              Core #3                          |
   |                                ....                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  reserved     | reserved  |T|S|     Type      |     Length    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      type of service/flow-id                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     unused    |       unused      |     Type      |   Length  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                     .....Security data.....                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5. CBT Control Packet Header

+o Vers: Version number -- this release specifies version 1.

+o type: indicates control message type (see sections 10.3).

+o code: indicates subcode of control message type.

+o # cores: number of core addresses carried by this control packet.

- +o header length: length of the header, for purpose of checksum calculation.

- +o checksum: the 16-bit one's complement of the one's complement of the CBT control header, calculated across all fields.

- +o group identifier: multicast group address.

- +o group mask: mask value for aggregated CBT joins/join-acks. Zero for non-aggregated joins/join-acks.

- +o packet origin: address of the CBT router that originated the control packet.

- +o primary core address: the address of the primary core for the group.

- +o target core address: desired core affiliation of control message.

- +o Core #N: IP address for each of a group's cores.

- +o T bit: indicates the presence (1) or absence (0) of Type of Service/flow-id value ("type", "length", "type of service/flow-id") .

- +o S bit: indicates the presence (1) or absence (0) of a security value ("type", "length", "security data").

10.3. CBT Control Message Types

There are ten types of CBT message. All are encoded in the CBT control header, shown in figure 5.

- +o JOIN-REQUEST (type 1): generated by a router and unicast to the specified core address. It is processed hop-by-hop on its way to the specified core. Its purpose is to establish the originating CBT router, and all intermediate CBT routers, as part of the corresponding delivery tree. Note that all cores for the corresponding group are carried in join-requests.

- +o JOIN-ACK (type 2): an acknowledgement to the above. The full list of core addresses is carried in a JOIN-ACK, together with the actual core affiliation (the join may have been terminated by an on-tree router on its journey to the specified core, and the terminating router may or may not be affiliated to the core specified in the original join). A JOIN-ACK traverses the reverse path as the corresponding JOIN-REQUEST, with each CBT router on the path processing the ack. It is the receipt of a JOIN-ACK that actually "fixes" tree state.

+o JOIN-NACK (type 3): a negative acknowledgement, indicating that the tree join process has not been successful.

+o QUIT-REQUEST (type 4): a request, sent from a child to a parent, to be removed as a child of that parent.

+o QUIT-ACK (type 5): acknowledgement to the above. If the parent, or the path to it is down, no acknowledgement will be received within the timeout period. This results in the child nevertheless removing its parent information.

+o FLUSH-TREE (type 6): a message sent from parent to all children, which traverses a complete branch. This message results in all tree interface information being removed from each router on the branch, possibly because of a re-configuration scenario.

+o CBT-ECHO-REQUEST (type 7): once a tree branch is established, this messsage acts as a "keepalive", and is unicast from child to parent (can be aggregated from one per group to one per link. See section 4).

+o CBT-ECHO-REPLY (type 8): positive reply to the above.

+o CBT-BR-KEEPALIVE (type 9): applicable to border routers only. See [11] for more information.

+o CBT-BR-KEEPALIVE-ACK (type 10): acknowledgement to the above.

10.3.1. CBT Control Message Subcodes

The JOIN-REQUEST has three valid subcodes:

- +o ACTIVE-JOIN (code 0) - sent from a CBT router that has no children for the specified group.

- +o REJOIN-ACTIVE (code 1) - sent from a CBT router that has at least one child for the specified group.

- +o REJOIN-NACTIVE (code 2) - generated by a router subsequent to receiving a join ack, subcode NORMAL, in response to a active-rejoin.

A JOIN-ACK has three valid subcodes:

- +o NORMAL (code 0) - sent by a core router, or on-tree router, acknowledging joins with subcodes ACTIVE-JOIN and REJOIN-ACTIVE.

- +o PRIMARY-REJOIN-ACK (code 1) - sent by a primary core to acknowledge the receipt of a join-request received with sub-code REJOIN-ACTIVE. This message traverses the reverse-path of the corresponding re-join, and is processed by each router on that path.

- +o PRIMARY-NACTIVE-ACK (code 2) - sent by a primary core to acknowledge the receipt of a join-request received with sub-code REJOIN-NACTIVE. This ack is unicast directly to the router that generated the rejoin-Nactive, i.e. the ack it is not processed hop-by-hop.

11. CBT Protocol Number

CBT has been assigned IP protocol number 7. CBT control messages are carried directly over IP.

12. Default Timer Values

There are several CBT control messages which are transmitted at fixed intervals. These values, retransmission times, and timeout values, are given below. Note these are recommended default values only, and are configurable with each implementation (all times are in seconds):

+o CBT-ECHO-INTERVAL 30 (time between sending successive CBT-ECHO-REQUESTs to parent).

+o PEND-JOIN-INTERVAL 5 (retransmission time for join-request if no ack rec'd)

+o PEND-JOIN-TIMEOUT 30 (time to try joining a different core, or give up)

+o EXPIRE-PENDING-JOIN 90 (remove transient state for join that has not been ack'd)

+o PEND_QUIT_INTERVAL 5 (retransmission time for quit-request if no ack rec'd)

+o CBT-ECHO-TIMEOUT 90 (time to consider parent unreachable)

+o CHILD-ASSERT-INTERVAL 90 (increment child timeout if no ECHO rec'd from a child)

+o CHILD-ASSERT-EXPIRE-TIME 180 (time to consider child gone)

+o IFF-SCAN-INTERVAL 300 (scan all interfaces for group presence. If none, send QUIT)

+o BR-KEEPALIVE-INTERVAL 200 (backup designated BR to designated BR keepalive interval)

+o BR-KEEPALIVE-RETRY-INTERVAL 30 (keepalive interval if BR fails to respond)

13. Interoperability Issues

Interoperability between CBT and DVMRP has recently been defined in [11].

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

14. CBT Security Architecture

see [4].

Acknowledgements

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

Thanks too to Sue Thompson (Bellcore). Her detailed reviews led to the identification of some subtle protocol flaws, and she suggested several simplifications.

Thanks also to the networking team at Bay Networks for their comments and suggestions, in particular Steve Ostrowski for his suggestion of using "native mode" as a router optimization, and Eric Crawley.

Thanks also to Ken Carlberg (SAIC) for reviewing the text, and generally providing constructive comments throughout.

I would also like to thank the participants of the IETF IDMR working group meetings for their general constructive comments and suggestions since the inception of CBT.

APPENDICES

DISCLAIMER: As of writing, the mechanisms described in Appendices A and B have not been tested, simulated, or demonstrated.

APPENDIX A

Dynamic Source-Migration of Cores

A.0 Abstract

This appendix describes CBT protocol mechanisms that allow a CBT multicast tree, initially constructed around a randomly-placed set of core router, to dynamically reconfigure itself in response to an active source, such that the CBT tree becomes rooted at the source's local CBT router. Henceforth, CBT emulates a shortest-path tree.

For clarity, the mechanisms are described in the context of "flat" multicasting, but are transferrable to a hierarchical model with only minor changes.

A.1 Motivation

One of the criticisms levelled against shared tree multicast schemes is that they potentially result in sub-optimal routes between receivers. Another criticism is that shared trees incur a high traffic concentration effect on the core routers. Given that any shared tree is likely to have two, three, or more cores which can be strategically placed in the network, as well as the fact that any on-tree router can act as a "branch point" (or "exploder point"), shared tree traffic concentration can be significantly reduced. This note nevertheless addresses both of these criticisms by describing new mechanisms that

+o allow a CBT to dynamically transition from a random configuration to one where any CBT router can become a core - more precisely, that which is local to a source, and...

+o remove the traffic concentration issue completely, as a result of the above; traffic concentration is not an issue with source-rooted trees.

The mechanisms described here are relevant to non-concurrent sources;

the concurrent-sender case is not addressed here, although experience with MBONE applications for the past several years suggests that most multicast applications are of the single, infrequently-changing sender type. Also, it is not necessarily implied that the initial CBT tree must be transitioned. Any transition is an "all-or-nothing" transition, meaning that either all the tree transitions, or none of it does (footnote 6).

A.2 Goals & Requirements

By means of the mechanisms described, this Appendix sets out to achieve the follwoing:

+o provide mechanisms that allow the dynamic transition from an initial CBT, constructed around a pre-configured set of cores, to a CBT that is rooted at a core attached to a sender's local subnetwork. This is source-rooted tree emulation.

+o ensure that these mechanisms do not impact CBT's simplicity or scalability.

+o eliminate completely the traffic concentration issue from CBT.

+o to eliminate the core placement/core advertisement problems.

+o ensure that the scheme is robust, such that if a source's local router (or link to it) should fail, the CBT self-organises itself and returns to its original configuration.

+o the mechanisms should provide the same even to non-member senders.

The above incurs a few additional requirements on existing baseline CBT mechanisms described in this specification:

+o a new JOIN-REQUEST subcode, REVERSE-JOIN

+o a new JOIN-ACK subcode, REVERSE-ACK

---

6 This is the expected behaviour of PIM Sparse Mode; on reciept of high-bandwidth traffic, most receivers' local routers will be configured to transition to source trees.

+o new JOIN-ACK subcode, CORE-MIGRATE

+o a "first-hop router" field needs to be included in the CBT data packet header.

+o a new message type:

- SOURCE-NOTIFICATION

+o CBT-mode data encapsulation is required until the local CBT router connected to an active source receives a JOIN-REQUEST, whose "target core address" field is one of its own IP addresses.

These new additions are explained in the next section.

A.3 Source-Tree Emulation Criteria

CBT routers are configured with a lower-bound data-rate threshold that is the expected boundary between low- and high-bandwidth data rate traffic. CBT also monitors the duration each sender sends. If this duration exceeds a pre-configured value (global across CBT), say 3 minutes, AND the data rate threshold is exceeded, the CBT tree transitions such that receivers become joined to the "core" local to the source's subnet, i.e. the CBT tree becomes source-rooted, but nevertheless remains a CBT.

A.4 Source-Migration Mechanisms

```
                                                    E o        o D
                                                       \     /
                                                        \   /
            L o                                          \ /
               \                                           o C
                \                  N                      /
                 \                                       /
                  \A(2)                            (1)B /
                   O===================================O
                   |                                   |
          M        |                                   |
                   |                                   |
                 K o                                   o H
                  /\                                  /\
                 /  \                                /  \
                /    \                              /    \
        s    J o      o I                        G o      o F
       ----------
```

Key: B = primary core
A = secondary core
s = sending host
J = sending host's local DR
M & N = network nodes not on original CBT tree

Figure A1: Original CBT Tree

In figure A1, host s starts sending native mode multicast data. CBT router J encapsulates it as CBT mode, inserting its own IP address in the "first-hop router" field of the CBT mode data packet header. This data packet flows over the CBT tree.

Note that tree migration can be disabled either by sending all packets in native mode, or by inserting NULL value into the "first-hop router" field. Since the first-hop router is the original encapsulating router (data packets are always originated from hosts in native mode), the first-hop router knows whether the sender's data rate warrants activating the "first-hop router" field; for the purpose of the ensuing protocol description, we assume this is the case.

Any router on the tree receiving the CBT mode data packet, inspects the "first-hop router" field of the CBT header, and compiles a join-request to send to it. In order to fully specify the join, it must inspect its underlying unicast routing table(s) to find the best next-hop to the source's first hop router. That next hop will be either on or off the existing CBT tree for the group. If the next hop is off-tree, the join generated is given a subcode of ACTIVE-JOIN (as per CBT spec), and a "target core address" of the source's first hop router. The join is then forwarded and processed according to the CBT specification. The primary core, and the original core list, remain specified in their respective fields of the CBT control packet header.

Using figure A1 to illustrate an example, node L's routing tables suggest that the best next-hop to J, the source's first hop router, is via node M, not yet on the tree. So, node L generates a join and forwards it to M, which forwards it to J. The join-ack (subcode NORMAL) returns to L via M on the reverse-path of the join. When the join-ack reaches L, L sends a QUIT-REQUEST to A, its old parent. The shortest-path branch now exists, L-M-J.

If the best next hop to the source's first hop router is via an existing on-tree interface, if that interface is the node's parent on the current tree, no further action need be taken, and no join need be sent towards the source, J.

However, the join's best next hop may be via an existing child interface - this is where the new join type, subcode REVERSE-JOIN, comes in. The purpose of this join type is to simply reverse the existing parent-child relationship between two adjacent on-tree routers; each end of the link between the two routers is re-labelled. This join must be acknowledged by means of a JOIN-ACK, subcode REVERSE-ACK. A reverse-join is only ever sent from a child to its parent.

Immediately subsequent to sending a reverse-join-ACK, the sending node's old parent interface is labelled as "pending child", and a timer is set on that interface. This is a delay timer, set at a default of 5 seconds, during which time a reverse-join is expected over that interface from the node's old parent. Should this timer expire, a REVERSE-ASSERT message is sent to the old parent (new child) to cause it to agree to the change in the parent-child relationship. A REVERSE-ASSERT must be ack'd (REVERSE-ASSERT-ACK). If, after (say) three retransmissions (at 5 sec intervals) no reverse-assert-ack has been received, a QUIT-REQUEST is sent to the old parent and the corresponding interface is removed from this node's current forwarding database.

Of course, if a node has already received a reverse-join during the period one of its other interfaces was changing its parent-child relationship with another of its neighbours, then the pending-child delay timer need not be activated.

Looking at figure A1 again, here's the process of how the parent-child relationships change on the tree when an active source, s, starts sending. Of course, links E-C, I-J, and L-J do not do this because they forge completely new paths towards the source's local router, J.

K sends a reverse-join to J. J acks this with a join-ack, subcode REVERSE-ACK. At this point, J is K's parent, and I is still K's child. K now sets the pending-child delay timer on its interface to A (K's old parent), and expects a reverse-join from A. If it weren't to arrive after the delay timer expires, plus several retransmissions of a reverse-assert control message, K can send a quit to A (it sends a quit because, as far as A is concerned, it thinks K is still its child) and removes the K-A interface from its CBT forwarding database. However, assuming a reverse-join does arrive at K from A before the delay timer expires, K acks the reverse-join and cancels the delay timer on that interface.

Next, let's consider CBT router (node) I. I's unicast routing table suggest it can reach J directly (next-hop) via a different interface than the I-K interface, so I sends a join-request, subcode active-join, to J, which acks it as normal. On receipt of the ack, I sends a quit to K and removes K as its parent from its database.

Now let's consider node L. Like I, it finds a new path to J, via M, so simply sends a new join to J, via M, and on receipt of the join-ack, sends a quit to A, and removes A from its forwarding database. A new, shortest-path, branch now exists, J-M-L.

Next let's consider A-B, the link between the cores. A is the secondary, and B is the primary, so A originally joined towards B. So, B sends a reverse-join to A. A sends a reverse-ack to B, so A is now B's parent, and B has children B-H, and B-C. Note that the role of primary and secondary is not affected - the target of B's join to A is the source's local router, J.

The existing branches D-C-B, F-H-B, and G-H-B, need not change any of their parent-child relationships, since each of these nodes' unicast routing tables indicate that the best next-hop a join-request, targetted at source J, would take, is via the corresponding existing parent.

For E, it sends a new join via N to J. On receipt of the join-ack, it sends a quit to C. A new branch has been created, E-N-J.

Each node on the tree now has a shortest-path to J, the source's local CBT router. Hence, J is the root ("core") of a shortest-path multicast tree.

Note that these new mechanisms augment the CBT protocol, and the baseline CBT protocol engine is not affected in any way by this add-on mechanism.

A.5 Robustness Issues

Some immediate questions might be:

+o what happens to the source-rooted tree if the source's local CBT router fails?

+o what happens if the source's local CBT router fails whilst the initial tree is transitioning?

+o what happens if the tree is partitioned, or not yet fully connected, when a source starts sending?

+o how do new receivers join an already-transitioned tree?

All of these questions are now addressed:

+o What happens to the source-rooted tree if the source's local CBT router fails?

A source-rooted CBT has a single point of failure - the root of the tree.

In spite of a source being joined, the corelist (primary & secondaries) is carried in CBT control packets, as per the CBT spec. However, the contents of the "target core address" field identifies the IP address of the source's local CBT router. So, in the event of a failure, the CBT routers still have all the information they need to rejoin the original tree, constructed around the corelist. Rejoining then, proceeds according to the rules of the CBT specification.

Of course, rejoining the original tree happens only after several attempts have been made to rejoin the source's "core".

+o What happens if the source's local CBT router fails whilst the initial tree is transitioning?

This really is no different to the above case. The parts of the tree that have transitioned will rejoin the original tree according to their corresponding corelist. Those parts of the tree in the process of transitioning may temporarily transition, but eventually those nodes will receive a FLUSH from a CBT router adjacent to the failed source router ("core"). They then rejoin the original tree.

+o What happens if the tree is partitioned, or not yet fully connected, when a source starts sending?

The problem here is that some parts of the network (CBT tree) may not receive CBT encapsulated mode data packets before the source's local DR starts forwarding data in native mode, and so those receivers will not know the IP address of the local DR to join to.

For example, assume a secondary core with downstream members cannot reach the primary. If the routers adjacent to the secondaries are all functioning correctly, the secondaries themselves may not be aware that a partition has occurred somewhere further upstream. So, what if a source downstream from a secondary, starts sending data after the partition has happened?

A new control message, the SOURCE-NOTIFICATION, is used to solve this problem. As soon as any core recieves CBT mode encapsulated data, it caches the source "core" IP address, and starts multicasting (to the group) SOURCE-NOTIFICATION messages, one every minute. Source-notifications contain the IP address of the source's local DR. A core continues to multicast source-notications at 1 minute intervals until the source has ceased transmitting data for more than 20 seconds.

Obviously, if a CBT is fully connected, the larger proportion of source-notifications will be redundant. However, this cost justifies the robustness the scheme provides.

If an off-tree source begins sending data, which first hits the tree at a secondary core with no receivers attached, the secondary does not trigger a join towards the primary, but instead just unicasts the data, in CBT mode, to the primary (as per CBT spec). The primary then forwards the data over any connected tree branches. Receivers can then begin transitioning. In this way, a transitioned CBT tree extends to the first hop router of a non-member sender.

Note that cores and on-tree routers only ever react to active sources iff they have an existing CBT forwarding database for the said group. For example, a primary core would not establish a shortest-path branch to a non-member sender unless it has at least one existing child registered for the corresponding group.

+o How do new receivers join an already-transitioned CBT?

New receivers will always attempt to join one of the cores in the corelist for a group. Two things can happen here: firstly, a new join, targetted at one of the cores in the corelist eventually reaches that target core. Secondly, the new join hits a router already established on-tree, but the router encountered is now joined to the source tree (source "core").

For the first scenario, all on-tree routers and all core routers maintain the address of which upstream core their CBT branch actually emanates from (as per CBT spec). When a new join arrives at one of the original cores, the core checks whether its own current core affiliation is to a core outside the corelist set. If so, that core is a source "core", so the core responds to the new join with a JOIN-ACK, subcode CORE-MIGRATE. This join-ack contains the address of the active source "core". This join-ack causes a join-request to be issued by one of the routers that receives it - the router whose path to the core (just joined) diverges from that to the source "core"; this can easily be gleaned from unicast routing. The router then simply directs it new join at the source "core", and on receipt of the join-ack, sends a quit to its now "old" parent.

For the second case, the solution is trivial; any on-tree router receiving a join targetted either at one of the original cores for the group, or the active source "core", simply acks (subcode NORMAL) the join and includes in the ack the source "core" affiliation (as per CBT spec).

A.6 Loops

It may seem that the potential for a transitioning tree to form loops, especially in the presence of reverse-joins, is greatly increased. This is probably NOT the case; "reversed branches" are those that are already part of a loop-free tree that CBT constructs around the original set of cores. Transitioned tree are just CBTs, whereby the core is simply rooted at the source. Loops are no more likely with these mechanisms then they are with baseline CBT. Note that these are assertions - formal proofs may be more appropriate.

APPENDIX B

Group State Aggregation

B.1 Introduction

Although the scalability of shared tree multicast schemes is attractive now, to scale over the longer-term, a combination of hierarchy (support mechanisms that facilitate domain-oriented multicasting), and group aggregation strategies, is required. If IP multicast is to have a long-term future in the Internet as a global transport mechanism, by far the most serious challenge is to address the issue of group state aggregation.

Shared trees were developed partly to address scalability with regards to multicast state maintained in the network, which resulted in an improvement in that state by a factor of the number of active sources (a source being a subnetwork aggregate). However, it is perceived that the number of sources sending to any one group will not grow as fast as the number of groups, indeed the latter will probably grow at several orders of magnitude faster [12]. Therefore, it is essential to contain this potential problem, particularly for the benefit of routers on wide-area links, by designing an effective group state aggregation mechanism, capable of collapsing group state.

Unlike unicast addresses, multicast addresses cannot be aggregated according to topological locality; multicast addresses are truly location-independent. Thus, it would not seem obvious how the problem can be addressed - clearly, it must be looked at in a different way.

In order to be effective, flexibility and efficiency must be facets of group aggregation; an aggregation scheme must be able to accommodate groups with wide-ranging characteristics in the least constraining way possible. For example, the trend towards small, non-local groups (e.g. 4 or 5 person audio/video conferences between different user groups spread over different countries/continents); it is these types of groups that are likely to result in an explosive growth in state. Also, these groups will, in all likelihood, utilize multicast addresses that are randomly spread across the multicast address space, making aggregation seemingly more difficult. An aggregation scheme must therefore account for this.

B.2 Design Overview

This scheme involves replacing a subset of individual tree state present on inter-domain links, and aggregating it over a single shared tree. The scheme does not yet specify how candidate groups for aggregation are arrived at, but an obvious scheme to would be to aggregate already-overlapping distribution trees. The pivotal idea behind this approach encompasses two inter-dependent strategies:

+o administratively defining a portion of the multicast address space for aggregate groups. For brevity, an example might be the range 238.0.0.0 - 238.255.255.255.

+o associated with each aggregate group address is a mask, specifying the portion of the address that it used to identify the aggregate group itself (the portion covered by the mask); the remaining address space is used as an index to an ordered list of groups with which the aggregate address is associated. The ordered list and its association with a group aggregate address is conveyed by means of a protocol message (TBD). The index is used to de-aggregate at region boundaries (border routers).

The scheme subscribes to the notion of aggregation-on-demand; a border router (BR) is configured with a threshold number of groups on a BRs external interface, above which it begins to solicit aggregations periodically, say once every hour.

As an example, say BR 123 wishes to aggregate 200 groups. BR 123 randomly chooses (or by some address allocation algorithm) a group aggregate address. It has been established that the number of groups for which aggregation is desired is 200. The nearest power of 2 value to 200 is 256 (2^8), and so the aggregate mask covers 24 bits, leaving 8 to specify each individual group's traffic flowing over the aggregate tree.

So we have:

Group aggregate address: 238.10.12.0

Group aggregate mask: 238.10.12/24

A data packet for the 30th listed group (listed in a protocol message (TBD) as described above) would be addressed to: 238.10.12.30.

Similarly, a data packet pertaining to the 150th listed group would be addressed to: 238.10.12.150, and so on.

All routers comprising the aggregate tree need only maintain the group aggregate address and mask, together with the aggregate tree's associated interfaces. If a number of individual shared trees have been replaced by an aggregate tree, then the core routers (RPs) of each of those shared trees must additionally maintain the complete list of groups associated with an <aggregate address/mask-len> so as to be able to "re-direct" any incoming joins for already aggregated groups. Similarly, border routers (BRs) are incurred the storage cost of maintaining the individual groups associated with an <aggregate address/mask-len>, so as to be able to aggregate and de-aggregate as data packets flow across a (sub)region's border.

B.3 Scaling Further

The scheme described can be applied recursively (to border routers) to accommodate a hierarchy containing an arbitrary number of levels.

The scheme described imposes two general requirements (or assumptions):

+o a well defined aggregate group address space for each level of hierarchy (or scope levels).

+o the ability to arbitrarily create boundaries in multicast routers, thereby separating different hierarchical levels.

The former will require consensus within the IETF and approval from the IANA. The latter capability is already available in multicast routers; boundaries are specified in a multicast routers configuration file. This capability is currently available in the best known multicast routing protocols: DVMRP, M-OSPF, PIM, and CBT.

Defining boundaries may require some degree of coordination; whenever a particular scoped level (boundary) is introduced which has multiple entry/exit multicast routers, these must all be configured such that their boundary definitions are identical, i.e. they must each be configured with the same boundary-address/mask (the range 239.0.0.0 - 239.255.255.255 is the IANA-defined multicast boundary address range).

Author Information:

Tony Ballardie,
Department of Computer Science,
University College London,
Gower Street,
London, WC1E 6BT,
ENGLAND, U.K.

Tel: ++44 (0)71 419 3462
e-mail: A.Ballardie@cs.ucl.ac.uk

Scott Reeve, Nitin Jain,
Bay Networks, Inc.
3, Federal Street,
Billerica, MA 01821,
USA.

Tel: ++1 508 670 8888
e-mail: {sreeve, njain}@BayNetworks.com

References

[1] T. Pusateri. Distance Vector Multicast Routing Protocol. Working draft, June 1996. (draft-ietf-idmr-dvmrp-v3-01.{ps,txt}).

[2] J. Moy. Multicast Routing Extensions to OSPF. Communications of the ACM, 37(8): 61-66, August 1994. Also RFC 1584, March 1994.

[3] D. Farinacci, S. Deering, D. Estrin, and V. Jacobson. Protocol Independent Multicast (PIM) Dense-Mode Specification. Working draft, July 1996. (draft-ietf-idmr-pim-dm-spec-02.{ps,txt}).

[4a] A. Ballardie. Core Based Tree (CBT) Multicast Architecture. Working draft, July 1996. (draft-ietf-idmr-cbt-arch-04.txt)

[4] A. J. Ballardie. Scalable Multicast Key Distribution; RFC 1949, SRI Network Information Center, 1996.

[5] A. J. Ballardie. "A New Approach to Multicast Communication in a Datagram Internetwork", PhD Thesis, 1995. Available via anonymous ftp from: cs.ucl.ac.uk:darpa/IDMR/ballardie-thesis.ps.Z.

[6] W. Fenner. Internet Group Management Protocol, version 2 (IGMPv2). Working draft, May 1996. (draft-idmr-igmp-v2-03.txt).

[7] B. Cain, S. Deering, A. Thyagarajan. Internet Group Management Protocol Version 3 (IGMPv3) (draft-cain-igmp-00.txt).

[8] M. Handley, J. Crowcroft, I. Wakeman. Hierarchical Rendezvous Point proposal, work in progress. (http://www.cs.ucl.ac.uk/staff/M.Handley/hpim.ps) and (ftp://cs.ucl.ac.uk/darpa/IDMR/IETF-DEC95/hpim-slides.ps).

[9] D. Estrin et al. USC/ISI, Work in progress. (http://netweb.usc.edu/pim/).

[10] D. Estrin et al. PIM Sparse Mode Specification. Working draft, July 1996. (draft-ietf-idmr-pim-sparse-spec-04.{ps,txt}).

[11] A. Ballardie. CBT - Dense Mode Interoperability: Border Router Specification; Working draft, July 1996. Also available from: ftp://cs.ucl.ac.uk/darpa/IDMR/draft-ietf-idmr-cbt-dm-interop-XX.txt

[12] S. Deering. Private communication, August 1996.