Core Based Trees (CBT)

- Scalable Multicast Routing -

by

A. Ballardie (UCL), P. Tsuchiya (Bellcore), J. Crowcroft (UCL)

Status of this Draft

Abstract

Multicasting is a technique used which allows for group communication either locally or on a wide-area scale. Most local-area networks such as Ethernet and Token Ring provide a multicast service, which has since been exploited by many applications and distributed systems. More recently, multicast capability has been extended to the internetwork using a combination of a distance-vector routing algorithm, a host-to-router group-membership reporting protocol, and the computation of sender-based multicast trees. This, and other approaches to internetwork multicasting, are not scalable to large groups, especially so for large numbers of groups.

This document provides a proposal for a new multicast routing algorithm which provides multicast capability in a datagram internetwork. It is scalable, low-cost and efficient - properties lacking in current internetwork multicast routing protocols.

Distribution of this draft is unlimited. A mailing list is in place in idmr@cs.ucl.ac.uk (subscribe to idmr-request@cs.ucl.ac.uk) to discuss this and other inter-domain multicast routing issues.

CBT Expires April 6th, 1993.

Contents

1 Introduction

Multicast transmission is an increasingly important function in data networks. For instance, SMDS specifies multicast as part of its service, and multicast is an important element for ATM networking, particularly for, but not limited to, video broadcast. Multicast is also increasingly used in the IP internet.

One of the central problems in multicast transmission is forming the multicast tree - the collection of nodes and links that the multicast packet traverses. In spite of the fact that substantial work has been done in the area of algorithms for forming multicast trees, significant problems remain to be solved. Paramount among these is the problem of scaling. Current multicast techniques either scale well but form grossly suboptimal trees, or they form good trees but scale poorly. (By scale poorly, we mean consume too many memory, bandwidth, or processing resources.)

For example, the multicast model that has been adopted for internetwork multicasting is based onformulating a *sender-based* multicast spanning tree for each member of a group to which a host is subscribed, using information extracted from the *distance-vecto* routing algorithm. This imposes considerable storage costs on participating routers. These costs include storing parent and child attributes (and corresponding next-hop link information) in existing route table entries, as well as some other protocol-dependent information. Furthermore, all routers in the internet that have multicast capability are required to participate in establishing a multicast tree for each group present in the internet, whether they are on the multicast tree or not[1]. This is clearly not scalable as the number of groups to which a host subscribes, and the number of hosts per group, increases.

The number of applications taking advantage of the internetwork multicast service is growing rapidly. Example applications include video-conferencing, audio-conferencing, distributed database updating/querying, and location mechanisms for distributed network services. It is therefore important that any internetwork multicast routing algorithm be **scalable**, **efficient** (in terms of multicast packet forwarding), **low-cost** (in terms of computational overhead and storage requirements), and be adaptable to security features - an important issue so far not addressed in multicasting.

This draft proposes a new algorithm for multicast group communication in the global internetwork, called **CBT** (Core Based Trees). It satisfies all of the requirements stated above.

[1]This overhead can be reduced by using an appropriate TTL value in the IP header, especially so for localised groups

2 CBT Design Goals

1. The design of any multicast routing algorithm should satisfy the requirements of the *Host Group Model* as specified in [1]. These mostly have to do with the flexibility of multicasting, for example:

   - **Senders need not be members.**
   - **Groups may have any number of members.**
   - **There are no topological restrictions on group membership.**
   - **Membership is dynamic and autonomous.**
   - **Host groups may be permanent or transient.**

2. **Scalability** is of paramount importance in any multicast routing algorithm. Good multicast routing algorithm scaling characteristics include **low memory**, **low bandwidth**, and **low processing** resouce requirements. Furthermore, the less routers need to know about the location of *remote* group members, the better.

3. **Robust Multicast Routing**. CBT uses a core-based approach to building the multicast spanning tree. Most groups will be small at the beginning of their lifetime, and so the multicast spanning tree for a group will be single-core in the beginning. The destination address in all multicast packets will be the address of the core. As groups become larger and more widespread, additional core(s) may be assigned to the group. Core placement will be chosen so as to enhance the optimality and robustness of the multicast delivery tree.

4. **Invisible Multicast Routing and Addressing to routers not on the CBT tree.** CBT allows for multicast routing and addressing to be completely invisible to underlying unicast routing and addressing to routers not on the CBT tree. No separate multicast routing table is required by routers (although minor extensions to the unicast routing table are required), and looking at the IP address field alone, multicast addresses are indistinguishable from IP unicast addresses. Multicast addresses are only recognised as such if the router in question is a member of the same group as the group identified in an incoming multicast packet, i.e. it is a router on the CBT tree for that group. The router in question can establish this by matching its group identifier with that present in the IP Options field. The group identifier is 32 bits long. If there is no match, then the packet is routed as a normal unicast packet towards the core addressed in the packet. We considered this addressing flexibility an important design goal since multicast addresses are no longer restricted to class D addresses, the administrative overhead of assigning them is reduced, and we see a degree of multicast address and routing transparency introduced not seen before in multicasting. Furthermore, by having a separate name space it is concievable to apply additional semantics to the name.

   Routers that are part of a CBT tree need only know which routers are its parent (any router has only one parent per CBT tree) and which are its children (the exception to this rule is for certain cores on the core tree which need to know about each other although they are not directly connected neighbours). Thus, CBT allows for a considerable amount of information hiding when compared with other network-level multicast approaches.

5. **Routing Algorithm/Protocol Independence**. CBT's flexible addressing method allows CBT to be installable anywhere, even across different routing algorithms. Furthermore, it is easy to incorporate into new protocols that may not have a separate address space set aside for multicasting (for example, ATM with E.164 addresses). This is obviously advantageous for inter-domain multicast routing, and also means it can be applied to technologies such as *SMDS* and *ATM networks*

6. **Independence from any new IP addressing scheme**. One of the biggest headaches currently facing the internet research community is that the 32-bit IP address space is fast exhausting. A considerable number of researchers are frantically trying to find the right addressing solution to satisfy future growth expectations and beyond. As far as we can foresee, CBT should function independent of addressing method. This is certainly the case for the proposals so far reviewed by the authors.

3 Assumptions of the CBT Model

- CBT is discussed here in the context of a connectionless datagram internetwork, although as we have already stated, the design of CBT should allow it to run over heterogeneous networks.

- CBT uses both **ICMP** and **IGMP** protocols, and although the both are standard parts of IP, the latter is present in many, but not all, implementations. We assume its presence for CBT.

- CBT requires certain messages not originating on the CBT tree, to traverse only tree branches once encountering the tree. Furthermore, messages originating in certain parts of the tree must only traverse certain branches, and any acknowledgements to these messages must similarly traverse the same branch(es) on the return-path. We have not assumed *symmetric routes* and have therefore made explicit provisions to create symmetric routes where necessary.

- CBT routers are not dedicated to CBT multicast routing. They are normal IP routers with normal IP routing capabilities. A router may be a CBT router for more than one group at any one time. CBT functionality then, is considered an extension to normal IP routing functionality. Ideally, all routers in the internetwork will have CBT functionality and each will make itself available to become a core for a group(s) at any time.

- Last, but by no means least, we assume the presence of a *multicast core-assignment service*, or *core server*. The specification of such a *core-assignment service* is outside the context of this draft. However, we make numerous assumptions/suggestions as to how it would/should operate and some of the services it should provide. We also include some of the messages that might be exchanged between such a service and CBT in order to provide a clearer picture, but these message are NOT part of CBT. Whether a dedicated *core-assignment service* is required, or directory services should assume this role, is open to debate, but clearly a *core-assignment service* would require global topology information in order to be able to assign cores reasonably optimally. For simplicity, we will refer to a dedicated *core-assignment service* throughout this draft as *multicast directory service*.

  We feel some kind of dedicated multicast service is necessary not only for CBT, but for all multicast protocols. If such a service were presently in operation, maybe *it* could assign the class D addresses required by current internet multicast routing protocols, which at present, come from "thin air".

4 CBT Design Overview

We will first of all provide a description of some of the terminology used in CBT. This is followed by a subsection on CBT describing what it is, what it aims to provide, and briefly how it works. Finally, a more detailed description of the algorithm is given. CBT encompasses a rather more complicated protocol for maintaining the connectivity of the backbone of the CBT tree (**core-tree**), called the *Core-tree Maintenance Protocol.* This may be mentioned in the description of the algorithm, but we have dedicated a complete section to it later on because of its importance and for reasons of clarity.

4.1 Some Terminology

CBT (core-based tree) is a multicast routing algorithm which results in a centre-based (core-based) multicast delivery tree. There is one CBT tree per group. A **core-based tree** can be of two types:

- single-core
- multi-core

Single-core CBT trees are for small, geographically local groups. As a group becomes larger and more widespread, additional cores may be assigned to the group concerned, thus making it a **multi-core** CBT tree.

We distinguish between three different types of router that make up a CBT tree:

- **major core** - these are the only routers to be assigned by directory service. They become the centre (or part of the centre in the case of multi-core trees) of a multicast CBT spanning tree. They are assigned so as to be strategically placed to form as near an optimal multicast spanning tree as possible for the group. All multicast packets contain, as their destination address, the (unicast) address of a major core.
- **minor core** - router(s) which form a virtual link between major core routers on a CBT tree. These routers are only present in multi-core CBT trees. When an additoinal major core is assigned to a single-core CBT tree, that major core must make a special *core-join request* to the existing major core. The path of routers the *core-join request-ACK* traverses become **minor cores** on the CBT tree.
- **non-core router** - all routers part of the CBT tree that are not major or minor cores are **non-core routers**.

The part(s) of the CBT spanning tree that link **major core** routers together, i.e. the branches of the tree that only contain **major** and **minor** cores, is known as the **core tree**.

- **major child** - the major core that has sent (and received and ACK for) a *core-join request* to another major core on the core tree.
- **major parent** - the major core which has received and ACK'd a *core-join request* from another major core, now on the core tree.

4.2 Packet Types

The following is a list of all CBT and CBT-related control packet types. Brief descriptions are given here, with fuller implementation details available in the appropriate later sections.

- **CBT control packet types**.

  1. *send-join request* - sent from a host to its local router requesting that router to formulate and send a *join request*. It is an indication of the host's firm intention to join a particular group[2].
  2. *send-join-ACK* - positive response from the local router to the host, formulated and sent by it *after* forwarding a *join-request* to the **major core** as specified in the *send-join request*. The host's group membership is pending.
  3. *join-request* - originated by (if received a *send-join request* from host on attached subnet), or forwarded by, a router that is not already part of the CBT tree for the group identified by **group-id** inside the packet. **Non-core router** status is pending.
  4. *join-ACK* - formulated by any router on the CBT tree that receives a *join-request*. All routers traversed by a *join-ACK* become **non-core routers** on the CBT tree for the group identified by **group-id**.
  5. *core-join request* - formulated and sent by a **major core** router that has been newly-assigned to a group. It will have been assigned as such by **directory service**, which also supplies it with the address of an existing **major core** to which the *core-join request* must be forwarded.

     Alternatively, a *core-join request* is formulated and sent by an existing **major core** to another existing **major core** when reachability between the two has been lost, or one of the **major core's** parents has become unavailable. Both scenarios amount to a **core-tree** partition.
  6. *core-join-ACK* - formulated and sent by either a **major core** (in the case it actually received the *core-join request*), or by a **minor core** (if that **minor core** terminated the *core-join request*), towards the source of the *core-join request*. All routers traversed by a *core-join-ACK* on its way to its destination become **minor core** routers, if are not already such.
  7. *termination-notification* - formulated by a **minor core** that has received (and thus terminated) a *core-join request*. It is forwarded to the **major core** addressed as the target of the of the *core-join request*.
  8. *termination-notification-ACK* - formulated by the **major core** that was the target of a *core-join request* terminated by a **minor core**. It is forwarded to that **minor core**.

[2] This will incur minor host modifications. Any host modifications are obviously undesirable, but we feel that if we are going to solve the problem of inter-domain multicasting for the long-term future to provide a scalable and simple approach, host modifications will be eventually necessary. The minor host changes will be with respect to IGMP

9. *core-state-notification* - originated by a **major core** when that **major core** has sent either a *core-join-ACK*, or a *termination-notification-ACK*. Disemminated on all **core-tree** interfaces.

   Alternatively, a **major core** will generate a *core-state-notification* when it loses reachability to a virtual neighbour.

10. *core-list* - generated by the originating **major core** of a *core-state-notification* packet. It is the result of a *core-state-notification* packet having stabilized. The *core-list* is multicast on the CBT tree.

11. *quit-request* - can be composed by all three types of CBT router. In the case of **major cores**, a *quit-request* is sent when a new *core-join request* arrives from a currently unreachable **major child** via a different interface than that leading to the child previously.

    In the case of **minor cores**, a *quit-request* is sent only when one is received on a child **core-tree** interface AND there are no member hosts on any attached subnets.

    In the case of **non-core** routers, a *quit-request* is sent whenever the **non-core** router's parent changes and the path leading to the new parent is different from the old, or a *quit request* is received from a downstream **non-core** AND the current router has no member hosts on any attached subnets.

Note: All CBT control packets will be sent encapsulated in IP datagrams.

- **ICMP**.

  1. *ICMP echo request* - sent both ways between **major child** and **major parent**, traversing only the **core-tree**, to test the "liveness" of each.
  2. *ICMP echo reply* - a positive response to the above, again, sent only via the **core-tree**.

- **IGMP**.

  1. *group-membership query* - composed by routers and sent locally to attached subnets to monitor the presence of group members. Failure to receive a response for a particular group query after a certain timeout period results in bf non-core router sending a *quit-request* to its parent.
  2. *group-membership report* - composed in response to receiving a *group-membership query* from the local designated router, to indicate that the group in question still has membership on this subnet.

- **CBT - Directory Service**.

  1. *group-initiation request* - composed by a host that wishes to create a group for a particular multicast application. It contains the address(es) of the other parties with which it wishes to communicate, and is forwarded to **directory service** as a request for a **major core** to be assigned. It expects the address of a new core to be returned.
  2. *group-join request* - composed by a host that wishes to join an existing group. It is a request for a **major core** address.

3. *core-request* - a request from **directory service** aimed at a router for it to become a **major core** for a group.

4.3 CBT Introduction

The aim of CBT is to provide a **core-based tree** for each multicast group. This is a centre-based shortest-path spanning tree, with branches emanating from the centre. The tree will normally start out by being a **single-core** tree but as the group becomes larger and more widespread, additional core(s) will be assigned to the tree for reasons of routing efficiency and robustness. All cores are assigned by a **multicast directory service**, the operation of which is outside the context of this draft. When more than a single core is present in a CBT tree, the branch(es) linking the cores together form the **core-tree**. The assigned cores for a particular CBT tree, known as **major core routers**, are likely to be geographically distant apart, and so the core tree which links these cores is likely to contain other routers, known as **minor core routers**. For reasons of brevity, we describe first the single-core case.

All branches of the tree consist of point-to-point links. When a host on a subnet somewhere in the internet, wishes to join an existing group, it sends a request to its local router to send a request to join the group. This *join-request* contains as its destination address, the address of the major core for the group, and is forwarded using the underlying unicast routing algorithm to the next hop on its way to the core. Every router that is traversed en-route will either be a router that is already part of the CBT tree, known as **non-core routers**, or will not be part of the CBT tree. If the *join-request* encounters a non-core router, that router *terminates* the **join-request**, and sends a *join-ACK* back to the source. All routers traversed by the *join-ACK* become **non-core routers** on receiving it. The *join-ACK* continues to be forwarded until it reaches the original source.

Routers that receive a *join-request* that are not non-cores for the CBT tree will forward it on its way to the destination core using normal unicast routing, and then await a *join-ACK* before becoming **non-cores** for the CBT tree (group).

Whenever a *join-ACK* is sent or received via an interface, that interface is marked as being a multicast interface for the corresponding CBT tree (group). In this way, branches are formed on the CBT tree. The diagram over illustrates a single-core CBT tree.

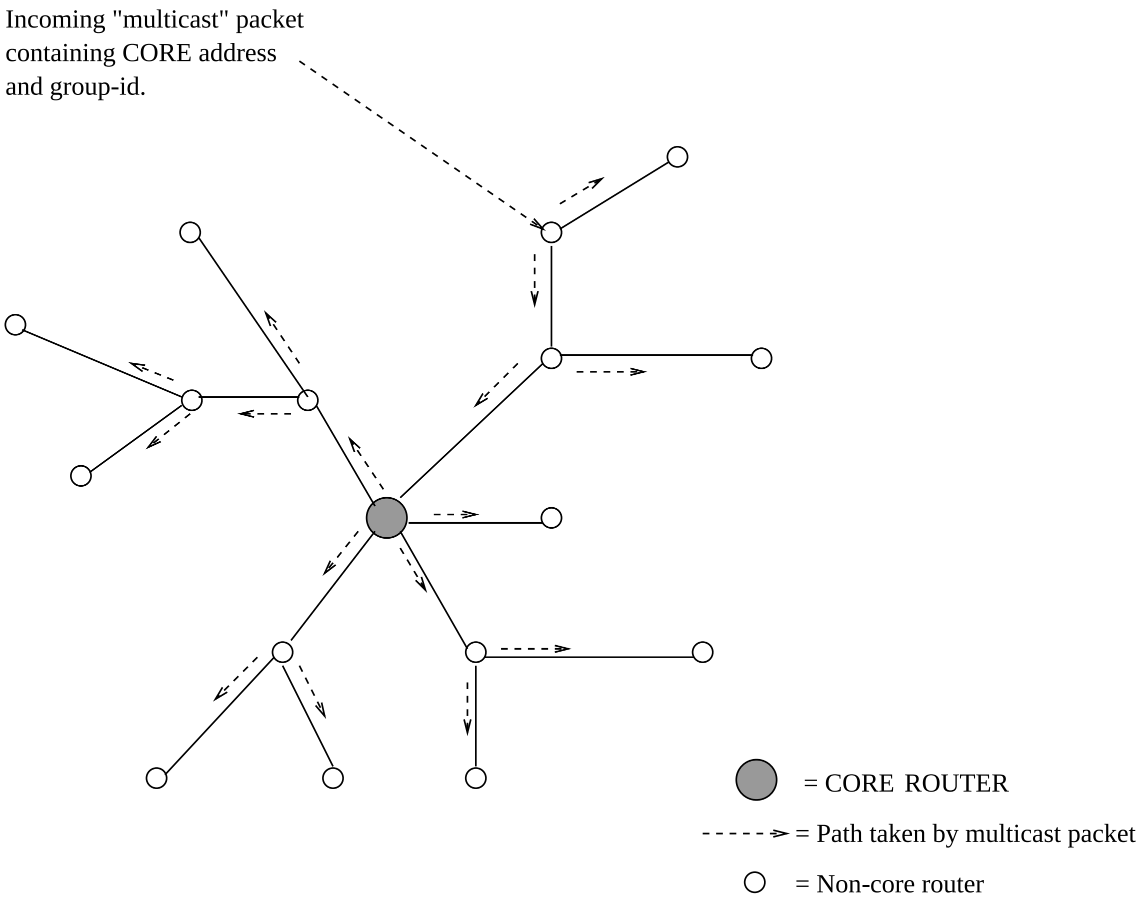


Figure 1: Single-core CBT Tree

Should the group become larger or/and more widespread, an additional core(s) may be assigned by directory service. This new **major core** must make a special join-request to the existing major core, to become part of the **core-tree**. This request is called a *core-join request*. All routers traversed en-route by the *core-join request* will become **minor core routers** on receiving a *core-join-ACK*. Existing **minor cores** on a CBT **core-tree** are obliged to *terminate core-join requests* and send a *core-join-ACK* back to the source, which, on receipt of it, becomes a new **major core** for the group. The new and existing **major cores** become (virtual) neighbours (major neighbours), the latter being the **major parent** and the former being the **major child**. On receiving the *core-join-ACK*, both parent and child majors can begin exchanging *Core-tree Maintenance Protocol* messages. The diagram over illustrates a multi-core CBT tree.

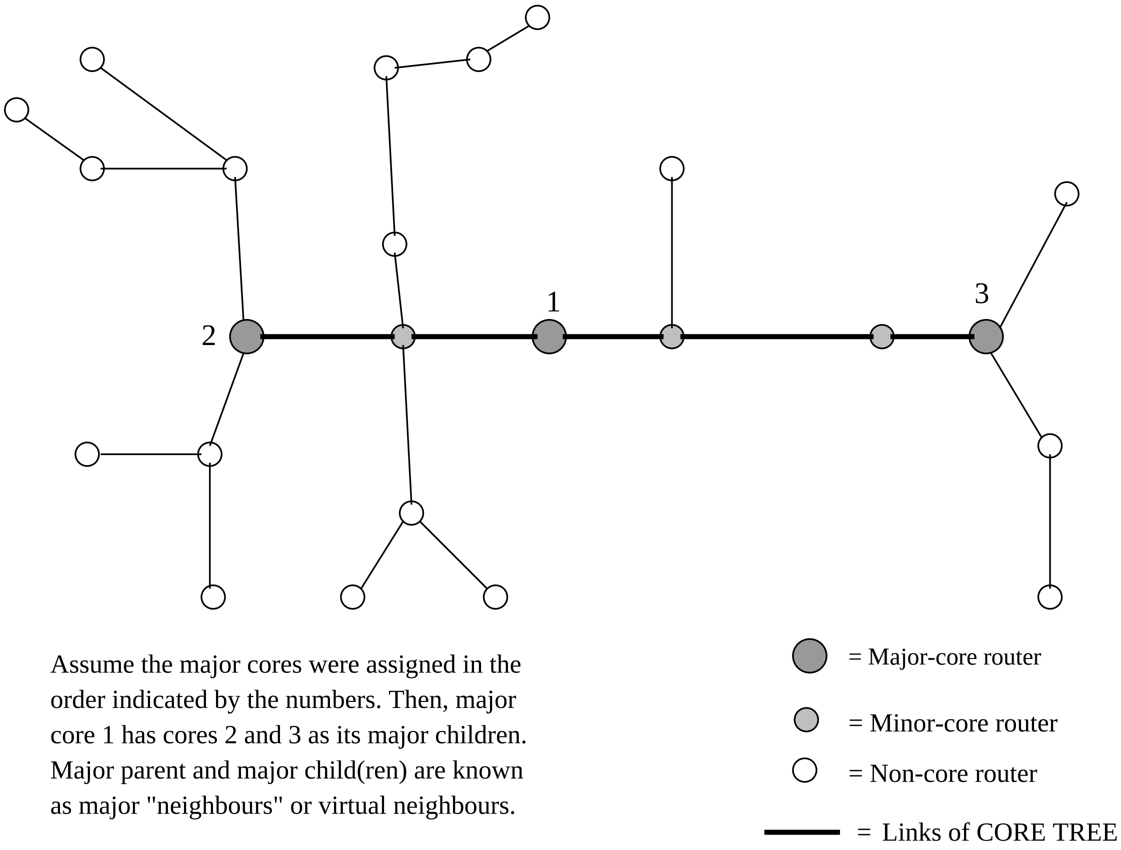


Figure 2: Multi-core CBT Tree

Should an existing **minor core** terminate a *core-join request*, it must send a *termination-notification* packet to the target of the join. When it receives a *termination-notification-ACK* in response, it may then generate a *core-join-ACK* which is forwarded to the source of the *core-join request*. The *termination-notification* packet indicates to the **major parent** that it has a new **major child**.

Multicast data packets can originate from members of the group or non-members outside the group that are not part of the CBT tree. In the former case, multicast data packets will originate from a member host on a CBT router's attached subnet and be disemminated on reciept by that local on all outgoing multicast interfaces identified by **group-id**. Eventually, the data packets will reach all nodes on the CBT tree.

In the latter case, multicast data packets will be forwarded as normal data packets, with the destination being a **major core** for the group, using the underlying unicast routing algorithm. The multicast data packet will first encounter the CBT tree either when the destination is reached, i.e. the **major core**, or it will hit another router (**minor** or **non-core**) that is part of the CBT tree. In either case, the multicast data packet will be forwarded on all outgoing interfaces associated with the **group-id** present in the data packet(s). In this way, the multicast data packets will eventually reach all nodes on the CBT tree.

4.4 CBT Algorithm

As specified above, it is assumed that hosts and routers are running the Internet Group Management Protocol - routers running the router part, and hosts the host part [1].

Whenever a host anywhere in the internet wants to initiate a group, there must be at least one other entity at the initial stage with which it wishes to communicate. Taking as an example, two hosts in different parts of the internet have agreed, by some external means (e.g. e-mail), that they wish to partake in a video conference.

On group initiation, one host must make a *group-initiation-request* to **multicast directory service** for a **group initiation**. If, as in this case, there is no CBT tree yet present for the application concerned, the host makes its *group initiation request* supplying all the host addresses it currently knows about with which it wishes to take part in the group communication. In this way, directory service can "magically" select a *major core* which is optimally placed between those participants and supply the address of that major core, together with the **group-identifier** for that group, to all hosts identified in the initial reqest.

If the requesting host knows, by some external means, that a group with which it wishes to communicate already exists, then it makes a *group-join reqest* to directory service which reqires no arguments. Directory service replies with the address of a major core and **group-identifier (group-id)** for the group.

Once the requesting host has received the address of a major core for the group, it must now send a *send-join-reqest*, which includes the **group-id** and **major core** address, to its local router. There are two scenarios to consider:

- The local router is not already a member for the group identified by the **group-id**. In this case, the local router formulates a *join-request* containing as its destination address, the address of the core supplied in the *send-join request* and also containing the **group-id**. This is forwarded using normal unicast routing on the next hop to its destination. At this point, the local router sends a *send-join-ACK* back to the local host, indicating to it that the *join-request* has been sent and group membership is pending. When a *join-ACK* is received by the local router, it is forwarded to the local host that initiated the *join-request*.

- The local router is already a member for the group identified by the **group-id**. In this case, the *send-join request* is not actually honoured, and the local router formulates a *send-join-ACK* and forwards it locally back to the host. Following this, the router formulates a "bogus" *join-ACK* (since no *join-request* was actually sent) and forwards it to the local host.

Only at this point, can an end-system host consider itself a member for the group, and as such can send/receive multicast packets for that group, as well as send positive replies to IGMP group membership queries.

Design decision: We decided a member host should not be permitted to forward multicast packets to the CBT tree for a group until it receives the **join-ACK**. Here we trade-off between reliablility and low join-latency. By forwarding multicast packets to an upstream router before being "ACK'd" would further reduce the degree of delivery probability associated with a connectionless datagram network.

When a router receives a *join-request* from a neighbour router, there are two scenarios to consider:

- **Scenario 1**: The router is already part of the CBT tree for the group identified by **group-id**, but it is not the **major core** to which the join was addressed. Thus, the router encountered is either a **minor core** or a **non-core** router on the tree. At this point, the *join-request* terminates.

  The *join-request* is terminated at this point because there is already a shortest-path from this router to the **core tree**. So, the interface over which the *join-reqest* arrives is marked as a child interface for the corresponding CBT tree, and all CBT packets for the group arriving over it are forwarded on that router's one and only interface leading towards the **core tree**.

  If the router receiving the *join-request* is indeed the **major core** to which it is addressed, then the **major core** must be the first router encountered on the CBT tree for that group. In this case, the **major core** simply returns a *join-ACK* which may traverse other routers before it reaches its destination.

  All routers that forward/originate a *join-request* that recieve/forward a *join-ACK* for that request, set the following flags:

  1. **Router status flag** - if not already set, is set to **non-core** in the **flag table**. Other router status flags are **major core** and **minor core**, but these are set under different circumstances.
     Note: Only one **router status flag** *can be set at any one time for any one CBT tree in the* **flag table**. *The setting of one flag may however, lead to the clearing of another.*
  2. **parent-branch** - this flag is set in routing table, and there can only be one branch for which this flag is set. It is set against the interface indicating the parent branch in a particular CBT tree identified by **group-id**. The routing table is a slightly modified unicast routing table.
  3. **child-branch** - this flag is also set in the routing table against the interface indicating a child branch in a particular CBT tree identified by *group-id.* For any one group, this flag may be set on multiple interfaces.

  The **flag table** is illustrated below:

ROUTER STATUS FLAG		
Non-core	Minor-core	Major-core
group-id	-	-
-	-	group-id
-	group-id	-
....	....	....

- **Scenario 2**: The *join-request* encounters a router that is not part of the CBT tree identified by **group id**. In this case the router forwards the *join-request* on the next hop towards the addressed core using the underlying unicast routing algorithm. As the *join-ACK* passes back along the point-to-point links that the *join-request* traversed, the routers encountered become **non-cores** and the appropriate flags are set in the **flag table**/routing table, as described above.

Note*: All* **join-request***'s contain an identifier in addition to the* **group-id** *to help distinguish between different join's for the same group, and match* **join-request***'s with* **join-ACK***'s. When a* **join-request** *is received by a router not already on the CBT tree, a copy of the* **join-request** *is cached, together with the incoming interface address. When it is forwarded, the outgoing interface address is added to the information contained in the cache. This is to ensure that the* **join-ACK** *is received and sent out on the correct interfaces.*

4.5 Some Examples

The diagram below shows a CBT tree for a group with **group-id** 12345. We will demonstrate two examples. The first will show the messages generated when a host, shown as "X" next to the bottom right router in the diagram, wishes to become a member of the group 12345.

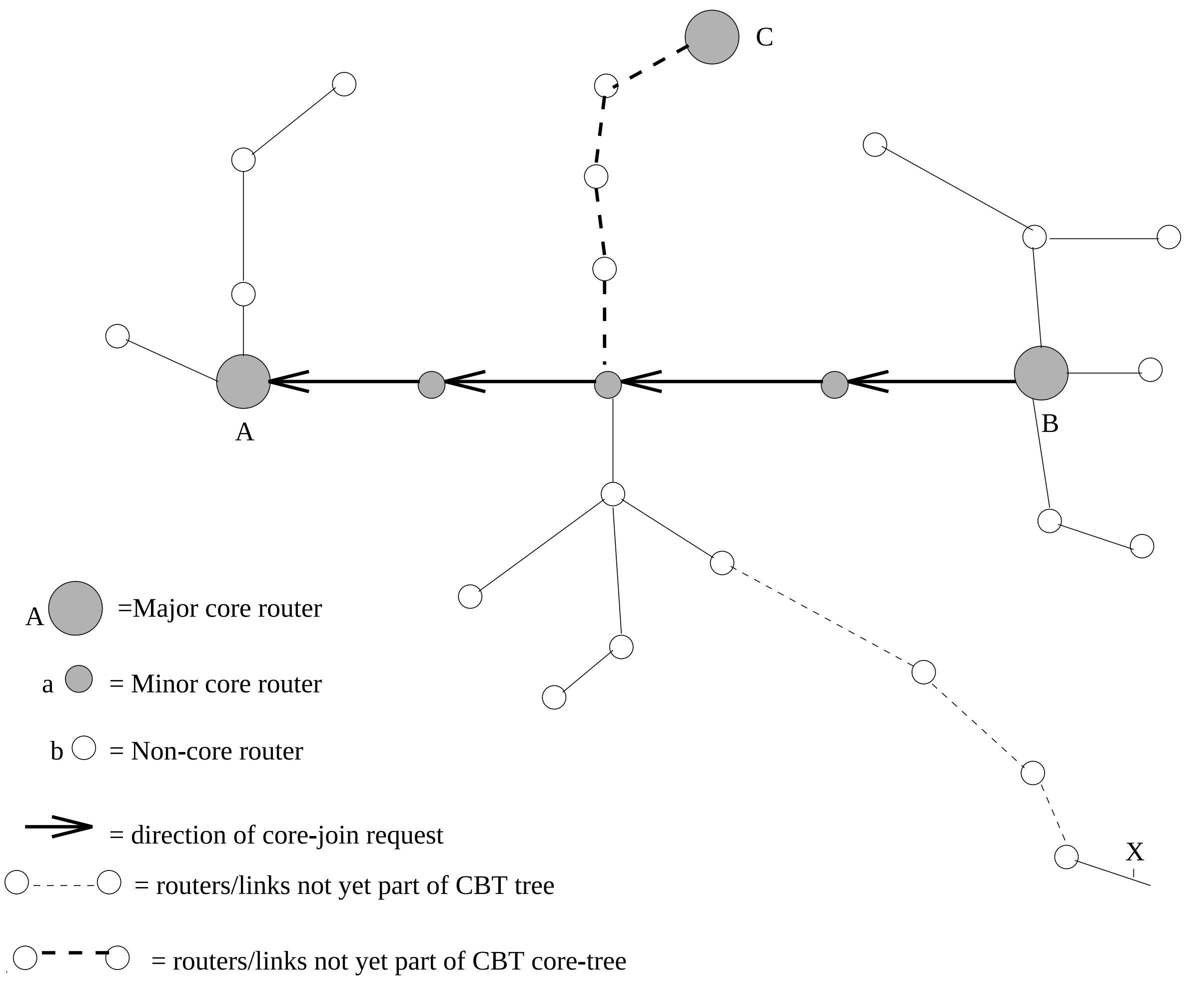


Figure 3: Example Multi-Core CBT Tree

The second example demonstrates how a new **major core** becomes part of the **core-tree**. The part of the diagram relevant for this is the top centre indicated by thick dotted lines.

- **Example 1**: This example demonstrates the sequence of events required for host "X" (bottom right in diagram), when it wishes to become a member for the group 12345.

The diagram illustrates the present state of the CBT tree for group 12345, but the dashed lines indicate the branches not yet part of the tree. The routers on the dotted lines are those traversed by any messages exchanged with the tree.

We assume the message exchange with **directory service** has already taken place, and has supplied **major core** B's address. The sequence of events is as follows:

1. Host "X" composes a *send-join request* containing the address of **major core** B, and the group-id 12345. It then forwards it to the local router shown in the diagram.
2. The local router composes a *join-request* with **major core** B as the destination address, and forwards it on the next-hop towards the core. There are two further hops to traverse (shown on the dotted line on bottom right of diagram) before the *join-request* hits the CBT tree. These two routers forward the *join-request* as a normal unicast packet, i.e. CBT is transparent to them as long as they are not part of the group identified by **group-id** in the OPTIONS field of the IP header.
3. After the local router has forwarded the *join-request*, it returns a *send-join-ACK* to the local host. This indicates to the local host that group membership is pending.
4. The **non-core** router at the end of the dashed line receives the *join-request.* It processes the IP header which includes the **group-id** in the OPTIONS field of the header, and realises it is a CBT packet pertaining to group 12345, of which it is a current member.
5. Since the **non-core** router already has a shortest-path branch to the **core tree**, it terminates the *join-request* and composes a *join-ACK*. This is forwarded back over the interface the join was received, towards the originator of the *join-request.* The current **non-core** router sets another child interface flag in its routing table, for the new branch of the CBT tree.
6. Each router that receives the *join-ACK* sets the parent/child flags in their routing tables, indicating the CBT tree interfaces. It also sets the **non-core** flag in the **flag table**.
7. When the originator receives the *join-ACK*, it marks the appropriate parent interface in its routing table, sets its **non-core** flag in its **flag table**, and forwards the *join-ACK* to the local host that requested membership.
8. From this point the host can send/receive multicast packets for the group identified by **group-id**. The host will also begin to answer *group-membership queries* as part of IGMP.

- **Example 2**: This example demonstrates the sequence of events required for a newly-assigned **major core** router to become part of the CBT **core-tree** for the group 12345. The CBT tree for group 12345 is the same as one used in Example 1 above.

  We assume the initial exchange with **directory service** has been completed, and the **major core** has been supplied with either the address of one existing **major core** (in this case, B), or it has been supplied with the complete list of all current **major cores** for the group, and makes its own choice (in this case, B). The path between the newly-assigned **major core** and **major core B** is shown by the thick dotted line. The sequence of events is as follows:

  1. The newly-assigned **major core** C now knows the address of **major core** B, to which it must forward a *core-join request.* It composes the request, and forwards it on the next hop towards B. The sequence of hops before encountering the CBT tree is shown by the thick dotted line.
  2. The next-hop recieves the *core-join request.* It forwards it using normal unicast routing, since it does not recognise CBT packets unless it is a router on the CBT tree for that group. Similarly for the next two hops, since they are not part of the CBT tree (yet) either.
  3. The *core-join request* hits the **core-tree** on its next-hop. All **minor cores** terminate **core-join requests**. The **minor core** caches the received *core-join request* and forwards a *termination-notification* to **major core** B. When the **minor core** receives a *termination-notification-ACK* it can then compose a *core-join-ACK* and forwards it on the next-hop to the new **major core**.
  4. Once **major core** B has sent the *termination-notification-ACK* it marks the interface over which it was sent in its routing table as the interface for **major child** C. It can proceed to send ICMP echo request/replies over that interface, but first waits to receive an echo request before itself sending one.
  5. The routers traversed by the *core-join-ACK* record over which interface it was received, set the appropriated parent flag in the routing table, set the minor core flag in the **flag table**, then set the child flags in the routing table after forwarding the *join-ACK*.
  6. When the new **major core** receives the *join-ACK*, it records the interface over which it was received, together with its new *major parent*, in the **parent/child table**. It then proceeds to send an ICMP echo request on the interface leading to its new **major parent**. A new **core-tree** branch is now complete.

5 The LAN Issue

Multicast packets will nearly always originate from a host on some local area network (LAN). For LAN's with only one router attached, there are no apparent problems with receiving/forwarding CBT multicast packets onto the LAN on thier way to their destination.

Today however, multi-access links are becoming more common on LAN's, i.e. LAN's now tend to have more than one router attached to them.

Although all routers on a multi-access link will be exchanging EGP/IGP messages to establish which are the unicast shortest-cost paths to destinations, there is often more than one valid route to a destination, for example, for different types of service (ToS) or/and policy considerations for inter-domain routing.

The latter points are of relevance to CBT; consider what would happen if a *join-request* from a particular **major core** from downstream (in relation to that core), arrived at a router on the multi-access LAN. To which router should the *join-request* be forwarded if there is more than one with a shortest-path to the core? If the router receiving the *join-request* were to select (or elect) a forwarding router on the LAN for that and all subsequent multicast packets, the problem is not solved, as we can see from the scenario that could then follow; consider another *join-request* for the same group arriving from downstream (in relation to the core tree) at a different router on the LAN (which is a perfectly valid scenario). To exacerbate the problems of our example, the *join-request* is for the same group, but addressed to a different **major core**.

Further assume that the router at which the *join-request* arrived, selects/elects a different router on the LAN to the one previously chosen by the router that received the first *join-request*. The LAN topology could be as shown in the diagram over ....

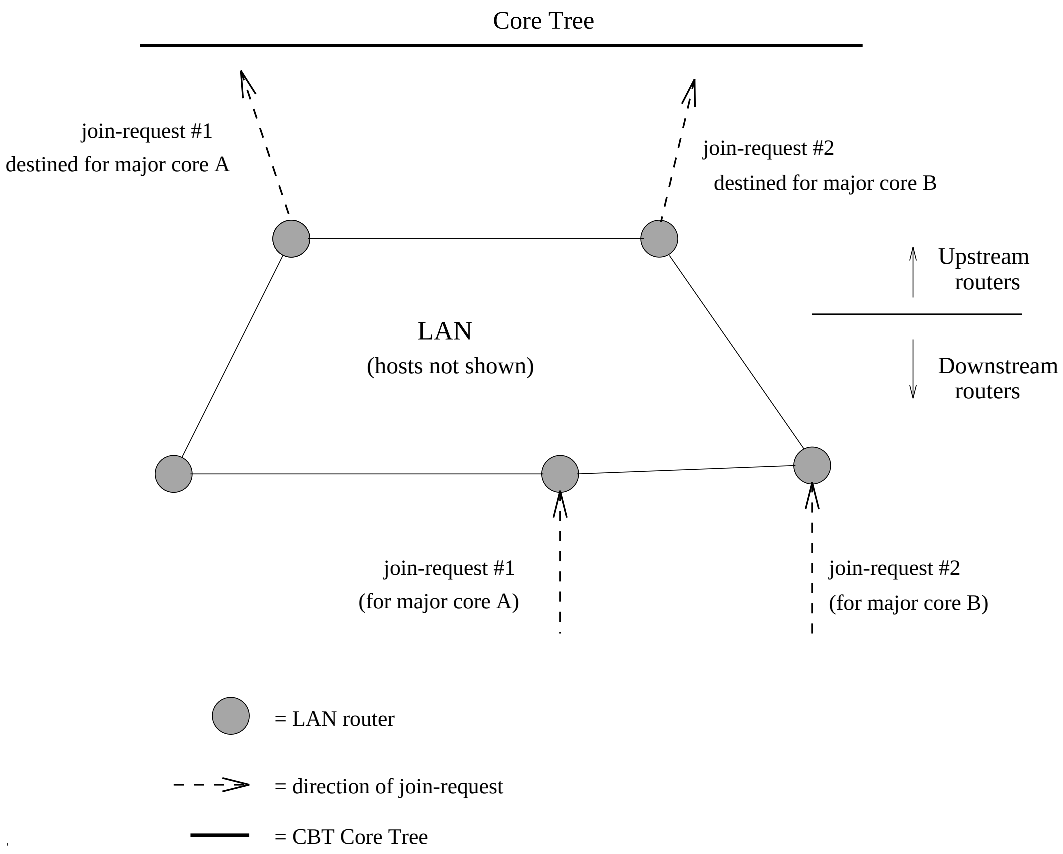


Figure 4: The Multi-Access LAN Problem

This obviously cannot be allowed to happen since not only would it cause "cycles" in the tree (resulting in looping), but would also result in duplicate multicast packets arriving on the LAN from upstream, and duplicates being sent from the LAN upstream.

The problem can be alleviated rather simply since we can take advantage of the way CBT uses unicast addressing to perform multicasting. This would certainly not be the case if CBT used *class D* addresses, since modern router interfaces are programmed to pick up all multicast packets.

What we must do is imagine a multi-access LAN as one large node (router) on the CBT tree. Any node on the tree has only one interface leading to/from the **core-tree**, and zero, one or more interfaces leading away from the **core-tree**.

The two problems described in our example are solved as follows:

Problem 1: A router has received a *join-request* from downstream (in relation to the core tree) and must decide between two or more routers on the LAN which each have equal-cost paths to the addressed **major core**.

The problem is solved by electing the router with the *lowest address* from those with equal-cost paths. Then, the *join-request* and all subsequent multicast packets pertaining to that group that arrive from downstream are encapsulated at the link level and addressed to the elected router.

Problem 2: As we have already explained, it is perfectly valid for a second *join-request* for the same group to arrive at a different router on the LAN than the previous one, and be addressed to a different **major core** for the same group.

If we imagine the LAN to be one large node (router) on the CBT tree, the solution should be obvious. As we explained in section 4.4, when a *join-request* arrives at a node that is already on the tree, whichever **major core** for the group the *join-request* is addressed to, the join is terminated at that node, and the path towards the **core tree** is via that node's parent interface for the tree. Thus, that node can ACK the *join-request*.

More details on the solutions to the above problems are given in the following subsection.

5.1 Further Details on the LAN Issue

As we described in the preceding section, we must imagine a multi-access LAN as one large node (router) on the CBT tree, with one interface leading to the core from the LAN, and from the core to the LAN.

There may be zero, one or more legitimate LAN interfaces (routers) leading away from the LAN for any particular group tree.

We now describe the details of two scenarios with respect to multi-access LAN CBT multicasting:

Scenario 1: There are no members for CBT group g on multi-access LAN l, and some host(s) on the LAN is/are sending to group g. In this case, there are no complications involved and the CBT multicast packet will be encapsulated as normal at the link level and addressed to some local router that has a shortest-path to the core. This router then forwards the packet on its way to the addressed **major core**.

If the chosen router happens not to have the shortest-path to the addressed core, that router will nevertheless forward the packet, and then send an *ICMP-Redirect* message to the originating host so that subsequent packets will be addressed to the more appropriate router[3].

Scenario 2: This is the more complicated scenario where a *join-request* is received by a router on a multi-access LAN. As we have stressed, there must be only one LAN interface to/from the core. Thus, when a *join-request* arrives at a router on a multi-access LAN, that router must take immediate action to avoid the problems occurring that we outlined in the previous section. Exactly what happens is described below.

[3]The router to which packets are sent for forwarding maybe DOWN. In the absence of a *router-discovery protocol*, hosts may never dynamically discover that the packets they are sending to it are disappearing "into a black hole".

On receipt of a *join-request*, router *r* performs the following:

- elects a router on the LAN that has the shortest-path to the addressed **major core**. If there is a tie, the router with the *lowest address* is elected as **designated core multicast router**[4].
- Router *r* sends a special *CBT LAN broadcast* containing the following information:
  1. **group-id** contained in the *join-request*
  2. designated core multicast router IP address
- The *join-request* is forwarded to the designated router.

All routers on the LAN are expected to store this information, together with a timer (of say, three minutes) for reference purposes, in case another *join-request* is received before the previous *join-request* has been ACK'd. When the *join-ACK* arrives from upstream, it is forwarded onto the LAN by the **designated router**, and is eventually picked up by the sending LAN router, which may forward it which may or may not forward it downstream, depending on whether it received a join, or originated one.

CBT LAN broadcast messages are sent by a single router on the LAN say, once every three minutes. They are stored by all LAN routers with a timer. Thus, the router responsible for broadcasting must continue to do so once within the timeout period, as long as it is a node on the CBT tree. Information in successive broadcasts overwrites previous information with respect to the group identified by **group-id** provided it is broadcast by the same source.

If another router on the LAN recieves a *join-request* for the same group, it may not have received a previously broadcast message from some other router on the LAN w.r.t. the same group (however, this is considered highly unlikely given LAN media bandwidths and reliability). If this second router begins broadcasting the same or different information w.r.t. the router on the LAN with the shortest-path to the core, all other routers would receive it and recognise it as being generated by a different source than the first. In this (again unlikely) case, the two broadcasting routers will choose between themselves, which is going to elect the **designated core multicast router** and be responsible for LAN broadcasts.

The diagram over shows a LAN with a **designated router**:

[4] The **designated core multicast router** will be responsible for operating the "Host Membership Protocol" (IGMP)

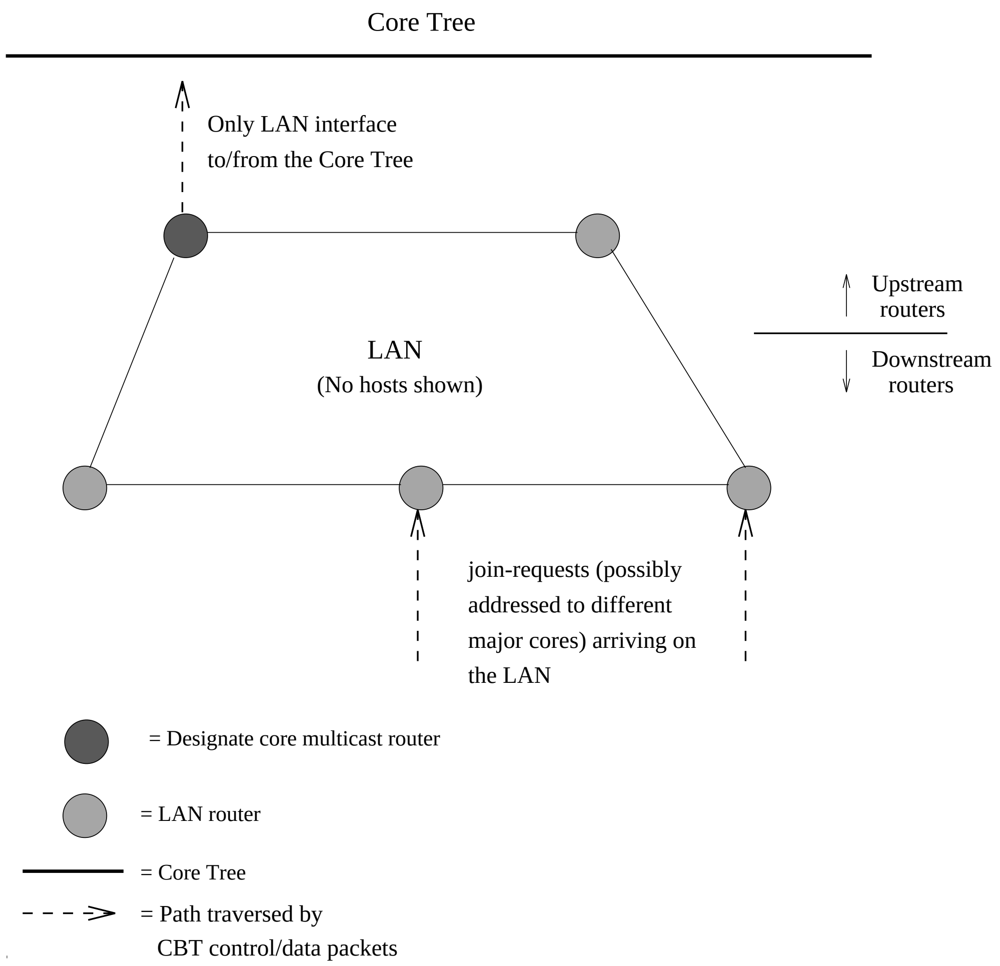


Figure 5: The Solution to the Multi-Access LAN Problem

If a *quit-request* is received between broadcasts, and there is more than one downstream LAN interface for the CBT tree, then multicasts received via those interfaces will continue to use the **designated core multicast router** until the entry expires for that group. This, and the case where the **designated router** fails, are now considered:

Scenario 1: A downstream CBT tree router interface has received a *quit-request*. It will encapsulate the *quit-request* and address it to the **designated router**. Since the **designated router** is also responsible for running IGMP, it will know if there are any more CBT routers on the LAN pertaining to that group. If there are, the *quit-request* is not forwarded further. If there are not, then it forwards the *quit-request* upstream and removes itself from the tree.

> **Note:** *If the* **quit-request** *has been sent by the router responsible for CBT LAN broadcasting, then assuming there are still members remaining on the LAN, it assumes responsibility for subsequent broadcasts.*

Scenario 2: The router responsible for CBT broadcasts is DOWN. This could potentially happen at any time. It is a fairly certain assumption that a dynamic routing protocol will be operating between the routers on the LAN which should indicate the status of the CBT LAN broadcasting router. If it is indeed DOWN, the **designated router** on the LAN assumes responsibility for that role.

5.2 Summary

In summary then, once a *join-request* has been received by a router on a multi-access LAN, all other routers on that LAN are informed of the **group-id** and the **designated router**, which is considered the only LAN interface to the **core tree**. After the *join-ACK* has been received and forwarded, all routers on the LAN have the information necessary to allow the LAN to simulate a single node on the CBT tree, with one interface leading to/from the core, and (potentially) several interfaces leading away from the core.

6 More on how a Router becomes a Major Core

Any available IP router can be requested to become a **major core** at any time by **directory service**. It is assumed that directory service will have a full list of most recently available routers (subject to routing update period), each of which it can request to become a **major core** for a particular group(s) at any time. **Directory service** make such a request by means of a *core-request*. Network managers have the authority to configure their routers to reject *core-request*'s if they so wish.

When a *core-request* is successfully accepted (by means of an ACK as opposed to a N-ACK), directory service makes an entry in its **group table**. This contains a link list for every group present in the internetwork. As long as a group is in existence it will have at least one link (for the case of a single-core tree) containing the address of the **major core**. As more **major core**'s become assigned to a group, their addresses are added to the corresponding link list. An entry in the **group table** of **directory service** would be as follows:

Group-id	$core_1$ addr.	$core_2$ addr.	$core_3$ addr.	NULL

On receipt of a successful ACK to a *core-request*, **directory service** will send a packet containing the address of an existing **major core** (or alternatively all the core addresses) that the new core should join. If this packet is empty, the new core assumes it is the core for a single-core tree and awaits *join-requests* that will be pending, since cores are only assigned as such when some end-system requested the presence of the group.

As the group becomes larger and more widespread, directory service can assign further **major cores** for the group. The assignment of these is left at the discretion of directory service. Additional core assignment becomes necessary when potential members request the address of a **major core** for the group but are geographically distant from present members. Alternatively, a new **major core** may be assigned for reasons of robustness.

Any newly-assigned core must send a *core-join request* to the **major core** determined by directory service (or alternatively determined by itself from a list supplied by directory service). This is described in the section 8.

7 Core Assignment and Optimality

It is the assumption of this draft that **major cores** will be optimally placed (or rather, near optimally placed) with respect to group members, whether groups be small or large. We have left this "magical assignment" to *multicast directory service*, but have not said how directory service should/could achieve near-optimal core placement. There is no doubt that such a specification is outside the context of a multicast routing protocol, but it is clearly a subject of future work.

However, in our opinion, the heuristic required to achieve near optimal core placement is likely to be simpler rather than the result of complex mathematics or graph theory. We may well propose a solution before a CBT implementation is complete.

Even with optimal core placement, it should be obvious that CBT does not always provide shortest paths between a multicast source and all destinations on the tree, as opposed to source-based multicast. There are some for whom this will be unacceptable, but we must focus on the problems at hand; firstly, it is highly probable, given optimal or near-optimal core placement, that the paths traversed by CBT multicast packets will be *mostly* shortest-paths, and so will compare favourably with source-based multicast trees. We have yet to carry out experimentation to prove this, but we don't expect an optimality margin any greater than 10-15%.

Second, we have to think of the problems at hand. As far as multicast routing is concerned, we should first and foremost ensure scalability, as well as simplicity, robustness and information hiding.

Thirdly (and rather more cynically), most routing algorithms/protocols are in one way or another sub-optimal, whether it be how they handle congestion, or react to topology changes. What, in the area of routing, is optimal in *all* cases!

Finally, there is the interesting case of locating network services distributed throughout the internetwork. So called *resource discovery* is a topic of current research, but there is no doubt that multicast is an important aspect of it, with for example, one multicast group linking together all locations relevant to a particular service. Such an application would require *permanent group* trees to be maintained. Again, assuming reasonably optimal core placement between the members of such a group, a *client* need only send a request by means of a CBT multicast packet, and unicast it towards a core for that particular tree. The first available service receiving the request on the tree would reply by means of a unicast to the sender. Used in this way, CBT must surely offer one of the most attractive solutions to the *resource discovery* problem.

7.1 Dynamically Improving Optimality

It may well be that when a *join-request* arrives at a router that is not already on the tree, that router's most optimal path to the addressed core is temporarily not available. In order to keep the join latency low, we decided that all routers must immediately forward *join-requests*, and in the above case, the join would be forwarded on the next-best path, which may be considerably less optimal.

We have made a provision for the tree to be able to re-configure itself so that the paths traversed to the **core tree** are mostly optimal. The overhead required to do this is negligible and the packet loss resulting from the transition may well be zero.

Consider the following, where router X (not yet on the tree) must forward an incoming *join-request* to next-hop Y on its way to **major core** A, since it's best next-hop, Z, is temporarily not available.

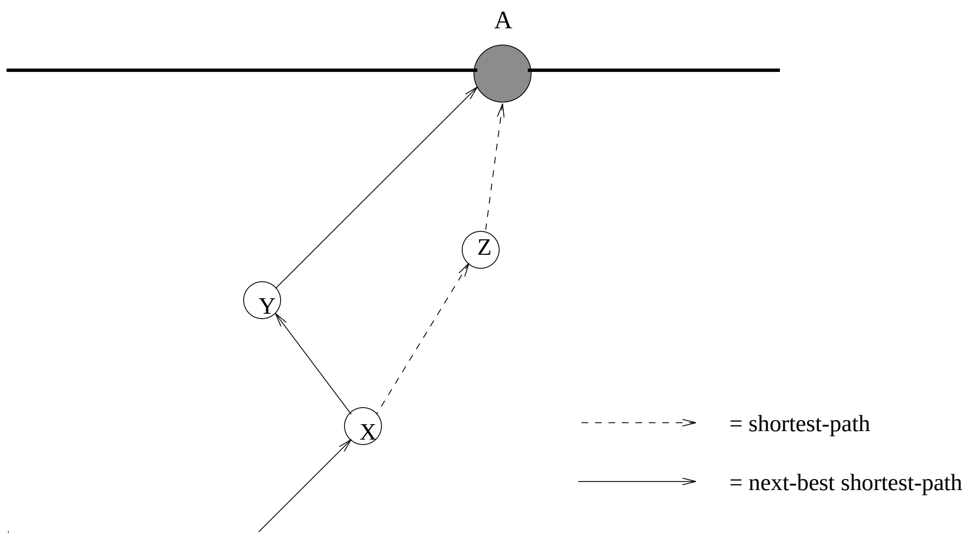


Figure 6: CBT branches may change dynamically

In the above case, the branch formed will be X, Y, A. However, the best path is X, Z, A. When Z becomes available again, X can send a second *join-request* to Z, which forwards it on towards A. A will then send a *join-ACK* back along the path A, Z, X. When X receives it, it can "keep it on hold" before making a new entry (parent interface in routing table) for that CBT tree.

In an instant when all the of X's CBT interfaces are silent, it could then make the transition to shortest-path by simply replacing the its parent entry in it's routing table. After the transtion it sends a *quit-request* out on the old interface.

8 Major Cores and Minor Cores: The *core-join request*

A *core-join request* will always be forwarded by **non-core routers**. However, **minor cores** will always *terminate* a *core-join request.* There are 5 scenarios to consider when a **core-join request** is sent to an existing **major core router**:

- **Scenario 1**: There is a point-to-point link between the sending **major core** and the recipient **major core**. In this case the *core-join request* arrives directly at the destination **major core**. The recipient core sends a *core-join-ACK* back to the originating core, and proceed to exchange *Core Tree Maintenance Protocol* messages.

- **Scenario 2**: The *core-join request* encounters a router that is not any part of the CBT tree for the group identified by **group-id** in the *core-join request.* The receiving router will cache the request (which contains additional identifying information) and the address of the incoming interface, forward it using normal unicast routing to the next hop on its way to its destination (**major core**), then add the outgoing interface address to the cached information. When/if a matching *core-join-ACK* is received, the cached information is used to ensure that it arrived on the correct interface. If so, the appropriate flags are set in the **flag table**/routing table, and a *core-join-ACK* is forwarded on the interface as specified in the cached information. After a *core-join-ACK* has been forwarded, it becomes a **minor core** on the **core-tree** for that group. **Minor cores** are obliged to forward *Core Tree Maintenance Protocol* messages.

  If, for any reason, a check fails, the packet is dropped.

  Note: *The interfaces over which a* **core-join-ACK** *is received/sent are recorded in the routing table by flags, one for each interface that corresponds to the core tree. It is necessary for a* **minor core** *to know its parent core-tree interface and child(ren) core-tree interface(s), and these are marked as such in the routing table.*

- **Scenario 3**: The *core-join request* encounters a **non-core** router on the CBT tree. **Non-core** routers are obliged to forward *core-join requests* to an upstream neighbour that is the next-hop on the CBT tree. Once a *core-join request* hits a **non-core** router, irrespective of the **major core** it is addressed to, it is forwarded on the **parent** interface of that **non-core router** towards. Thus, the *core-join request* may no longer be destined for the **major core** addressed in the IP header. The *core-join request* continues it's journey on a branch of the CBT tree for the appropriate group, and will eventually hit either a **minor core** or a **major core**.

  If a **major core** is encountered, whether it is the one addressed or not, it is ACK'd by it. The ACK traverses the same branch back to the source and all routers along the way change their status to **minor core**, except the final recipient router, which now becomes a new **major core**. In this way, a new branch of the **core tree** is formed.

  Design decision: In this way, a **core-join request** will only traverse existing branches of the tree and in doing so prevent *core-join requests* "crossing over" **non-core** routers, which could potentially lead to "cycles" in the tree.

Note: The *core-join ACK* must contain an explicit field as to the true identity of the parent **major core**. The reason should be apparent from **scenario 5**.

- **Scenario 4**: The first router encountered by a *core-join request* is a **major core** different from that in the destination field of the request. This is a somewhat unlikely scenario since it would indicate that **directory service** had made an error in its choice of new **major core**. **Major cores** are expected to be geographically isolated from each other, and so it is unlikely that **directory service** would assign a router to become a **major core** so near to an existing **major core**, for any one group.

  However, if the first router encountered were indeed a different **major core** than the one addressed in the *core-join request*, it will terminate it and ACK it. The source address of the ACK is the true identity of the **parent**.

- **Scenario 5**: The *core-join request* encounters is a **minor core** for the group. The *core-join request* is *terminated* at this point. The recipient **minor core** next formulates a *termination-notification* packet which contains among other things, the address of the source (**major core**), and forwards it over the core tree interface which leads to the addressed core (as indicated in its routing table). By doing this, *Core Tree Maintenance Protocol* message exchanges may be more evenly distributed. The *termination-notification* packet serves to indicate to the addressed **major core** that it has a new **major child** with which to exchange *Core Tree Maintenance Protocol* messages.

  On receipt of the *termination-notification-ACK*, the **minor core** formulates a *core-join-ACK* and forwards it on the interface over which it received the *core-join request.* As already mentioned, any routers on or off the tree that the *core-join-ACK* traverses becomes a **minor core** for the group, so the necessary updates must be made in the **flag table** of each router. In this way, new branches of the **core-tree** are formed, and the **core-tree** branch interfaces are recorded by each **minor core** by setting the appropriate flags in the routing table.

Note: The *core-join-ACK* formulated by the **minor core** must have a field containing the identity of the true *major parent* of the new child. By assuming the source address of the ACK would be clearly incorrect.

9 Core Tree Maintenance Protocol

An introduction to the *Core Tree Maintenance Protocol* is given below, but the actual detailed working of the protocol is made clear in the sections on the *core-state-notification* packet and *core-list packet.*

ICMP messages form the basis of the *Core Tree Maintenance Protocol.* However, other types of message are generated as part core-tree maintenance, so we shall refer to these messages collectively as *Core Tree Maintenance Protocol* messages. Only **major cores** are ever the source/destination of *Core Tree Maintenance Protocol* messages. More precisely, only **major parents** and thier respective **major child(ren)** exchange *Core Tree Maintenance Protocol* messages.

For any group, say group A, the *Core Tree Maintenance Protocol* messages must traverse only the **core-tree** for group A. Only in this way will **core-tree** partitions be noticable.

Design decision: This is an example of why it is necessary to distinguish between **minor cores** and **non-cores**. The latter will never receive **Core Tree Maintenance Protocol** messages, and so need not know how to deal with them.

Whenever a **major core** sends/receives a *core-join request*, it records the destination address present in the *core-join-ACK* (or the source address in the case of receiving a *core-join-ACK*) and the interface over which it forwards the ACK (or received it). This information is recorded by **major cores** in the **major parent-child table**. This not only gives information on who a **major core**'s **major parent/major child(ren)** are, but also the interfaces over which they are reachable, since **major parent/child(ren)** will not be directly connected neighbours. The interfaces will obviously be the CBT **core-tree** interfaces.

Note: If a **major core** receives a *termination-notification* packet from some **minor core**, and replies with a *termination-notification-ACK*, address and interface information is similarly recorded, as above, in the **major parent-child table**. The **major parent-child table** is illustrated below:

Group-id	major-parent	i/f	$major - child_1$	i/f	$major - child_2$	i/f	....	....

A **core-list packet** contains the addresses of the **major cores** for a CBT tree that are considered to be currently available. A **core-list** packet is only generated in response to the addition or removal of **major cores** and is the result the result of a *core-state-notification* packet having stabilized. A **core-list** packet is multicast by one **major core**. This process will be more fully explained in the sections 10 and 11.

Non-core routers cache and forward **core-list** packets on all outgoing CBT tree interfaces, i.e. multicast. **Non-core** routers need take no action on receiving a **core-list** packet unless its parent is a **major core** and has become unavailable. The parent may have become unavailable since the generation of the **core-list**, or the **core-list** was generated as a result of *that* **major core** being removed. In either case, it is the responsibility of the **non-core** router to send a new *join-request* to the "nearest" **major core** listed in the **core-list packet**.

If a **major core**, to which a **non-core router** is attached becomes unreachable, it is the responsibility of that **non-core router** to send a new *join-request* towards another **major core** selected from its current **core-list** packet. In this way, only the **non-core** router in the immediate vacinity of a CBT partition need take action to ensure that the branch becomes re-attached to the tree. Thus, the partition remains transparent to other **non-cores** not in the direct vacinity of the partition.

Minor cores ignore **core-list** packets. **Minor cores** play no direct part in repairing partitions of a **core-tree**, but do serve to help prune the **core-tree** after a failure has occured.

Major cores cache the most recently received **core-list** packet. A **major core** must use its cached *core-list* if it establishes that it's parent is DOWN, in order to restore **core tree** connectivity. When reachability is lost between **major child** and **major parent** because of an intermediate link failure, it is the responsibility of the **major child** only, to re-establish a (virtual) link with its **major parent**. The *core-list* is not required for this.

Design decision: Indeed, if both **major cores** did try to re-establish connectivity with each other, each would become the parent (and child) of the other.

Major-parent and **major-child(ren)** constantly test each others "liveness" by exchanging **ICMP echo request/replies**. If a reply is not received after a certain timeout period, one of two kinds of failure is assumed:

- **Scenario 1**: **Major core failure**.
- **Scenario 2**: **Minor core failure** (or some link between parent and child **major cores** is down.

Scenario 1: **Major core failure** - A **major parent** (or child) can establish to a high degree of probability the reason for not receiving a response to a *ping* message. It can assume failure after trying to reach the parent/child via all other possible routes. Again, if no reply is received within a small timeout period, the parent/child is assumed DOWN.

Only the **major child(ren)** of the failed **major core** is responsible for re-establishing full connectivity of the **core-tree**. Each child does so by making *core-join request* to another **major core** as described in the section 11 on the *core-list packet*. After a **major core** has received the *core-join-ACK*, its new **parent-core** generates a *core-state-notification* packet which is disemminated on *all* **core-tree** interfaces for that group. When the *core-state-notification* packet stabilizes, i.e. the packet sent out on a **core-tree** interface is the same as that received, the contents of it are copied to a **core-list packet** and multicast on the CBT tree. Only the **major core** that originated the *core-state-notification* packet generates and multicasts the **core-list packet**.

Scenario 2: **Minor core (or link) failure** - if, on the other hand, an **ICMP** reply is received via a route not on the **core tree**, then the core tree link is assumed broken. As in the previous case, it is the responsiblility of the child to re-establish core tree connectivity. The underlying unicast routing algorithm will provide a new route to the parent **major core**. The new route may or may not include hops that were part of the old **core-tree**. What follows is described in the next section on the *quit-request*.

9.1 The *quit-request*

If it has been decided that a core tree link is broken, the **major child** sends a new *core-join request* towards its **major parent**. If the first next-hop of the new route is different from the old, it must also send a *quit-request* to the old next-hop which formed part of the original **core-tree**.

Quit-requests can have a *cascading* effect in that whenever any router on the CBT tree receives one from a child router, it too sends one to its parent iff it has no further children and no member hosts on attached subnets. The exception to this is when a *quit-request* arrives at a **non-core** router from a downstream **non-core** router, and the directly connected parent is either a **minor** or **major core**. In both cases the *quit-request* is sent to the parent which removes the child flags as appropriate from the routing table, but no further *quit-requests* are generated from this point.

A *quit-request* is only permitted in the direction child to parent, and never in the reverse direction. Furthermore, *quit-requests* are always terminated at a **major core**.

10 *core-state-notification* (CSN) packet

A *core-state-notification* packet is generated only by **major cores** as the result of a **major core** having accepted a new *core-join request*, or by a **major core** as the result of that core sending a *quit-request*. The CSN packet contains a unique identifier as well as the **group-id** and originator's address, both of which remain constant until the CSN packet stabilizes.

Once a new **major core** has been ACK'd by its **major parent** (either explicitly or by having received a *core-join-ACK* from a **minor core** in response to the **minor core** having received a *termination-notification-ACK* packet), the parent generates a *core-state-notification* packet containing a list of its major core children.

Alternatively, a **major core** has sent one of it's children a *quit-request*. In this case too, the sending core generates a CSN packet.

The CSN packet is distributed on each **core-tree** interface for the group. Exactly how the CSN packet works is described in the next section.

10.1 How the CSN Packet Works

In a multi-core CBT tree, all **major cores** have at least one **major core** (virtual) neighbour. When the state of the core tree changes as the result of a new core addition or core removal, a mechanism is needed to inform the rest of the CBT tree of the new state. This mechanism is the *core-state notification* packet exchange which results in a *core-list* packet being multicast on the tree.

It works as follows; whenever a **major core** accepts a new child, or sends a *quit-request* to remove one, it must take action on reporting it. The parent to the new addition/removal generates a *core-state notification* packet reporting itself and its children that are UP. This CSN packet is broadcast on all **core tree** interfaces, but not on other CBT tree interfaces. The aim of the CSN packet is for it to arrive in a state where each child is terminated by a "NULL" - this "NULL" entry can only be made by a child if it has itself no children. If a **major core** has no children, then a "NULL" follows it's own entry. When a **major core** cannot append information to a broadcast CSN packet is it forwarded on outgoing interfaces only if it is as large or larger than the CSN packet the node currently knows about. If it is not the CSN packet arriving is dropped. In this way, a slow responder cannot flood the the core tree with out-of-date packets. If however, a broadcast arrives which is as big or bigger than the one it currently knows about AND can modify it, the resulting CSN packet is broadcast on all outgoing **core tree** interfaces.

The CSN packet is considered to have stabilized when all child entries in it are suffixed with "NULL", and when childless parent entries are also suffixed with "NULL". A stabilized CSN packet is copied to a *core-list* packet and multicast on the CBT tree by the originator of the first CSN packet. A diagram is given in section 11.

A diagram may have been appropriate here, but because of the potential for each **major core** to be slow/faster than the others, and as broadcasts can arrive in different orderings because of faster/slower nodes, we thought a diagram may have been misleading, so we hope the explanation suffices.

11 *core-list packet*

As far as **major cores** are concerned, the **core-list** packet is crucial to core-tree maintenance, allowing **major cores**, where appropriate, to re-establish **core-tree** connectivity after it has failed.

The structure of the **core-list** packet is simple, but is catalytic in repairing **core-tree** connectivity. As we do not envisage the number of **major cores** to be more than a handful for very large groups, the **core-list** is uncomplicated and remains even in the worst case, quite small. It consists of a concatenation of **major cores** and their **major core children** in the order they joined the parent.For example, take the following (large for demonstration purposes) **core-tree**:

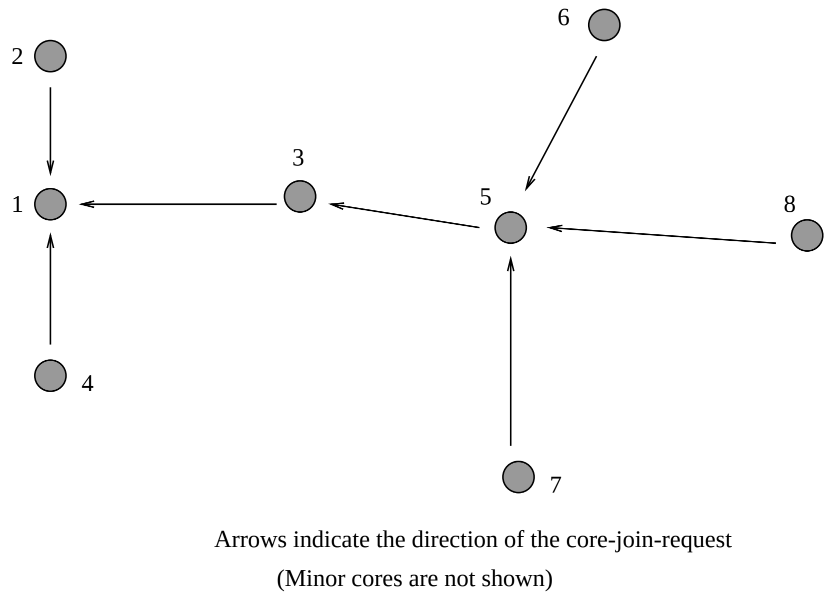


Figure 7: CBT Core Tree

The numbers only serve to illustrate the example and indicate the order in which **major cores** were assigned to the group. The **core-list** packet for this group will be as shown over:

1						
2	NULL	3	NULL	4	NULL	
2						
NULL						
3						
5	NULL					
4						
NULL						
5						
6	NULL	7	NULL	8	NULL	
6						
NULL						
7						
NULL						
8						
NULL						

Figure 8: CBT core-list packet

If a **major core** fails and it has no **major core** children, the failed **major core** is a *leaf* on the CBT **core-tree**. In this case, no action is necessary since **core-tree** connectivity is maintained. If there are **non-core** router children hanging off it, it will be the responsibility of the **non-core** whose parent has failed, to make a *join-request* elsewhere.

Assuming that the failed **major core** has at least one major child, it may be necessary for the child/children to restore **core-tree** connectivity. The first **major child** that joins a **major core** is tagged to indicate it is the oldest child for that core. While there is only one child, it is also the youngest, so it is also tagged as such. The oldest child will obviously retain the oldest tag (as long as it is alive), but as new major children join, the youngest tag changes.

Returning to the example of the failed **major core**. Its youngest major child must send a *core-join request* to the next-oldest, which then does the same. This continues until the oldest child has been reached. The **major core** this child sends a *core-join request* to depends on whether the failed parent was the child of any other **major core**. There are two scenarios:

- **If the failed major core has a parent**, the youngest child must join that parent.
- **If the failed major core has no parent**, then the youngest child of the failed core need not make a *core-join request* to any other **major core**, since original **core-tree** connectivity will have been re-established simply by having linked the children together, if indeed, there were any present.

Design decision: When a **core-tree** partition occurs, simply having a **major core** in the vacinity of the partition send a new **core-join request** to *any* of the other **major cores** listed in the **core-list** packet would NOT ensure full **core-tree** connectivity, since a request could potentially be made to a **major core** on the same side of the partition. The method described above ensures that full **core-tree** connectivity is restored after a partition.

Let us take the previous example again to show this:

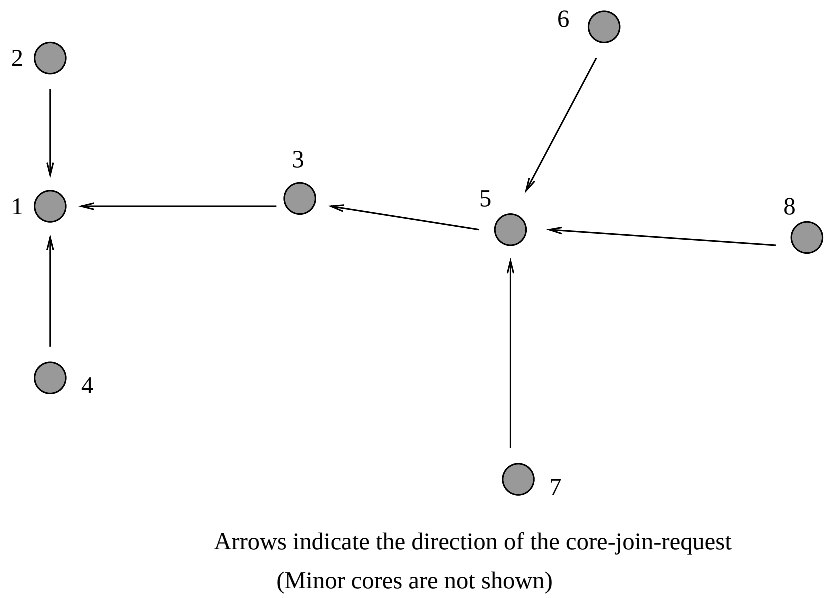
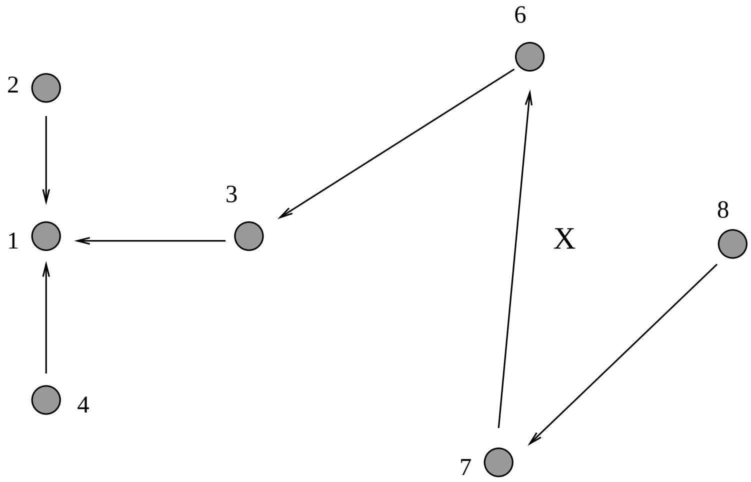


Figure 9: CBT Core Tree

When 5 fails: From the **core-list**, **major core** 5 has 3 major core children (6, 7, and 8). Sequence of events:

- 8 joins 7, 7 joins 6.
- 6 scans the **core-list** to establish whether 5 is the child of any other **major core**.
- There is a hit. 3 is the parent to 5.
- 6 joins 3 to re-establish **core-tree** connectivity.

The new **core-tree** is shown over:



The arrows indicate the direction of the core-join requests
(Minor cores are not shown)
X = position in core-tree of failed major core

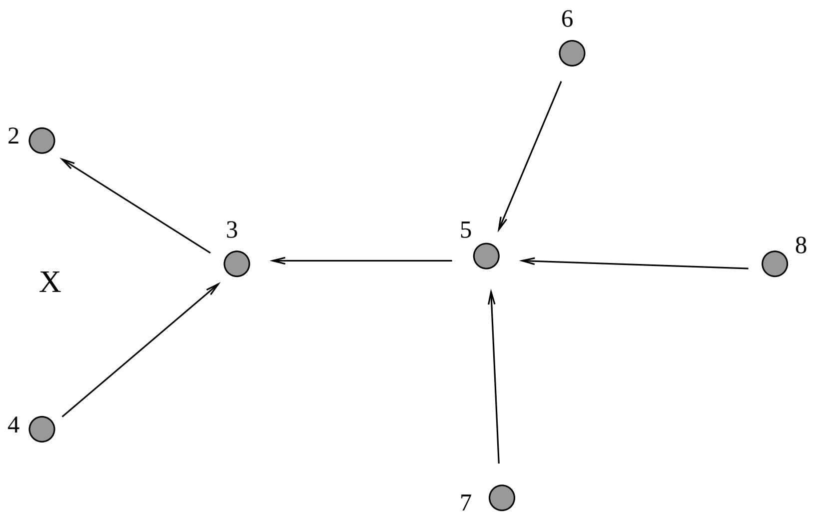
Figure 10: Repaired CBT Core Tree

Another example is given over ....

Another example. Assume **major core** 1 fails. Sequence of events:

- 4 joins 3, 3 joins 2.
- 2 scans the **core-list** to establish whether 1 is the parent of any other **major core**.
- There is no hit. Thus, no action is taken, and **core-tree** connectivity is re-established.

The resulting **core-tree** is as shown below:



Arrows indicate the direction of the core-join requests.
(Minor cores are not shown here)
X = indicates the position of the failed major core.

Figure 11: Repaired CBT Core Tree

12 Acknowledgements

There are several people whom we would especially like to thank for their valuable contributions to CBT. Firstly, Ian Wakeman (UCL) whose suggestion it was to separate the **group identifier** from the IP address and place it in the options field of the IP header. We considered this an important design issue offering several advantages over the original design.

Secondly, Joel Halpern (Network Systems Corporation) and Steve Deering (Xerox PARC). Joel identified many of the problems encountered when CBT multicast packets traverse or originate on a multi-access LAN. Both he and Steve offered valuable insights into the LAN multicast problems and made many comments and suggestions as to how these problems could be solved.

Finally, we would like to thank all those on the IDMR mailing-list who contributed to, or/and made comments on, CBT, however small these may have been. We consider these, and all ongoing comments and suggestions important in devoloping a standard inter-domain multicast routing protocol that will cater for the requirements of the internet community for the forseeable future and beyond.

We encourage continued participation via IDMR on CBT and all other inter-domain multicast routing issues.

A Authors' Addresses

Anthony J. Ballardie,
Department of Computer Science,
University College London,
Gower St.,
London WC1E 6BT,
ENGLAND.

e-mail: A.Ballardie@uk.ac.ucl.cs
Office Tel: ++44 (0)71 387 7050 ext. 3701

Paul F. Tsuchiya,
Bellcore,
445 South St. 2L-281,
Morristown,
New Jersey 07960,
U.S.A.

e-mail: tsuchiya@thumper.bellcore.com
Office Tel: 1-201-829-4484

Jon Crowcroft,
Department of Computer Science,
University College London,
Gower St.,
London WC1E 6BT,
ENGLAND.

e-mail: J.Crowcroft@uk.ac.ucl.cs
Office Tel: ++44 (0)71 380 7296

References

[1] S. E. Deering. *Multicast Routing in a Datagram Internetwork.* PhD thesis, Stanford University, California, U.S.A., 1991.