

Scalable Internet Multicast Routing

M. Parsa, and J.J. Garcia-Luna-Aceves
Department of Computer Engineering
University of California
Santa Cruz, CA 95064
courant, jj@cse.ucsc.edu

Abstract

In distributed network applications such as multiparty teleconferencing, users often need to send the same message to several other users. To achieve such one-to-many or many-to-many communication efficiently in wide-area internetworks, it is imperative to support multicast, i.e., concurrent sending of messages from one source to multiple receivers.

The current IP architecture for multicast routing, the Core-Based Tree (CBT) protocol, and Protocol Independent Multicast (PIM) protocol have a number of limitations for very large internets. To eliminate their limitations, we propose a new multicast routing protocol, called Scalable Internet Multicast Protocol (SIMP).

1 Introduction

Multicast service in WANs has been achieved by extending two common routing algorithms used by network layer routers, i.e., distance-vector and link-state algorithms [2]. This service enables sources to send a single copy of a message to a special address that represents a set of destinations, and the message is delivered to each destination in the set.

Based on Deering's work [2], the TCP/IP suite constructs multicast delivery trees on-demand [6, 8]. For example, the Multicast Backbone, Mbone, in today's Internet consists of a set of routers running DVMRP. However, there are several shortcomings with the existing IP architecture for multicast routing. First, the architecture is inefficient when group members are sparse and widely-distributed in the internet. Because it is assumed that most of the networks have group members, all corners of the internetwork are flooded periodically with messages when the state information for a multicast tree times out. Second, packets traverse paths that do not lead to any receivers or sources, consuming network resources. Therefore, routers not on the multicast tree incur memory and processing overhead to construct and maintain the tree for the lifetime of the group. Third, the multicast routing information is stored for each multicast source. If there are S sources and G groups, the multicast protocols scale as $\Theta(SG)$.

To overcome the above shortcomings of the IP architecture, two protocols have been recently proposed: core-based tree (CBT) architecture [1] and protocol independent multicast (PIM) architecture [3]. Although both approaches constitute a substantial improvement

over the current multicast architecture, each protocol has several disadvantages.

The disadvantages of CBT follow: There is only a single shared tree per group. The shared tree imposes a hard constraint on the optimality of routes to all receivers from multiple sources. Furthermore, the communication traffic of different sources for a group tends to concentrate on a few links in a network [9], further decreasing the capacity of a network. The tree is rooted ad hoc at a special router called the core. The optimality of routes can suffer from poor selection of cores. In a dynamic network, CBT requires a dynamic core selection and placement to achieve some upper bound on delay to receivers; otherwise, the paths in the tree can arbitrarily deviated from the shortest paths to receivers in the network. However, to have the core router topologically centered is NP-hard in a dynamic network. As a core-based tree is rooted at the (primary) core, in case of the core failure, the tree may have to be flushed to be re-constructed rooted at an alternate backup core. This response to core failure disrupts the communication of all sources to group members. As we demonstrate in a subsequent section, even with the loop-detection scheme of CBT [1], transient loops can be created in the multicast tree. This can lead to replicates of data packets loading the network links used by the multicast tree. The data packets sent unicast towards the core of a tree by a source can also suffer from loops and be delayed.

The disadvantages of PIM are the following: PIM relies on periodic refresh of state information, called soft-state mechanism, which adds to the control message overhead. Moreover, to achieve stable behavior, the timers controlling the periodic refresh of state information have to be slow, which necessarily incurs long latencies for "seeing" network disturbances and dynamics. Even under stable network topology and group membership, there is a control message overhead in maintaining a multicast tree. The part of a shared tree upstream of RPs is not shared between different sources. That is, routers upstream of an RP on a path from a source have per source state information for the same group. Using a shared tree then does not limit the resource consumption of the shared tree over having a tree per source. This also makes the placement of RPs more critical for the efficiency of a shared-tree. In using a shared tree, an inappropriate heuristic for RP selection can seriously affect the scalability and overhead of a shared tree. As we demonstrate subsequently, data and control messages can traverse transient loops. This can cause receivers to select alternate distal RPs, giving the receivers poor distribution. The interaction of PIM's

*This work was supported in part by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175

timeout mechanism and the oscillatory behavior of unicast transient loops can degrade the efficiency of PIM.

We have developed a new multicast routing protocol called Scalable Internet Multicast Protocol (SIMP), which solves the shortcomings of the current IP architecture, PIM, and CBT. SIMP offers a flexible, simple and unified approach to the construction of multicast trees that are shared among group members or that are shortest-path trees to receivers. It is easy to accommodate the needs of a wide range of multicast applications from sparse widely-distributed replicated databases to delay-sensitive interactive applications. The shared tree of a group scales as $\Theta(1)$ with respect to the number of group sources S . SIMP can be source-initiated or receiver-initiated. SIMP is robust and adapts under dynamic network conditions, such as topology or link cost changes, to maintain multicast routing. SIMP has fast response time to network conditions since network events such as link failures are propagated as fast as messages can travel, as opposed to timers expiring to reflect the new state. Under stable network conditions, SIMP has no maintenance or control message overhead.

The next section presents examples of looping problems in CBT and PIM. Section 3 describes the multicast and network models used in SIMP. Section 4 gives an overview of SIMP. A detailed description and verification of SIMP is given elsewhere [7].

2 Looping problems in CBT and PIM

2.1 Looping in CBT

In CBT, the initiator of a group selects a set of core routers. The identity of all cores, which are explicitly ranked, is communicated to all cores by the initiator. Each core joins the highest ranked (operational) core to form a core backbone. Receivers become part of the core tree by sending explicit join messages to the highest ranked core. The join messages are acknowledged by routers on the tree and the flow of the acknowledgement message to receivers establishes tree edges.

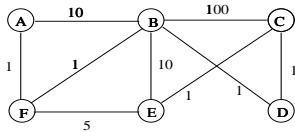


Figure 1: Example.

In CBT, a source sends its data unicast to the core of the multicast tree for a group. The data can experience arbitrary routing table looping on the way to the core from the source, if the underlying unicast routing is not loop-free. This can cause undue congestion and increased delays in the network. There is no mechanism in CBT to prevent this. Moreover, the mechanism to detect loops in the process of reconfiguring the core tree in response to network dynamics may prove inadequate. The basic method for avoiding loops in CBT is timeout of unacknowledged join requests that never reach the multicast tree; however, CBT needs and incorporates an additional measure because of the potential for loop formation in the multicast tree when the tree partitions due to link failures. Because the originating router of a join request that reaches a tree can be the root of that

tree. Unfortunately, all of this is insufficient to prevent looping.

Consider the example network topology shown in Fig. 1. In this example, links (B, D) and (C, E) have failed, and we assume that enough time has elapsed that the unicast routing tables have converged. Let router A be a member receiver and D the primary core for a group. Fig. 2 (a) shows the multicast tree created by CBT, where the thick arrows are the tree edges and they point from a router to its parent in the tree. Let the links (B, D) and (C, E) recover at time t_1 after the tree has been established. Then, at time $t_2 > t_1$, link (C, D) fails. According to CBT, when the link fails, router C is responsible for re-attaching itself (and its subtree) to the multicast tree. Thus, router C tests the highest ranking core, i.e., router D , for reachability by sending a ping message and waiting for a response. Since D is up, C will get a response and decides that core D is reachable. Thus, it sends an active re-join request (as it has a child in the tree) towards D via next-hop E , as shown in Fig. 2 (b). When the active re-join request reaches F , it is acknowledged. The acknowledgement traverses the path that the active re-join request took and establishes the edges in the tree. At this point, there is a loop in the multicast tree as shown in Fig. 2 (c), and data packets that are on the subtree and the new data packets that hit it will loop around. The loop detection mechanism in CBT is based on F and all its ancestors forwarding an inactive re-join up to their parents. This way, when C gets its own re-join request, it detects the loop and sends a quit request to its parent E , and tries again. Depending on the delays in the network, the inactive re-join can reach to C after the acknowledgement for the active re-join, so that the loop can persist for a significant time in CBT. In the example, all the retries by C will fail and C will have to flush the tree.

Similar behavior can be shown to occur if the primary core is down and the secondary core $C2$ is in the subtree of the router C trying to re-join. In fact, all active re-joins sent by C will form a loop. Therefore, the subtree at C has to be deconstructed and flushed.

2.2 Looping in PIM

In PIM, receivers join a multicast tree of a group by sending join messages to special routers, called rendezvous points (RPs), for the group. They also send join messages to sources from which they want the shortest path. The join messages establish multicast forwarding information in routers. The receivers send the join messages periodically to refresh the state information regarding the multicast tree. An RP sends reachability messages periodically to its subtree to inform the receivers in its subtree that it is alive. To provide a level of reliability several RPs are used for each group. A source sending to group transmits its unicast data to all the RPs for a group. When an RP gets data from a source, it sends a join message to the source to establish state information about the multicast tree. This is also done periodically to maintain the tree. Garbage collection is when state information times out at routers.

Unicast routing-table looping affects and degrades the performance of PIM in several important ways. Consider the network topology used in the previous section, shown in Fig. 1. Receiver A sends a join upstream towards RP D as shown in Fig. 3 (a). The join message is forwarded by router B to E due to routing-table

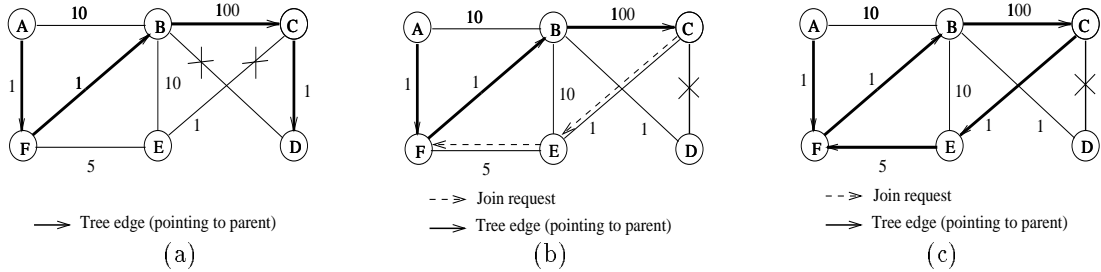


Figure 2: Transient loops in CBT.

looping. This results in establishing E as a child of router F , thereby creating an incorrect forwarding state information at F and E . Although not in this small example, this looping of join messages can be arbitrarily windy and can thus set up incorrect state information in many routers. As a result, data packets are forwarded on paths in the network that do not lead to any receivers due to incorrect information. Shown in Fig. 3(c), the path (E, F) is such a path in the example. This behavior will persist until the incorrect state information times out. Moreover, as a result the loop, routers upstream of B may time out and flush their state information. Thus, data packets cannot flow downstream and they are discarded. Data packets can also experience looping, if router B picks as its parent router which is a descendent on the multicast tree, e.g., E , as shown in Fig. 3(b). Because looping behavior is oscillatory by counting-to-infinity, incorrect data packet forwarding can persist for some time. Thus, the data packets from B can loop back to B and traverse the whole subtree at B again.

Next, let's consider reachability messages, which are sent by RPs to their subtrees to inform the receivers that they are alive. As a result of looping conditions, the periodic reachability messages that are sent downstream can get discarded by the reverse path forwarding (RPF) check. Because the information on the next-hop to an RP may not be correct due to routing table looping. This may lead downstream routers to think that the RP is down, to look up an alternate possibly distal RP, and to start sending joins to the other RP.

3 Multicast and network models

Like CBT and PIM, SIMP adopts the host group multicast model [2], which is used in the existing IP architecture. The model defines the service interface to the users of the internetwork. Each multicast address identifies a group of receivers to which a multicast packet is delivered with “best effort.” The set of the receivers of a multicast packet is called a host group. To send messages to a group, a sender specifies the destination of the messages with the multicast address of the group; it does not need to know the addresses of the individual members of the group. Any sender can send to a group, whether or not it is a member of the group. The number and location of members in a group can be arbitrary and dynamic. The membership of different groups may also overlap.

A protocol (e.g., IGMP [2]) is assumed for the routers to monitor the presence of group members on their attached subnetworks and to propagate and exchange multicast information. Furthermore, for any multi-

access LAN with two or more routers, there is a designated router (DR), just as in CBT and PIM, to act on the behalf of the end hosts on the LAN to start, join, or end a multicast communication and to transmit communication packets of a group. A simple DR election mechanisms suffices, e.g., the router with the largest IP address becomes a DR, or the Hello protocol.

The network consists of an arbitrary interconnection of routers by local area networks (LANs) or point-to-point links. A network is represented by a graph $G = (V, E)$, where nodes represent routers, and edges represent links. The links are bidirectional and have a time-dependent positive cost which is the same in both directions. A link is operational if it is operational in both directions. SIMP assumes there is an underlying protocol that maintains up and down states for links and routers. Thus, a router knows within a finite time if its adjacent links are operational. All SIMP control messages sent over operational links are received correctly and in proper order within a finite time; this service is provided by an underlying protocol. All message arrivals, link failures and recoveries, link-cost changes, are processed one at a time and in the order of their occurrence.

4 Overview of SIMP

SIMP is based on diffusing computations [4, 5]. A router initiates a computation by sending queries to its neighbors and waits for replies from the neighbors to detect termination. A query specifies a computation requested from the receiving router. A query or a reply may have additional information germane to the particular computation requested.

Each router has a link-cost table giving the cost of the adjacent links. It is assumed that link costs are symmetric in both directions. Each router x knows the next-hop router and the distance cost to destinations from unicast routing-table URT^x .

The multicast tree for a group is rooted at a router, which is usually a source for the group. The tree-predecessor and tree-successors are defined with respect to the root. The tree-predecessor of a router x is the parent of x , and a tree-successor of a router x is a router a child of x . In each router x , SIMP maintains a multicast routing table (MRT^x) for recording the tree-predecessor and tree-successor set of a multicast tree at x . This information is used in forwarding data packets. The information about a multicast tree is stored in MRT^x is indexed by (s, g) , where s is the root of a tree for group g . When the source and group of a packet match an entry (s, g) , and the packet arrived via the tree-predecessor of the entry, the packet is forwarded on

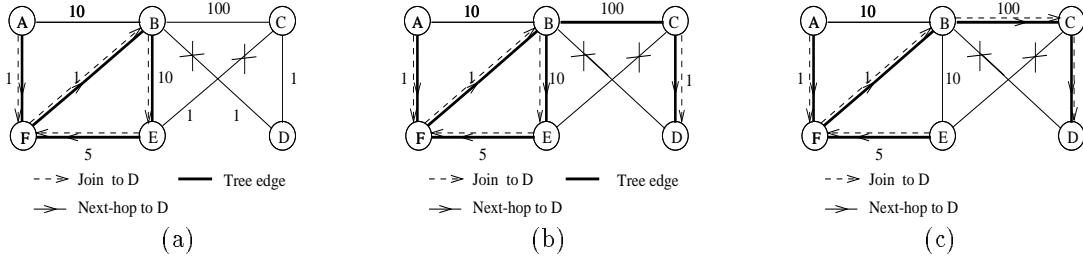


Figure 3: Transient loops in PIM.

all tree-successors of the entry. When the source does not match any entry but the group matches a shared entry, which is identified as having the bit *shared* bit set, the packet is forwarded using the shared tree. Using the shared tree, the data packet is forwarded to all neighboring routers in the tree except the router from which the packet was received.

There are several different computations in SIMP. A query specifies the computation to be carried out. The Expand computation is used by a router to grow or establish a multicast subtree at the router. The Join computation is used by router to become part of the multicast tree. The Terminate computation is used by a router to tear down its subtree. The Root-update computation is used by a router to update cost and root information in its multicast subtree. The Prune computation is used by a leaf router on the tree to remove itself from the multicast tree. In this paper, we have enough space to only describe source-initiated tree construction in SIMP. The complete description is given elsewhere [7].

The source-initiated tree construction is well-suited for small groups, where it is manageable for the source to know the identity of the receivers. A good example of such application is video conferencing, which will ordinarily involve fewer than hundreds of sites. This also gives the source more control over the distribution. The Expand computations are the primary means used in the source-initiated multicast tree creation. The computation proceeds with the transmission and reception of Expand queries and replies. In all queries and replies, the source s and group address g are specified along with other germane information.

All routers are initialized in the idle state. A router involved in an Expand computation is said to be in an expand-active state. A router x in an expand-active state has a tentative predecessor ep^x , called e -predecessor, to which it owes a reply; and has tentative successors, called e -successors, from which it expects replies. When the router finishes its computation and becomes part of the multicast tree, it is defined to be in an expand-passive state.

The description of SIMP is given with respect to a generic multicast tree for a group g and rooted at source s . To avoid cumbersome notation, the subscripting by (s, g) as shown here in the names of auxiliary and state variables and routing information associated with a given multicast tree is suppressed in the description and discussion of SIMP. The following notation is used throughout:

$d^{x,y}$: link cost from router x to y .
 $ep^{x,i}$: e -predecessor of x for initiator i .
 $p_{s,g}^x$: tree-predecessor of x on the path from s .

$D^{x,y}$: path cost from router x to y in a multicast tree.
 $D_{s,g}^{x,y}(M)$: path cost from router x to y in query or reply M .
 EA^x : active expand set of x .
 $EA^{x,y}$: active expand set of x for neighbor y .
 $ED^{x,y}$: expand done set of x for entry (s, g) .
 $EQ^{x,y}$: expand query from x to neighbor y .
 $ER^{x,y}$: expand reply from x to neighbor y .
 $R(M)$: set of destinations in a query or reply M .
 UR^x : set of unreachable destinations at router x .

To start a multicast session, source router s becomes expand-active for a set of members in the group and sends expand queries towards the members. When a member router receives an expand query for itself, it replies with an expand-ack, thereby establishing an edge of the multicast tree. As the replies flow to the source, the edges belonging to the multicast tree are established. When the source gets the replies to all its queries, it becomes expand-passive and the multicast tree spanning all reachable members is established.

When s becomes expand-active, it creates an (s, g) entry in MRT^s which will contain all the pertinent information regarding the multicast tree and its computation. It sets a bit *expand* for the entry to indicate that it is created by an expand computation. If s is going to be the root of the shared tree for a group, it also sets a bit *shared* for the entry. Let's assume for simplicity that s is creating a shared tree. Source s sends query $EQ^{s,x}$ to x which is the next-hop for a subset $EA^{s,x}$ of group members, specifying the subset, the cost $D^{s,x}$ (i.e., the link cost $d^{s,x}$), and the initiator s . It also sets bit *shared* in $EQ^{s,x}$. Each member u for which s sends a query is inserted in set EA^s . Source s designates each neighbor x that receives a query as an e -successor. Its tree-predecessor and e -predecessor are set to null. Source s uses a reply counter for each e -successor to know when it has all the replies from the e -successor. It increments the counter for an e -successor by one when it sends a query to the e -successor. When it receives the reply from the e -successor, it decrements the counter by a value given in the reply.

When an idle router x receives an expand query $EQ^{y,x}$, it becomes expand-active and creates an (s, g) entry in MRT^x . It sets the *expand* bit for the entry. If the query indicates that the tree is shared, x also sets the *shared* bit for the entry. It initializes $ep^{x,s} \leftarrow p^x \leftarrow y$. Router x uses a query counter to know how many queries it has received from y . It is incremented whenever x receives a query. Router x specifies the value of the counter in its reply to y , and zeroes out the counter after giving its reply to y . Thereby, x aggregates its replies to y . If $R(EQ^{y,x}) = x$, router x replies with an expand-ack, in which it specifies itself, the initiator s , and the value of the query counter, and

it becomes expand-passive. If $R(EQ^{y,x}) \neq x$, router x takes the the same steps as s and sends expand queries towards the reachable members in $R(EQ^{y,x})$. For message efficiency, the members for which the next-hop is y are considered unreachable; thus, no query is sent for them. Source s is specified as the initiator in the queries.

Let x in an expand-active state receives another query $EQ^{z,x}$ such that $R(EQ^{z,x}) \neq \emptyset$. It compares $D^{sx}(EQ^{z,x})$ to $D^{s,x}$ to decide its response. First consider the case $z \neq ep^{x,s}$. If $D^{sx}(EQ^{z,x}) < D^{s,x}$, it sends an expand-nack to $ep^{x,s}$, specifying the initiator s , and the value of the query counter. It may optionally specify in the expand-nack the members for which it has received replies, i.e., ED^x . The expand-nack indicates to $ep^{x,s}$ that x is not a tree-successor for $ep^{x,s}$. Router x also sends a prune request to p^x , if $p^x \neq ep^{x,s}$. The prune request removes x from the tree-successor set of p^x . If z is a tree-successor of x , router x removes it from the set. It sets $ep^{x,s} \leftarrow p^x \leftarrow z$ and $D^{s,x} \leftarrow D^{sx}(EQ^{z,x})$. On the other hand, if $D^{sx}(EQ^{z,x}) \geq D^{s,x}$, it sends an expand-nack to z .

Irrespective of $D^{sx}(EQ^{z,x})$, x updates EA^x . Let $X = \{u \mid \text{member } u \in R(EQ^{z,x}) \text{ and } u \notin EA^x\}$. Router x forwards queries for the reachable members in X as before. A basic expand query EQ is defined to be an expand query such that $R(EQ) = \emptyset$. If x updated $D^{s,x}$, it sends a basic expand query to each tree-successor and to each e -successor that is not the next-hop for a member in X .

Now consider the case $z = ep^{x,s}$. If $D^{sx}(EQ^{z,x}) < D^{s,x}$, router x updates $D^{s,x} \leftarrow D^{sx}(EQ^{z,x})$. However, if $D^{sx}(EQ^{z,x}) \geq D^{s,x}$, it saves the value of $D^{sx}(EQ^{z,x})$ until it becomes expand-passive to update $D^{s,x}$. Router x takes the same steps and forwards queries for the reachable members in X as before, where X is defined as above. It sends basic expand queries also as above. The cost specified in the expand queries forwarded is calculated using $D^{sx}(EQ^{z,x})$, e.g., the cost in an expand query to y equals $D^{sx}(EQ^{z,x}) + d^{x,y}$.

Router x in an expand-passive state can receive query $EQ^{z,x}$. If $D^{sx}(EQ^{z,x}) < D^{s,x}$, x sends a prune request to p^x , sets $p^x \leftarrow z$, updates $D^{s,x} \leftarrow D^{sx}(EQ^{z,x})$. However, if $D^{sx}(EQ^{z,x}) \geq D^{s,x}$, no prune request is sent, nor p^x and $D^{s,x}$ are changed. Router x does not have to remember $D^{sx}(EQ^{z,x})$, but it might be useful information while $p^x \neq ep^x$. Lets assume x saves $D^{sx}(EQ^{z,x})$ in variable $D_*^{s,x}$. Next, x sets $ep^{x,s} \leftarrow z$. Note that the tree-predecessor and e -predecessor may become different in the course of tree construction. Irrespective of $D^{sx}(EQ^{z,x})$, it becomes expand-active, initializes $EA^x \leftarrow R(EQ^{z,x})$, and sends the necessary queries as before.

A router x in an expand-passive or -active state may receive a basic expand query $EQ^{z,x}$. First, consider the case when x is expand-passive. Router x may receive a basic expand query only from $z = p^x$, in which case it becomes expand-active. If $D^{sx}(EQ^{z,x}) < D^{s,x}$, it updates $D^{s,x} \leftarrow D^{sx}(EQ^{z,x})$. Otherwise, it waits until it has all its replies from its e -successors to update $D^{s,x}$. Router x forwards a basic expand query to each of its tree-successors. The cost specified in the queries forwarded is calculated using $D^{sx}(EQ^{z,x})$, e.g., the cost in a query to y equals $D^{sx}(EQ^{z,x}) + d^{x,y}$.

Next, consider when x is expand-active. Router x may receive a basic expand query only from p^x or $ep^{x,s}$. In the following, when we use a basic expand query from

$ep^{x,s}$, i.e., $EQ^{ep^{x,s},x}$, it implies that $ep^{x,s} \neq p^x$. Moreover, whenever basic expand queries need to be sent, the cost in the queries are calculated using the cost of the received basic expand query. If $D^{sx}(EQ^{ep^{x,s},x}) < D^{s,x}$, x sends a prune request to p^x , sets $p^x \leftarrow ep^{x,s}$, and sends basic expand queries to its tree-, and e -successors. If $D^{sx}(EQ^{ep^{x,s},x}) \geq D^{s,x}$, it does nothing other than increment the appropriate query counter. Since x was expand-active for $ep^{x,s}$ before receiving the query, it will send a reply to $ep^{x,s}$. When x sends it reply to $ep^{x,s}$, the value of the query counter specified in the reply accounts for the basic expand query. If $D^{sx}(EQ^{p^x,x}) < D^{s,x}$, router x updates $D^{s,x}$. However, if $D^{sx}(EQ^{p^x,x}) \geq D^{s,x}$, it saves the value of $D^{sx}(EQ^{p^x,x})$ to update $D^{s,x}$ when it becomes expand-passive. In either case, it sends basic expand queries to each of its tree-, e -successors. If $D^{sx}(EQ^{p^x,x}) \geq D_*^{s,x}$, it sends a prune request to p^x and sets $p^x \leftarrow ep^{x,s}$.

Whenever a router x receives a reply from e -successor z , it adds the set of members in the reply to set ED^x . Once x has received all the replies to its expand computation with initiator i , it gives a reply to $ep^{x,i}$, specifying ED^x , i , and the query counter in the reply. If $ep^{x,i} \neq p^x$, the reply to $ep^{x,i}$ is an expand-nack. If $ep^{x,i} = p^x$ and x has neither local host members nor tree-successors, the reply is also an expand-nack. If $ep^{x,i} = p^x$ and x has either local host members or tree-successors, the reply is an expand-ack. Router x zeroes out its query counter for initiator i after giving its reply, does the necessary clean-up, and becomes expand-passive if it replied with an expand-ack; otherwise, it cleans up all the state information and becomes idle.

When s gets all its replies, it finds the potentially unreachable set $UR^s \leftarrow EA^s - ED^s$. The receivers in UR^s are potentially not on the multicast tree. Some receivers in UR^s may already be on the tree, but, if their expand-acks were lost, they appear in UR^s . Source s starts a new computation to reach the receiver in UR^s using alternative next-hops. If s has exhausted all its neighbors in trying to reach a router in UR^s , it decides the router to be unreachable.

References

- [1] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core-based Trees (CBT). In *Proc. of SIGCOMM '93*, pages 85–95, 1993.
- [2] S. E. Deering. Host Extensions for IP Multicasting. *RFC 1112*, Aug. 1988.
- [3] S. E. Deering et al. An Architecture for Wide-area Multicast Routing. *ACM Computer Communication Review*, 24(4):126–135, 1994.
- [4] E. W. Dijkstra and C. S. Scholten. Termination Detection for Diffusing Computation. *Inform. Process. Lett.*, 11(1):1–4, 1980.
- [5] J. J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Trans. on Networking*, 1(1):130–141, 1993.
- [6] J. Moy. Multicast Extension to OSPF. *Internet Draft*, 1992.
- [7] M. Parsa and J.J. Garcia-Luna-Aceves. A Protocol for Scalable Internet Multicast Routing. *Submitted for publication*, 1995.
- [8] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. *RFC 1075*, Nov 1988.
- [9] L. Wei and D. Estrin. The Trade-offs of Multicast Trees and Algorithms. In *Proc. of ICCCN*, 1994.