A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing

Deborah Estrin, Mark Handley, Ahmed Helmy, Polly Huang, David Thaler[*]

Abstract

Current multicast routing protocols can be classified into three types according to how the multicast tree is established: broadcast and prune (e.g., DVMRP, PIM-DM), membership advertisement (e.g., MOSPF), and rendezvous-based (e.g., CBT, PIM-SM). Rendezvous-based protocols associate with each logical multicast group address, a physical unicast address, referred to as the 'core' or 'rendezvous point' (RP). Members first join a multicast tree rooted at this rendezvous point in order to receive data packets sent to the group. Rendezvous mechanisms are well suited to large wide-area networks because they distribute group-specific data and membership information only to those routers that are on the multicast distribution tree. However, rendezvous protocols require a *bootstrap mechanism* to map each logical multicast address to its current physical rendezvous-point address. The bootstrap mechanism must adapt to network and router failures but should minimize unnecessary changes in the group-to-RP mapping. In addition, the bootstrap mechanism should be transparent to the hosts.

This paper describes and analyzes the bootstrap mechanism developed for PIM-SM. The mechanism employs an algorithmic mapping of multicast group to rendezvous point address, based on a set of available RPs distributed throughout a multicast domain. The primary evaluation measures are convergence time, message distribution overhead, balanced assignment of groups to RPs, and host impact. The mechanism as a whole, and the design lessons in particular, are applicable to other rendezvous-based multicast routing protocols as well.

1 Introduction

Multicast routing is a critical technology for the evolving Internet. Previous papers have described techniques that use some form of broadcast to build a multicast distribution tree from active sources to current group members. DVMRP [1] and PIM-DM [2] broadcast initial data packets, which in turn trigger and store prune state in those parts of the network that do not lead to group members. MOSPF broadcasts membership information so that intermediate nodes can construct source specific distribution trees on the fly. These multicast routing protocols are efficient when each group is widely represented or the bandwidth is universally plentiful.

[*]Authors listed in alphabetical order.

In contrast, Core Based Trees (CBT) [3] and Protocol Independent Multicast-Sparse Mode (PIM-SM, or PIM for short) [4], do not use broadcast. Rather they use explicit joining to a designated core or rendezvous point whereby new receivers hear from all group sources and new sources reach all group members. Such rendezvous-based multicast routing protocols are better suited for wide area multicast, where group members may be sparsely distributed and where bandwidth is a scarce resource. The PIM architecture [5, 6, 4] described the rationale and design details by which the multicast trees are built and maintained. In this paper we describe the bootstrap mechanisms– in particular, the distribution of rendezvous-point information and the use of that information to map specific groups to RPs in a consistent, distributed, and robust manner. We focus on the robustness and control message overhead of the bootstrap mechanisms because these are the main determinants of scalability.

After a short overview of PIM-SM we motivate the design of its bootstrap mechanisms and outline the remainder of the paper.

1.1 Protocol Independent Multicast–Overview

In this section we provide an overview of the architectural components of PIM-SM (referred to hereafter as PIM) [6, 4, 7]. The architecture described here operates in each multicast domain, which in this context is a contiguous set of routers that all implement PIM and operate within a common boundary defined by PIM Multicast Border Routers [7].

As shown in Figure 1, when a receiver's local router (A) discovers it has local receivers, it starts sending periodic join-messages toward a group-specific Rendezvous Point (RP). Each router along the path toward the RP builds a wildcard (any-source) *route entry* for the group and sends the join-messages on toward the RP. A *route entry* is the forwarding state held in a router to maintain the distribution tree. Typically it includes the data source address, the multicast group address, the incoming interface from which packets are accepted, the list of outgoing interfaces to which packets are sent, and various timers and flags. In this case, the wildcard route entry's data source is "any source", the incoming interface points toward the RP and the outgoing interfaces point to the neighboring downstream routers that have sent join messages toward the RP. This state forms a shared, RP-rooted, distribution tree that reaches all group members.

When a data source first sends to a group, its local router (D) unicasts register-messages to the RP with the source's data packets encapsulated within. If the data rate is high, the RP can send source-specific join-messages back towards the source. These will instantiate source specific route entries in the routers on the path, and the source's data packets will then follow the resulting forwarding state and travel unencapsulated to the RP.

Data packets reaching the RP are forwarded natively down the RP-rooted distribution tree toward group members. If the data rate warrants it, routers with local receivers can join a source-specific, shortest path, distribution tree, and prune this source's packets off of the shared RP-rooted tree. However, for low data rate sources, PIM routers need not join a source-specific shortest path tree and data packets can be delivered via the shared RP-tree.

If a sender or receiver has more than one local router, one of them is elected the Designated Router (DR), and only this router participates in this process.

Figure 1: How senders rendezvous with receivers

1.2 Bootstrap mechanisms–architectural issues

Rendezvous-based protocols require that every router consistently maps an active multicast group address to the same RP. There are many ways to perform this task:

- Application level advertisements to distribute RP information along with the multicast group to potential participant applications.
- A routing protocol to advertise group-to-RP mappings to routers.
- A distributed directory in which routers can look-up group-to-RP mappings on-demand.
- An algorithmic mapping from multicast address to RP address.

Application-level advertisements require changing the IP multicast service model, and require adding application-level dependencies on the choice of multicast routing protocol.

Routing-protocol based advertisements scale badly because each router needs to maintain the mapping for all potential multicast groups. This may additionally add unpredictable delays between choosing a multicast address and being able to use it.

On-demand look-up using a distributed directory (such as DNS) scales better, but introduces a startup delay from when a source starts sending to when the directory responds with the group-to-RP mapping. This is a problem for sources wishing to use multicast for resource location, for example, where only a brief query may be sent, and then the sender goes silent again. Such bursty sources may send for less time than the time taken to acquire the mapping. If the inter-burst period is longer than the routers state timeout period, then all the source's packets may be lost deterministically. We refer to this problem as the *bursty source problem.*

All three of these mechanisms do not allow applications to algorithmically generate and use multicast addresses without announcing them to all potential recipients (a bootstrapping problem in some circumstances).

Moreover, all three of these mechanisms introduce a dependency on application-level directory assistance in some way (if only to insert the mapping into the database). Ideally we would like multicast to be as low a level service as possible, depending only on unicast routing and routers themselves.

The use of an *algorithmic mapping* from group to RP address allows us to address these competing design issues. This approach entails periodic distribution of a set of reachable, RPs to all routers in the domain, and use of a common hash function to map any multicast address to one router from the set of RPs. The algorithmic mapping approach requires routers to store and maintain liveness information for a relatively small and stable set of RPs per multicast domain. This approach scales well, is purely a low-level routing function, and does not require any changes to hosts.

For such an algorithmic mapping to work, the set of RPs must be distributed consistently to all routers within a domain. This must be done robustly and efficiently. If routers have inconsistent sets of RPs, groups may fail to rendezvous.

The distribution of the set of RPs within a domain cannot be supported by rendezvous-based multicast without manual configuration, which is not robust to failures or misconfiguration. The set of RPs could be distributed using a separate broadcast-and-prune multicast routing protocol but this would introduce an unnecessary dependency. A third alternative, adopted here, uses a simple bridge-like flooding mechanism to distribute the set of RPs efficiently within each domain (see section 2.1).

Finally, given the choice of an algorithmic mapping, we must use one that will achieve the goal of mapping groups evenly across RPs in the domain's RP set, thus reducing traffic concentration at individual RPs. We also desire characteristics such as minimal group disruption when RP reachability changes, the ability to map related groups to the same RP, and efficient implementation.

The remainder of this paper provides a detailed evaluation of the design, robustness, and performance of the bootstrap mechanisms. Section 2 describes these mechanisms in detail and section 3 evaluates them in terms of robustness and efficiency. Section 4 details the algorithmic mapping and its evaluation. Section 5 summarizes our results and proposals for future research. Supplemental appendices provide elaborate details of protocol models, mathematical analyses, and simulation detail.

2 Bootstrap mechanisms: Design description and rationale

This section describes how the set of RPs is distributed throughout a domain. We elaborate on the design rationale of the bootstrap mechanisms. The three critical elements of the design are the bootstrap router, the candidate RPs, and the RP-Set. The following subsections describe how these elements fit together to realize the bootstrap mechanisms.

2.1 BSR election

A **BootStrap Router (BSR)** is a dynamically-elected PIM router within a PIM domain. It collects the set of potential RPs, and distributes the resulting set of RPs **(RP-Set)** to all the PIM routers in the PIM domain. Centralizing the RP-Set distribution reduces the opportunities for inconsistency, while dynamic election provides robustness in the face of network changes and failures.

A small set of routers within a PIM domain are configured as candidate bootstrap routers **(Candidate-BSRs)**, and each is given a (not necessarily unique) BSR-priority. From these candidates, a single bootstrap router (BSR) is elected for that domain. Should the current BSR fail, or another Candidate-BSR with higher priority be added to the network, a new BSR is elected automatically.

The mechanism is based upon a bridge-like spanning-tree election algorithm; bootstrap messages originated by the Candidate-BSRs travel hop-by-hop, and the most preferred[1] candidate is elected as the BSR for the domain. This "dense" distribution mechanism is efficient since all PIM routers in the domain require the messages, which corresponds to a densely populated group.

The elected BSR originates periodic bootstrap messages to capture network dynamics. If a PIM domain partitions, each area separated from the old BSR will elect its own BSR, which will distribute an RP-Set containing RPs that are reachable within that partition. When the partition heals, an election will occur with the next periodic bootstrap message and only one of the BSRs will continue to send out bootstrap messages. As is expected at the time of a partition or healing, some disruption in packet delivery may occur. This time will be on the order of the region's end-to-end delay and the bootstrap router timeout values (see section 3.1).

The BSR election mechanism is integrated with the RP-Set distribution mechanism described in section 2.3. Complete mechanistic details are given in appendix A and [7].

2.2 RP-Set construction using Candidate-RP-Advertisements

A set of routers within a PIM domain are configured as *candidate RPs*. Typically these are the same routers that are configured as Candidate-BSRs[2]. Once a BSR for the domain is elected, candidate RPs periodically unicast Candidate-RP-Advertisement messages to the elected BSR. These messages include the address of the advertising candidate RP and provide a liveness indication to the BSR.

The BSR chooses a subset[3] of (or all) the live candidate RPs to form the RP-Set, which is then distributed in the bootstrap messages.

2.3 RP-Set distribution

The bootstrap messages originated by the BSR carry the BSR's address and priority, and the RP-Set. A bootstrap message is multicast out all interfaces with a TTL of 1, to all directly connected PIM routers [7].

Before accepting a bootstrap message, a PIM router performs two checks:

1. Only those bootstrap messages that arrive from the Reverse Path Forwarding (RPF) neighbor[4] towards the BSR are processed, others are discarded. The RPF check prevents looping of bootstrap messages. Persistent looping would lead to usage of obsolete information.

   Unicast routing dynamics may cause transient loops but these do not affect the eventual correctness of the mechanism.

[1]The preferred router is the one with highest configured priority or highest addressed if priorities are equal.

[2]Any PIM router can be configured as a candidate RP or Candidate-BSR. However, to maximize reachability, only well-connected stable routers should be configured as candidates.

[3]The BSR attempts to choose the same subset each time if possible.

[4]The RPF neighbor for an IP address X is the the next-hop router used to forward packets toward X.

2. If the RPF check passes, a check is performed against the previously stored active BSR information (priority and address). Every time a PIM router gets a message from the elected BSR, it restarts a bootstrap-timer. So long as the timer has not expired, the BSR is considered active, and only messages from a preferred Candidate-BSR, or from the active BSR, are accepted. If the bootstrap-timer has expired, this indicates that the stored BSR is no longer active, and the next bootstrap message received will be accepted. Appendix A presents a detailed state transition diagram for this mechanism, as well as the timer actions.

When a bootstrap message is accepted by a PIM router, the BSR and RP-Set information within that router are updated, and the bootstrap-timer restarted. The bootstrap message is then forwarded out all interfaces except the receiving interface.

To achieve better convergence for newly started PIM routers, a Designated Router may send a bootstrap message carrying RP-Set information to new PIM neighbors. In this case, a newly started PIM router, having no RP-Set information, accepts the first bootstrap message it gets. In general, however, bootstrap messages are only sent periodically, and are not triggered. This minimizes the overhead associated with the mechanism (for detailed analysis of overhead see section 3.2).

When a BSR becomes unreachable and bootstrap messages stop, routers continue to use the stored RP-Set until they get new information. This way, multicast data distribution is not disrupted as long as the RP for the group is still reachable. For further discussion of detailed failure scenarios see section 3.1.

2.4 RP-Set processing

The bootstrap message indicates liveness of the RP in the RP-Set. If an RP is included, then it is tagged as 'up' at the routers, while RPs not included in the message are removed from the set of RPs over which the group-to-RP mapping algorithm functions.

This mechanism adapts efficiently to network dynamics, including partitions. The BSR maintains a timer for each RP, called the *RP-timer*. When an RP becomes unreachable, the BSR stops receiving advertisements from it. Consequently, the BSR's RP-timer for that RP expires, and the RP is removed from the RP-Set. Routers receiving the new RP-Set (that does not contain the failed RP), re-map the affected groups to other reachable RPs from the new RP-Set.

Since the set of RPs are known to be live prior to initiating a multicast group, this scheme requires no start up phase, and hence is suitable for applications with strict start up delay bounds. The bursty source problem is also eliminated.

This RP liveness mechanism requires only a bootstrap message, Candidate-RP-Advertisement message and an RP-timer at the BSR, simplifying complex reachability mechanisms described in a previous PIM specification [8], and obviating the need for RP-reachability, RP-list, and Poll messages, in addition to various flags and timers.

2.5 Group-to-RP mapping

When a Designated Router (DR) receives an IGMP Host-Membership-Report [9] from a directly connected receiver for a group for which it has no state, the DR uses an algorithmic mapping to bind the group to one of the RPs in the RP-Set. The DR then sends a join message towards that RP. When the DR receives a data packet from a directly connected sender for such a group, it performs the same algorithmic mapping and sends the data packet encapsulated in a Register message directly to the RP. All PIM routers within a domain use the same mapping algorithm, which we present in section 4.

3 Evaluation metrics of the Bootstrap Mechanisms

One of the major design goals of the bootstrap mechanisms is *scalability.* In the next two subsections, we show a detailed evaluation of the bootstrap mechanism, with emphasis on two critical dimensions of scalability– *robustness* and *overhead.*

For robustness, we study responsiveness of the protocol to router failures and topology changes. In particular, we explain the transient protocol behavior due to various network events, and evaluate the time taken to converge on, and join to, a consistent RP-Set. The study establishes an upper bound on average *convergence time.*

For overhead, we evaluate the domain resources (state and bandwidth) consumed by the bootstrap mechanisms. The state consumed is the memory required to maintain the BSR and RP-Set information in each router, and is simply proportional to the size of the RP-Set. The analysis of the bandwidth overhead, however, is more involved, because it is affected by the underlying topology.

3.1 Evaluation of robustness

When an RP or BSR changes state (i.e., becomes reachable or unreachable) data loss may occur. The duration of time during which loss may occur is a function of the convergence time; much as it is for unicast routing.

Occasional RP failure or unreachability is unavoidable. When this happens, receivers using the shared RP tree may lose data for groups that map to the failed RP. Meanwhile, the RP-timer associated with this RP, at the BSR, times out, and the RP is removed from the RP-Set. Up to this point, the RP-Set remains consistent among all routers. When distribution of the new RP-Set starts, the groups that mapped to the failed RP may temporarily use inconsistent RP-Sets, if some routers receive the new RP-Set before others. After the distribution is completed, all routers converge on the new RP-Set. When affected groups re-join to the new RP, data will be received again.

In this section we enumerate the various network events that can result in data loss. For each event we formulate an expression for the average convergence time. We use the following terms in our analysis:

- **Convergence time** (*'Conv'*): The time taken for all routers within a domain to converge on, and join to, a consistent RP-Set of reachable RPs, after network changes. This represents an upper bound on the time taken for group participants' DRs to re-join to a single reachable RP. This time is measured from the point when the RP-Set becomes inconsistent or an RP within the RP-Set becomes unreachable.
- **Candidate-RP-Advertisement period** (*'RPAdvPeriod'*): The period at which Candidate-RP-Advertisement messages are sent. (e.g. 60 seconds[5]).

[5]All default values mentioned here are set as per the PIMv2 specification[7].

- **RP timeout** (*'RPTimeout'*): The time interval after which the BSR times out an RP. Typically, this timer is set to 2.5 times the Candidate-RP-Advertisement period (e.g. 150 seconds).
- **Bootstrap period** (*'BootstrapPeriod'*): The period at which the elected BSR originates periodic bootstrap messages (e.g. 60 seconds). We call the state in which the elected BSR originates periodic messages the *steady state.*
- **Bootstrap timeout** (*'BootstrapTimeout'*): The time after which a Candidate-BSR originates a bootstrap message to elect a new BSR. We call the time during which the election is in process the *transient state.* (e.g. 150 seconds = 2.5 times the bootstrap period).
- **RP-Set distribution time** (*'SetDist'*): The time to distribute an RP-Set throughout the domain or the time for all routers within the domain to converge on the distributed RP-Set.
- **Join latency** (*'JoinLat'*): The time taken to establish a new multicast branch leading to a receiver, by processing hop-by-hop PIM join messages. The join messages may, in worst case, travel all the way to the RP, if no closer point exists on the multicast tree[6].
- **Join period** (*'JoinPeriod'*): The time interval at which periodic join-messages are sent (e.g. 60 seconds).

At the end of this section we present sample convergence times for various network topologies and loss probabilities.

3.1.1 Convergence time due to various network events

When network dynamics affect BSR and RP reachability, data loss may occur. The bootstrap mechanism described here was designed to adapt to these changes in a timely fashion. The convergence time varies depending upon the particular network event.

Changes in BSR A change in the BSR without a change in the RP-Set does not partition groups[7], nor cause data loss. RPs continue to support active groups without disruption to data delivery, so long as the RPs are reachable during the BSR re-election.

Addition of a new RP When a new candidate RP becomes reachable, it may immediately obtain the BSR address from its neighbors or wait for the periodic bootstrap messages. The new candidate RP then sends a Candidate-RP-Advertisement to the BSR. Provided that the Candidate-RP-Advertisement message is not lost, and the BSR includes it in a new RP-Set, the new RP-Set will be distributed during the next RP-Set cycle. After the RP-Set distribution time (*'SetDist'*), the domain converges on the updated RP-Set. The last converged router needs time *'JoinLat'* to join to the new RP.

Group partition may occur during the distribution of the new RP-Set, as some routers get the new information before others. There is potential data loss for some group members during the time when group partition occurs.

[6]If no join messages are lost, this value is on the order of the end-to-end delay of a domain.

[7]A group partitions when mutually reachable receivers and senders map to different RPs, or map to an unreachable RP, and hence fail to rendezvous.

Hence, the convergence time becomes:

$$RPAddConv = SetDist + JoinLat$$

Deletion of an RP In the following analyses we use the notation '(a,b)' to denote a time interval between 'a' and 'b'. If an RP becomes unreachable from the BSR, its associated RP-timer at the BSR expires within the interval ((*RPTimeout* - *RPAdvPeriod*), *RPTimeout*); at best, the RP failure happens right before the RP sends a Candidate-RP-Advertisement, where the time interval between the point of failure and RP-timer timeout is *'RPTimeout-RPAdvPeriod'*. The BSR will send the new RP-Set at the next RP-Set cycle (0, *BootstrapPeriod*). After *'SetDist'*, the PIM domain converges on the updated RP-Set. The last converged router needs time *'JoinLat'* to join to the new RP.

We note that potential data loss starts from the point of RP failure (or unreachability) but only affects groups that map to the unreachable RP. After all PIM routers converge on the updated RP-Set and affected groups re-join to their new RP, the data loss for the affected groups stops. During the RP-Set Distribution Time, groups mapping to the unreachable RP are partitioned.

The convergence time becomes:

$$\begin{aligned} RPDelConv &= ((RPTimeout - RPAdvPeriod), RPTimeout) + \\ &\quad (0, BootstrapPeriod) + SetDist + JoinLat \end{aligned}$$

Network partitions In addition to single BSR change and RP change, we are interested in events such as network partition and partition healing that may sometimes cause simultaneous changes in both RP and BSR reachability.

When a domain partitions, it is divided into two separate regions, one of which will have the old BSR and some subset of the RPs in the RP-Set, and the other will have the remainder of the RPs in the RP-Set. We will look at each "side" of the partition in turn.

For the region containing the original BSR, any groups using RPs in that region will be unaffected, and hence their convergence time is '0'. Any groups using RPs that are now-unreachable will see convergence times equivalent to those experienced when an RP is deleted.

On the other side of the partition (without the original BSR) all groups using RPs which are still reachable (on the same side) will remain unaffected and hence also have a convergence time of '0'. Groups using now-unreachable RPs, however, experience longer convergence times. A new BSR is elected after *'BootstrapTimeout'*. After *'RPTimeout'*, the new BSR times out the RP-timer. Then, the new BSR updates the RP-Set and distributes the new RP-Set within the interval (0, *BootstrapPeriod*). After *'SetDist'*, the PIM domain converges on the updated, smaller, RP-Set. The last converged router needs time *'JoinLat'* to join to the new RP.

In this case, the convergence time is given by:

$$\begin{aligned} PartitionConv &= BootstrapTimeout + RPTimeout + \\ &\quad (0, BootstrapPeriod) + SetDist + JoinLat \end{aligned}$$

When a partition heals, the two previously-partitioned regions merge. Each region will originally have its own BSR and set of RPs. Of the two BSRs, one will be more preferred than the other and become the BSR of the combined region. At the point where the two regions become mutually reachable, groups are, by definition, partitioned and hence the convergence time starts from this point.

When the partition heals, the more preferred BSR sends a bootstrap message within the interval (0, *BootstrapPeriod*). All routers in the other region receive this message after *SetDist*. It then takes *JoinLat* for them to join to a new RP. Thus, the initial convergence time is given by:

$$HealConv \quad = \quad (0, BootstrapPeriod) + SetDist + JoinLat$$

Meanwhile, the candidate RPs in the other region start sending their Candidate-RP-Advertisement messages to the more preferred BSR. Then, one *BootstrapPeriod* after the first bootstrap message, the more preferred BSR may issue an updated RP-Set with RPs from both previously-disconnected regions. This can cause another temporary group partition which is equivalent to that experienced when a new RP is added.

3.1.2 Evaluation of RP-Set distribution time and join latency

The expressions derived above are all a function of RP-Set distribution time and join latency, both of which depend upon the characteristics of the domain topology and the number of successive bootstrap message losses.

In the absence of packet loss, the average RP-Set distribution time grows linearly with the end-to-end delay of the domain. Therefore, we represent the distribution time for a particular topology in the absence of packet loss as a constant C_b. This constant covers transmission, processing and propagation delays.

Next, we consider the effect of packet loss. In Appendix B, we show that the expected RP-Set distribution time is bounded by:

$$E[SetDist] \leq C_b + BootstrapPeriod \cdot \left(\frac{1}{(1-P)^{N-1}} - 1 \right)$$

where $E[SetDist]$ is the expected value of *SetDist*, N is the number of nodes in the domain, and P is the probability of packet loss on a link.

PIM join messages are processed hop-by-hop, and establish multicast state within intermediate routers. If the first join message reaches the RP successfully, the value of *JoinLat* is simply the time to establish the shared multicast tree within intermediate routers[8]. Such time includes transmission, processing times and propagation delays, and is represented by the constant C_j.

As with bootstrap messages, PIM join message loss contributes to overall convergence time. Using similar terminology, we can likewise express the expected join latency (*'JoinLat'*) in the presence of packet loss as being bounded by:

$$E[JoinLat] \leq C_j + JoinPeriod \cdot \left(\frac{1}{(1-P)^{N-1}} - 1 \right)$$

Given these formulations for the expected RP-Set distribution time and join latency, Figure 2 shows combined delays as a function of network topology and packet loss (per link) probabilities. We present examples for the combination of RP-Set distribution time and join latency because together they are the common variables in most of the convergence formulas. These results were based on several simplifications and assumptions. The constants (C_b and C_j) were assumed to be

[8]Note that this represents an upper bound on the join latency (*JoinLat*) for all routers, since in general join messages only reach the nearest point of the established tree, and need not reach the RP per se.

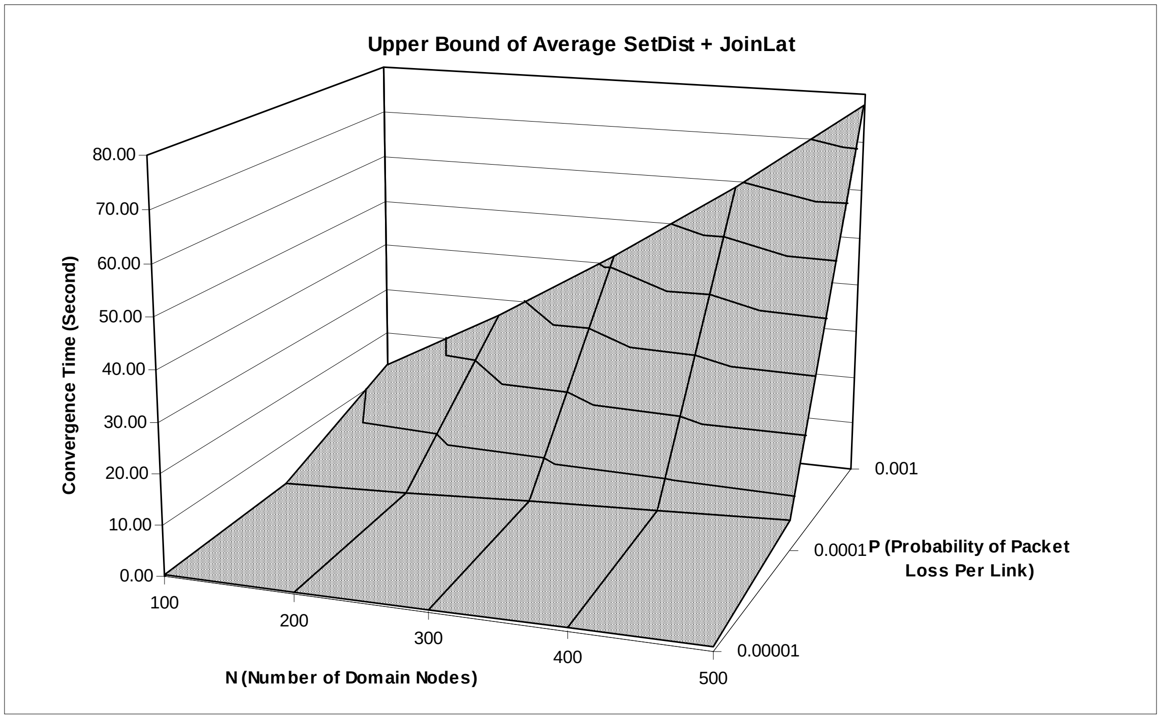


Figure 2: Combined RP-Set distribution time and join latency '*SetDist + JoinLat*'

negligible as compared to the other terms, and default values from the PIM-SM specification [7] were used for the *JoinPeriod, BootstrapPeriod, RPTimeout, BootstrapTimeout, and RPAdvPeriod.* Values used for the packet loss probability per link were restricted to those ranges considered typical of current networks.

3.1.3 Summary of convergence time

To conclude our discussion of convergence time, Figure 3 shows the results of substituting numbers in Figure 2 into the convergence formulas for the various network events. The results shown were obtained using the following simplified equations, based on default timer values:

$$\begin{aligned}
E[RPAddConv] &\leq UpperBound(E[SetDist + JoinLat]) \\
E[RPDelConv] &\leq 150 + UpperBound(E[RPAddConv]) \\
E[PartitionConv] &\leq 330 + UpperBound(E[RPAddConv]) \\
E[HealConv] &\leq 30 + UpperBound(E[RPAddConv])
\end{aligned}$$

A network administrator could use this analysis to determine the maximum desired domain size given a tolerable convergence time and observed probability of packet loss per link.

3.2 Evaluation of control message overhead

Typically, state and bandwidth overheads are considered. For the bootstrap mechanisms, state overhead is simply proportional to the number of RPs within the RP-Set. Bandwidth overhead is

Upper Bound on Avg RPAddConv(in sec)

N\P	0.00001	0.0001	0.001
100	0.12	1.19	12.49
200	0.24	2.41	26.44
300	0.36	3.64	41.85
400	0.48	4.89	58.88
500	0.60	6.14	77.70

Upper Bound on Avg RPDelConv(in sec)

N\P	0.00001	0.0001	0.001
100	150.12	151.19	162.49
200	150.24	152.41	176.44
300	150.36	153.64	191.85
400	150.48	154.89	208.88
500	150.60	156.14	227.70

Upper Bound on Avg PartitionConv(in sec)

N\P	0.00001	0.0001	0.001
100	330.12	331.19	342.49
200	330.24	332.41	356.44
300	330.36	333.64	371.85
400	330.48	334.89	388.88
500	330.60	336.14	407.70

Upper Bound on Avg HealConv(in sec)

N\P	0.00001	0.0001	0.001
100	30.12	31.19	42.49
200	30.24	32.41	56.44
300	30.36	33.64	71.85
400	30.48	34.89	88.88
500	30.60	36.14	107.70

N: Number of Nodes in Domain
P = Probability of Packet Loss Per Link

Figure 3: Summary of convergence time

a function of Candidate-RP-Advertisement and bootstrap messages. Candidate-RP-Advertisement overhead is proportional to the number of candidate RPs, and the unicast distances between candidate RPs and the BSR. A more complex issue is the overhead due to distribution of bootstrap messages. The number of RPs in the domain included in the RP-Set affects the size of bootstrap messages. We assume the number of RPs is constant for the remainder of the comparisons.

3.2.1 Bootstrap message overhead in steady state

In steady state, when there is no BSR election, bootstrap messages will be originated periodically every *BootstrapPeriod* by the BSR. Each router accepts only those bootstrap messages sent by the Reverse Path Forwarding neighbor towards the BSR. The accepted bootstrap message is then forwarded to all interfaces except the receiving interface.

This behavior has the following property: *If the unicast routing is stable and the link layer does not cause any duplicate packet transmission, each node will forward bootstrap messages only once over each interface except the incoming interface.*

By the above property, it is easy to show that the total number of bootstrap messages sent within the domain equals:

$$C_{BSR} + \sum_{i \neq C_{BSR}} (C_i - 1) = \left(\sum C_i\right) - (N - 1)$$

where C_i is the degree of router i, and N is total number of routers in the PIM domain.

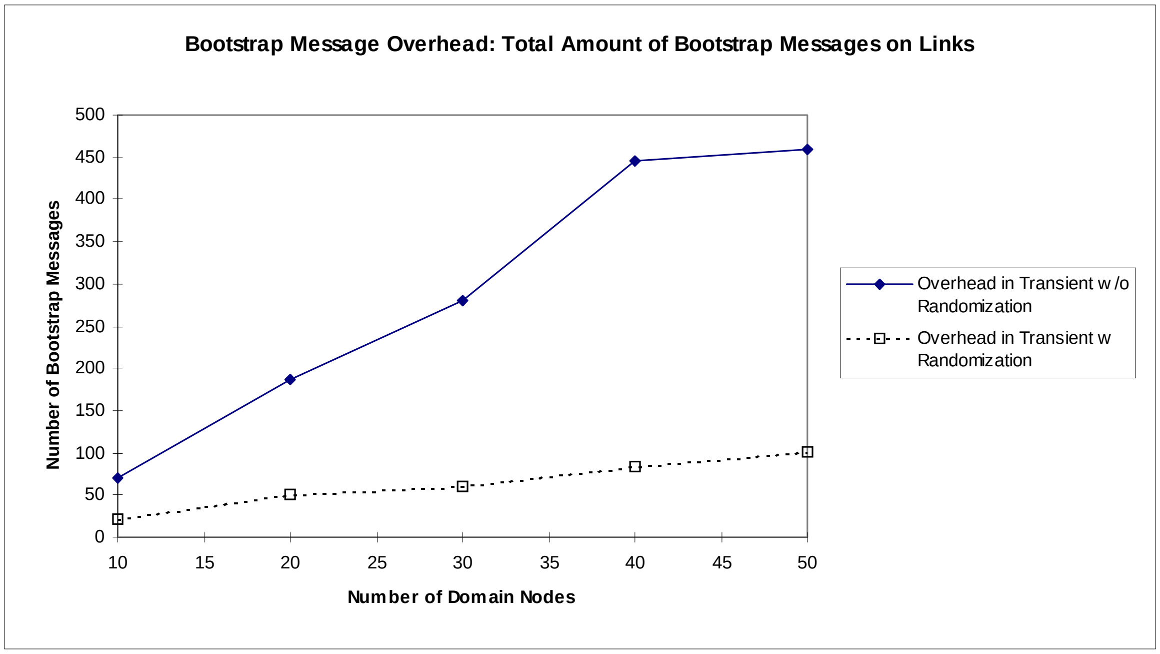


Figure 4: Distribution of bootstrap overhead

3.2.2 Bootstrap message overhead in transient state

When the bootstrap-timer expires in all Candidate-BSRs, during transient state after a network partition for example, each Candidate-BSR originates a bootstrap message. This bootstrap message is suppressed when it reaches any PIM router having higher preference or which has received bootstrap messages from another Candidate-BSR with higher preference.

Since the bootstrap-timers are synchronized by the elected BSR's bootstrap messages, this mechanism may incur a lot of overhead, as shown in Figure 4[9]. Y-axis values are the total number of bootstrap messages generated or forwarded, during a partition healing, until convergence[10]. However, adding randomization to bootstrap message origination decreases the overhead of bootstrap messages significantly. Further, if the random wait value is a function of the priority and address of the Candidate-BSR, the behavior of bootstrap messages in transient states tends to that of the steady state. In Figure 4, the dotted line represents this case. The *RandomWait* value is set such that the highest addressed Candidate-BSR is the first to originate a bootstrap message. The function used here is[11]:

$$RandomWait = log_2(1 + ElectedBSRAddress - CandidateBSRAddress)$$

[9]Simulations used random graphs generated using RTG[10]. As a worst case scenario, all routers were considered Candidate-BSRs.

[10]Simulations were performed using PIMSIM[11]; a discrete event-driven packet-level simulator based on the Maryland Routing Simulator(MaRS)[12, 13]. Future simulations will use the Network Simulator (ns), version 2 [14].

[11]This function could be extended easily to accommodate various priorities as well.

4 Group-to-Rendezvous Point Mapping Algorithm

Having shown that all routers converge on a common RP-Set, under all kinds of conditions, we now consider the group to RP mapping using this RP-Set.

Ideally, a good mapping of groups to RPs meets the following requirements:

- Group balancing under any multicast address allocation scheme. By this, we mean that when the number of groups is large, no single RP is serving significantly many more groups than any other RP in the RP-Set. While this balances the *number* of groups as opposed to balancing the actual load, the actual load will be balanced roughly when the number of groups is large, provided that the mapping is independent of the load represented by a group.
- Minimal disruption of groups when there's a change in the RP-Set. By this, we mean that the number of active multicast groups (and hence, the number of shared RP-trees) affected by a change in the RP-Set must be as small as possible.
- A set of related groups can map to the same RP, giving the same latency and fate-sharing characteristics, and allowing state aggregation. This is useful when data is split across multiple groups because of different media types or for hierarchical encoding (e.g., [15, 16, 17]).
- Efficient implementation. This means there must be a fast implementation requiring only 32-bit integer arithmetic.

We employ hash function theory to realize the desired algorithm. In the following subsections we provide background information on theory and implementation of conventional hash functions, followed by an explanation of the selected Highest Random Weight (HRW) scheme. Then, we evaluate the scaling properties of the chosen algorithm.

4.1 Hash Function Theory and Implementation

A conventional hash function maps a key k to one of n "buckets" by computing a bucket number i as a function of k, i.e., $i = h(k)$. Typically, such functions are defined as $f(k)$ modulo n, where f is some function of k (e.g., $f(k) = k$ when k is an integer). For our purposes, a key k corresponds to a multicast group address, and a bucket corresponds to an RP from the RP-Set. The problem with using a Modulo-n function for Group-to-RP mapping, however, comes when the RP-Set changes.

We first define *disruption* as the fraction of keys which map to a different bucket when the set of buckets changes, i.e., the number of multicast groups which get remapped to a different RP when the RP-Set changes. Since the RP-Set may change whenever a candidate RP goes up or down, the number of buckets (n) over which the hash function will operate can change over time. For example, when a single RP fails and the number of RPs in the RP-Set changes from n to $n-1$, all groups would be remapped except those for which $f(k) \bmod n = f(k) \bmod (n-1)$. When $f(k)$ is uniformly distributed, the disruption will thus be $\frac{n-1}{n}$, or almost all the groups. Clearly, this is insufficient for our purposes.

A much better scheme, which we employ, is the Highest Random Weight (HRW) algorithm[18]. HRW defines the hash function so as to give the bucket with the maximum value of a function f, i.e.:

$$h = i : f(k, l(i)) > f(k, l(j))), \quad 0 \le i, j < n$$

where the bucket label $l(i)$ is a function of the bucket number (for our purposes, $l(i)$ is the unicast IP address of the i^{th} RP in the RP-Set) and f is a function of both the key and the bucket label. If f is sufficiently random with respect to k and l, this has the desirable property that the disruption caused by m RPs changing is just m/n. This successfully achieves the minimum bound on disruption as shown in [18].

We now need to find an appropriate function for f that satisfies the above constraint. For this, we use the following function:

$$f(k, l(i)) = (1103515245 \cdot ((1103515245k + 12345) \text{ XOR } l(i)) + 12345) \mod 2^{31}$$

which is derived from the BSD `rand()` function, and is shown in [18] to perform well as a hash function compared with other pseudo-random number generators. It also lends itself well to an efficient 32-bit implementation.

When a router detects that one or more (m) RPs have failed (by their absence from a newly received RP-Set), only those groups which previously mapped to those RPs must be rehashed. For each of these $(\frac{m}{n})g$ groups, h is recomputed in $O(n)$ time, for a total processing requirement of $O(mg)$ work.

On the other hand, when a router detects that one or more (m) RPs have been added to the RP-Set, again only $(\frac{m}{n})g$ groups will be affected, but all groups must be rehashed to determine which of them are affected. This can be optimized by storing the values of i and $f(k, l(i))$ in per-group state. Then for each group k, only $f(k, l(j))$, where $l(j)$ is the address of a *new* RP, need be calculated to determine which groups are affected. This again can be done in $O(mg)$ time.

Finally, to allow state aggregation, we now modify the definition of the key k to be:

$$k = (\text{group address } \& \text{ mask})$$

The mask determines how much state aggregation will exist (i.e, how many groups will always map to the same RP). This mask is specified by the BSR and is included with the RP-Set distributed to all other routers in the domain. For example, if this mask is (hex) FFFFFFFC (the recommended default), then sets of four consecutive group addresses will map to the same RP. A multicast address allocation scheme may take advantage of this when allocating multiple multicast addresses for the same session by allocating a set of addresses which have the longest possible prefix in common.

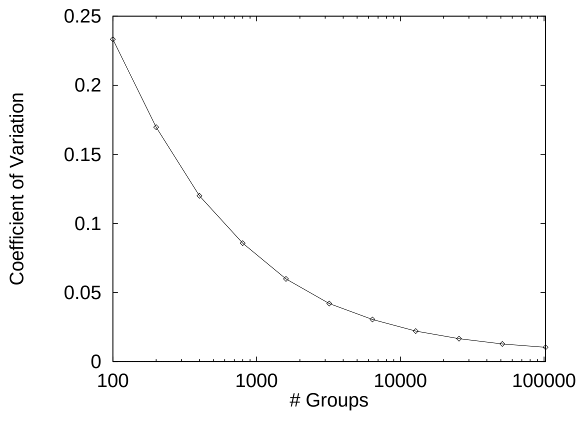
4.2 Evaluation of the Hash Function

To determine how well our hash function scales with respect to group balancing, we use the *coefficient of variation*, defined as the ratio between the standard deviation and the mean (σ/μ), of the number of active groups mapped to each RP in the RP-Set. For group balancing to scale, we desire that:

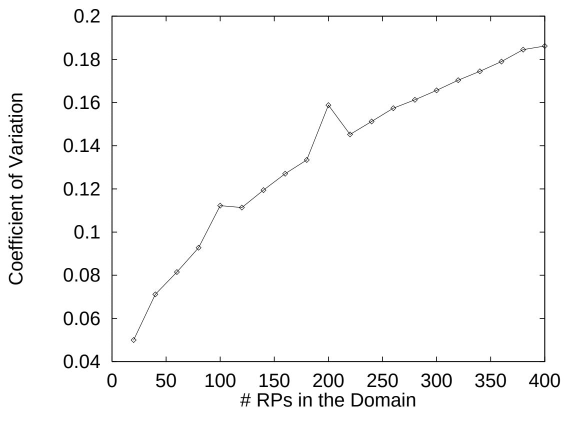
$$\lim_{g \to \infty} \sigma/\mu = 0$$

where g is the number of active groups. We emphasize that it is important that this be true *regardless of the distribution of k*. In other words, it must be true under any multicast address allocation scheme. When the number of groups is small, the coefficient of variation is less important since the RPs will be comparatively lightly loaded.

We ran several simulations to verify that σ/μ does indeed approach 0 in the limit. Figure 5(a) shows the results of varying the number of groups on σ/μ among 10 RPs. In this simulation, both multicast group addresses and addresses of RPs were randomly chosen from the available



(a)



(b)

Figure 5: Group balancing effectiveness

address spaces. Each point represents an average over 500 trials. As can be seen, the groups are indeed balanced as $g \to \infty$. Other simulations using sequential multicast group addresses and RP addresses produced similar results. This is not surprising, since the randomness inherent in the function f results in a mapping which is relatively independent of the distribution of its inputs.

Figure 5(b) shows the results of varying the number of RPs in the RP-Set, with the total number of groups remaining constant at 5000. Again, each point represents an average over 500 trials. From this, we see that σ/μ grows with the number of RPs. However, since the load on each RP decreases as the number of RPs grows, this imbalance becomes less important.

5 Summary and Future Work

Rendezvous-based multicast routing protocols are likely to be an important infrastructure technology for the evolving Internet and therefore it is important to carefully evaluate their robustness and efficiency. While previous studies described and justified PIM's use of explicit join mechanism and the manner in which it builds and maintains distribution trees, this is the first treatment of the other critical component of the architecture; the *bootstrap mechanisms*.

In this paper, we have presented architectural and mechanistic details of the bootstrap mechanisms, as well as systematic analysis and evaluation of the robustness and performance of these mechanisms. Moreover, these bootstrap mechanisms could be readily used with other rendezvous-based multicast protocols, such as CBT.

Currently under study are the extensions of the bootstrap mechanisms to a global inter-domain context and questions of multicast group address allocation and state aggregation.

References

[1] D. Waitzman S. Deering, C. Partridge. Distance Vector Multicast Routing Protocol, November 1988. RFC1075.

[2] D. Estrin, D. Farinacci, A. Helmy, V. Jacobson, and L. Wei. Protocol Independent Multicast-Dense Mode (PIM-DM) : Protocol Specification. *Proposed Experimental RFC*, September 1996.

[3] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core Based Trees. In *Proceedings of the ACM SIGCOMM*, San Francisco, 1993.

[4] S. Deering, D. Estrin, D. Farinacci, M. Handley, A. Helmy, V. Jacobson, C. Liu, P. Sharma, D. Thaler, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Motivation and Architecture. *Experimental RFC*, October 1996.

[5] S. Deering, D Estrin, D. Farrinacci, V. Jacobson, C. Liu, and L. Wei. An Architecture for Wide-area Multicast Routing. In *Proceedings of the ACM SIGCOMM '94*, London, 1994.

[6] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The PIM Architecture for Wide-Area Multicast Routing. *ACM Transactions on Networks*, April 1996.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification. *Experimental RFC*, December 1996.

[8] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol Independent Multicast-Sparse Mode (PIM-SM) : Protocol Specification. *Internet Draft*, December 1995.

[9] W. Fenner. Internet Group Management Protocol, Version 2. *Internet Draft*, January 1997.

[10] Liming Wei. Thesis Dissertation - Scalable Multicast Routing: Tree Types And Protocol Dynamics, December 1995.

[11] Liming Wei. The design of the USC PIM SIMulator (PIMSIM). Technical Report USC TR-95-604, Computer Science Department, University of Southern California, Feburary 1995.

[12] Cengiz Alaettinoglu, A Udaya Shanker, Klaudia Dussa-Zieger, and Ibrahim Matta. Mars (Maryland Routing Simulator) - version 1.0 user's manual. Technical Report CS-TR-2687, Computer Science Department, University of Mariland, June 1991.

[13] Cengiz Alaettinoglu, A Udaya Shanker, Klaudia Dussa-Zieger, and Ibrahim Matta. Design and implementation of mars: A routing testbed. Technical Report CS-TR-2964, Computer Science Department, University of Mariland, September 1992.

[14] Steven McCanne. The UCB/LBNL Network Simulator(ns), November 1996.

[15] N. Shacham. Multipoint communication by hierarchically encoded data. In *Proceedings of the IEEE INFOCOM '92*, pages 2107–2114, 1996.

[16] D. Taubman and A. Zakhor. Multi-rate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588, September 1994.

[17] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven Layered Multicast. In *Proceedings of the ACM SIGCOMM '96*, August 1996.

[18] David Thaler and Chinya V. Ravishankar. A Name-Based Mapping Scheme for Rendezvous. Technical Report CSE-TR-316-96, University of Michigan, November 1996.

Appendices

A BSR Election and RP-Set Distribution

For simplicity, the bootstrap message is used in both the BSR election and the RP-Set distribution mechanisms. These mechanisms are described by the following state machine, illustrated in figure 6. The protocol transitions for a Candidate-BSR are given in state diagram (a). For routers not configured as Candidate-BSRs, the protocol transitions are given in state diagram (b).

State variables

LclBSR = Local concatenated BSR priority and BSR IP address
LclRP-Set = Local RP-Set
RxdBSR = Received concatenated BSR priority and BSR IP address
RxdRP-Set = Received RP-Set
Bootstrap-Period = 60 seconds
Bootstrap-Timeout = 2.5 x Bootstrap-Period = 150 seconds

Predicates

Name	*Meaning*
P	$RxdBSR \geq LclBSR$

Incoming events

Name	*Interface*	*Meaning*
PrefBSMRxd	RPF nbr toward included BSR	Bootstrap msg rcvd satisfying **P**
BSMRxd	RPF nbr toward included BSR	Bootstrap message received
TExp	Timer provider machinery	Bootstrap timer expired

States

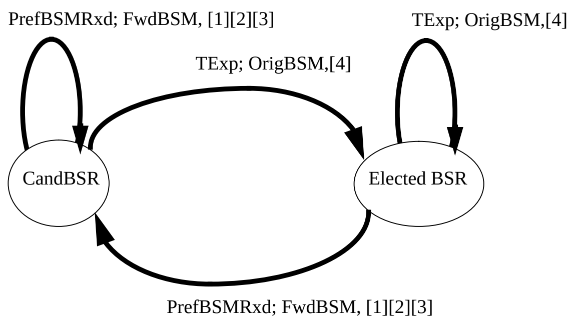
Name	*Meaning*
AxptPref	Accept only Bootstrap messages from preferred or equal BSR
AxptAny	Accept any RP-Set messages coming thru the right interface
CandBSR	Candidate bootstrap router
ElectedBSR	Elected bootstrap router

Outgoing events

Name	*Interface*	*Meaning*
FwdBSM	All interfaces except receiving interface	Forward Bootstrap message
OrigBSM	All interfaces	Originate Bootstrap message

Specific actions

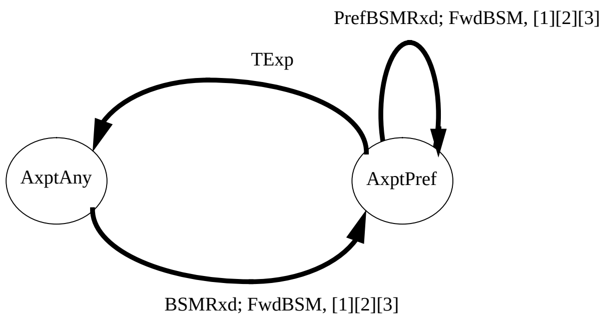
[1] = Restart Bootstrap timer at *Bootstrap-Timeout*
[2] = *(LclBSR = RxdBSR)*
[3] = (*LclRP-Set = RxdRP-Set*)
[4] = Restart Bootstrap timer at *Bootstrap-Period*



Initial state: CandBSR; LclBSR = Local address, LclRP-Set = empty

State transition diagram for a Candidate BSR

(a)



Initial state: AxptAny; LclBSR = 0, LclRP-Set = empty

State transition diagram for a router not configured as C-BSR

(b)

Figure 6: State Diagram for the BSR election and RP-Set distribution mechanisms

Each PIM router keeps a bootstrap-timer, initialized to [Bootstrap-Timeout], in addition to a local BSR field *'LclBSR'* (initialized to a local address if Candidate-BSR, or to 0 otherwise), and a local RP-Set *'LclRP-Set'* (initially empty). The main stimuli to the state machine are timer events and arrival of bootstrap messages:

Initial states and timer events

1. If the router is a Candidate-BSR:
   (a) The router operates initially in the 'CandBSR' state, where it does not originate any bootstrap messages.
   (b) If the bootstrap-timer expires, and the current state is 'CandBSR', the router originates a bootstrap message carrying the local RP-Set and its own BSR priority and address, restarts the bootstrap-timer at [Bootstrap-Period] seconds, and transits into the 'ElectedBSR' state. Note that the actual sending of the bootstrap message may be delayed by a random value to reduce transient control overhead.
   (c) If the bootstrap-timer expires, and the current state is 'ElectedBSR', the router originates a bootstrap message, and restarts the RP-Set timer at [Bootstrap-Period]. No state transition is incurred.

       This way, the elected BSR originates periodic bootstrap messages every [Bootstrap-Period].

2. If a router is not a Candidate-BSR:
   (a) The router operates initially in the 'AxptAny' state. In such state, a router accepts the first bootstrap message from the The Reverse Path Forwarding (RPF) neighbor toward the included BSR. The RPF neighbor in this case is the next hop router en route to the included BSR.
   (b) If the bootstrap-timer expires, and the current state is 'AxptPref'– where the router accepts only preferred bootstrap messages (those that carry BSR-priority and address higher than, or equal to, *'LclBSR'*) from the RPF neighbor toward the included BSR– the router transits into the 'AxptAny' state.

       In this case, if an elected BSR becomes unreachable, the routers start accepting bootstrap messages from another Candidate-BSR after the bootstrap-timer expires. All PIM routers within a domain converge on the preferred reachable Candidate-BSR.

Receiving bootstrap message To avoid loops, an RPF check is performed on the included BSR address. Upon receiving a bootstrap message from the RPF neighbor toward the included BSR, the following actions are taken:

1. If the router is not a Candidate-BSR:
   (a) If the current state is 'AxptAny', the router accepts the bootstrap message, and transits into the 'AxptPref' state.
   (b) If the current state is 'AxptPref', and the bootstrap message is preferred, the message is accepted. No state transition is incurred.

2. If the router is a Candidate-BSR, and the bootstrap message is preferred, the message is accepted. Further, if this happens when the current state is 'Elected BSR', the router transits into the 'CandBSR' state.

When a bootstrap message is accepted, the router restarts the bootstrap-timer at [Bootstrap-Timeout], stores the received BSR priority and address in *'LclBSR'*, and the received RP-Set in *'LclRP-Set'*, and forwards the bootstrap message out all interfaces except the receiving interface.

If a bootstrap message is rejected, no state transitions are triggered.

B Analysis of RP-Set distribution time and join latency

We first define the following terms:

- N : number of routers in the PIM domain.
- C_b: average RP-Set distribution time without loss of bootstrap messages.
- P_i : the probability that a given bootstrap message crossing router i's RPF link towards the BSR is lost.
- $Pr(\text{no bootstrap loss})$: the probability that a given bootstrap message reaches all other routers in the PIM domain.

$$Pr(\text{no bootstrap loss}) = \prod_{i=1}^{N-1} (1 - P_i)$$

If no messages are dropped, the average RP-Set distribution time *SetDist* should be no greater than the end-to-end delay, and equal to C_b. If the first bootstrap message is lost anywhere in the domain, convergence takes at least another *BootstrapPeriod*. The second message will then complete the convergence if there is no loss between the BSR and any routers which did not receive the first message. The number of times the bootstrap message must be sent until it reaches all routers is then bounded above by a geometric distribution function. The upper bound on the expected number of trials until success is thus given by:

$$\text{Expected trials until success} \leq \frac{1}{Pr(\text{no bootstrap loss})}$$

Note that this is simply an upper bound, since in fact each node need only get the message once. Thus, once an entire subtree has received the message, later retransmissions dropped within the subtree are irrelevant.

If we assume for the sake of simplicity that P_i is the same for all links, then we get:

$$\begin{aligned} \text{Expected trials until success} &\leq \frac{1}{(1-P)^{N-1}} \\ E[SetDist] &\leq C_b + BootstrapPeriod \cdot \left(\frac{1}{(1-P)^{N-1}} - 1\right) \end{aligned}$$

We can apply the same process to finding the average join latency. Let C_j be the average transmission, processing and propagation delay experienced from the DR to the RP. In the worst case (a linear topology), the join message again crosses $N-1$ links, giving us upper bounds of:

$$\begin{aligned} \text{Expected trials until success} &\leq \frac{1}{(1-P)^{N-1}} \\ E[JoinLat] &\leq C_j + JoinPeriod \cdot \left(\frac{1}{(1-P)^{N-1}} - 1\right) \end{aligned}$$