

Network Working Group
Request for Comments: 4601
Obsoletes: 2362
Category: Standards Track

B. Fenner
AT&T Labs - Research
M. Handley
UCL
H. Holbrook
Arastra
I. Kouvelas
Cisco
August 2006

Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document specifies Protocol Independent Multicast - Sparse Mode (PIM-SM). PIM-SM is a multicast routing protocol that can use the underlying unicast routing information base or a separate multicast-capable routing information base. It builds unidirectional shared trees rooted at a Rendezvous Point (RP) per group, and optionally creates shortest-path trees per source.

This document obsoletes RFC 2362, an Experimental version of PIM-SM.

Charter Communications, Inc. - Exhibit 1007, part 1 Charter Communications, Inc. v. Iarnach Technologies, Ltd.

Table of Contents

1. Introduction	4
2. Terminology	4
2.1. Definitions	4
2.2. Pseudocode Notation	5
3. PIM-SM Protocol Overview	5
3.1. Phase One: RP Tree	6
3.2. Phase Two: Register-Stop	6
3.3. Phase Three: Shortest-Path Tree.	7
3.4. Source-Specific Joins	7
3.5. Source-Specific Prunes	8
3.6. Multi-Access Transit LANs	8
3.7. RP Discovery	8
4. Protocol Specification	9
4.1. PIM Protocol State	9
4.1.1. General Purpose State	10
4.1.2. (*,*,RP) State.	10
4.1.3. (*,G) State	11
4.1.4. (S,G) State	12
4.1.5. (S,G,rpt) State	13
4.1.6. State Summarization Macros	14
4.2. Data Packet Forwarding Rules	18
4.2.1. Last-Hop Switchover to the SPT	20
4.2.2. Setting and Clearing the (S,G) SPTbit.	20
4.3. Designated Routers (DR) and Hello Messages	21
4.3.1. Sending Hello Messages	21
4.3.2. DR Election	23
4.3.3. Reducing Prune Propagation Delay on LANs	24
4.3.4. Maintaining Secondary Address Lists	26
4.4. PIM Register Messages	27
4.4.1. Sending Register Messages from the DR	28
4.4.2. Receiving Register Messages at the RP	32
4.5. PIM Join/Prune Messages	33
4.5.1. Receiving (*,*,RP) Join/Prune Messages	34
4.5.2. Receiving (*,G) Join/Prune Messages	36
4.5.3. Receiving (S,G) Join/Prune Messages	39
4.5.4. Receiving (S,G,rpt) Join/Prune Messages	42
4.5.5. Sending (*,*,RP) Join/Prune Messages	47
4.5.6. Sending (*,G) Join/Prune Messages	50
4.5.7. Sending (S,G) Join/Prune Messages	53
4.5.8. (S,G,rpt) Periodic Messages	57
4.5.9. State Machine for (S,G,rpt) Triggered Messages	58
4.5.10. Background: (*,*,RP) and (S,G,rpt) Interaction.	62
4.6. PIM Assert Messages.	63
4.6.1. (S,G) Assert Message State Machine	63
4.6.2. (*,G) Assert Message State Machine	69
4.6.3. Assert Metrics	74
4.6.4. AssertCancel Messages	75
4.6.5. Assert State Macros	76
4.7. PIM Bootstrap and RP Discovery	78
4.7.1. Group-to-RP Mapping	79

4.7.2. Hash Function	79
4.8. Source-Specific Multicast	80
4.8.1. Protocol Modifications for SSM Destination Addresses.	80
4.8.2. PIM-SSM-Only Routers	81
4.9. PIM Packet Formats	82
4.9.1. Encoded Source and Group Address Formats	83
4.9.2. Hello Message Format	85
4.9.3. Register Message Format	88
4.9.4. Register-Stop Message Format	90
4.9.5. Join/Prune Message Format.	90
4.9.5.1. Group Set Source List Rules	93
4.9.5.2. Group Set Fragmentation	95
4.9.6. Assert Message Format	96
4.10. PIM Timers	97
4.11. Timer Values	98
5. IANA Considerations	101
5.1. PIM Address Family	101
5.2. PIM Hello Options	101
6. Security Considerations	101
6.1. Attacks Based on Forged Messages.	102
6.1.1. Forged Link-Local Messages	102
6.1.2. Forged Unicast Messages	102
6.2. Non-Cryptographic Authentication Mechanisms	102
6.3. Authentication Using IPsec	103
6.3.1. Protecting Link-Local Multicast Messages	103
6.3.2. Protecting Unicast Messages	103
6.3.2.1. Register Messages	103
6.3.2.2. Register-Stop Messages	104
6.4. Denial-of-Service Attacks	104
7. Acknowledgements	104
8. Normative References	105
9. Informative References	105
10. Appendix A: PIM Multicast Border Router Behavior	106
10.1. Sources External to the PIM-SM Domain	106
10.2. Sources Internal to the PIM-SM Domain	106
11. Index	108

List of Figures

Figure 1. Per-(S,G) register state machine at a DR	28
Figure 2. Downstream per-interface (*,*,RP) state machine	34
Figure 3. Downstream per-interface (*,G) state machine.	37
Figure 4. Downstream per-interface (S,G) state machine.	40
Figure 5. Downstream per-interface (S,G,rpt) state machine.	43
Figure 6. Upstream (*,*,RP) state machine	47
Figure 7. Upstream (*,G) state machine	50
Figure 8. Upstream (S,G) state machine	54
Figure 9. Upstream (S,G,rpt) state machine for triggered messages.	58
Figure 10. Per-interface (S,G) Assert State machine	63
Figure 11. Per-interface (*,G) Assert State machine	70

1. Introduction

This document specifies a protocol for efficiently routing multicast groups that may span wide-area (and inter-domain) internets. This protocol is called Protocol Independent Multicast - Sparse Mode (PIM-SM) because, although it may use the underlying unicast routing to provide reverse-path information for multicast tree building, it is not dependent on any particular unicast routing protocol.

PIM-SM version 2 was originally specified in RFC 2117 and was revised in RFC 2362, both Experimental RFCs. This document is intended to obsolete RFC 2362, to correct a number of deficiencies that have been identified with the way PIM-SM was previously specified, and to bring PIM-SM onto the IETF Standards Track. As far as possible, this document specifies the same protocol as RFC 2362 and only diverges from the behavior intended by RFC 2362 when the previously specified behavior was clearly incorrect. Routers implemented according to the specification in this document will be able to interoperate successfully with routers implemented according to RFC 2362.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant PIM-SM implementations.

2.1. Definitions

The following terms have special significance for PIM-SM:

Rendezvous Point (RP):

An RP is a router that has been configured to be used as the root of the non-source-specific distribution tree for a multicast group. Join messages from receivers for a group are sent towards the RP, and data from senders is sent to the RP so that receivers can discover who the senders are and start to receive traffic destined for the group.

Designated Router (DR):

A shared-media LAN like Ethernet may have multiple PIM-SM routers connected to it. A single one of these routers, the DR, will act on behalf of directly connected hosts with respect to the PIM-SM protocol. A single DR is elected per interface (LAN or otherwise) using a simple election process.

MRIB

Multicast Routing Information Base. This is the multicast topology table, which is typically derived from the unicast routing table, or routing protocols such as Multiprotocol BGP (MBGP) that carry multicast-specific topology information. In PIM-SM, the MRIB is used to decide where to send Join/Prune messages. A secondary function of the MRIB is to provide routing metrics for destination addresses; these metrics are used when sending and processing Assert messages.

RPF Neighbor

RPF stands for "Reverse Path Forwarding". The RPF Neighbor of a router with respect to an address is the neighbor that the MRIB indicates should be used to forward packets to that address. In the case of a PIM-SM multicast group, the RPF neighbor is the router that a Join message for that group would be directed to, in the absence of modifying Assert state.

- TIB** Tree Information Base. This is the collection of state at a PIM router that has been created by receiving PIM Join/Prune messages, PIM Assert messages, and Internet Group Management Protocol (IGMP) or Multicast Listener Discovery (MLD) information from local hosts. It essentially stores the state of all multicast distribution trees at that router.
- MFIB** Multicast Forwarding Information Base. The TIB holds all the state that is necessary to forward multicast packets at a router. However, although this specification defines forwarding in terms of the TIB, to actually forward packets using the TIB is very inefficient. Instead, a real router implementation will normally build an efficient MFIB from the TIB state to perform forwarding. How this is done is implementation-specific and is not discussed in this document.
- Upstream**
Towards the root of the tree. The root of tree may be either the source or the RP, depending on the context.
- Downstream**
Away from the root of the tree.
- GenID**
Generation Identifier, used to detect reboots.
- PMBR**
PIM Multicast Border Router, joining a PIM domain with another multicast domain.

2.2. Pseudocode Notation

We use set notation in several places in this specification.

$A \oplus B$ is the union of two sets, A and B.

$A \ominus B$ is the elements of set A that are not in set B.

NULL is the empty set or list.

In addition, we use C-like syntax:

= denotes assignment of a variable.

== denotes a comparison for equality.

≠ denotes a comparison for inequality.

Braces { and } are used for grouping.

3. PIM-SM Protocol Overview

This section provides an overview of PIM-SM behavior. It is intended as an introduction to how PIM-SM works, and it is **NOT** definitive. For the definitive specification, see Section 4.

PIM relies on an underlying topology-gathering protocol to populate a routing table with routes. This routing table is called the Multicast Routing Information Base (MRIB). The routes in this table may be taken directly from the unicast routing table, or they may be different and provided by a separate routing protocol such as MBGP [10]. Regardless of how it is created, the primary role of the MRIB in the PIM protocol is to provide the next-hop router along a multicast-capable path to each destination subnet. The MRIB is used to determine the next-hop neighbor to which any PIM Join/Prune message is sent. Data flows along the reverse path of the Join messages. Thus, in contrast to the unicast RIB, which specifies the next hop that a data packet would take to get to some subnet, the MRIB gives

reverse-path information and indicates the path that a multicast data packet would take *from* its origin subnet to the router that has the MRIB.

Like all multicast routing protocols that implement the service model from RFC 1112 [3], PIM-SM must be able to route data packets from sources to receivers without either the sources or receivers knowing a priori of the existence of the others. This is essentially done in three phases, although as senders and receivers may come and go at any time, all three phases may occur simultaneously.

3.1. Phase One: RP Tree

In phase one, a multicast receiver expresses its interest in receiving traffic destined for a multicast group. Typically, it does this using IGMP [2] or MLD [4], but other mechanisms might also serve this purpose. One of the receiver's local routers is elected as the Designated Router (DR) for that subnet. On receiving the receiver's expression of interest, the DR then sends a PIM Join message towards the RP for that multicast group. This Join message is known as a (*,G) Join because it joins group G for all sources to that group. The (*,G) Join travels hop-by-hop towards the RP for the group, and in each router it passes through, multicast tree state for group G is instantiated. Eventually, the (*,G) Join either reaches the RP or reaches a router that already has (*,G) Join state for that group. When many receivers join the group, their Join messages converge on the RP and form a distribution tree for group G that is rooted at the RP. This is known as the RP Tree (RPT), and is also known as the shared tree because it is shared by all sources sending to that group. Join messages are resent periodically so long as the receiver remains in the group. When all receivers on a leaf-network leave the group, the DR will send a PIM (*,G) Prune message towards the RP for that multicast group. However, if the Prune message is not sent for any reason, the state will eventually time out.

A multicast data sender just starts sending data destined for a multicast group. The sender's local router (DR) takes those data packets, unicast-encapsulates them, and sends them directly to the RP. The RP receives these encapsulated data packets, decapsulates them, and forwards them onto the shared tree. The packets then follow the (*,G) multicast tree state in the routers on the RP Tree, being replicated wherever the RP Tree branches, and eventually reaching all the receivers for that multicast group. The process of encapsulating data packets to the RP is called registering, and the encapsulation packets are known as PIM Register packets.

At the end of phase one, multicast traffic is flowing encapsulated to the RP, and then natively over the RP tree to the multicast receivers.

3.2. Phase Two: Register-Stop

Register-encapsulation of data packets is inefficient for two reasons:

- Encapsulation and decapsulation may be relatively expensive operations for a router to perform, depending on whether or not the router has appropriate hardware for these tasks.
- Traveling all the way to the RP, and then back down the shared tree may result in the packets traveling a relatively long distance to reach receivers that are close to the sender. For some applications, this increased latency or bandwidth consumption is undesirable.

Although Register-encapsulation may continue indefinitely, for these reasons, the RP will normally choose to switch to native forwarding. To do this, when the RP receives a register-encapsulated data packet from source S on group G, it will normally initiate an (S,G) source-specific Join towards S. This Join message travels hop-by-hop towards S, instantiating (S,G) multicast tree state in the routers along the path. (S,G) multicast tree

state is used only to forward packets for group G if those packets come from source S. Eventually the Join message reaches S's subnet or a router that already has (S,G) multicast tree state, and then packets from S start to flow following the (S,G) tree state towards the RP. These data packets may also reach routers with (*,G) state along the path towards the RP; if they do, they can shortcut onto the RP tree at this point.

While the RP is in the process of joining the source-specific tree for S, the data packets will continue being encapsulated to the RP. When packets from S also start to arrive natively at the RP, the RP will be receiving two copies of each of these packets. At this point, the RP starts to discard the encapsulated copy of these packets, and it sends a Register-Stop message back to S's DR to prevent the DR from unnecessarily encapsulating the packets.

At the end of phase 2, traffic will be flowing natively from S along a source-specific tree to the RP, and from there along the shared tree to the receivers. Where the two trees intersect, traffic may transfer from the source-specific tree to the RP tree and thus avoid taking a long detour via the RP.

Note that a sender may start sending before or after a receiver joins the group, and thus phase two may happen before the shared tree to the receiver is built.

3.3. Phase Three: Shortest-Path Tree

Although having the RP join back towards the source removes the encapsulation overhead, it does not completely optimize the forwarding paths. For many receivers, the route via the RP may involve a significant detour when compared with the shortest path from the source to the receiver.

To obtain lower latencies or more efficient bandwidth utilization, a router on the receiver's LAN, typically the DR, may optionally initiate a transfer from the shared tree to a source-specific shortest-path tree (SPT). To do this, it issues an (S,G) Join towards S. This instantiates state in the routers along the path to S. Eventually, this join either reaches S's subnet or reaches a router that already has (S,G) state. When this happens, data packets from S start to flow following the (S,G) state until they reach the receiver.

At this point, the receiver (or a router upstream of the receiver) will be receiving two copies of the data: one from the SPT and one from the RPT. When the first traffic starts to arrive from the SPT, the DR or upstream router starts to drop the packets for G from S that arrive via the RP tree. In addition, it sends an (S,G) Prune message towards the RP. This is known as an (S,G,rpt) Prune. The Prune message travels hop-by-hop, instantiating state along the path towards the RP indicating that traffic from S for G should NOT be forwarded in this direction. The prune is propagated until it reaches the RP or a router that still needs the traffic from S for other receivers.

By now, the receiver will be receiving traffic from S along the shortest-path tree between the receiver and S. In addition, the RP is receiving the traffic from S, but this traffic is no longer reaching the receiver along the RP tree. As far as the receiver is concerned, this is the final distribution tree.

3.4. Source-Specific Joins

IGMPv3 permits a receiver to join a group and specify that it only wants to receive traffic for a group if that traffic comes from a particular source. If a receiver does this, and no other receiver on the LAN requires all the traffic for the group, then the DR may omit performing a (*,G) join to set up the shared tree, and instead issue a source-specific (S,G) join only.

The range of multicast addresses from 232.0.0.0 to 232.255.255.255 is currently set aside for source-specific multicast in IPv4. For groups in this range, receivers should only issue source-specific IGMPv3 joins. If a PIM router receives a non-source-specific join for a group in this range, it should ignore it, as described in Section 4.8.

3.5. Source-Specific Prunes

IGMPv3 also permits a receiver to join a group and to specify that it only wants to receive traffic for a group if that traffic *does not* come from a specific source or sources. In this case, the DR will perform a (*,G) join as normal, but may combine this with an (S,G,rpt) prune for each of the sources the receiver does not wish to receive.

3.6. Multi-Access Transit LANs

The overview so far has concerned itself with point-to-point transit links. However, using multi-access LANs such as Ethernet for transit is not uncommon. This can cause complications for three reasons:

- Two or more routers on the LAN may issue (*,G) Joins to different upstream routers on the LAN because they have inconsistent MRIB entries regarding how to reach the RP. Both paths on the RP tree will be set up, causing two copies of all the shared tree traffic to appear on the LAN.
- Two or more routers on the LAN may issue (S,G) Joins to different upstream routers on the LAN because they have inconsistent MRIB entries regarding how to reach source S. Both paths on the source-specific tree will be set up, causing two copies of all the traffic from S to appear on the LAN.
- A router on the LAN may issue a (*,G) Join to one upstream router on the LAN, and another router on the LAN may issue an (S,G) Join to a different upstream router on the same LAN. Traffic from S may reach the LAN over both the RPT and the SPT. If the receiver behind the downstream (*,G) router doesn't issue an (S,G,rpt) prune, then this condition would persist.

All of these problems are caused by there being more than one upstream router with join state for the group or source-group pair. PIM does not prevent such duplicate joins from occurring; instead, when duplicate data packets appear on the LAN from different routers, these routers notice this and then elect a single forwarder. This election is performed using PIM Assert messages, which resolve the problem in favor of the upstream router that has (S,G) state; or, if neither or both router has (S,G) state, then the problem is resolved in favor of the router with the best metric to the RP for RP trees, or the best metric to the source to source-specific trees.

These Assert messages are also received by the downstream routers on the LAN, and these cause subsequent Join messages to be sent to the upstream router that won the Assert.

3.7. RP Discovery

PIM-SM routers need to know the address of the RP for each group for which they have (*,G) state. This address is obtained automatically (e.g., embedded-RP), through a bootstrap mechanism, or through static configuration.

One dynamic way to do this is to use the Bootstrap Router (BSR) mechanism [11]. One router in each PIM domain is elected the Bootstrap Router through a simple election process. All the routers in the domain that are configured to be candidates to be RPs periodically unicast their candidacy to the BSR. From the candidates, the BSR picks an RP-

set, and periodically announces this set in a Bootstrap message. Bootstrap messages are flooded hop-by-hop throughout the domain until all routers in the domain know the RP-Set.

To map a group to an RP, a router hashes the group address into the RP-set using an order-preserving hash function (one that minimizes changes if the RP-Set changes). The resulting RP is the one that it uses as the RP for that group.

4. Protocol Specification

The specification of PIM-SM is broken into several parts:

- Section 4.1 details the protocol state stored.
- Section 4.2 specifies the data packet forwarding rules.
- Section 4.3 specifies Designated Router (DR) election and the rules for sending and processing Hello messages.
- Section 4.4 specifies the PIM Register generation and processing rules.
- Section 4.5 specifies the PIM Join/Prune generation and processing rules.
- Section 4.6 specifies the PIM Assert generation and processing rules.
- Section 4.7 specifies the RP discovery mechanisms.
- The subset of PIM required to support Source-Specific Multicast, PIM-SSM, is described in Section 4.8.
- PIM packet formats are specified in Section 4.9.
- A summary of PIM-SM timers and their default values is given in Section 4.10.
- Appendix A in Section 10 specifies the PIM Multicast Border Router behavior.

4.1. PIM Protocol State

This section specifies all the protocol state that a PIM implementation should maintain in order to function correctly. We term this state the *Tree Information Base* (TIB), as it holds the state of all the multicast distribution trees at this router. In this specification, we define PIM mechanisms in terms of the TIB. However, only a very simple implementation would actually implement packet forwarding operations in terms of this state. Most implementations will use this state to build a multicast forwarding table, which would then be updated when the relevant state in the TIB changes.

Although we specify precisely the state to be kept, this does not mean that an implementation of PIM-SM needs to hold the state in this form. This is actually an abstract state definition, which is needed in order to specify the router's behavior. A PIM-SM implementation is free to hold whatever internal state it requires and will still be conformant with this specification so long as it results in the same externally visible protocol behavior as an abstract router that holds the following state.

We divide TIB state into four sections:

(*,*,RP) state

State that maintains per-RP trees, for all groups served by a given RP.

(*,G) state

State that maintains the RP tree for G.

(S,G) state

State that maintains a source-specific tree for source S and group G.

(S,G,rpt) state

State that maintains source-specific information about source S on the RP tree for G.

For example, if a source is being received on the source-specific tree, it will normally have been pruned off the RP tree. This prune state is (S,G,rpt) state.

The state that should be kept is described below. Of course, implementations will only maintain state when it is relevant to forwarding operations; for example, the "NoInfo" state might be assumed from the lack of other state information rather than being held explicitly.

4.1.1. General Purpose State

A router holds the following non-group-specific state:

For each interface:

- Effective Override Interval
- Effective Propagation Delay
- Suppression state: One of {"Enable", "Disable"}

Neighbor State:

For each neighbor:

- Information from neighbor's Hello
- Neighbor's GenID.
- Neighbor Liveness Timer (NLT)

Designated Router (DR) State:

- Designated Router's IP Address
- DR's DR Priority

The Effective Override Interval, the Effective Propagation Delay and the Interface suppression state are described in Section 4.3.3. Designated Router state is described in Section 4.3.

4.1.2. (*,*,RP) State

For every RP, a router keeps the following state:

(*,*,RP) state:

For each interface:

PIM (*,*,RP) Join/Prune State:

- State: One of {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}
- Prune-Pending Timer (PPT)
- Join/Prune Expiry Timer (ET)

Not interface specific:

Upstream (*,*,RP) Join/Prune State:

- State: One of {"NotJoined(*,*,RP)", "Joined(*,*,RP)"}
- Upstream Join/Prune Timer (JT)
- Last RPF Neighbor towards RP that was used

PIM (*,*,RP) Join/Prune state is the result of receiving PIM (*,*,RP) Join/Prune messages on this interface and is specified in Section 4.5.1.

The upstream (*,*,RP) Join/Prune State reflects the state of the upstream (*,*,RP) state machine described in Section 4.5.5.

The upstream (*,*,RP) Join/Prune Timer is used to send out periodic Join(*,*,RP) messages, and to override Prune(*,*,RP) messages from peers on an upstream LAN interface.

The last RPF neighbor towards the RP is stored because if the MRIB changes, then the RPF neighbor towards the RP may change. If it does so, then we need to trigger a new Join(*,*,RP) to the new upstream neighbor and a Prune(*,*,RP) to the old upstream neighbor. Similarly, if a router detects through a changed GenID in a Hello message that the upstream neighbor towards the RP has rebooted, then it should re-instantiate state by sending a Join(*,*,RP). These mechanisms are specified in Section 4.5.5.

4.1.3. (*,G) State

For every group G, a router keeps the following state:

(*,G) state:

For each interface:

Local Membership:

State: One of {"NoInfo", "Include"}

PIM (*,G) Join/Prune State:

- State: One of {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}
- Prune-Pending Timer (PPT)
- Join/Prune Expiry Timer (ET)

(*,G) Assert Winner State

- State: One of {"NoInfo" (NI), "I lost Assert" (L), "I won Assert" (W)}
- Assert Timer (AT)
- Assert winner's IP Address (AssertWinner)
- Assert winner's Assert Metric (AssertWinnerMetric)

Not interface specific:

Upstream (*,G) Join/Prune State:

- State: One of {"NotJoined(*,G)", "Joined(*,G)"}
- Upstream Join/Prune Timer (JT)
- Last RP Used
- Last RPF Neighbor towards RP that was used

Local membership is the result of the local membership mechanism (such as IGMP or MLD) running on that interface. It need not be kept if this router is not the DR on that interface unless this router won a (*,G) assert on this interface for this group, although implementations may optionally keep this state in case they become the DR or assert winner. We recommend storing this information if possible, as it reduces latency converging to stable operating conditions after a failure causing a change of DR. This information is used by the `pim_include(*,G)` macro described in Section 4.1.6.

PIM (*,G) Join/Prune state is the result of receiving PIM (*,G) Join/Prune messages on this interface and is specified in Section 4.5.2. The state is used by the macros that calculate the outgoing interface list in Section 4.1.6, and in the `JoinDesired(*,G)` macro (defined in Section 4.5.6) that is used in deciding whether a Join(*,G) should be sent upstream.

(*,G) Assert Winner state is the result of sending or receiving (*,G) Assert messages on this interface. It is specified in Section 4.6.2.

The upstream (*,G) Join/Prune State reflects the state of the upstream (*,G) state machine described in Section 4.5.6.

The upstream (*,G) Join/Prune Timer is used to send out periodic Join(*,G) messages, and to override Prune(*,G) messages from peers on an upstream LAN interface.

The last RP used must be stored because if the RP-Set changes (Section 4.7), then state must be torn down and rebuilt for groups whose RP changes.

The last RPF neighbor towards the RP is stored because if the MRIB changes, then the RPF neighbor towards the RP may change. If it does so, then we need to trigger a new Join(*,G) to the new upstream neighbor and a Prune(*,G) to the old upstream neighbor. Similarly, if a router detects through a changed GenID in a Hello message that the upstream neighbor towards the RP has rebooted, then it should re-instantiate state by sending a Join(*,G). These mechanisms are specified in Section 4.5.6.

4.1.4. (S,G) State

For every source/group pair (S,G), a router keeps the following state:

(S,G) state:

For each interface:

Local Membership:

State: One of {"NoInfo", "Include"}

PIM (S,G) Join/Prune State:

- State: One of {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}
- Prune-Pending Timer (PPT)
- Join/Prune Expiry Timer (ET)

(S,G) Assert Winner State

- State: One of {"NoInfo" (NI), "I lost Assert" (L), "I won Assert" (W)}
- Assert Timer (AT)
- Assert winner's IP Address (AssertWinner)
- Assert winner's Assert Metric (AssertWinnerMetric)

Not interface specific:

Upstream (S,G) Join/Prune State:

- State: One of {"NotJoined(S,G)", "Joined(S,G)"}
- Upstream (S,G) Join/Prune Timer (JT)
- Last RPF Neighbor towards S that was used
- SPTbit (indicates (S,G) state is active)
- (S,G) Keepalive Timer (KAT)

Additional (S,G) state at the DR:

- Register state: One of {"Join" (J), "Prune" (P), "Join-Pending" (JP), "NoInfo" (NI)}
- Register-Stop timer

Additional (S,G) state at the RP:

- PMBR: the first PMBR to send a Register for this source with the Border bit set.

Local membership is the result of the local source-specific membership mechanism (such as IGMP version 3) running on that interface and specifying that this particular source should be included. As stored here, this state is the resulting state after any IGMPv3 inconsistencies have been resolved. It need not be kept if this router is not the DR on that interface unless this router won a (S,G) assert on this interface for this group. However, we recommend storing this information if possible, as it reduces latency converging to stable operating conditions after a failure causing a change of DR. This information is used by the `pim_include(S,G)` macro described in Section 4.1.6.

PIM (S,G) Join/Prune state is the result of receiving PIM (S,G) Join/Prune messages on this interface and is specified in Section 4.5.2. The state is used by the macros that calculate the outgoing interface list in Section 4.1.6, and in the `JoinDesired(S,G)` macro (defined in Section 4.5.7) that is used in deciding whether a Join(S,G) should be sent upstream.

(S,G) Assert Winner state is the result of sending or receiving (S,G) Assert messages on this interface. It is specified in Section 4.6.1.

The upstream (S,G) Join/Prune State reflects the state of the upstream (S,G) state machine described in Section 4.5.7.

The upstream (S,G) Join/Prune Timer is used to send out periodic Join(S,G) messages, and to override Prune(S,G) messages from peers on an upstream LAN interface.

The last RPF neighbor towards S is stored because if the MRIB changes, then the RPF neighbor towards S may change. If it does so, then we need to trigger a new Join(S,G) to the new upstream neighbor and a Prune(S,G) to the old upstream neighbor. Similarly, if the router detects through a changed GenID in a Hello message that the upstream neighbor towards S has rebooted, then it should re-instantiate state by sending a Join(S,G). These mechanisms are specified in Section 4.5.7.

The SPTbit is used to indicate whether forwarding is taking place on the (S,G) Shortest Path Tree (SPT) or on the (*,G) tree. A router can have (S,G) state and still be forwarding on (*,G) state during the interval when the source-specific tree is being constructed. When SPTbit is FALSE, only (*,G) forwarding state is used to forward packets from S to G. When SPTbit is TRUE, both (*,G) and (S,G) forwarding state are used.

The (S,G) Keepalive Timer is updated by data being forwarded using this (S,G) forwarding state. It is used to keep (S,G) state alive in the absence of explicit (S,G) Joins. Amongst other things, this is necessary for the so-called "turnaround rules" — when the RP uses (S,G) joins to stop encapsulation, and then (S,G) prunes to prevent traffic from unnecessarily reaching the RP.

On a DR, the (S,G) Register State is used to keep track of whether to encapsulate data to the RP on the Register Tunnel; the (S,G) Register-Stop timer tracks how long before encapsulation begins again for a given (S,G).

On an RP, the PMBR value must be cleared when the Keepalive Timer expires.

4.1.5. (S,G,rpt) State

For every source/group pair (S,G) for which a router also has (*,G) state, it also keeps the following state:

(S,G,rpt) state:

For each interface:

Local Membership:

State: One of {"NoInfo", "Exclude"}

PIM (S,G,rpt) Join/Prune State:

- State: One of {"NoInfo", "Pruned", "Prune-Pending"}
- Prune-Pending Timer (PPT)
- Join/Prune Expiry Timer (ET)

Not interface specific:

Upstream (S,G,rpt) Join/Prune State:

- State: One of {"RPTNotJoined(G)", "NotPruned(S,G,rpt)", "Pruned(S,G,rpt)"}
- Override Timer (OT)

Local membership is the result of the local source-specific membership mechanism (such as IGMPv3) running on that interface and specifying that although there is (*,G) Include state, this particular source should be excluded. As stored here, this state is the resulting state after any IGMPv3 inconsistencies between LAN members have been resolved. It need not be kept if this router is not the DR on that interface unless this router won a (*,G) assert on this interface for this group. However, we recommend storing this information if possible, as it reduces latency converging to stable operating conditions after a failure causing a change of DR. This information is used by the `pim_exclude(S,G)` macro described in Section 4.1.6.

PIM (S,G,rpt) Join/Prune state is the result of receiving PIM (S,G,rpt) Join/Prune messages on this interface and is specified in Section 4.5.4. The state is used by the macros that calculate the outgoing interface list in Section 4.1.6, and in the rules for adding Prune(S,G,rpt) messages to Join(*,G) messages specified in Section 4.5.8.

The upstream (S,G,rpt) Join/Prune state is used along with the Override Timer to send the correct override messages in response to Join/Prune messages sent by upstream peers on a LAN. This state and behavior are specified in Section 4.5.9.

4.1.6. State Summarization Macros

Using this state, we define the following "macro" definitions, which we will use in the descriptions of the state machines and pseudocode in the following sections.

The most important macros are those that define the outgoing interface list (or "olist") for the relevant state. An olist can be "immediate" if it is built directly from the state of the relevant type. For example, the `immediate_olist(S,G)` is the olist that would be built if the router only had (S,G) state and no (*,G) or (S,G,rpt) state. In contrast, the "inherited" olist inherits state from other types. For example, the `inherited_olist(S,G)` is the olist that is relevant for forwarding a packet from S to G using both source-specific and group-specific state.

There is no `immediate_olist(S,G,rpt)` as (S,G,rpt) state is negative state; it removes interfaces in the (*,G) olist from the olist that is actually used to forward traffic. The `inherited_olist(S,G,rpt)` is therefore the olist that would be used for a packet from S to G forwarding on the RP tree. It is a strict subset of $(\text{immediate_olist}(*,*,RP) \oplus \text{immediate_olist}(*,G))$.

Generally speaking, the inherited olists are used for forwarding, and the immediate_olist are used to make decisions about state maintenance.

```

immediate_olist(*,*,RP) =
    joins(*,*,RP)

immediate_olist(*,G) =
    joins(*,G)  $\oplus$  pim_include(*,G)  $\ominus$  lost_assert(*,G)

immediate_olist(S,G) =
    joins(S,G)  $\oplus$  pim_include(S,G)  $\ominus$  lost_assert(S,G)

inherited_olist(S,G,rpt) =
    ( joins(*,*,RP(G))  $\oplus$  joins(*,G)  $\ominus$  prunes(S,G,rpt) )
     $\oplus$  ( pim_include(*,G)  $\ominus$  pim_exclude(S,G) )
     $\ominus$  ( lost_assert(*,G)  $\oplus$  lost_assert(S,G,rpt) )

inherited_olist(S,G) =
    inherited_olist(S,G,rpt)  $\oplus$ 
    joins(S,G)  $\oplus$  pim_include(S,G)  $\ominus$  lost_assert(S,G)

```

The macros `pim_include(*,G)` and `pim_include(S,G)` indicate the interfaces to which traffic might be forwarded because of hosts that are local members on that interface. Note that normally only the DR cares about local membership, but when an assert happens, the assert winner takes over responsibility for forwarding traffic to local members that have requested traffic on a group or source/group pair.

```

pim_include(*,G) =
    { all interfaces I such that:
      ( ( I_am_DR( I ) AND lost_assert(*,G,I) == FALSE )
        OR AssertWinner(*,G,I) == me )
      AND local_receiver_include(*,G,I) }

pim_include(S,G) =
    { all interfaces I such that:
      ( (I_am_DR( I ) AND lost_assert(S,G,I) == FALSE )
        OR AssertWinner(S,G,I) == me )
      AND local_receiver_include(S,G,I) }

pim_exclude(S,G) =
    { all interfaces I such that:
      ( (I_am_DR( I ) AND lost_assert(*,G,I) == FALSE )
        OR AssertWinner(*,G,I) == me )
      AND local_receiver_exclude(S,G,I) }

```

The clause "`local_receiver_include(S,G,I)`" is true if the IGMP/MLD module or other local membership mechanism has determined that local members on interface I desire to receive traffic sent specifically by S to G. "`local_receiver_include(*,G,I)`" is true if the IGMP/MLD module or other local membership mechanism has determined that local members on interface I desire to receive all traffic sent to G (possibly excluding traffic from a specific set of sources). "`local_receiver_exclude(S,G,I)`" is true if "`local_receiver_include(*,G,I)`" is true but none of the local members desire to receive traffic from S.

The set "joins(*,*,RP)" is the set of all interfaces on which the router has received (*,*,RP) Joins:

```
joins(*,*,RP) =
    { all interfaces I such that
      DownstreamJPState(*,*,RP,I) is either Join or
      Prune-Pending }
```

DownstreamJPState(*,*,RP,I) is the state of the finite state machine in Section 4.5.1.

The set "joins(*,G)" is the set of all interfaces on which the router has received (*,G) Joins:

```
joins(*,G) =
    { all interfaces I such that
      DownstreamJPState(*,G,I) is either Join or Prune-
      Pending }
```

DownstreamJPState(*,G,I) is the state of the finite state machine in Section 4.5.2.

The set "joins(S,G)" is the set of all interfaces on which the router has received (S,G) Joins:

```
joins(S,G) =
    { all interfaces I such that
      DownstreamJPState(S,G,I) is either Join or Prune-
      Pending }
```

DownstreamJPState(S,G,I) is the state of the finite state machine in Section 4.5.3.

The set "prunes(S,G,rpt)" is the set of all interfaces on which the router has received (*,G) joins and (S,G,rpt) prunes.

```
prunes(S,G,rpt) =
    { all interfaces I such that
      DownstreamJPState(S,G,rpt,I) is Prune or PruneTmp }
```

DownstreamJPState(S,G,rpt,I) is the state of the finite state machine in Section 4.5.4.

The set "lost_assert(*,G)" is the set of all interfaces on which the router has received (*,G) joins but has lost a (*,G) assert. The macro lost_assert(*,G,I) is defined in Section 4.6.5.

```
lost_assert(*,G) =
    { all interfaces I such that
      lost_assert(*,G,I) == TRUE }
```

The set "lost_assert(S,G,rpt)" is the set of all interfaces on which the router has received (*,G) joins but has lost an (S,G) assert. The macro lost_assert(S,G,rpt,I) is defined in Section 4.6.5.

```
lost_assert(S,G,rpt) =
    { all interfaces I such that
      lost_assert(S,G,rpt,I) == TRUE }
```

The set "lost_assert(S,G)" is the set of all interfaces on which the router has received (S,G) joins but has lost an (S,G) assert. The macro lost_assert(S,G,I) is defined in Section 4.6.5.

```
lost_assert(S,G) =
    { all interfaces I such that
      lost_assert(S,G,I) == TRUE }
```

The following pseudocode macro definitions are also used in many places in the specification. Basically, RPF' is the RPF neighbor towards an RP or source unless a PIM-Assert has overridden the normal choice of neighbor.

```

neighbor RPF' (*,G) {
    if ( I_Am_Assert_Loser(*, G, RPF_interface(RP(G))) ) {
        return AssertWinner(*, G, RPF_interface(RP(G)))
    } else {
        return NBR( RPF_interface(RP(G)), MRIB.next_hop( RP(G) ) )
    }
}

neighbor RPF' (S,G,rpt) {
    if( I_Am_Assert_Loser(S, G, RPF_interface(RP(G))) ) {
        return AssertWinner(S, G, RPF_interface(RP(G)))
    } else {
        return RPF' (*,G)
    }
}

neighbor RPF' (S,G) {
    if ( I_Am_Assert_Loser(S, G, RPF_interface(S)) ) {
        return AssertWinner(S, G, RPF_interface(S))
    } else {
        return NBR( RPF_interface(S), MRIB.next_hop( S ) )
    }
}

```

RPF'(*,G) and RPF'(S,G) indicate the neighbor from which data packets should be coming and to which joins should be sent on the RP tree and SPT, respectively.

RPF'(S,G,rpt) is basically RPF'(*,G) modified by the result of an Assert(S,G) on RPF_interface(RP(G)). In such a case, packets from S will be originating from a different router than RPF'(*,G). If we only have active (*,G) Join state, we need to accept packets from RPF'(S,G,rpt) and add a Prune(S,G,rpt) to the periodic Join(*,G) messages that we send to RPF'(*,G) (see Section 4.5.8).

The function MRIB.next_hop(S) returns an address of the next-hop PIM neighbor toward the host S, as indicated by the current MRIB. If S is directly adjacent, then MRIB.next_hop(S) returns NULL. At the RP for G, MRIB.next_hop(RP(G)) returns NULL.

The function NBR(I, A) uses information gathered through PIM Hello messages to map the IP address A of a directly connected PIM neighbor router on interface I to the primary IP address of the same router (Section 4.3.4). The primary IP address of a neighbor is the address that it uses as the source of its PIM Hello messages. Note that a neighbor's IP address may be non-unique within the PIM neighbor database due to scope issues. The address must, however, be unique amongst the addresses of all the PIM neighbors on a specific interface.

I_Am_Assert_Loser(S, G, I) is true if the Assert state machine (in Section 4.6.1) for (S,G) on Interface I is in "I am Assert Loser" state.

I_Am_Assert_Loser(*, G, I) is true if the Assert state machine (in Section 4.6.2) for (*,G) on Interface I is in "I am Assert Loser" state.

4.2. Data Packet Forwarding Rules

The PIM-SM packet forwarding rules are defined below in pseudocode.

`inif` is the incoming interface of the packet.

`S` is the source address of the packet.

`G` is the destination address of the packet (group address).

`RP` is the address of the Rendezvous Point for this group.

`RPF_interface(S)` is the interface the MRIB indicates would be used to route packets to `S`.

`RPF_interface(RP)` is the interface the MRIB indicates would be used to route packets to `RP`, except at the `RP` when it is the decapsulation interface (the "virtual" interface on which register packets are received).

First, we restart (or start) the Keepalive Timer if the source is on a directly connected subnet.

Second, we check to see if the SPTbit should be set because we've now switched from the RP tree to the SPT.

Next, we check to see whether the packet should be accepted based on TIB state and the interface that the packet arrived on.

If the packet should be forwarded using (S,G) state, we then build an outgoing interface list for the packet. If this list is not empty, then we restart the (S,G) state Keepalive Timer.

If the packet should be forwarded using (*,*,RP) or (*,G) state, then we just build an outgoing interface list for the packet. We also check if we should initiate a switch to start receiving this source on a shortest path tree.

Finally we remove the incoming interface from the outgoing interface list we've created, and if the resulting outgoing interface list is not empty, we forward the packet out of those interfaces.

On receipt of data from S to G on interface iif:

```

if( DirectlyConnected(S) == TRUE AND iif == RPF_interface(S) ) {
    set KeepaliveTimer(S,G) to Keepalive_Period
    # Note: a register state transition or UpstreamJPState(S,G)
    # transition may happen as a result of restarting
    # KeepaliveTimer, and must be dealt with here.
}

if( iif == RPF_interface(S) AND UpstreamJPState(S,G) == Joined AND
    inherited_olist(S,G) ≠ NULL ) {
    set KeepaliveTimer(S,G) to Keepalive_Period
}

Update_SPTbit(S,G,iif)
oiflist = NULL

if( iif == RPF_interface(S) AND SPTbit(S,G) == TRUE ) {
    oiflist = inherited_olist(S,G)
} else if( iif == RPF_interface(RP(G)) AND SPTbit(S,G) == FALSE ) {
    oiflist = inherited_olist(S,G,rpt)
    CheckSwitchToSpt(S,G)
} else {
    # Note: RPF check failed
    # A transition in an Assert FSM may cause an Assert(S,G)
    # or Assert(*,G) message to be sent out interface iif.
    # See section 4.6 for details.
    if ( SPTbit(S,G) == TRUE AND iif is in inherited_olist(S,G) ) {
        send Assert(S,G) on iif
    } else if ( SPTbit(S,G) == FALSE AND
        iif is in inherited_olist(S,G,rpt) ) {
        send Assert(*,G) on iif
    }
}

oiflist = oiflist ∅ iif
forward packet on all interfaces in oiflist

```

This pseudocode employs several "macro" definitions:

`DirectlyConnected(S)` is TRUE if the source S is on any subnet that is directly connected to this router (or for packets originating on this router).

`inherited_olist(S,G)` and `inherited_olist(S,G,rpt)` are defined in Section 4.1.

Basically, `inherited_olist(S,G)` is the outgoing interface list for packets forwarded on (S,G) state, taking into account (*,*,RP) state, (*,G) state, asserts, etc.

`inherited_olist(S,G,rpt)` is the outgoing interface list for packets forwarded on (*,*,RP) or (*,G) state, taking into account (S,G,rpt) prune state, asserts, etc.

`Update_SPTbit(S,G,iif)` is defined in Section 4.2.2.

`CheckSwitchToSpt(S,G)` is defined in Section 4.2.1.

UpstreamJPState(S,G) is the state of the finite state machine in Section 4.5.7.

Keepalive_Period is defined in Section 4.10.

Data-triggered PIM-Assert messages sent from the above forwarding code should be rate-limited in a implementation-dependent manner.

4.2.1. Last-Hop Switchover to the SPT

In Sparse-Mode PIM, last-hop routers join the shared tree towards the RP. Once traffic from sources to joined groups arrives at a last-hop router, it has the option of switching to receive the traffic on a shortest path tree (SPT).

The decision for a router to switch to the SPT is controlled as follows:

```
void
CheckSwitchToSpt(S,G) {
    if ( ( pim_include(*,G)  $\ominus$  pim_exclude(S,G)
           $\oplus$  pim_include(S,G)  $\neq$  NULL )
        AND SwitchToSptDesired(S,G) ) {
        # Note: Restarting the KAT will result in the SPT switch
        set KeepaliveTimer(S,G) to Keepalive_Period
    }
}
```

SwitchToSptDesired(S,G) is a policy function that is implementation defined. An "infinite threshold" policy can be implemented by making SwitchToSptDesired(S,G) return false all the time. A "switch on first packet" policy can be implemented by making SwitchToSptDesired(S,G) return true once a single packet has been received for the source and group.

4.2.2. Setting and Clearing the (S,G) SPTbit

The (S,G) SPTbit is used to distinguish whether to forward on (*,*,RP)/(*,G) or on (S,G) state. When switching from the RP tree to the source tree, there is a transition period when data is arriving due to upstream (*,*,RP)/(*,G) state while upstream (S,G) state is being established, during which time a router should continue to forward only on (*,*,RP)/(*,G) state. This prevents temporary black-holes that would be caused by sending a Prune(S,G,rpt) before the upstream (S,G) state has finished being established.

Thus, when a packet arrives, the (S,G) SPTbit is updated as follows:

```
void
Update_SPTbit(S,G,iif) {
    if ( iif == RPF_interface(S)
        AND JoinDesired(S,G) == TRUE
        AND ( DirectlyConnected(S) == TRUE
              OR RPF_interface(S)  $\neq$  RPF_interface(RP(G))
              OR inheritedolist(S,G,rpt) == NULL
              OR ( ( RPF'(S,G) == RPF'(*,G) ) AND
                  ( RPF'(S,G)  $\neq$  NULL ) )
              OR ( I_Am_Assert_Loser(S,G,iif) ) ) {
        Set SPTbit(S,G) to TRUE
    }
}
```

Additionally, a router can set SPTbit(S,G) to TRUE in other cases, such as when it receives

an Assert(S,G) on RPF_interface(S) (see Section 4.6.1).

JoinDesired(S,G) is defined in Section 4.5.7 and indicates whether we have the appropriate (S,G) Join state to wish to send a Join(S,G) upstream.

Basically, Update_SPTbit will set the SPTbit if we have the appropriate (S,G) join state, and if the packet arrived on the correct upstream interface for S, and if one or more of the following conditions applies:

1. The source is directly connected, in which case the switch to the SPT is a no-op.
2. The RPF interface to S is different from the RPF interface to the RP. The packet arrived on RPF_interface(S), and so the SPT must have been completed.
3. Noone wants the packet on the RP tree.
4. $RPF'(S,G) == RPF'(*,G)$. In this case, the router will never be able to tell if the SPT has been completed, so it should just switch immediately.

In the case where the RPF interface is the same for the RP and for S, but $RPF'(S,G)$ and $RPF'(*,G)$ differ, we wait for an Assert(S,G), which indicates that the upstream router with (S,G) state believes the SPT has been completed. However, item (3) above is needed because there may not be any (*,G) state to trigger an Assert(S,G) to happen.

The SPTbit is cleared in the (S,G) upstream state machine (see Section 4.5.7) when JoinDesired(S,G) becomes FALSE.

4.3. Designated Routers (DR) and Hello Messages

A shared-media LAN like Ethernet may have multiple PIM-SM routers connected to it. A single one of these routers, the DR, will act on behalf of directly connected hosts with respect to the PIM-SM protocol. Because the distinction between LANs and point-to-point interfaces can sometimes be blurred, and because routers may also have multicast host functionality, the PIM-SM specification makes no distinction between the two. Thus, DR election will happen on all interfaces, LAN or otherwise.

DR election is performed using Hello messages. Hello messages are also the way that option negotiation takes place in PIM, so that additional functionality can be enabled, or parameters tuned.

4.3.1. Sending Hello Messages

PIM Hello messages are sent periodically on each PIM-enabled interface. They allow a router to learn about the neighboring PIM routers on each interface. Hello messages are also the mechanism used to elect a Designated Router (DR), and to negotiate additional capabilities. A router must record the Hello information received from each PIM neighbor.

Hello messages **MUST** be sent on all active interfaces, including physical point-to-point links, and are multicast to the 'ALL-PIM-ROUTERS' group address ('224.0.0.13' for IPv4 and 'ff02::d' for IPv6).

We note that some implementations do not send Hello messages on point-to-point interfaces. This is non-compliant behavior. A compliant PIM router MUST send Hello messages, even on point-to-point interfaces.

A per-interface Hello Timer (HT(I)) is used to trigger sending Hello messages on each active interface. When PIM is enabled on an interface or a router first starts, the Hello Timer of that interface is set to a random value between 0 and Triggered_Hello_Delay. This prevents synchronization of Hello messages if

multiple routers are powered on simultaneously. After the initial randomized interval, Hello messages must be sent every `Hello_Period` seconds. The Hello Timer should not be reset except when it expires.

Note that neighbors will not accept Join/Prune or Assert messages from a router unless they have first heard a Hello message from that router. Thus, if a router needs to send a Join/Prune or Assert message on an interface on which it has not yet sent a Hello message with the currently configured IP address, then it **MUST** immediately send the relevant Hello message without waiting for the Hello Timer to expire, followed by the Join/Prune or Assert message.

The `DR_Priority` Option allows a network administrator to give preference to a particular router in the DR election process by giving it a numerically larger DR Priority. The `DR_Priority` Option **SHOULD** be included in every Hello message, even if no DR Priority is explicitly configured on that interface. This is necessary because priority-based DR election is only enabled when all neighbors on an interface advertise that they are capable of using the `DR_Priority` Option. The default priority is 1.

The `Generation_Identifier` (GenID) Option **SHOULD** be included in all Hello messages. The GenID option contains a randomly generated 32-bit value that is regenerated each time PIM forwarding is started or restarted on the interface, including when the router itself restarts. When a Hello message with a new GenID is received from a neighbor, any old Hello information about that neighbor **SHOULD** be discarded and superseded by the information from the new Hello message. This may cause a new DR to be chosen on that interface.

The LAN Prune Delay Option **SHOULD** be included in all Hello messages sent on multi-access LANs. This option advertises a router's capability to use values other than the defaults for the `Propagation_Delay` and `Override_Interval`, which affect the setting of the Prune-Pending, Upstream Join, and Override Timers (defined in Section 4.10).

The Address List Option advertises all the secondary addresses associated with the source interface of the router originating the message. The option **MUST** be included in all Hello messages if there are secondary addresses associated with the source interface and **MAY** be omitted if no secondary addresses exist.

To allow new or rebooting routers to learn of PIM neighbors quickly, when a Hello message is received from a new neighbor, or a Hello message with a new GenID is received from an existing neighbor, a new Hello message should be sent on this interface after a randomized delay between 0 and `Triggered_Hello_Delay`. This triggered message need not change the timing of the scheduled periodic message. If a router needs to send a Join/Prune to the new neighbor or send an Assert message in response to an Assert message from the new neighbor before this randomized delay has expired, then it **MUST** immediately send the relevant Hello message without waiting for the Hello Timer to expire, followed by the Join/Prune or Assert message. If it does not do this, then the new neighbor will discard the Join/Prune or Assert message.

Before an interface goes down or changes primary IP address, a Hello message with a zero HoldTime should be sent immediately (with the old IP address if the IP address changed). This will cause PIM neighbors to remove this neighbor (or its old IP address) immediately. After an interface has changed its IP address, it MUST send a Hello message with its new IP address. If an interface changes one of its secondary IP addresses, a Hello message with an updated Address_List option and a non-zero HoldTime should be sent immediately. This will cause PIM neighbors to update this neighbor's list of secondary addresses immediately.

4.3.2. DR Election

When a PIM Hello message is received on interface I, the following information about the sending neighbor is recorded:

`neighbor.interface`

The interface on which the Hello message arrived.

`neighbor.primary_ip_address`

The IP address that the PIM neighbor used as the source address of the Hello message.

`neighbor.genid`

The Generation ID of the PIM neighbor.

`neighbor.dr_priority`

The DR Priority field of the PIM neighbor, if it is present in the Hello message.

`neighbor.dr_priority_present`

A flag indicating if the DR Priority field was present in the Hello message.

`neighbor.timeout`

A timer value to time out the neighbor state when it becomes stale, also known as the Neighbor Liveness Timer.

The Neighbor Liveness Timer (NLT(N,I)) is reset to `Hello_Holdtime` (from the Hello Holdtime option) whenever a Hello message is received containing a Holdtime option, or to `Default_Hello_Holdtime` if the Hello message does not contain the Holdtime option.

Neighbor state is deleted when the neighbor timeout expires.

The function for computing the DR on interface I is:

```
host
DR(I) {
    dr = me
    for each neighbor on interface I {
        if ( dr_is_better( neighbor, dr, I ) == TRUE ) {
            dr = neighbor
        }
    }
    return dr
}
```

The function used for comparing DR "metrics" on interface I is:

```
bool
dr_is_better(a,b,I) {
    if( there is a neighbor n on I for which n.dr_priority_present
        is false ) {
        return a.primary_ip_address > b.primary_ip_address
    } else {
        return ( a.dr_priority > b.dr_priority ) OR
            ( a.dr_priority == b.dr_priority AND
              a.primary_ip_address > b.primary_ip_address )
    }
}
```

The trivial function I_am_DR(I) is defined to aid readability:

```
bool
I_am_DR(I) {
    return DR(I) == me
}
```

The DR Priority is a 32-bit unsigned number, and the numerically larger priority is always preferred. A router's idea of the current DR on an interface can change when a PIM Hello message is received, when a neighbor times out, or when a router's own DR Priority changes. If the router becomes the DR or ceases to be the DR, this will normally cause the DR Register state machine to change state. Subsequent actions are determined by that state machine.

We note that some PIM implementations do not send Hello messages on point-to-point interfaces and thus cannot perform DR election on such interfaces. This is non-compliant behavior. DR election MUST be performed on ALL active PIM-SM interfaces.

4.3.3. Reducing Prune Propagation Delay on LANs

In addition to the information recorded for the DR Election, the following per neighbor information is obtained from the LAN Prune Delay Hello option:

```
neighbor.lan_prune_delay_present
    A flag indicating if the LAN Prune Delay option was present in the Hello
    message.

neighbor.tracking_support
    A flag storing the value of the T bit in the LAN Prune Delay option if it is present
    in the Hello message. This indicates the neighbor's capability to disable Join
    message suppression.

neighbor.propagation_delay
    The Propagation Delay field of the LAN Prune Delay option (if present) in the
    Hello message.

neighbor.override_interval
    The Override_Interval field of the LAN Prune Delay option (if present) in the
    Hello message.
```

The additional state described above is deleted along with the DR neighbor state when the neighbor timeout expires.

Just like the DR_Priority option, the information provided in the LAN Prune Delay option is not used unless all neighbors on a link advertise the option. The function below computes this state:

```
bool
lan_delay_enabled(I) {
    for each neighbor on interface I {
        if ( neighbor.lan_prune_delay_present == false ) {
            return false
        }
    }
    return true
}
```

The Propagation Delay inserted by a router in the LAN Prune Delay option expresses the expected message propagation delay on the link and should be configurable by the system administrator. It is used by upstream routers to figure out how long they should wait for a Join override message before pruning an interface.

PIM implementers should enforce a lower bound on the permitted values for this delay to allow for scheduling and processing delays within their router. Such delays may cause received messages to be processed later as well as triggered messages to be sent later than intended. Setting this Propagation Delay to too low a value may result in temporary forwarding outages because a downstream router will not be able to override a neighbor's Prune message before the upstream neighbor stops forwarding.

When all routers on a link are in a position to negotiate a Propagation Delay different from the default, the largest value from those advertised by each neighbor is chosen. The function for computing the Effective_Propagation_Delay of interface I is:

```
time_interval
Effective_Propagation_Delay(I) {
    if ( lan_delay_enabled(I) == false ) {
        return Propagation_delay_default
    }
    delay = Propagation_Delay(I)
    for each neighbor on interface I {
        if ( neighbor.propagation_delay > delay ) {
            delay = neighbor.propagation_delay
        }
    }
    return delay
}
```

To avoid synchronization of override messages when multiple downstream routers share a multi-access link, sending of such messages is delayed by a small random amount of time. The period of randomization should represent the size of the PIM router population on the link. Each router expresses its view of the amount of randomization necessary in the Override Interval field of the LAN Prune Delay option.

When all routers on a link are in a position to negotiate an Override Interval different from the default, the largest value from those advertised by each neighbor is chosen. The function for computing the Effective Override Interval of interface I is:

```
time_interval
Effective_Override_Interval(I) {
    if ( lan_delay_enabled(I) == false ) {
        return t_override_default
    }
    delay = Override_Interval(I)
    for each neighbor on interface I {
        if ( neighbor.override_interval > delay ) {
            delay = neighbor.override_interval
        }
    }
    return delay
}
```

Although the mechanisms are not specified in this document, it is possible for upstream routers to explicitly track the join membership of individual downstream routers if Join suppression is disabled. A router can advertise its willingness to disable Join suppression by using the T bit in the LAN Prune Delay Hello option. Unless all PIM routers on a link negotiate this capability, explicit tracking and the disabling of the Join suppression mechanism are not possible. The function for computing the state of Suppression on interface I is:

```
bool
Suppression_Enabled(I) {
    if ( lan_delay_enabled(I) == false ) {
        return true
    }
    for each neighbor on interface I {
        if ( neighbor.tracking_support == false ) {
            return true
        }
    }
    return false
}
```

Note that the setting of Suppression_Enabled(I) affects the value of t_suppressed (see Section 4.10).

4.3.4. Maintaining Secondary Address Lists

Communication of a router's interface secondary addresses to its PIM neighbors is necessary to provide the neighbors with a mechanism for mapping next_hop information obtained through their MRIB to a primary address that can be used as a destination for Join/Prune messages. The mapping is performed through the NBR macro. The primary address of a PIM neighbor is obtained from the source IP address used in its PIM Hello messages. Secondary addresses are carried within the Hello message in an Address List Hello option. The primary address of the source interface of the router MUST NOT be listed within the Address List Hello option.

In addition to the information recorded for the DR Election, the following per neighbor information is obtained from the Address List Hello option:

`neighbor.secondary_address_list`

The list of secondary addresses used by the PIM neighbor on the interface through which the Hello message was transmitted.

When processing a received PIM Hello message containing an Address List Hello option, the list of secondary addresses in the message completely replaces any previously associated secondary addresses for that neighbor. If a received PIM Hello message does not contain an Address List Hello option, then all secondary addresses associated with the neighbor must be deleted. If a received PIM Hello message contains an Address List Hello option that includes the primary address of the sending router in the list of secondary addresses (although this is not expected), then the addresses listed in the message, excluding the primary address, are used to update the associated secondary addresses for that neighbor.

All the advertised secondary addresses in received Hello messages must be checked against those previously advertised by all other PIM neighbors on that interface. If there is a conflict and the same secondary address was previously advertised by another neighbor, then only the most recently received mapping **MUST** be maintained, and an error message **SHOULD** be logged to the administrator in a rate-limited manner.

Within one Address List Hello option, all the addresses **MUST** be of the same address family. It is not permitted to mix IPv4 and IPv6 addresses within the same message. In addition, the address family of the fields in the message **SHOULD** be the same as the IP source and destination addresses of the packet header.

4.4. PIM Register Messages

The Designated Router (DR) on a LAN or point-to-point link encapsulates multicast packets from local sources to the RP for the relevant group unless it recently received a Register-Stop message for that (S,G) or (*,G) from the RP. When the DR receives a Register-Stop message from the RP, it starts a Register-Stop Timer to maintain this state. Just before the Register-Stop Timer expires, the DR sends a Null-Register Message to the RP to allow the RP to refresh the Register-Stop information at the DR. If the Register-Stop Timer actually expires, the DR will resume encapsulating packets from the source to the RP.

4.4.1. Sending Register Messages from the DR

Every PIM-SM router has the capability to be a DR. The state machine below is used to implement Register functionality. For the purposes of specification, we represent the mechanism to encapsulate packets to the RP as a Register-Tunnel interface, which is added to or removed from the (S,G) olist. The tunnel interface then takes part in the normal packet forwarding rules as specified in Section 4.2.

If register state is maintained, it is maintained only for directly connected sources and is per-(S,G). There are four states in the DR's per-(S,G) Register state machine:

Join (J)

The register tunnel is "joined" (the join is actually implicit, but the DR acts as if the RP has joined the DR on the tunnel interface).

Prune (P)

The register tunnel is "pruned" (this occurs when a Register-Stop is received).

Join-Pending (JP)

The register tunnel is pruned but the DR is contemplating adding it back.

NoInfo (NI)

No information. This is the initial state, and the state when the router is not the DR.

In addition, a Register-Stop Timer (RST) is kept if the state machine is not in the NoInfo state.

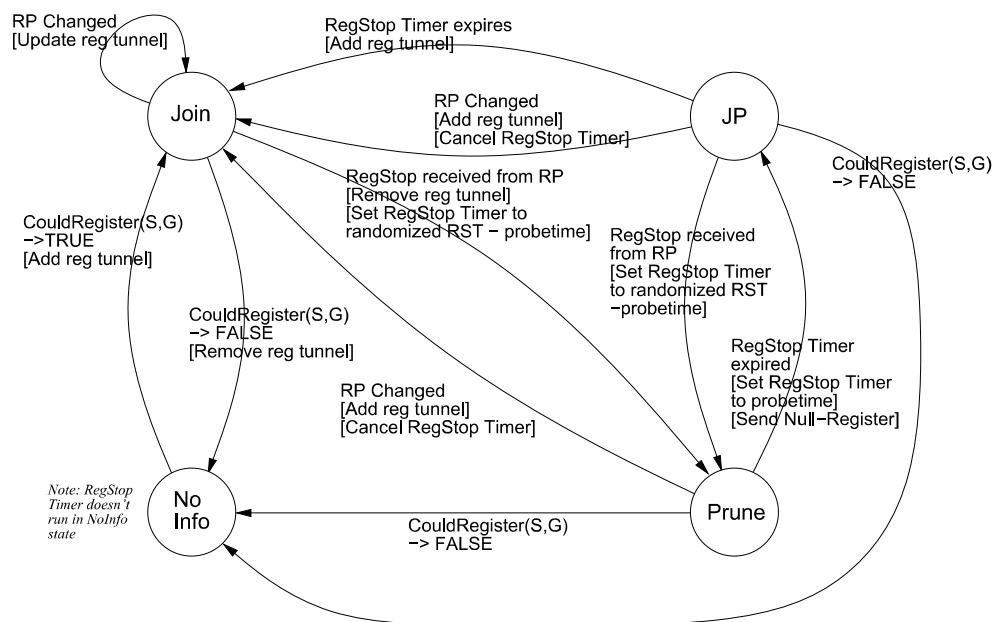


Figure 1: Per-(S,G) register state machine at a DR

Per-(S,G) register state machine at a DR in tabular form:

Prev State	Event				
	Register-Stop Timer expires	Could Register → True	Could Register → False	Register-Stop received	RP changed
NoInfo (NI)	-	→ J state add reg tunnel	-	-	-
Join (J)	-	-	→ NI state remove reg tunnel	→ P state remove reg tunnel; set Register-Stop Timer(*)	→ J state update reg tunnel
Join-Pending (JP)	→ J state add reg tunnel	-	→ NI state	→ P state set Register-Stop Timer(*)	→ J state add reg tunnel; cancel Register-Stop Timer
Prune (P)	→ JP state set Register-Stop Timer(**); send Null-Register	-	→ NI state	-	→ J state add reg tunnel; cancel Register-Stop Timer

Notes:

- (*) The Register-Stop Timer is set to a random value chosen uniformly from the interval ($0.5 * \text{Register_Suppression_Time}$, $1.5 * \text{Register_Suppression_Time}$) minus $\text{Register_Probe_Time}$.

Subtracting off $\text{Register_Probe_Time}$ is a bit unnecessary because it is really small compared to $\text{Register_Suppression_Time}$, but this was in the old spec and is kept for compatibility.

- (**) The Register-Stop Timer is set to $\text{Register_Probe_Time}$.

The following three actions are defined:

Add Register Tunnel

A Register-Tunnel virtual interface, VI, is created (if it doesn't already exist) with its encapsulation target being RP(G). $\text{DownstreamJPState}(S,G,VI)$ is set to Join state, causing the tunnel interface to be added to $\text{immediate_olist}(S,G)$ and $\text{inherited_olist}(S,G)$.

Remove Register Tunnel

VI is the Register-Tunnel virtual interface with encapsulation target of RP(G). $\text{DownstreamJPState}(S,G,VI)$ is set to NoInfo state, causing the tunnel interface to be removed from $\text{immediate_olist}(S,G)$ and $\text{inherited_olist}(S,G)$. If $\text{DownstreamJPState}(S,G,VI)$ is NoInfo for all (S,G), then VI can be deleted.

Update Register Tunnel

This action occurs when RP(G) changes.

VI_old is the Register-Tunnel virtual interface with encapsulation target old_RP(G). A Register-Tunnel virtual interface, VI_new, is created (if it doesn't already exist) with its encapsulation target being new_RP(G). DownstreamJPState(S,G,VI_old) is set to NoInfo state and DownstreamJPState(S,G,VI_new) is set to Join state. If DownstreamJPState(S,G,VI_old) is NoInfo for all (S,G), then VI_old can be deleted.

Note that we cannot simply change the encapsulation target of VI_old because not all groups using that encapsulation tunnel will have moved to the same new RP.

CouldRegister(S,G)

The macro "CouldRegister" in the state machine is defined as:

```
bool CouldRegister(S,G) {
    return ( I_am_DR( RPF_interface(S) ) AND
             KeepaliveTimer(S,G) is running AND
             DirectlyConnected(S) == TRUE )
}
```

Note that on reception of a packet at the DR from a directly connected source, `KeepaliveTimer(S,G)` needs to be set by the packet forwarding rules *before* computing `CouldRegister(S,G)` in the register state machine, or the first packet from a source won't be registered.

Encapsulating Data Packets in the Register Tunnel

Conceptually, the Register Tunnel is an interface with a smaller MTU than the underlying IP interface towards the RP. IP fragmentation on packets forwarded on the Register Tunnel is performed based upon this smaller MTU. The encapsulating DR may perform Path MTU Discovery to the RP to determine the effective MTU of the tunnel. Fragmentation for the smaller MTU should take both the outer IP header and the PIM register header overhead into account. If a multicast packet is fragmented on the way into the Register Tunnel, each fragment is encapsulated individually so it contains IP, PIM, and inner IP headers.

In IPv6, the DR **MUST** perform Path MTU discovery, and an ICMP Packet Too Big message **MUST** be sent by the encapsulating DR if it receives a packet that will not fit in the effective MTU of the tunnel. If the MTU between the DR and the RP results in the effective tunnel MTU being smaller than 1280 (the IPv6 minimum MTU), the DR **MUST** send Fragmentation Required messages with an MTU value of 1280 and **MUST** fragment its PIM register messages as required, using an IPv6 fragmentation header between the outer IPv6 header and the PIM Register header.

The TTL of a forwarded data packet is decremented before it is encapsulated in the Register Tunnel. The encapsulating packet uses the normal TTL that the router would use for any locally-generated IP packet.

The IP ECN bits should be copied from the original packet to the IP header of the encapsulating packet. They **SHOULD NOT** be set independently by the encapsulating router.

The Diffserv Code Point (DSCP) should be copied from the original packet to the IP header of the encapsulating packet. It **MAY** be set independently by the encapsulating router, based upon static configuration or traffic classification. See [12] for more discussion on setting the DSCP on tunnels.

Handling Register-Stop(*,G) Messages at the DR

An old RP might send a Register-Stop message with the source address set to all zeros. This was the normal course of action in RFC 2362 when the Register message matched against (*,G) state at the RP, and it was defined as meaning "stop encapsulating all sources for this group". However, the behavior of such a Register-Stop(*,G) is ambiguous or incorrect in some circumstances.

We specify that an RP should not send Register-Stop(*,G) messages, but for compatibility, a DR should be able to accept one if it is received.

A Register-Stop(*,G) should be treated as a Register-Stop(S,G) for all (S,G) Register state machines that are not in the NoInfo state. A router should not apply a Register-Stop(*,G) to sources that become active after the Register-Stop(*,G) was received.

4.4.2. Receiving Register Messages at the RP

When an RP receives a Register message, the course of action is decided according to the following pseudocode:

```
packet_arrives_on_rp_tunnel( pkt ) {
    if( outer.dst is not one of my addresses ) {
        drop the packet silently.
        # Note: this may be a spoofing attempt
    }
    if( I_am_RP(G) AND outer.dst == RP(G) ) {
        sentRegisterStop = FALSE;
        if ( register.borderbit == TRUE ) {
            if ( PMBR(S,G) == unknown ) {
                PMBR(S,G) = outer.src
            } else if ( outer.src != PMBR(S,G) ) {
                send Register-Stop(S,G) to outer.src
                drop the packet silently.
            }
        }
        if ( SPTbit(S,G) OR
            ( SwitchToSptDesired(S,G) AND
              ( inherited_olist(S,G) == NULL ))) {
            send Register-Stop(S,G) to outer.src
            sentRegisterStop = TRUE;
        }
        if ( SPTbit(S,G) OR SwitchToSptDesired(S,G) ) {
            if ( sentRegisterStop == TRUE ) {
                set KeepaliveTimer(S,G) to RP_Keepalive_Period;
            } else {
                set KeepaliveTimer(S,G) to Keepalive_Period;
            }
        }
        if( !SPTbit(S,G) AND ! pkt.NullRegisterBit ) {
            decapsulate and forward the inner packet to
            inherited_olist(S,G,rpt) # Note (†)
        }
    } else {
        send Register-Stop(S,G) to outer.src
        # Note (*)
    }
}
```

outer.dst is the IP destination address of the encapsulating header.

outer.src is the IP source address of the encapsulating header, i.e., the DR's address.

I_am_RP(G) is true if the group-to-RP mapping indicates that this router is the RP for the group.

Note (*): This may block traffic from S for Register_Suppression_Time if the DR learned about a new group-to-RP mapping before the RP did. However, this doesn't matter unless we figure out some way for the RP also to accept (*,G) joins when it doesn't

yet realize that it is about to become the RP for G. This will all get sorted out once the RP learns the new group-to-rp mapping. We decided to do nothing about this and just accept the fact that PIM may suffer interrupted (*,G) connectivity following an RP change.

Note (†): Implementations are advised not to make this a special case, but to arrange that this path rejoin the normal packet forwarding path. All of the appropriate actions from the "On receipt of data from S to G on interface iif" pseudocode in Section 4.2 should be performed.

KeepaliveTimer(S,G) is restarted at the RP when packets arrive on the proper tunnel interface and the RP desires to switch to the SPT or the SPTbit is already set. This may cause the upstream (S,G) state machine to trigger a join if the inherited_olist(S,G) is not NULL.

An RP should preserve (S,G) state that was created in response to a Register message for at least (3 * Register_Suppression_Time); otherwise, the RP may stop joining (S,G) before the DR for S has restarted sending registers. Traffic would then be interrupted until the Register-Stop Timer expires at the DR.

Thus, at the RP, KeepaliveTimer(S,G) should be restarted to (3 * Register_Suppression_Time + Register_Probe_Time).

When forwarding a packet from the Register Tunnel, the TTL of the original data packet is decremented after it is decapsulated.

The IP ECN bits should be copied from the IP header of the Register packet to the decapsulated packet.

The Diffserv Code Point (DSCP) should be copied from the IP header of the Register packet to the decapsulated packet. The RP MAY retain the DSCP of the inner packet or re-classify the packet and apply a different DSCP. Scenarios where each of these might be useful are discussed in [12].

4.5. PIM Join/Prune Messages

A PIM Join/Prune message consists of a list of groups and a list of Joined and Pruned sources for each group. When processing a received Join/Prune message, each Joined or Pruned source for a Group is effectively considered individually, and applies to one or more of the following state machines. When considering a Join/Prune message whose Upstream Neighbor Address field addresses this router, (*,G) Joins and Prunes can affect both the (*,G) and (S,G,rpt) downstream state machines, while (*,*,RP), (S,G), and (S,G,rpt) Joins and Prunes can only affect their respective downstream state machines. When considering a Join/Prune message whose Upstream Neighbor Address field addresses another router, most Join or Prune messages could affect each upstream state machine.

In general, a PIM Join/Prune message should only be accepted for processing if it comes from a known PIM neighbor. A PIM router hears about PIM neighbors through PIM Hello messages. If a router receives a Join/Prune message from a particular IP source address and it has not seen a PIM Hello message from that source address, then the Join/Prune message SHOULD be discarded without further processing. In addition, if the Hello message from a neighbor was authenticated using IPsec AH (see Section 6.3), then all Join/Prune messages from that neighbor MUST also be authenticated using IPsec AH.

We note that some older PIM implementations incorrectly fail to send Hello messages on point-to-point interfaces, so we also RECOMMEND that a configuration option be provided to allow interoperation with such older routers, but that this configuration option SHOULD

NOT be enabled by default.

4.5.1. Receiving (*,*,RP) Join/Prune Messages

The per-interface state machine for receiving (*,*,RP) Join/Prune Messages is given below. There are three states:

NoInfo (NI)

The interface has no (*,*,RP) Join state and no timers running.

Join (J)

The interface has (*,*,RP) Join state, which will cause the router to forward packets destined for any group handled by RP from this interface except if there is also (S,G,rpt) prune information (see Section 4.5.4) or the router lost an assert on this interface.

Prune-Pending (PP)

The router has received a Prune(*,*,RP) on this interface from a downstream neighbor and is waiting to see whether the prune will be overridden by another downstream router. For forwarding purposes, the Prune-Pending state functions exactly like the Join state.

In addition, the state machine uses two timers:

ExpiryTimer (ET)

This timer is restarted when a valid Join(*,*,RP) is received. Expiry of the ExpiryTimer causes the interface state to revert to NoInfo for this RP.

Prune-Pending Timer (PPT)

This timer is set when a valid Prune(*,*,RP) is received. Expiry of the Prune-Pending Timer causes the interface state to revert to NoInfo for this RP.

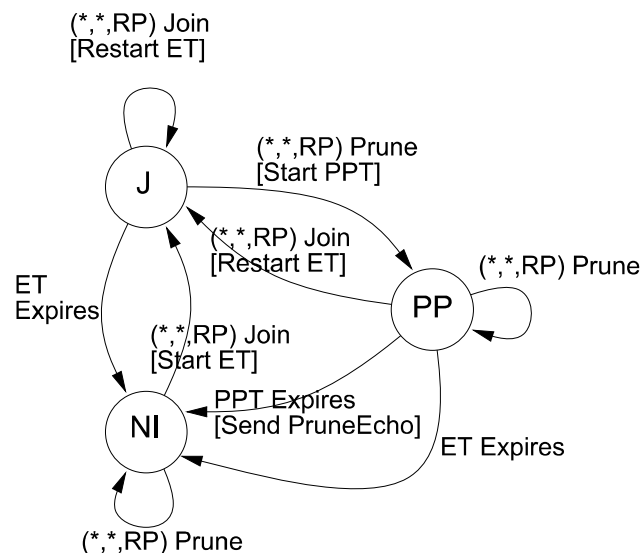


Figure 2: Downstream per-interface (*,*,RP) state machine

Downstream per-interface (*,*,RP) state machine in tabular form:

Prev State	Event			
	Receive Join(*,*,RP)	Receive Prune(*,*,RP)	Prune-Pending Timer Expires	Expiry Timer Expires
NoInfo (NI)	→ J state start Expiry Timer	→ NI state	-	-
Join (J)	→ J state restart Expiry Timer	→ PP state start Prune- Pending Timer	-	→ NI state
Prune-Pending (PP)	→ J state restart Expiry Timer	→ PP state	→ NI state Send Prune- Echo(*,*,RP)	→ NI state

The transition events "Receive Join(*,*,RP)" and "Receive Prune(*,*,RP)" imply receiving a Join or Prune targeted to this router's primary IP address on the received interface. If the upstream neighbor address field is not correct, these state transitions in this state machine must not occur, although seeing such a packet may cause state transitions in other state machines.

On unnumbered interfaces on point-to-point links, the router's address should be the same as the source address it chose for the Hello message it sent over that interface. However, on point-to-point links we also recommend that for backwards compatibility PIM Join/Prune messages with an upstream neighbor address field of all zeros are also accepted.

Transitions from NoInfo State

When in NoInfo state, the following event may trigger a transition:

Receive Join(*,*,RP)

A Join(*,*,RP) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,*,RP) downstream state machine on interface I transitions to the Join state. The Expiry Timer (ET) is started and set to the HoldTime from the triggering Join/Prune message.

Note that it is possible to receive a Join(*,*,RP) message for an RP for which we do not have information telling us that it is an RP. In the case of (*,*,RP) state, so long as we have a route to the RP, this will not cause a problem, and the transition should still take place.

Transitions from Join State

When in Join state, the following events may trigger a transition:

Receive Join(*,*,RP)

A Join(*,*,RP) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,*,RP) downstream state machine on interface I remains in Join state, and the Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Receive Prune(*,*,RP)

A Prune(*,*,RP) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,*,RP) downstream state machine on interface I transitions to the Prune-Pending state. The Prune-Pending Timer is started. It is set to the J/P_Override_Interval(I) if the router has more than one neighbor on that interface; otherwise, it is set to zero, causing it to expire immediately.

Expiry Timer Expires

The Expiry Timer for the (*,*,RP) downstream state machine on interface I expires.

The (*,*,RP) downstream state machine on interface I transitions to the NoInfo state.

Transitions from Prune-Pending State

When in Prune-Pending state, the following events may trigger a transition:

Receive Join(*,*,RP)

A Join(*,*,RP) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,*,RP) downstream state machine on interface I transitions to the Join state. The Prune-Pending Timer is canceled (without triggering an expiry event). The Expiry Timer is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Expiry Timer Expires

The Expiry Timer for the (*,*,RP) downstream state machine on interface I expires.

The (*,*,RP) downstream state machine on interface I transitions to the NoInfo state.

Prune-Pending Timer Expires

The Prune-Pending Timer for the (*,*,RP) downstream state machine on interface I expires.

The (*,*,RP) downstream state machine on interface I transitions to the NoInfo state. A PruneEcho(*,*,RP) is sent onto the subnet connected to interface I.

The action "Send PruneEcho(*,*,RP)" is triggered when the router stops forwarding on an interface as a result of a prune. A PruneEcho(*,*,RP) is simply a Prune(*,*,RP) message sent by the upstream router on a LAN with its own address in the Upstream Neighbor Address field. Its purpose is to add additional reliability so that if a Prune that should have been overridden by another router is lost locally on the LAN, then the PruneEcho may be received and cause the override to happen. A PruneEcho(*,*,RP) need not be sent on an interface that contains only a single PIM neighbor during the time this state machine was in Prune-Pending state.

4.5.2. Receiving (*,G) Join/Prune Messages

When a router receives a Join(*,G), it must first check to see whether the RP in the message matches RP(G) (the router's idea of who the RP is). If the RP in the message does not match RP(G), the Join(*,G) should be silently dropped. (Note that other source list entries, such as (S,G,rpt) or (S,G), in the same Group-Specific Set should still be processed.) If a

router has no RP information (e.g., has not recently received a BSR message), then it may choose to accept Join(*,G) and treat the RP in the message as RP(G). Received Prune(*,G) messages are processed even if the RP in the message does not match RP(G).

The per-interface state machine for receiving (*,G) Join/Prune Messages is given below. There are three states:

NoInfo (NI)

The interface has no (*,G) Join state and no timers running.

Join (J)

The interface has (*,G) Join state, which will cause the router to forward packets destined for G from this interface except if there is also (S,G,rpt) prune information (see Section 4.5.4) or the router lost an assert on this interface.

Prune-Pending (PP)

The router has received a Prune(*,G) on this interface from a downstream neighbor and is waiting to see whether the prune will be overridden by another downstream router. For forwarding purposes, the Prune-Pending state functions exactly like the Join state.

In addition, the state machine uses two timers:

Expiry Timer (ET)

This timer is restarted when a valid Join(*,G) is received. Expiry of the Expiry Timer causes the interface state to revert to NoInfo for this group.

Prune-Pending Timer (PPT)

This timer is set when a valid Prune(*,G) is received. Expiry of the Prune-Pending Timer causes the interface state to revert to NoInfo for this group.

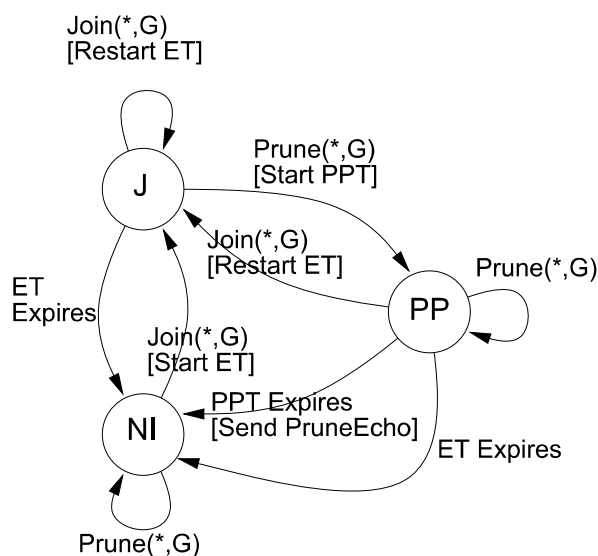


Figure 3: Downstream per-interface (*,G) state machine

Downstream per-interface (*,G) state machine in tabular form:

Prev State	Event			
	Receive Join(*,G)	Receive Prune(*,G)	Prune-Pending Timer Expires	Expiry Timer Expires
NoInfo (NI)	→ J state start Expiry Timer	→ NI state	-	-
Join (J)	→ J state restart Expiry Timer	→ PP state start Prune- Pending Timer	-	→ NI state
Prune-Pending (PP)	→ J state restart Expiry Timer	→ PP state	→ NI state Send Prune- Echo(*,G)	→ NI state

The transition events "Receive Join(*,G)" and "Receive Prune(*,G)" imply receiving a Join or Prune targeted to this router's primary IP address on the received interface. If the upstream neighbor address field is not correct, these state transitions in this state machine must not occur, although seeing such a packet may cause state transitions in other state machines.

On unnumbered interfaces on point-to-point links, the router's address should be the same as the source address it chose for the Hello message it sent over that interface. However, on point-to-point links we also recommend that for backwards compatibility PIM Join/Prune messages with an upstream neighbor address field of all zeros are also accepted.

Transitions from NoInfo State

When in NoInfo state, the following event may trigger a transition:

Receive Join(*,G)

A Join(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,G) downstream state machine on interface I transitions to the Join state. The Expiry Timer (ET) is started and set to the HoldTime from the triggering Join/Prune message.

Transitions from Join State

When in Join state, the following events may trigger a transition:

Receive Join(*,G)

A Join(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,G) downstream state machine on interface I remains in Join state, and the Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Receive Prune(*,G)

A Prune(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,G) downstream state machine on interface I transitions to the Prune-Pending state. The Prune-Pending Timer is started. It is set to the J/P_Override_Interval(I) if the router has more than one neighbor on that

interface; otherwise, it is set to zero, causing it to expire immediately.

Expiry Timer Expires

The Expiry Timer for the (*,G) downstream state machine on interface I expires.

The (*,G) downstream state machine on interface I transitions to the NoInfo state.

Transitions from Prune-Pending State

When in Prune-Pending state, the following events may trigger a transition:

Receive Join(*,G)

A Join(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (*,G) downstream state machine on interface I transitions to the Join state.

The Prune-Pending Timer is canceled (without triggering an expiry event). The Expiry Timer is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Expiry Timer Expires

The Expiry Timer for the (*,G) downstream state machine on interface I expires.

The (*,G) downstream state machine on interface I transitions to the NoInfo state.

Prune-Pending Timer Expires

The Prune-Pending Timer for the (*,G) downstream state machine on interface I expires.

The (*,G) downstream state machine on interface I transitions to the NoInfo state. A PruneEcho(*,G) is sent onto the subnet connected to interface I.

The action "Send PruneEcho(*,G)" is triggered when the router stops forwarding on an interface as a result of a prune. A PruneEcho(*,G) is simply a Prune(*,G) message sent by the upstream router on a LAN with its own address in the Upstream Neighbor Address field. Its purpose is to add additional reliability so that if a Prune that should have been overridden by another router is lost locally on the LAN, then the PruneEcho may be received and cause the override to happen. A PruneEcho(*,G) need not be sent on an interface that contains only a single PIM neighbor during the time this state machine was in Prune-Pending state.

4.5.3. Receiving (S,G) Join/Prune Messages

The per-interface state machine for receiving (S,G) Join/Prune messages is given below and is almost identical to that for (*,G) messages. There are three states:

NoInfo (NI)

The interface has no (S,G) Join state and no (S,G) timers running.

Join (J)

The interface has (S,G) Join state, which will cause the router to forward packets from S destined for G from this interface if the (S,G) state is active (the SPTbit is set) except if the router lost an assert on this interface.

Prune-Pending (PP)

The router has received a Prune(S,G) on this interface from a downstream neighbor and is waiting to see whether the prune will be overridden by another

downstream router. For forwarding purposes, the Prune-Pending state functions exactly like the Join state.

In addition, there are two timers:

Expiry Timer (ET)

This timer is set when a valid Join(S,G) is received. Expiry of the Expiry Timer causes this state machine to revert to NoInfo state.

Prune-Pending Timer (PPT)

This timer is set when a valid Prune(S,G) is received. Expiry of the Prune-Pending Timer causes this state machine to revert to NoInfo state.

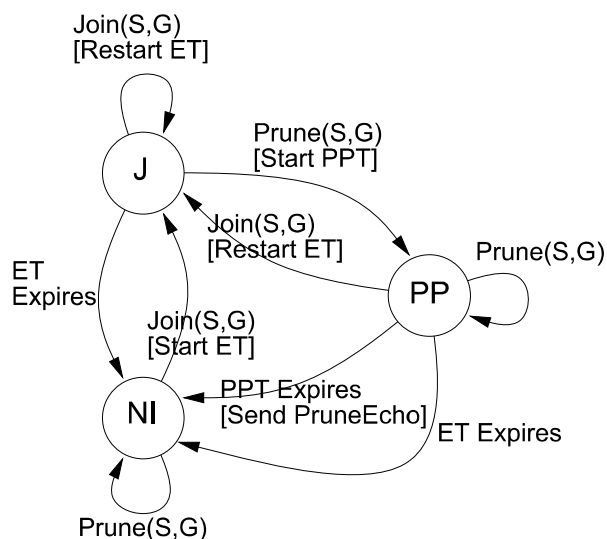


Figure 4: Downstream per-interface (S,G) state machine

Downstream per-interface (S,G) state machine in tabular form:

Prev State	Event			
	Receive Join(S,G)	Receive Prune(S,G)	Prune-Pending Timer Expires	Expiry Timer Expires
NoInfo (NI)	→ J state start Expiry Timer	→ NI state	-	-
Join (J)	→ J state restart Expiry Timer	→ PP state start Prune-Pending Timer	-	→ NI state
Prune-Pending (PP)	→ J state restart Expiry Timer	→ PP state	→ NI state Send Prune-Echo(S,G)	→ NI state

The transition events "Receive Join(S,G)" and "Receive Prune(S,G)" imply receiving a Join or Prune targeted to this router's primary IP address on the received interface. If the upstream neighbor address field is not correct, these state transitions in this state machine must not occur, although seeing such a packet may cause state transitions in other state machines.

On unnumbered interfaces on point-to-point links, the router's address should be the same as the source address it chose for the Hello message it sent over that interface. However, on point-to-point links we also recommend that for backwards compatibility PIM Join/Prune messages with an upstream neighbor address field of all zeros are also accepted.

Transitions from NoInfo State

When in NoInfo state, the following event may trigger a transition:

Receive Join(S,G)

A Join(S,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G) downstream state machine on interface I transitions to the Join state. The Expiry Timer (ET) is started and set to the HoldTime from the triggering Join/Prune message.

Transitions from Join State

When in Join state, the following events may trigger a transition:

Receive Join(S,G)

A Join(S,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G) downstream state machine on interface I remains in Join state, and the Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Receive Prune(S,G)

A Prune(S,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G) downstream state machine on interface I transitions to the Prune-Pending state. The Prune-Pending Timer is started. It is set to the J/P_Override_Interval(I) if the router has more than one neighbor on that interface; otherwise, it is set to zero, causing it to expire immediately.

Expiry Timer Expires

The Expiry Timer for the (S,G) downstream state machine on interface I expires.

The (S,G) downstream state machine on interface I transitions to the NoInfo state.

Transitions from Prune-Pending State

When in Prune-Pending state, the following events may trigger a transition:

Receive Join(S,G)

A Join(S,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G) downstream state machine on interface I transitions to the Join state. The Prune-Pending Timer is canceled (without triggering an expiry event). The Expiry Timer is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Expiry Timer Expires

The Expiry Timer for the (S,G) downstream state machine on interface I expires.

The (S,G) downstream state machine on interface I transitions to the NoInfo state.

Prune-Pending Timer Expires

The Prune-Pending Timer for the (S,G) downstream state machine on interface I expires.

The (S,G) downstream state machine on interface I transitions to the NoInfo state. A PruneEcho(S,G) is sent onto the subnet connected to interface I.

The action "Send PruneEcho(S,G)" is triggered when the router stops forwarding on an interface as a result of a prune. A PruneEcho(S,G) is simply a Prune(S,G) message sent by the upstream router on a LAN with its own address in the Upstream Neighbor Address field. Its purpose is to add additional reliability so that if a Prune that should have been overridden by another router is lost locally on the LAN, then the PruneEcho may be received and cause the override to happen. A PruneEcho(S,G) need not be sent on an interface that contains only a single PIM neighbor during the time this state machine was in Prune-Pending state.

4.5.4. Receiving (S,G,rpt) Join/Prune Messages

The per-interface state machine for receiving (S,G,rpt) Join/Prune messages is given below. There are five states:

NoInfo (NI)

The interface has no (S,G,rpt) Prune state and no (S,G,rpt) timers running.

Prune (P)

The interface has (S,G,rpt) Prune state, which will cause the router not to forward packets from S destined for G from this interface even though the interface has active (*,G) Join state.

Prune-Pending (PP)

The router has received a Prune(S,G,rpt) on this interface from a downstream neighbor and is waiting to see whether the prune will be overridden by another downstream router. For forwarding purposes, the Prune-Pending state functions exactly like the NoInfo state.

PruneTmp (P')

This state is a transient state that for forwarding purposes behaves exactly like the Prune state. A (*,G) Join has been received (which may cancel the (S,G,rpt) Prune). As we parse the Join/Prune message from top to bottom, we first enter this state if the message contains a (*,G) Join. Later in the message, we will normally encounter an (S,G,rpt) prune to reinstate the Prune state. However, if we reach the end of the message without encountering such a (S,G,rpt) prune, then we will revert to NoInfo state in this state machine.

As no time is spent in this state, no timers can expire.

Prune-Pending-Tmp (PP')

This state is a transient state that is identical to P' except that it is associated with the PP state rather than the P state. For forwarding purposes, PP' behaves exactly like PP state.

In addition, there are two timers:

Expiry Timer (ET)

This timer is set when a valid Prune(S,G,rpt) is received. Expiry of the Expiry Timer causes this state machine to revert to NoInfo state.

Prune-Pending Timer (PPT)

This timer is set when a valid Prune(S,G,rpt) is received. Expiry of the Prune-Pending Timer causes this state machine to move on to Prune state.

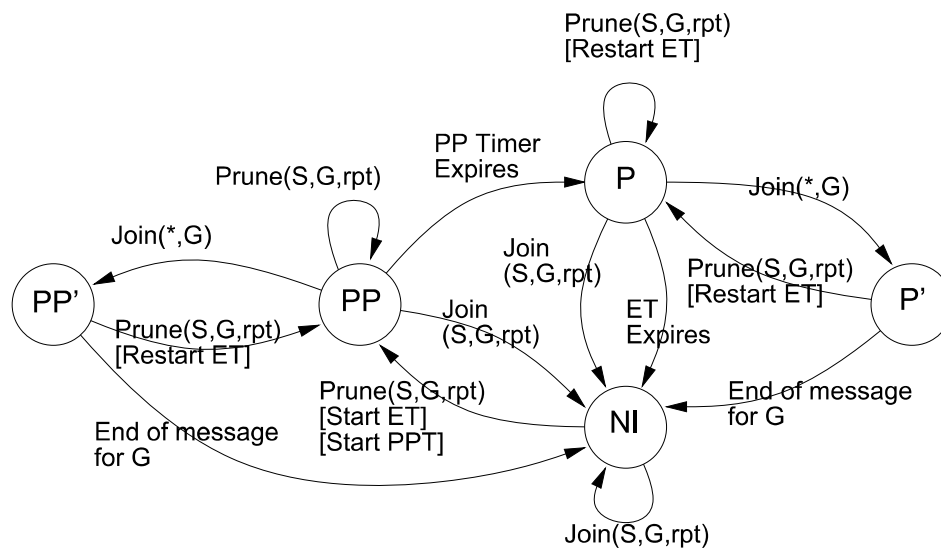


Figure 5: Downstream per-interface (S,G,rpt) state machine

Downstream per-interface (S,G,rpt) state machine in tabular form:

Prev State	Event					
	Receive Join(*,G)	Receive Join (S,G,rpt)	Receive Prune (S,G,rpt)	End of Message	Prune-Pending Timer Expires	Expiry Timer Expires
NoInfo (NI)	-	-	→ PP state start Prune-Pending Timer; start Expiry Timer	-	-	-
Prune (P)	→ P' state	→ NI state	→ P state restart Expiry Timer	-	-	→ NI state
Prune-Pending (PP)	→ PP' state	→ NI state	-	-	→ P state	-
PruneTmp (P')	-	-	→ P state restart Expiry Timer	→ NI state	-	-
Prune-Pending-Tmp (PP')	-	-	→ PP state restart Expiry Timer	→ NI state	-	-

The transition events "Receive Join(S,G,rpt)", "Receive Prune(S,G,rpt)", and "Receive Join(*,G)" imply receiving a Join or Prune targeted to this router's primary IP address on the received interface. If the upstream neighbor address field is not correct, these state transitions in this state machine must not occur, although seeing such a packet may cause state transitions in other state machines.

On unnumbered interfaces on point-to-point links, the router's address should be the same as the source address it chose for the Hello message it sent over that interface. However, on point-to-point links we also recommend that PIM Join/Prune messages with an upstream neighbor address field of all zeros are also accepted.

Transitions from NoInfo State

When in NoInfo (NI) state, the following event may trigger a transition:

Receive Prune(S,G,rpt)

A Prune(S,G,rpt) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I transitions to the Prune-Pending state. The Expiry Timer (ET) is started and set to the HoldTime from the triggering Join/Prune message. The Prune-Pending Timer is started. It is set to the J/P_Override_Interval(I) if the router has more than one neighbor on that interface; otherwise, it is set to zero, causing it to expire immediately.

Transitions from Prune-Pending State

When in Prune-Pending (PP) state, the following events may trigger a transition:

Receive Join(*,G)

A Join(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I transitions to Prune-Pending-Tmp state whilst the remainder of the compound Join/Prune message containing the Join(*,G) is processed.

Receive Join(S,G,rpt)

A Join(S,G,rpt) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I transitions to NoInfo state. ET and PPT are canceled.

Prune-Pending Timer Expires

The Prune-Pending Timer for the (S,G,rpt) downstream state machine on interface I expires.

The (S,G,rpt) downstream state machine on interface I transitions to the Prune state.

Transitions from Prune State

When in Prune (P) state, the following events may trigger a transition:

Receive Join(*,G)

A Join(*,G) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I transitions to PruneTmp state whilst the remainder of the compound Join/Prune message containing the Join(*,G) is processed.

Receive Join(S,G,rpt)

A Join(S,G,rpt) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I transitions to NoInfo state. ET and PPT are canceled.

Receive Prune(S,G,rpt)

A Prune(S,G,rpt) is received on interface I with its Upstream Neighbor Address set to the router's primary IP address on I.

The (S,G,rpt) downstream state machine on interface I remains in Prune state.

The Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

Expiry Timer Expires

The Expiry Timer for the (S,G,rpt) downstream state machine on interface I expires.

The (S,G,rpt) downstream state machine on interface I transitions to the NoInfo state.

Transitions from Prune-Pending-Tmp State

When in Prune-Pending-Tmp (PP') state and processing a compound Join/Prune message, the following events may trigger a transition:

Receive Prune(S,G,rpt)

The compound Join/Prune message contains a Prune(S,G,rpt).

The (S,G,rpt) downstream state machine on interface I transitions back to the Prune-Pending state. The Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

End of Message

The end of the compound Join/Prune message is reached.

The (S,G,rpt) downstream state machine on interface I transitions to the NoInfo state. ET and PPT are canceled.

Transitions from PruneTmp State

When in PruneTmp (P') state and processing a compound Join/Prune message, the following events may trigger a transition:

Receive Prune(S,G,rpt)

The compound Join/Prune message contains a Prune(S,G,rpt).

The (S,G,rpt) downstream state machine on interface I transitions back to the Prune state. The Expiry Timer (ET) is restarted, set to maximum of its current value and the HoldTime from the triggering Join/Prune message.

End of Message

The end of the compound Join/Prune message is reached.

The (S,G,rpt) downstream state machine on interface I transitions to the NoInfo state. ET is canceled.

Notes:

Receiving a Prune(*,G) does not affect the (S,G,rpt) downstream state machine.

Receiving a Join(*,*,RP) does not affect the (S,G,rpt) downstream state machine. If a router has originated Join(*,*,RP) and pruned a source off it using Prune(S,G,rpt), then to receive that source again it should explicitly re-join using Join(S,G,rpt) or Join(*,G). In some LAN topologies it is possible for a router sending a new Join(*,*,RP) to have to wait as much as a Join/Prune Interval before noticing that it needs to override a neighbor's preexisting Prune(S,G,rpt). This is considered acceptable, as (*,*,RP) state is intended to be used only in long-lived and persistent scenarios.

4.5.5. Sending (*,*,RP) Join/Prune Messages

The per-interface state machines for (*,*,RP) hold join state from downstream PIM routers. This state then determines whether a router needs to propagate a Join(*,*,RP) upstream towards the RP.

If a router wishes to propagate a Join(*,*,RP) upstream, it must also watch for messages on its upstream interface from other routers on that subnet, and these may modify its behavior. If it sees a Join(*,*,RP) to the correct upstream neighbor, it should suppress its own Join(*,*,RP). If it sees a Prune(*,*,RP) to the correct upstream neighbor, it should be prepared to override that prune by sending a Join(*,*,RP) almost immediately. Finally, if it sees the Generation ID (see Section 4.3) of the correct upstream neighbor change, it knows that the upstream neighbor has lost state, and it should be prepared to refresh the state by sending a Join(*,*,RP) almost immediately.

In addition, if the MRIB changes to indicate that the next hop towards the RP has changed, the router should prune off from the old next hop and join towards the new next hop.

The upstream (*,*,RP) state machine contains only two states:

Not Joined

The downstream state machines and local membership information do not indicate that the router needs to join the (*,*,RP) tree for this RP.

Joined

The downstream state machines and local membership information indicate that the router should join the (*,*,RP) tree for this RP.

In addition, one timer JT(*,*,RP) is kept that is used to trigger the sending of a Join(*,*,RP) to the upstream next hop towards the RP, NBR(RPF_interface(RP), MRIB.next_hop(RP)).

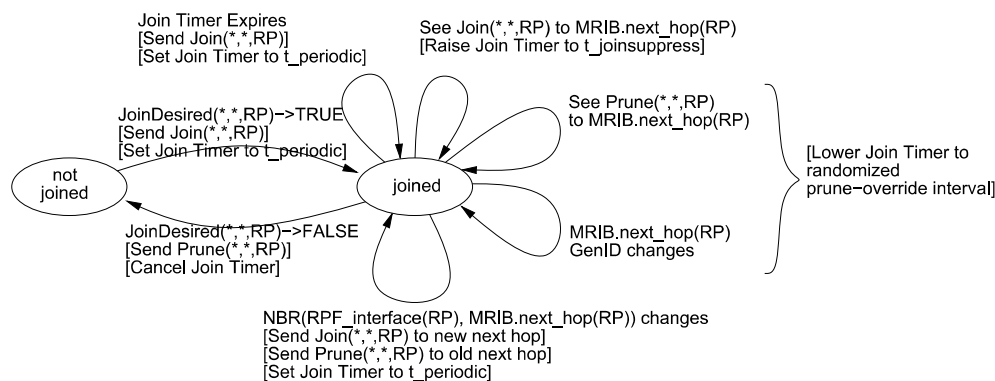


Figure 6: Upstream (*,*,RP) state machine

Upstream (*,*,RP) state machine in tabular form:

Prev State	Event	
	JoinDesired(*,*,RP) →True	JoinDesired(*,*,RP) →False
NotJoined (NJ)	→ J state Send Join(*,*,RP); Set Join Timer to t_periodic	-
Joined (J)	-	→ NJ state Send Prune(*,*,RP); Cancel Join Timer

In addition, we have the following transitions, which occur within the Joined state:

In Joined (J) State		
Timer Expires	See Join(*,*,RP) to MRIB. next_hop(RP)	See Prune(*,*,RP) to MRIB. next_hop(RP)
Send Join(*,*,RP); Set Join Timer to t_periodic	Increase Join Timer to t_joinsuppress	Decrease Join Timer to t_override

In Joined (J) State	
NBR(RPF_interface(RP), MRIB.next_hop(RP)) changes	MRIB.next_hop(RP) GenID changes
Send Join(*,*,RP) to new next hop; Send Prune(*,*,RP) to old next hop; set Join Timer to t_periodic	Decrease Join Timer to t_override

This state machine uses the following macro:

```
bool JoinDesired(*,*,RP) {
    if immediate_olist(*,*,RP) != NULL
        return TRUE
    else
        return FALSE
}
```

JoinDesired(*,*,RP) is true when the router has received (*,*,RP) Joins from any downstream interface. Note that although JoinDesired is true, the router's sending of a Join(*,*,RP) message may be suppressed by another router sending a Join(*,*,RP) onto the upstream interface.

Transitions from NotJoined State

When the upstream (*,*,RP) state machine is in NotJoined state, the following event may trigger a state transition:

JoinDesired(*,*,RP) becomes True

The downstream state for (*,*,RP) has changed so that at least one interface is in immediateolist(*,*,RP), making JoinDesired(*,*,RP) become True.

The upstream (*,*,RP) state machine transitions to Joined state. Send Join(*,*,RP) to the appropriate upstream neighbor, which is NBR(RPF_interface(RP), MRIB.next_hop(RP)). Set the Join Timer (JT) to expire after t_periodic seconds.

Transitions from Joined State

When the upstream (*,*,RP) state machine is in Joined state, the following events may trigger state transitions:

JoinDesired(*,*,RP) becomes False

The downstream state for (*,*,RP) has changed so no interface is in immediateolist(*,*,RP), making JoinDesired(*,*,RP) become False.

The upstream (*,*,RP) state machine transitions to NotJoined state. Send Prune(*,*,RP) to the appropriate upstream neighbor, which is NBR(RPF_interface(RP), MRIB.next_hop(RP)). Cancel the Join Timer (JT).

Join Timer Expires

The Join Timer (JT) expires, indicating time to send a Join(*,*,RP)

Send Join(*,*,RP) to the appropriate upstream neighbor, which is NBR(RPF_interface(RP), MRIB.next_hop(RP)). Restart the Join Timer (JT) to expire after t_periodic seconds.

See Join(*,*,RP) to MRIB.next_hop(RP)

This event is only relevant if RPF_interface(RP) is a shared medium. This router sees another router on RPF_interface(RP) send a Join(*,*,RP) to NBR(RPF_interface(RP), MRIB.next_hop(RP)). This causes this router to suppress its own Join.

The upstream (*,*,RP) state machine remains in Joined state.

Let t_joinsuppress be the minimum of t_suppressed and the HoldTime from the Join/Prune message triggering this event.

If the Join Timer is set to expire in less than t_joinsuppress seconds, reset it so that it expires after t_joinsuppress seconds. If the Join Timer is set to expire in more than t_joinsuppress seconds, leave it unchanged.

See Prune(*,*,RP) to MRIB.next_hop(RP)

This event is only relevant if RPF_interface(RP) is a shared medium. This router sees another router on RPF_interface(RP) send a Prune(*,*,RP) to NBR(RPF_interface(RP), MRIB.next_hop(RP)). As this router is in Joined state, it must override the Prune after a short random interval.

The upstream (*,*,RP) state machine remains in Joined state. If the Join Timer is set to expire in more than t_override seconds, reset it so that it expires after t_override seconds. If the Join Timer is set to expire in less than t_override seconds, leave it unchanged.

NBR(RPF_interface(RP), MRIB.next_hop(RP)) changes

A change in the MRIB routing base causes the next hop towards the RP to change.

The upstream (*,*,RP) state machine remains in Joined state. Send Join(*,*,RP) to the new upstream neighbor, which is the new value of NBR(RPF_interface(RP), MRIB.next_hop(RP)). Send Prune(*,*,RP) to the old upstream neighbor, which is the old value of NBR(RPF_interface(RP), MRIB.next_hop(RP)). Set the Join Timer (JT) to expire after t_periodic seconds.

MRIB.next_hop(RP) GenID changes

The Generation ID of the router that is MRIB.next_hop(RP) changes. This normally means that this neighbor has lost state, and so the state must be refreshed.

The upstream (*,*,RP) state machine remains in Joined state. If the Join Timer is set to expire in more than t_override seconds, reset it so that it expires after t_override seconds.

4.5.6. Sending (*,G) Join/Prune Messages

The per-interface state machines for (*,G) hold join state from downstream PIM routers. This state then determines whether a router needs to propagate a Join(*,G) upstream towards the RP.

If a router wishes to propagate a Join(*,G) upstream, it must also watch for messages on its upstream interface from other routers on that subnet, and these may modify its behavior. If it sees a Join(*,G) to the correct upstream neighbor, it should suppress its own Join(*,G). If it sees a Prune(*,G) to the correct upstream neighbor, it should be prepared to override that prune by sending a Join(*,G) almost immediately. Finally, if it sees the Generation ID (see Section 4.3) of the correct upstream neighbor change, it knows that the upstream neighbor has lost state, and it should be prepared to refresh the state by sending a Join(*,G) almost immediately.

If a (*,G) Assert occurs on the upstream interface, and this changes this router's idea of the upstream neighbor, it should be prepared to ensure that the Assert winner is aware of downstream routers by sending a Join(*,G) almost immediately.

In addition, if the MRIB changes to indicate that the next hop towards the RP has changed, and either the upstream interface changes or there is no Assert winner on the upstream interface, the router should prune off from the old next hop and join towards the new next hop.

The upstream (*,G) state machine only contains two states:

Not Joined

The downstream state machines indicate that the router does not need to join the RP tree for this group.

Joined

The downstream state machines indicate that the router should join the RP tree for this group.

In addition, one timer JT(*,G) is kept that is used to trigger the sending of a Join(*,G) to the upstream next hop towards the RP, RPF'(*,G).

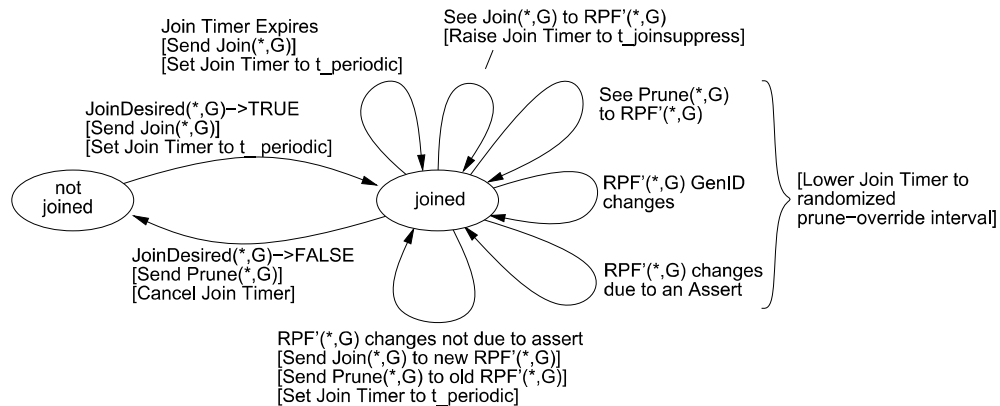


Figure 7: Upstream (*,G) state machine

Upstream (*,G) state machine in tabular form:

Prev State	Event	
	JoinDesired(*,G) →True	JoinDesired(*,G) →False
NotJoined (NJ)	→ J state Send Join(*,G); Set Join Timer to t_periodic	-
Joined (J)	-	→ NJ state Send Prune(*,G); Cancel Join Timer

In addition, we have the following transitions, which occur within the Joined state:

In Joined (J) State			
Timer Expires	See Join(*,G) to RPF'(*,G)	See Prune(*,G) to RPF'(*,G)	RPF'(*,G) changes due to an Assert
Send Join(*,G); Set Join Timer to t_periodic	Increase Join Timer to t_joinsuppress	Decrease Join Timer to t_override	Decrease Join Timer to t_override

In Joined (J) State	
RPF'(*,G) changes not due to an Assert	RPF'(*,G) GenID changes
Send Join(*,G) to new next hop; Send Prune(*,G) to old next hop; Set Join Timer to t_periodic	Decrease Join Timer to t_override

This state machine uses the following macro:

```
bool JoinDesired(*,G) {
    if (immediate_olist(*,G) != NULL OR
        (JoinDesired(*,*,RP(G)) AND
         AssertWinner(*, G, RPF_interface(RP(G))) != NULL))
        return TRUE
    else
        return FALSE
}
```

JoinDesired(*,G) is true when the router has forwarding state that would cause it to forward traffic for G using shared tree state. Note that although JoinDesired is true, the router's sending of a Join(*,G) message may be suppressed by another router sending a Join(*,G) onto the upstream interface.

Transitions from NotJoined State

When the upstream (*,G) state machine is in NotJoined state, the following event may trigger a state transition:

JoinDesired(*,G) becomes True

The macro JoinDesired(*,G) becomes True, e.g., because the downstream state for (*,G) has changed so that at least one interface is in immediate_olist(*,G).

The upstream (*,G) state machine transitions to Joined state. Send Join(*,G) to the appropriate upstream neighbor, which is RPF'(*,G). Set the Join Timer (JT) to expire after t_periodic seconds.

Transitions from Joined State

When the upstream (*,G) state machine is in Joined state, the following events may trigger state transitions:

JoinDesired(*,G) becomes False

The macro JoinDesired(*,G) becomes False, e.g., because the downstream state for (*,G) has changed so no interface is in immediate_olist(*,G).

The upstream (*,G) state machine transitions to NotJoined state. Send Prune(*,G) to the appropriate upstream neighbor, which is RPF'(*,G). Cancel the Join Timer (JT).

Join Timer Expires

The Join Timer (JT) expires, indicating time to send a Join(*,G)

Send Join(*,G) to the appropriate upstream neighbor, which is RPF'(*,G).

Restart the Join Timer (JT) to expire after t_periodic seconds.

See Join(*,G) to RPF'(*,G)

This event is only relevant if RPF_interface(RP(G)) is a shared medium. This router sees another router on RPF_interface(RP(G)) send a Join(*,G) to RPF'(*,G). This causes this router to suppress its own Join.

The upstream (*,G) state machine remains in Joined state.

Let t_joinsuppress be the minimum of t_suppressed and the HoldTime from the Join/Prune message triggering this event. If the Join Timer is set to expire in less than t_joinsuppress seconds, reset it so that it expires after t_joinsuppress seconds. If the Join Timer is set to expire in more than t_joinsuppress seconds, leave it unchanged.

See Prune(*,G) to RPF'(*,G)

This event is only relevant if RPF_interface(RP(G)) is a shared medium. This router sees another router on RPF_interface(RP(G)) send a Prune(*,G) to RPF'(*,G). As this router is in Joined state, it must override the Prune after a short random interval.

The upstream (*,G) state machine remains in Joined state. If the Join Timer is set to expire in more than t_override seconds, reset it so that it expires after t_override seconds. If the Join Timer is set to expire in less than t_override seconds, leave it unchanged.

RPF'(*,G) changes due to an Assert

The current next hop towards the RP changes due to an Assert(*,G) on the RPF_interface(RP(G)).

The upstream (*,G) state machine remains in Joined state. If the Join Timer is set to expire in more than t_override seconds, reset it so that it expires after t_override seconds. If the Join Timer is set to expire in less than t_override seconds, leave it unchanged.

RPF'(*,G) changes not due to an Assert

An event occurred that caused the next hop towards the RP for G to change. This may be caused by a change in the MRIB routing database or the group-to-RP mapping. Note that this transition does not occur if an Assert is active and the upstream interface does not change.

The upstream (*,G) state machine remains in Joined state. Send Join(*,G) to the new upstream neighbor, which is the new value of RPF'(*,G). Send Prune(*,G) to the old upstream neighbor, which is the old value of RPF'(*,G). Use the new value of RP(G) in the Prune(*,G) message or all zeros if RP(G) becomes unknown (old value of RP(G) may be used instead to improve behavior in routers implementing older versions of this spec). Set the Join Timer (JT) to expire after t_periodic seconds.

RPF'(*,G) GenID changes

The Generation ID of the router that is RPF'(*,G) changes. This normally means that this neighbor has lost state, and so the state must be refreshed.

The upstream (*,G) state machine remains in Joined state. If the Join Timer is set to expire in more than t_override seconds, reset it so that it expires after t_override seconds.

4.5.7. Sending (S,G) Join/Prune Messages

The per-interface state machines for (S,G) hold join state from downstream PIM routers. This state then determines whether a router needs to propagate a Join(S,G) upstream towards the source.

If a router wishes to propagate a Join(S,G) upstream, it must also watch for messages on its upstream interface from other routers on that subnet, and these may modify its behavior. If it sees a Join(S,G) to the correct upstream neighbor, it should suppress its own Join(S,G). If it sees a Prune(S,G), Prune(S,G,rpt), or Prune(*,G) to the correct upstream neighbor towards S, it should be prepared to override that prune by scheduling a Join(S,G) to be sent almost immediately. Finally, if it sees the Generation ID of its upstream neighbor change, it knows that the upstream neighbor has lost state, and it should refresh the state by scheduling a Join(S,G) to be sent almost immediately.

If a (S,G) Assert occurs on the upstream interface, and this changes the this router's idea of the upstream neighbor, it should be prepared to ensure that the Assert winner is aware of downstream routers by scheduling a Join(S,G) to be sent almost immediately.

In addition, if MRIB changes cause the next hop towards the source to change, and either the upstream interface changes or there is no Assert winner on the upstream interface, the router should send a prune to the old next hop and a join to the new next hop.

The upstream (S,G) state machine only contains two states:

Not Joined

The downstream state machines and local membership information do not indicate that the router needs to join the shortest-path tree for this (S,G).

Joined

The downstream state machines and local membership information indicate that the router should join the shortest-path tree for this (S,G).

In addition, one timer JT(S,G) is kept that is used to trigger the sending of a Join(S,G) to the upstream next hop towards S, RPF'(S,G).

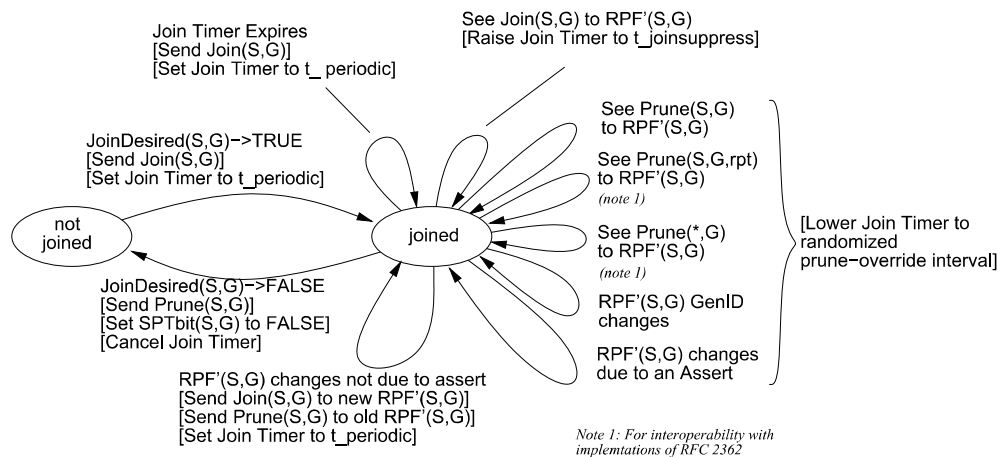


Figure 8: Upstream (S,G) state machine

Upstream (S,G) state machine in tabular form:

Prev State	Event	
	JoinDesired(S,G) →True	JoinDesired(S,G) →False
NotJoined (NJ)	→ J state Send Join(S,G); Set Join Timer to t_periodic	-
Joined (J)	-	→ NJ state Send Prune(S,G); Set SPTbit(S,G) to FALSE; Cancel Join Timer

In addition, we have the following transitions, which occur within the Joined state:

In Joined (J) State			
Timer Expires	See Join(S,G) to RPF'(S,G)	See Prune(S,G) to RPF'(S,G)	See Prune (S,G,rpt) to RPF'(S,G)
Send Join(S,G); Set Join Timer to t_periodic	Increase Join Timer to t_joinsuppress	Decrease Join Timer to t_override	Decrease Join Timer to t_override

In Joined (J) State			
See Prune(*,G) to RPF'(S,G)	RPF'(S,G) changes not due to an Assert	RPF'(S,G) GenID changes	RPF'(S,G) changes due to an Assert
Decrease Join Timer to t_override	Send Join(S,G) to new next hop; Send Prune(S,G) to old next hop; Set Join Timer to t_periodic	Decrease Join Timer to t_override	Decrease Join Timer to t_override

This state machine uses the following macro:

```
bool JoinDesired(S,G) {
    return( immediate_olist(S,G) ≠ NULL
           OR ( KeepaliveTimer(S,G) is running
               AND inherited_olist(S,G) ≠ NULL ) )
}
```

JoinDesired(S,G) is true when the router has forwarding state that would cause it to forward traffic for G using source tree state. The source tree state can be as a result of either active source-specific join state, or the (S,G) Keepalive Timer and active non-source-specific state. Note that although JoinDesired is true, the router's sending of a Join(S,G) message may be suppressed by another router sending a Join(S,G) onto the upstream interface.

Transitions from NotJoined State

When the upstream (S,G) state machine is in NotJoined state, the following event may trigger a state transition: