

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 17, 2012

Yiqun Cai
Sri Vallepalli
Heidi Ou
Cisco Systems, Inc.
Andy Green
British Telecom
October 15, 2011

Protocol Independent Multicast DR Load Balancing
draft-hou-pim-drlb-00.txt

Abstract

On a multi-access network such as an Ethernet, one of the PIM routers is elected as a Designated Routers (DR). The PIM DR has two roles in the PIM protocol. On the first hop network, the PIM DR is responsible for registering an active source to the RP if the group is operated in PIM SM. On the last hop network, the PIM DR is responsible for tracking local multicast listeners and forwarding traffic to these listeners if the group is operated in PIM SM/SSM/DM. In this document, we propose a modification to the PIM protocol that allows multiple of these last hop routers to be selected so that the forwarding load can be distributed to and handled among these routers. A router responsible for forwarding for a particular group is called a Group Designated Router (GDR).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 17, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Terminologies	3
2. Introduction	3
3. Applicability	6
4. Functional Overview	6
4.1. GDR Candidates	7
4.2. Hash Mask	7
4.3. PIM Hello Options	8
5. Packet Format	8
5.1. PIM DR Load Balancing GDR (LBGDR) Hello TLV	8
5.2. PIM DR Load Balancing Hash Masks (LBM) Hello TLV	9
6. Protocol Specification	9
6.1. PIM DR Operation	10
6.2. PIM GDR Candidate Operation	10
6.3. PIM Assert Modification	11
7. IANA Considerations	12
8. Security Considerations	12
9. Acknowledgement	12
10. References	12
10.1. Normative Reference	12
10.2. Informative References	12
Authors' Addresses	13

1. Terminologies

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

With respect to PIM, this document follows the terminology that has been defined in [RFC4601].

This document also introduces the following new acronyms:

- o GDR: GDR stands for "Group Designated Router". For each multicast group, a hash algorithm (described below) is used to select one of the routers as GDR. The GDR is responsible for initiating the forwarding tree building for the corresponding group.
- o GDR Candidate: a last hop router that has potential to become a GDR. A GDR Candidate must have the same DR priority as the DR router. It must send and process received new PIM Hello Options as defined in this document. There might be more than one GDR candidate on a LAN. But only one can become GDR for a specific multicast group.

2. Introduction

On a multi-access network such as an Ethernet, one of the PIM routers is elected as a Designated Routers (DR). The PIM DR has two roles in the PIM protocol. On the first hop network, the PIM DR is responsible for registering an active source to the RP if the group is operated in PIM SM. On the last hop network, the PIM DR is responsible for tracking local multicast listeners and forwarding to these listeners if the group is operated in PIM SM/SSM/DM.

Considering the following last hop network in Figure 1.

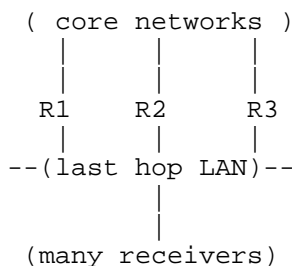


Figure 1: Last Hop Network

Assuming R1 is elected as the Designated Router. According to [RFC4601], R1 will be responsible for forwarding to the last hop LAN. In addition to keeping track of IGMP and MLD membership reports, R1 is also responsible for initiating the creation of source and/or shared trees towards the senders or the RPs.

Forcing sole data plane forwarding responsibility on the PIM DR proves a limitation in the protocol. In comparison, even though an OSPF DR, or an IS-IS DIS, handles additional duties while running the OSPF or IS-IS protocols, they are not required to be solely responsible for forwarding packets for the network. On the other hand, on a last hop LAN, only the PIM DR is asked to forward packets while the other routers handle only control traffic (and perhaps dropping packets due to RPF failures). The forwarding load of a last hop LAN is concentrated on a single router.

This leads to several issues. One of the issues is that the aggregated bandwidth will be limited to what R1 can handle towards this particular interface. These days, it is very common that the last hop LAN usually consists of switches that run IGMP/MLD or PIM snooping. This allows the forwarding of multicast packets to be restricted only to segments leading to receivers who have indicated their interest in multicast groups using either IGMP or MLD. The emergence of the switched Ethernet allows the aggregated bandwidth to exceed, some times by a large number, that of a single link. For example, let us modify Figure 1 and introduce an Ethernet switch in Figure 2.

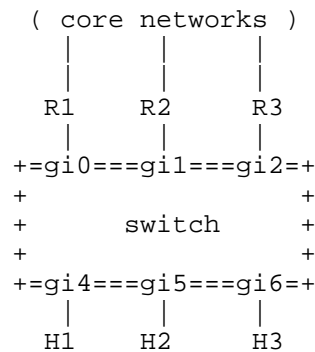


Figure 2: Last Hop Network with Ethernet Switch

Let us assume that each individual link is a Gigabit Ethernet. Each router, R1, R2 and R3, and the switch have enough forwarding capacity that can handle hundreds of Gigabits of data.

Let us further assume that each of the hosts requests 500 mbps of data and different traffic is requested by each host. This represents a total 1.5 gbps of data, which is under what each switch or the combined uplink bandwidth across the routers can handle, even under failure of a single router.

On the other hand, the link between R1 and switch, via port gi0, can only handle a throughput of 1gbps. And if R1 is the only router, the PIM DR elected using the procedure defined by RFC 4601, at least 500 mbps worth of data will be lost because the only link that can be used to draw the traffic from the routers to the switch is via gi0. In other words, the entire network's throughput is limited by the single connection between the PIM DR and the switch (or the last hop LAN as in Figure 1).

The problem may also manifest itself in a different way. For example, R1 happens to forward 500 mbps worth of unicast data to H1, and at the same time, H2 and H3 each requests 300 mbps of different multicast data. Once again packet drop happens on R1 while in the mean time, there is sufficient forwarding capacity left on R2 and R3 and link capacity between the switch and R2/R3.

Another important issue is related to failover. If R1 is the only forwarder on the last hop network, in the event of a failure when R1 goes out of service, multicast forwarding for the entire network has to be rebuilt by the newly elected PIM DR. However, if there was a way that allowed multiple routers to forward to the network for different groups, failure to one of the routers would only lead to

disruption to a subset of the flows, therefore improving the overall resilience of the network.

In this document, we propose a modification to the PIM protocol that allows multiple of these routers, called Group Designated Router (GDR) to be selected so that the forwarding load can be distributed to and handled by a number of routers.

3. Applicability

The proposed change described in this draft applies to PIM last hop routers only.

It doesn't alter the behavior of a PIM DR on the first hop network. This is because the source tree is built using the IP address of the sender, not the IP address of the PIM DR that sends the registers towards the RP. The load balancing between first hop routers can be achieved naturally if an IGP provides equal cost multiple paths (which it usually does in practice). And distributing the load to do registering doesn't justify the additional complexity required to support it.

4. Functional Overview

In existing PIM DR election, when multiple last hop routers are connected to a multi-access network (for example, an Ethernet), one of them is selected to act as PIM DR. The PIM DR is responsible for sending Join/Prune messages to the RP or source. To elect the PIM DR, each PIM router on the network examines the received PIM Hello messages and compares its DR priority and IP address with those of its neighbors. The router with the highest DR priority is the PIM DR. If there are multiple such routers, IP address is used as the tie breaker, as described in [RFC4601].

In order to share forwarding load among last hop routers, besides the normal PIM DR election, the GDR is also elected on the last hop multi-access network. There is only one PIM DR on the multi-access network, but there might be multiple GDR candidates.

For each multicast group, a hash algorithm is used to select one of the routers to be the GDR. Hash Masks are defined for Source, Group and RP separately, in order to handle different PIM modes. The masks are announced in PIM Hello as a new Load Balancing Hash Mask TLV (LBM TLV). Last hop routers with this new TLV and with the same DR priority as the PIM DR are GDR candidates.

A simple hash algorithm based on the announced Source, Group or RP masks allow one GDR to be assigned to a corresponding multicast group, and that GDR is responsible for initiating the creation of multicast forwarding tree for the group.

4.1. GDR Candidates

GDR is the new concept introduced by this draft. To become a candidate GDR, a router must have the same DR priority as the DR. For example, if there are 4 routers on the LAN: R1, R2, R3 and R4. R1, R2 and R3 have the same DR priority while R4's DR priority is less preferred. In this example, only R1, R2 and R3 will be eligible for GDR election. R4 is not because R4 will not become a PIM DR unless all of R1, R2 and R3 go out of service.

Further assuming router R1 wins the PIM DR election. In its Hello packet, R1 will include the identity of R1, R2 and R3 (the GDR candidates) besides its own Load Balancing Hash Mask TLV. The order of the GDR candidates is converted to the ordinal number associated with each GDR candidate. For example, addresses advertised by R1 is R1, R2, R3, the ordinal number assigned to R1 is 0, to R2 is 1 and to R3 is 2.

4.2. Hash Mask

A Hash Mask is used to extract a number of bits from the corresponding IP address field (32 for v4, 128 for v6), and calculate a hash value. A hash value is used to select GDR from GDR Candidates advertised in order by PIM DR. For example, 0.255.0.0 defines a Hash Mask for an IPv4 address that masks the first, the third and the fourth octets.

There are three Hash Masks defined,

- o RP Hash Mask
- o Source Hash Mask
- o Group Hash Mask

The Hash Masks must be configured on the PIM routers that can potentially become a PIM DR.

For ASM groups, a hash value is calculated using the following formula:

- o $\text{hashvalue_RP} = ((\text{RP_address} \& \text{RP_hashmask}) \gg N) \% M$

RP_address is the address of the RP defined for the group. N is the number of bits that are 0 from the right. M is the number of GDR

candidates as described above.

If RP_hashmask is 0, a hash value is also calculated using the group Hash Mask in a similar fashion

o $\text{hashvalue_Group} = ((\text{Group_address} \& \text{Group_hashmask}) \gg N) \% M$

For SSM groups, a hash value is calculated using both the source and group Hash Mask

o $\text{hashvalue_SG} = (((\text{Source_address} \& \text{Source_hashmask}) \gg N_S) \wedge ((\text{Group_address} \& \text{Group_hashmask}) \gg N_G)) \% M$

4.3. PIM Hello Options

When a non-DR PIM router that supports this draft sends a PIM Hello, it includes a new option, called "Load Balancing Hash Masks TLV (LBM TLV)". The LBM TLV consists of three Hash Masks as defined above.

Besides this new LBM TLV, the elected PIM DR router also includes a "Load Balancing GDR TLV (LBGDR TLV)" in its PIM Hello. The LBGDR TLV consists of the sorted addresses of all GDR candidates on the last hop network.

The elected PIM DR router uses LBM TLV to calculate its LBGDR TLV. The GDR candidates use LBM TLV and LBGDR TLV advertised by DR PIM router to calculate hash value.

5. Packet Format

5.1. PIM DR Load Balancing GDR (LBGDR) Hello TLV

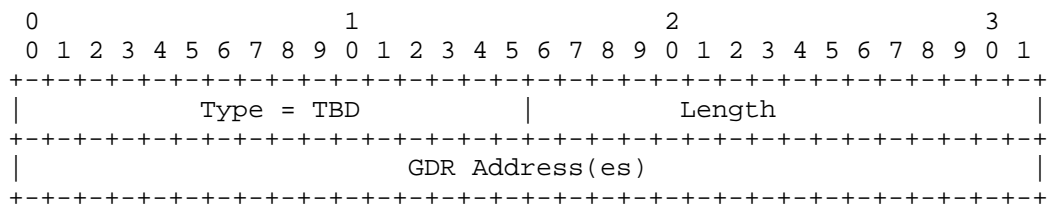


Figure 3: GDR Hello TLV

Type: TBD
Length:
GDR Address (32/128 bits): Address(es) of GDR candidates. All addresses must be in the same address family. The addresses are sorted from high to low. The order is used as the ordinal number, starting from 0, in hash value calculation.

This LBGDR TLV should only be advertised by the elected PIM DR router.

5.2. PIM DR Load Balancing Hash Masks (LBM) Hello TLV

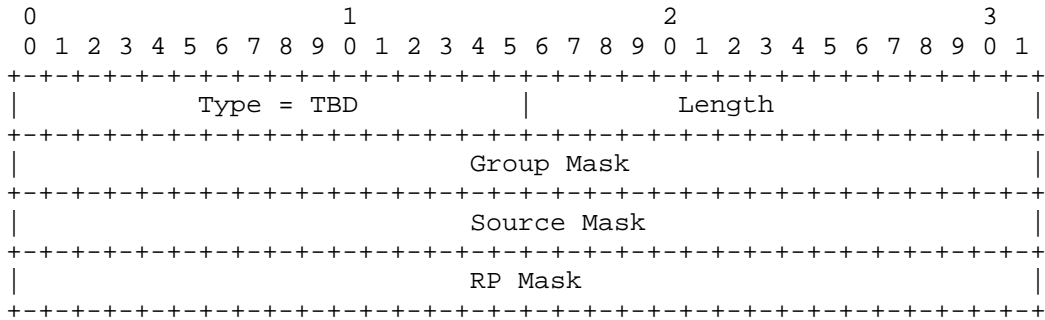


Figure 4: Hash Masks Hello TLV

Type: TBD.
Length:
Group Mask (32/128 bits): Mask
Source Mask (32/128 bits): Mask
RP Mask (32/128 bits): Mask

All masks must be in the same address family, with the same length.

This LBM TLV should be advertised by last hop routers, which support this draft.

6. Protocol Specification

6.1. PIM DR Operation

The DR elect process is still the same as defined in [RFC4601]. A DR supports this draft advertises a new Hello Option LBGRD TLV to includes all GDR candidates. Moreover, same as non-DR routers, DR also advertises LBM TLV Hello Option to indicate its capability of supporting this draft.

LBGRD TLV is composed by sorting the addresses of all GDR candidates. LBM TLV on PIM DR contains value of masks from user configuration.

If a PIM DR receives a neighbor Hello with LBGRD TLV, the PIM DR should ignore the TLV.

If a PIM DR receives a neighbor Hello with LBM TLV, and the neighbor has the same DR priority as PIM DR itself, the PIM DR should consider the neighbor as a GDR candidate and insert the neighbor's address into the sorted list of LBGRD TLV.

6.2. PIM GDR Candidate Operation

When an IGMP join is received, without this proposal, router R1 (the PIM DR) will handle the join and potentially run into the issues described earlier. Using this proposal, a simple algorithm is used to determine which router is going to be responsible for building forwarding trees on behalf of the host.

The algorithm works as follows, assuming the router in question is X and a GDR Candidate:

- o If the group is ASM, and if the RP Hash Mask announced by the PIM DR is not 0.0.0.0, calculate the value of hashvalue_RP. If hashvalue_RP is the same as the ordinal number assigned implicitly to X by PIM DR, X becomes the GDR.
- o If the group is ASM and if the RP Hash Mask announced by the PIM DR is 0, obtain the value of hashvalue_Group. And compare that to the ordinal value of X assigned by the PIM DR to decide if X is the GDR
- o If the group is SSM, then use hashvalue_SG to determine if X is the GDR.

If X is the GDR for the group, X will be responsible for building the forwarding tree.

A router that supports this draft advertises LBM TLV in its Hello, even the router may not be a GDR candidate.

A GDR candidate may receive a LBM TLV from PIM DR router, with a

different Hash Masks as advertised in its own Hello LBM TLV. The GDR candidate must use the Hash Masks advertised by the PIM DR Hello to calculate the hash value.

A GDR candidate may receive a LBGDR TLV from a non-DR PIM router. The GDR candidate must ignore such LBGDR TLV.

A GDR candidate may receive Hello from the elected PIM DR, and the PIM DR doesn't support this draft. The GDR election described by this draft will not take place, that is only the PIM DR joins the multicast tree.

6.3. PIM Assert Modification

When routers restart, GDR may change for a specific group, which might cause packet drops.

For example, if there are two streams G1 and G2, and R1 is the GDR for G1 and R2 is the GDR for G2. When R3 comes up online, it is possible that R2 becomes GDR for G1 and R3 becomes GDR for G2, and rebuilding of the forwarding trees for G1 and G2 will lead to potential packet loss.

This is not a typical deployment scenario but it still might happen. Here we describe a mechanism to minimize the impact.

When the role of GDR changes as above, instead of immediately stopping forwarding, R1 and R2 continue forwarding to G1 and G2 respectively, while in the same time, R2 and R3 build forwarding trees for G1 and G2 respectively. This will lead to PIM Asserts.

The same tie breakers are used to select an Assert winner with one modification. That is, instead of comparing IP addresses as the last resort, a router considers whether the sender of an Assert is a GDR. In this example, R1 will let R2 be the assert winner for G1, and R2 will do the same for R3 for G2. This will cause some duplicates in the network while minimizing packet loss.

If a router on the LAN doesn't support this draft, the Assert modification described above will not take place, that is only the IP address of an Assert sender is used as the tie breaker. For example, if R4, with preferred IP address, doesn't understand GDR and sends Assert for G1, R2, the GDR for G1, will grant R4 as the Assert winner, clear OIF on R2.

7. IANA Considerations

Two new PIM Hello Option Types are required to assign to the DR Load Balancing messages. According to [HELLO-OPT], this document recommends 32(0x20) as the new "PIM DR Load Balancing GDR Hello Option", and 33(0x21) as the new "PIM DR Load Balancing Hash Masks Hello Option" .

8. Security Considerations

Security of the PIM DR Load Balancing Hello message is only guaranteed by the security of PIM Hello packet, so the security considerations for PIM Hello packets as described in PIM-SM [RFC4601] apply here.

9. Acknowledgement

The authors would like to thank Steve Simlo, Taki Millonis for helping with the original idea, ??? for their review comments.

10. References

10.1. Normative Reference

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.

10.2. Informative References

- [RFC3973] Adams, A., Nicholas, J., and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)", RFC 3973, January 2005.
- [RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.
- [HELLO-OPT] IANA, "PIM Hello Options", PIM-HELLO-OPTIONS per RFC4601 <http://www.iana.org/assignments/pim-hello-options>, March 2007.

Authors' Addresses

Yiqun Cai
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: ycai@cisco.com

Sri Vallepalli
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: svallepa@cisco.com

Heidi Ou
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: hou@cisco.com

Andy Green
British Telecom
Adastral Park
Ipswich IP5 2RE
United Kingdom

Email: andy.da.green@bt.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2012

Yiqun Cai
Liming Wei
Heidi Ou
Cisco Systems, Inc.
Vishal Arya
Sunil Jethwani
DIRECTV Inc.
October 29, 2011

Protocol Independent Multicast ECMP Redirect
draft-ietf-pim-ecmp-01.txt

Abstract

A PIM router uses RPF procedure to select an upstream interface and router to build forwarding state. When there are equal cost multiple paths (ECMP), existing implementations often use hash algorithms to select a path. Such algorithms do not allow the spread of traffic among the ECMPs according to administrative metrics. This usually leads to inefficient or ineffective use of network resources. This document introduces the ECMP Redirect, a mechanism to improve the RPF procedure over ECMPs. It allows ECMP path selection to be based on administratively selected metrics, such as data transmission delays, path preferences and routing metrics.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 1, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements Notation	3
2. Introduction	3
2.1. Overview	3
2.2. Applicability	4
3. Protocol Specification	5
3.1. ECMP Bundle	5
3.2. Sending ECMP Redirect	5
3.3. Receiving ECMP Redirect	6
3.4. Transient State	6
3.5. Interoperability	7
3.6. Packet Format	7
3.6.1. PIM ECMP Redirect Hello Option	7
3.6.2. PIM ECMP Redirect Format	8
4. IANA Considerations	9
5. Security Considerations	9
6. Acknowledgement	9
7. References	10
7.1. Normative Reference	10
7.2. Informative References	10
Authors' Addresses	10

1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

A PIM [RFC4601] router uses RPF procedure to select an upstream interface and a PIM neighbor on that interface to build forwarding state. When there are equal cost multiple paths (ECMP) upstream, existing implementations often use hash algorithms to select a path. Such algorithms do not allow the spread of traffic among the ECMP according to administrative metrics. This usually leads to inefficient or ineffective use of network resources. This document introduces the ECMP Redirect, a mechanism to improve the RPF procedure over ECMP. It allows ECMP path selection to be based on administratively selected metrics, such as data transmission delays, path preferences and routing metrics, or a combination of metrics.

ECMPs are frequently used in networks to provide redundancy and to increase available bandwidth. A PIM router selects a path in the ECMP based on its own implementation specific choice. The selection is a local decision. One way is to choose the PIM neighbor with the highest IP address, another is to pick the PIM neighbor with the best hash value over the destination and source addresses.

While implementations supporting ECMP have been deployed widely, the existing RPF selection methods have weaknesses. The lack of administratively effective ways to allocate traffic over alternative paths is a major issue. For example, there is no straightforward way to tell two downstream routers to select either the same or different RPF neighbor routers for the same traffic flows.

With the ECMP Redirect mechanism introduced here, the upstream routers use a new PIM ECMP Redirect message to instruct the downstream routers on how to tie-break among the upstream neighbors. The PIM ECMP Redirect message conveys the tie-break information based on metrics selected administratively.

2.1. Overview

The existing PIM Assert mechanism allows the upstream router to detect the existence of multiple forwarders for the same multicast flow onto the same downstream interface. The upstream router sends a PIM Assert message containing a routing metric for the downstream routers to use for tie-breaking among the multiple upstream

forwarders on the same RPF interface.

With ECMP interfaces between the downstream and upstream routers, the PIM ECMP Redirect mechanism works in a similar way, but extends the ability to resolve the selection of forwarders among different interfaces in the ECMP.

When a PIM router downstream of the ECMP interfaces creates a new (*,G) or (S,G) entry, it will populate the RPF interface and RPF neighbor information according to the rules specified by [RFC4601]. This router will send its initial joins to that RPF neighbor.

When the RPF neighbor router receives the join message and finds that the receiving interface is one of the ECMP interfaces, it will check if the same flow is already being forwarded out of another ECMP interface. If so, this RPF neighbor router will send a PIM ECMP Redirect message onto the interface the join was received on. The PIM ECMP Redirect message contains the address of the desired RPF neighbor, an interface ID [RFC6395], along with other parameters used as tie breakers. In essence, a PIM ECMP Redirect message is sent by an upstream router to notify downstream routers to redirect PIM Joins to the new RPF neighbor via a different interface. When the downstream routers receive this message, they should trigger PIM Joins toward the new RPF neighbor specified in the packet.

This new PIM ECMP Redirect message has similar functions as existing PIM Assert message,

1. It is sent by an upstream router;
2. It is used to influence the RPF selection by downstream routers;
And
3. A tie breaker metric is used.

However, the existing Assert message is used to select an upstream router within the same multi-access network (such as a LAN) while the new Redirect message is used to select both a network and an upstream router.

One advantage of this design is that the control messages are only sent when there is need to "re-balance" the traffic. This reduces the amount of control traffic.

2.2. Applicability

The use of ECMP Redirect applies to shared trees or source trees built with procedures described in [RFC4601]. The use of ECMP Redirect in "Protocol Independent Multicast - Dense Mode" [RFC3973] or in "Bidirectional Protocol Independent Multicast" [RFC5015] is not

considered.

The enhancement described in this document can be applicable to a number of scenarios. For example, it allows a network operator to use ECMP paths and have the ability to perform load splitting based on bandwidth. To do this, the downstream routers perform RPF selection with bandwidth instead of IP addresses as a tie breaker. The ECMP Redirect mechanism assures that all downstream routers select the desired network link and upstream router whenever possible. Another example is for a network operator to impose a transmission delay limit on certain links. The ECMP Redirect mechanism provides a mean for an upstream router to instruct a downstream router to choose a different RPF path.

This specification does not dictate the scope of applications of this mechanism.

3. Protocol Specification

3.1. ECMP Bundle

An ECMP bundle is a set of PIM enabled interfaces on a router, where all interfaces belonging to the same bundle share the same routing metric. The ECMP paths reside between the upstream and downstream routers over the ECMP bundle.

There can be one or more ECMP bundles on any router, while one individual interface can only belong to a single bundle.

ECMP bundles are created on a router via configuration.

3.2. Sending ECMP Redirect

ECMP Redirects are sent by a preferred upstream router in a rate limited fashion, under the following conditions,

- o It detects a PIM Join on a non-desired outgoing interface; or
- o It detects multicast traffic on a non-desired outgoing interface.

In both cases, an ECMP Redirect is sent to the non-desired interface. An outgoing interface is considered "non-desired" when,

- o The upstream router is already forwarding the same flow out of another interface belonging to the same ECMP bundle;
- o The upstream router is not forwarding the flow yet out any interfaces of the ECMP bundle, but there is another interface with more desired attributes.

An upstream router may choose not to send ECMP Redirects if it becomes aware that some of the downstream routers do not support the new message, or unreachable via some links in ECMP bundle.

3.3. Receiving ECMP Redirect

When a downstream router receives an ECMP Redirect, and detects the desired RPF path from its upstream router's point of view is different from its current one, it should choose to prune from the current path and join to the new path. The exact order of such actions is implementation specific.

If a downstream router receives multiple ECMP Redirects sent by different upstream routers, it SHOULD use the Preference, Metric, or other fields as specified below, as the tie breakers to choose the most preferred RPF interface and neighbor.

If an upstream router receives an ECMP Redirect from another upstream router, it SHOULD NOT change its forwarding behavior even if the ECMP Redirect makes it a less preferred RPF neighbor on the receiving interface.

3.4. Transient State

During a transient network outage with a single link cut in an ECMP bundle, a downstream router may lose connection to its RPF neighbor and the normal ECMP Redirect operation may be interrupted temporarily. In such an event, the following actions are recommended.

The downstream router may re-select a new RPF neighbor. Among all ECMP upstream routers, the one on the same LAN as the previous RPF neighbor is preferred.

If there is no upstream router reachable on the same LAN, the downstream router will select a RPF neighbor on a different LAN. Among all ECMP upstream routers, the one served as RPF neighbor before the link failure is preferred. Such a router can be identified by the Router ID which is part of the Interface ID in the PIM ECMP Redirect Hello option.

During normal ECMP Redirect operations, when PIM Joins for the same (*,G) or (S,G) are received on a different LAN, an upstream router will send ECMP Redirect to prune the non-preferred LAN. Such ECMP Redirects during partial network outage can be suppressed if the upstream router decides that the non-preferred PIM Join is from a router that is not reachable via the preferred LAN. This check can be performed by retrieving the downstream's Router ID, using the

source address in the PIM join, and searching neighbors on the preferred LAN for one with the same router ID.

3.5. Interoperability

If a PIM router supports this draft, it MUST send the new Hello option ECMP-Redirect-Supported TLV in its PIM Hello messages. A PIM router sends ECMP Redirects on an interface only when it detects that all neighbors have sent this Hello option. If a PIM router detects that any of its neighbor does not support this Hello option, it MUST not send ECMP Redirects, however, it SHOULD still process any ECMP Redirects received.

3.6. Packet Format

3.6.1. PIM ECMP Redirect Hello Option

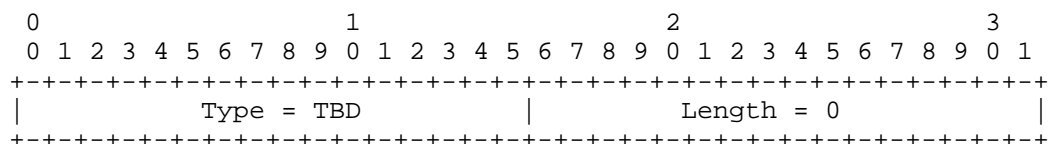


Figure 1: ECMP Redirect Hello Option

Type: TBD.
Length: 0

3.6.2. PIM ECMP Redirect Format

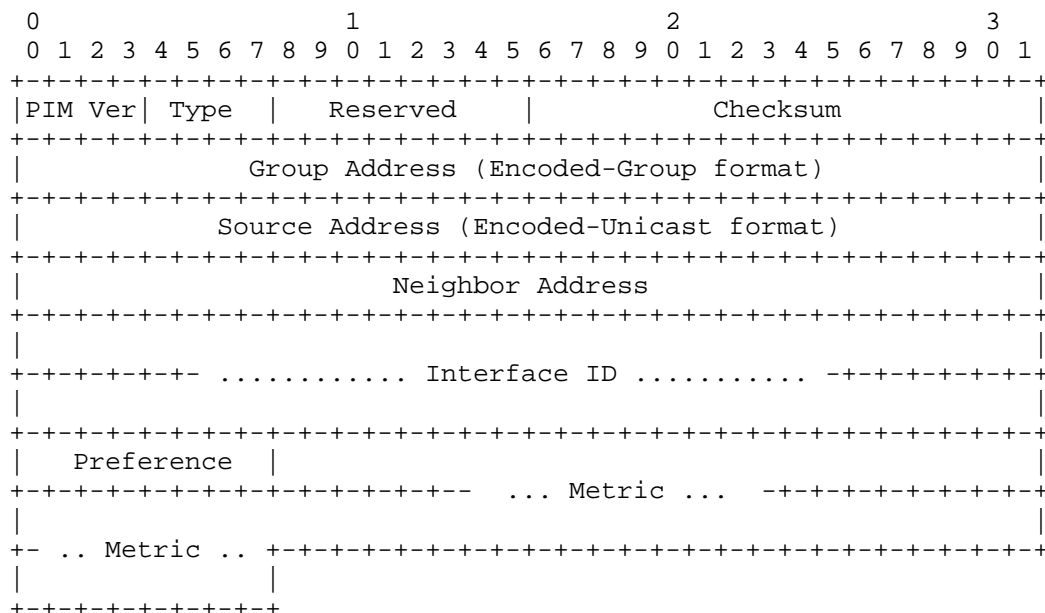


Figure 2: ECMP Redirect Message Format

Type: TBD

Neighbor Address (32/128 bits): Address of desired upstream neighbor where the downstream receiver should redirect PIM Joins to. This address MUST be associated with an interface in the same ECMP bundle as the ECMP Redirect message's outgoing interface. If the "Interface ID" field (see below) is ignored, this "Neighbor Address" field uniquely identifies a LAN and an upstream router to which a downstream router should redirect its Join messages to, and an ECMP Redirect message MUST be discarded if the "Neighbor Address" field in the message does not match cached neighbor address.

Interface ID (64 bits): This field is used in IPv4 when one or more RPF neighbors in the ECMP bundle are unnumbered, or in IPv6 where link local addresses are in use. For other IPv4 usage, this field is zero'ed when sent, and ignored when received. If the "Router ID" part of the "Interface ID" is zero, the field must be ignored. See [RFC6395] for details of its assignment and usage in PIM Hellos. If the "Interface ID" is not ignored, the receiving router of this message MUST use the "Interface ID", instead of

"Neighbor Address", to identify the new RPF neighbor, and an ECMP Redirect message MUST be discarded if the "Interface ID" field in the message does not match cached interface ID.

Preference (8 bits): The first tie breaker when ECMP Redirects from multiple upstream routers are compared against each other.

Numerically smaller value is preferred. A reserved value (15) is used to indicate the metric value following the "Preference" field is a timestamp, taken at the moment the sending router started to forward out of this interface.

Metric (64 bits): The second tie breaker if the the "Preference" values are the same. Numerically smaller metric is preferred. This "Metric" can contain path parameters defined by users. When both "Preference" and "Metric" values are the same, "Neighbor Address" or "Interface ID" field is used as the third tie-breaker, depends on which field is used to identify the RPF neighbor, and the bigger value wins.

4. IANA Considerations

A new PIM Hello Option type is requested to assign to the PIM ECMP Redirect Hello Option. According to [HELLO-OPT], this document recommends 32 (0x20) as the new "PIM ECMP Redirect Hello Option Type".

A new PIM Type is requested to assign to the ECMP Redirect messages. According to [RFC6166], this document recommends 11 (0xB) as the new "PIM ECMP Redirect Type".

5. Security Considerations

Security of the ECMP Redirect is only guaranteed by the security of the PIM packet, the security considerations for PIM Assert packets as described in [RFC4601] apply here. Spoofed ECMP Redirect packets may cause the downstream routers to send PIM Joins to an undesired upstream router, and trigger more ECMP Redirect messages.

6. Acknowledgement

The authors would like to thank Apoorva Karan for helping with the original idea, Eric Rosen, Isidor Kouvelas, Toerless Eckert, Stig Venaas and Jeffrey Zhang for their review comments.

7. References

7.1. Normative Reference

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.

7.2. Informative References

- [RFC3973] Adams, A., Nicholas, J., and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)", RFC 3973, January 2005.
- [RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.
- [RFC6166] Venaas, S., "A Registry for PIM Message Types", RFC 6166, April 2011.
- [RFC6395] Gulrajani, S. and S. Venaas, "An Interface Identifier (ID) Hello Option for PIM", RFC 6395, October 2011.
- [HELLO-OPT] IANA, "PIM Hello Options", PIM-HELLO-OPTIONS per RFC4601 <http://www.iana.org/assignments/pim-hello-options>, October 2011.

Authors' Addresses

Yiqun Cai
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: ycai@cisco.com

Liming Wei
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: lwei@cisco.com

Heidi Ou
Cisco Systems, Inc.
Tasman Drive
San Jose, CA 95134
USA

Email: hou@cisco.com

Vishal Arya
DIRECTV Inc.
2230 E Imperial Hwy
El Segundo, CA 90245
USA

Email: varya@directv.com

Sunil Jethwani
DIRECTV Inc.
2230 E Imperial Hwy
El Segundo, CA 90245
USA

Email: sjethwani@directv.com

PIM Working Group
Internet-Draft
Intended status: Informational
Expires: April 12, 2012

H. Asaeda
Keio University
N. Leymann
Deutsche Telekom AG
October 10, 2011

IGMP/MLD-Based Explicit Membership Tracking Function for Multicast
Routers
draft-ietf-pim-explicit-tracking-00

Abstract

This document describes the IGMP/MLD-based explicit membership tracking function for multicast routers. The explicit tracking function is useful for accounting and contributes to saving network resource and fast leaves (i.e. shortened leave latency).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 12, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Terminology	5
3. Explicit Tracking Function	6
3.1. Reducing the Number of Specific Queries	6
3.2. Shortening Leave Latencies	6
3.3. Considerations	7
4. Membership State Information	9
5. Multicast Router Behavior	10
6. Interoperability and Compatibility	11
7. Security Considerations	12
8. Acknowledgements	13
9. References	14
9.1. Normative References	14
9.2. Informative References	14
Authors' Addresses	15

1. Introduction

The Internet Group Management Protocol (IGMP) [2] for IPv4 and the Multicast Listener Discovery Protocol (MLD) [3] for IPv6 are the standard protocols used by listener hosts and multicast routers. When a host starts listening particular multicast channels, it sends IGMP/MLD State-Change Report messages specifying the corresponding channel information as the join/leave request to its upstream router (i.e., an adjacent multicast router or IGMP/MLD proxy [8]). This "unsolicited" Report is sent only once upon reception.

IGMP/MLD are non-reliable protocols; the unsolicited Report messages may be lost or not be reached to upstream routers. To recover the problem, the routers need to update membership information by sending IGMP/MLD General Query messages periodically. Member hosts then reply with "solicited" Report messages whenever they receive the Query messages.

Multicast routers are able to periodically maintain the multicast listener (or membership) state of downstream hosts attached on the same link by getting unsolicited Report messages and synchronize the actual membership state within the General Query timer interval (i.e., [Query Interval] value defined in [2][3].) However, this approach does not guarantee that the membership state is always perfectly synchronized. To minimize the possibility of having the outdated membership information, routers may shorten the periodic General Query timer interval. Unfortunately, this would increase the number of transmitted solicited Report messages and induce network congestion. And the more the network congestion is occurred, the more IGMP/MLD Report messages may be lost and the membership state information may be outdated in the router.

The IGMPv3 [2] and MLDv2 [3] protocols can provide the capability of keeping track of downstream (adjacent) multicast listener state to multicast routers. This document describes the "IGMP/MLD-based explicit member tracking function" for multicast routers and details the way for routers to implement the function. By enabling the explicit tracking function, routers can keep track of the downstream multicast membership state. This function implements the following requirements:

- o Per-host accounting
- o Reducing the number of transmitted Query and Report messages
- o Shortening leave latencies

- o Maintaining multicast channel characteristics (or statistics)

where this document mainly focuses on the above second and third bullets in the following sections.

The explicit tracking function does not change message formats used by the standard IGMPv3 [2] and MLDv2 [3], and their lightweight version protocols [4]. It does not change a multicast data sender's and receiver's behavior as well.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

3. Explicit Tracking Function

3.1. Reducing the Number of Specific Queries

The explicit tracking function reduces the number of Group-Specific or Group-and-Source Specific Query messages transmitted from a router, and then the number of Current-State Report messages transmitted from member hosts. As the result, network resources used for IGMP/MLD query-and-reply communications between a router and member hosts can be saved.

According to [2] and [3], whenever a router receives the State-Change Report, it sends the corresponding Group-Specific or Group-and-Source Specific Query messages to confirm whether the Report sender is the last member host or not. After getting these Query messages, all member hosts joining the corresponding channel reply with own Current-State Report messages. This condition requires transmitting a number of Current-State Report messages and consumes network resources especially when many hosts have been joining the same channel.

On the other hand, if a router enables the explicit tracking function, it does not need to always ask Current-State Report message transmission to the member hosts whenever it receives the State-Change Report. This is because the explicit tracking function works with the expectation that the State-Change Report sender is the last remaining member of the channel. Even if this expectation is wrong (i.e., the State-Change Report sender was not the sole member), other members remaining in the same channel will reply with identical Report messages, so the end result is the same and no problem occurs. (Section 4 details the point.)

In addition, the processing of IGMP membership or MLD listener reports consumes CPU resources on the IGMP/MLD querier devices itself. Therefore, the explicit tracking function reduces not only the network load but also the CPU load on the querier devices as well.

3.2. Shortening Leave Latencies

The explicit tracking function works with the expectation that the State-Change Report sender is the last remaining member of the channel. Thanks to this functionality, a router can tune timers and values related to decide that the State-Change Report sender was the sole member.

The [Last Member Query Interval] (LMQI) and [Last Listener Query Interval] (LLQI) values specify the maximum time allowed before

sending a responding Report. The [Last Member Query Count] (LMQC) and [Last Listener Query Count] (LLQC) are the number of Group-Specific Queries or Group-and-Source Specific Queries sent before the router assumes there are no local members. The [Last Member Query Time] (LMQT) and [Last Listener Query Time] (LLQT) values are the total time the router should wait for a report, after the Querier has sent the first query.

The default values for LMQI/LLQI defined in the standard specifications [2][3] are 1 second. For the router enabling the explicit tracking function, LMQI/LLQI would be set to 1 second or shorter. The LMQC/LLQC may be set to "1" for the router, whereas their default values are the [Robustness Variable] value whose default value is "2". Smaller LMQC/LLQC give smaller LMQT/LLQT; this condition shortens the leave latencies.

3.3. Considerations

As with the basic concepts of IGMP and MLD, the explicit tracking function does not guarantee the membership state is always perfectly synchronized; routers enabling the explicit tracking function still need to send IGMPv3/MLDv2 Query messages and inquire solicited IGMPv3/MLDv2 Report messages from downstream members to maintain downstream membership state.

- o IGMP/MLD messages are non-reliable and may be lost in the transmission, therefore routers need to confirm the membership by sending Query messages.
- o To preserve compatibility with older versions of IGMP/MLD, routers need to support downstream hosts that are not upgraded to the latest versions of IGMP/MLD and run the report suppression mechanism.
- o It is impossible to identify hosts when hosts send IGMP reports with a source address of 0.0.0.0.

Regarding the last bullet, the IGMPv3 specification [2] mentions that an IGMPv3 Report is usually sent with a valid IP source address, although it permits that a host uses the 0.0.0.0 source address (as it happens that the host has not yet acquired an IP address), and routers MUST accept a report with a source address of 0.0.0.0. The MLDv2 specification [3] mentions that an MLDv2 Report MUST be sent with a valid IPv6 link-local source address, although an MLDv2 Report can be sent with the unspecified address (::), if the sending interface has not acquired a valid link-local address yet. [3] also mentions that routers silently discard a message that is not sent with a valid link-local address or sent with the unspecified address,

without taking any action, because of the security consideration.

Another concern is that the explicit tracking function requires additional processing capability and a possibly large memory for routers to keep all membership states. Especially when a router needs to maintain a large number of member hosts, this resource requirement may be potentially-impacted. Operators may decide to disable this function when their routers do not have enough memory resources.

4. Membership State Information

The explicit tracking function is implemented with the following membership state information:

(S, G, number of receivers, (receiver records))

where each receiver record is of the form:

(IGMP/MLD Membership/Listener Report sender's address)

This state information must work properly when a receiver (i.e., Report sender) sends the same Report messages multiple times.

In the state information, each "S" and "G" indicates a single IPv4/IPv6 address. "S" is set to "Null" for an Any-Source Multicast (ASM) communication (i.e., (*,G) join reception). In order to simplify the implementation, the explicit tracking function does not keep the state of (S,G) join with EXCLUDE filter mode. If a router receives (S,G) join/leave request with EXCLUDE filter mode from the downstream hosts, it translates the join/leave request to (*,G) join state/leave request and records the state and the receivers' addresses into the maintained membership state information. Note that this membership state translation does not change the routing protocol behavior; the routing protocol must deal with the original join/leave request and translate the request only for the membership state information.

5. Multicast Router Behavior

The explicit tracking function makes routers expect whether the State-Change Report sender is the last remaining member of the channel. Therefore the router transmits a corresponding Current-State Report message only when the router thinks that the State-Change Report sender is the last remaining member of the channel. This contributes to saving the network resources and also shortening leave latency.

To synchronize the membership state information, when a multicast router receives a Current-State or State-Change Report message, it adds the receiver IP address to or delete from the receiver records or creates the corresponding membership state information. If there are no more receiver records left, the membership state information is deleted from the router.

However, the membership state information may be still outdated in the router. It may be happened especially in a mobile multicast environment that some member hosts have joined to or left from the network without sending State-Change Report messages. Or, some State-Change Report messages are lost due to network congestion. Therefore, the router enabling the explicit tracking function MUST send the periodic General Query regularly.

Regarding the leave latency, as specified in Section 3.2, the explicit tracking function contributes to the fast leave by setting LMQI/LLQI to "1" second or shorter and LMQC/LLQC to "1". However, if LMQC/LLQC is configured "2" or bigger value, then the router's behavior MAY be changed from the standard specification. According to [2] and [3], a router sends a Group- (and-Source) Specific Query [LMQC - 1] or [LLQC - 1] times when it receives State Change Report message (e.g. leave request) from a member host, in order to confirm whether or not the host is the only remaining member. However, this document RECOMMENDS that if the router enabling the explicit tracking function receives the corresponding Current State Report before the Specific Query retransmission, it cancels sending the same Specific Query for other [LMQC - 1] or [LLQC - 1] times.

Note that there is some risk that a router misses or loses Report messages sent from remaining members if the router adopts small LMQC/LLQC; however the wrong expectation would be lower happened for the router enabling the explicit tracking function. And to avoid the problem, a router can start sending a Group- (and-Source) Specific Query message when it expects the number of the remaining members is small, such as 5, but not 0.

6. Interoperability and Compatibility

The explicit tracking function does not work with the older versions of IGMP or MLD, IGMPv1 [5], IGMPv2 [6] or MLDv1 [7], because a member host using these protocols adopts a report suppression mechanism by which a host would cancel sending a pending membership Reports if a similar Report was observed from another member on the network.

If a multicast router enabling the explicit tracking function changes its compatibility mode to the older versions of IGMP or MLD, the router should turn off the explicit tracking function but should not flush the maintained membership state information (i.e., keep the current membership state information as is). When the router changes back to IGMPv3 or MLDv2 mode, it would resume the function with the kept membership state information, even if the state information is outdated. This manner would give "smooth state transition" that does not initiate the membership state from scratch and synchronizes the actual membership state smoothly.

There are several points TBD in the further discussions regarding the interoperability and compatibility issues. At first, it is necessary whether a multicast router enabling the explicit tracking function needs to detect adjacent routers that do not support the explicit tracking function on the link or not. After the clarification, this document will describe the method how to detect them. It would be done by a new signaling message, but the new message leads compatibility problems for older routers or other routing protocols such as PIM-DM. All of these discussions are TBD.

7. Security Considerations

There is no additional security considerations.

8. Acknowledgements

Toerless Eckert, Stig Venaas, and others provided many constructive and insightful comments.

9. References

9.1. Normative References

- [1] Bradner, S., "Key words for use in RFCs to indicate requirement levels", RFC 2119, March 1997.
- [2] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [3] Vida, R. and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, June 2004.
- [4] Liu, H., Cao, W., and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols", RFC 5790, February 2010.

9.2. Informative References

- [5] Deering, S., "Host Extensions for IP Multicasting", RFC 1112, August 1989.
- [6] Fenner, W., "Internet Group Management Protocol, Version 2", RFC 2373, July 1997.
- [7] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.
- [8] Fenner, B., He, H., Haberman, B., and H. Sandick, "Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying")", RFC 4605, August 2006.

Authors' Addresses

Hitoshi Asaeda
Keio University
Graduate School of Media and Governance
5322 Endo
Fujisawa, Kanagawa 252-0882
Japan

Email: asaeda@wide.ad.jp
URI: <http://web.sfc.wide.ad.jp/~asaeda/>

Nicolai Leymann
Deutsche Telekom AG
Winterfeldtstrasse 21-27
Berlin 10781
Germany

Email: n.leymann@telekom.de

Network Working Group
Internet Draft
Intended Status: Draft Standard
Expires: April 29, 2012

B. Fenner
AT&T Labs - Research
M. Handley
UCL
H. Holbrook
Arastra
I. Kouvelas
R. Parekh
Cisco Systems, Inc.
Z. Zhang
Juniper Networks
L. Zheng
Huawei Technologies
October 27, 2011

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

draft-ietf-pim-rfc4601bis-01

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 29, 2012.

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 addresses errata filed against RFC 4601, and removes

the optional (*,*,RP) feature that lacks sufficient deployment experience.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	6
2.	Terminology	6
2.1.	Definitions	6
2.2.	Pseudocode Notation	7
3.	PIM-SM Protocol Overview	8
3.1.	Phase One: RP Tree	8
3.2.	Phase Two: Register-Stop	9
3.3.	Phase Three: Shortest-Path Tree	10
3.4.	Source-Specific Joins	11
3.5.	Source-Specific Prunes	11
3.6.	Multi-Access Transit LANS	12
3.7.	RP Discovery	12
4.	Protocol Specification	13
4.1.	PIM Protocol State	14
4.1.1.	General Purpose State	15
4.1.2.	(*,G) State	16
4.1.3.	(S,G) State	17
4.1.4.	(S,G,rpt) State	20
4.1.5.	State Summarization Macros	21
4.2.	Data Packet Forwarding Rules	25
4.2.1.	Last-Hop Switchover to the SPT	27
4.2.2.	Setting and Clearing the (S,G) SPTbit	28
4.3.	Designated Routers (DR) and Hello Messages	29
4.3.1.	Sending Hello Messages	29
4.3.2.	DR Election	31
4.3.3.	Reducing Prune Propagation Delay on LANS	33
4.3.4.	Maintaining Secondary Address Lists	36
4.4.	PIM Register Messages	37
4.4.1.	Sending Register Messages from the DR	37
4.4.2.	Receiving Register Messages at the RP	41
4.5.	PIM Join/Prune Messages	43
4.5.1.	Receiving (*,G) Join/Prune Messages	43
4.5.2.	Receiving (S,G) Join/Prune Messages	48
4.5.3.	Receiving (S,G,rpt) Join/Prune Messages	51
4.5.4.	Sending (*,G) Join/Prune Messages	56
4.5.5.	Sending (S,G) Join/Prune Messages	61
4.5.6.	(S,G,rpt) Periodic Messages	66
4.5.7.	State Machine for (S,G,rpt) Triggered Messages	67
4.6.	PIM Assert Messages	71
4.6.1.	(S,G) Assert Message State Machine	72
4.6.2.	(*,G) Assert Message State Machine	79
4.6.3.	Assert Metrics	85
4.6.4.	AssertCancel Messages	87
4.6.5.	Assert State Macros	87
4.7.	PIM Bootstrap and RP Discovery	91
4.7.1.	Group-to-RP Mapping	92

4.7.2.	Hash Function	93
4.8.	Source-Specific Multicast	94
4.8.1.	Protocol Modifications for SSM Destination Addresses	94
4.8.2.	PIM-SSM-Only Routers	95
4.9.	PIM Packet Formats	96
4.9.1.	Encoded Source and Group Address Formats	98
4.9.2.	Hello Message Format	101
4.9.3.	Register Message Format	105
4.9.4.	Register-Stop Message Format	108
4.9.5.	Join/Prune Message Format	108
4.9.5.1.	Group Set Source List Rules	111
4.9.5.2.	Group Set Fragmentation	114
4.9.6.	Assert Message Format	115
4.10.	PIM Timers	116
4.11.	Timer Values	118
5.	IANA Considerations	123
5.1.	PIM Address Family	123
5.2.	PIM Hello Options	124
6.	Security Considerations	124
6.1.	Attacks Based on Forged Messages	124
6.1.1.	Forged Link-Local Messages	124
6.1.2.	Forged Unicast Messages	125
6.2.	Non-Cryptographic Authentication Mechanisms	125
6.3.	Authentication Using IPsec	126
6.3.1.	Protecting Link-Local Multicast Messages	126
6.3.2.	Protecting Unicast Messages	127
6.3.2.1.	Register Messages	127
6.3.2.2.	Register-Stop Messages	127
6.4.	Denial-of-Service Attacks	128
7.	Acknowledgements	128
8.	Normative References	129
9.	Informative References	129
	Authors' Addresses	131

List of Figures

Figure 1. Per-(S,G) register state machine at a DR	38
Figure 2. Downstream per-interface (*,G) state machine	45
Figure 3. Downstream per-interface (S,G) state machine	49
Figure 4. Downstream per-interface (S,G,rpt) state machine	53
Figure 5. Upstream (*,G) state machine	58
Figure 6. Upstream (S,G) state machine	62
Figure 7. Upstream (S,G,rpt) state machine for triggered messages	67
Figure 8. Per-interface (S,G) Assert State machine	72
Figure 9. Per-interface (*,G) Assert State machine	80

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 specified in RFC 4601 as a Proposed Standard. This document is intended to address the reported errata and to remove the optional (*,*,RP) feature that lacks sufficient deployment experience, to advance PIM-SM to Draft Standard.

This document specifies the same protocol as RFC 4601 and implementations per the specification in this document will be able to interoperate successfully with implementations per RFC 4601.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

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 the 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.

2.2. Pseudocode Notation

We use set notation in several places in this specification.

$A (+) B$ is the union of two sets, A and B.

$A (-) B$ 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.

Unless otherwise noted, operations specified by statements having multiple (+) and (-) operators should be evaluated from left to right, i.e. A (+) B (-) C is the set resulting from union of sets A and B minus elements in set C.

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:

- o 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.
- o 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:

- o 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.
- o 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.
- o 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 for 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:

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

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 three sections:

(*,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:

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

Neighbor State:

For each neighbor:

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

Designated Router (DR) State:

- o Designated Router's IP Address
- o 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. (*,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:

o State: One of {"NoInfo" (NI), "Join" (J), "Prune-Pending" (PP)}

o Prune-Pending Timer (PPT)

o Join/Prune Expiry Timer (ET)

(*,G) Assert Winner State

o State: One of {"NoInfo" (NI), "I lost Assert" (L), "I won Assert" (W)}

o Assert Timer (AT)

o Assert winner's IP Address (AssertWinner)

o Assert winner's Assert Metric (AssertWinnerMetric)

Not interface specific:

Upstream (*,G) Join/Prune State:

o State: One of {"NotJoined(*,G)", "Joined(*,G)"}

o Upstream Join/Prune Timer (JT)

o Last RP Used

o 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. It is RECOMMENDED to store 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 4.1.5.

PIM (*,G) Join/Prune state is the result of receiving PIM (*,G) Join/Prune messages on this interface and is specified in Section 4.5.1. The state is used by the macros that calculate the outgoing interface list in Section 4.1.5, and in the `JoinDesired(*,G)` macro (defined in Section 4.5.4) 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 4.5.4.

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 4.5.4.

4.1.3. (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:

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

(S,G) Assert Winner State

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

Not interface specific:

Upstream (S,G) Join/Prune State:

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

Additional (S,G) state at the DR:

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

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 an (S,G) assert on this interface for this group. However, it is RECOMMENDED to store 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 4.1.5.

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 4.1.5, and in the `JoinDesired(S,G)` macro (defined in Section 4.5.5) 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 4.5.5.

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 4.5.5.

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).

4.1.4. (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:

- o State: One of {"NoInfo", "Pruned", "Prune-Pending"}

- o Prune-Pending Timer (PPT)

- o Join/Prune Expiry Timer (ET)

Not interface specific:

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

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

- o 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 4.1.5.

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 4.5.3. The state is used by the macros that calculate the outgoing interface list in Section 4.1.5, and in the rules for adding Prune(S,G,rpt) messages to Join(*,G) messages specified in Section 4.5.6.

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 4.5.7.

4.1.5. 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 `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(*,G) =  
    joins(*,G) (+) pim_include(*,G) (-) lost_assert(*,G)  
  
immediate_olist(S,G) =  
    joins(S,G) (+) pim_include(S,G) (-) lost_assert(S,G)
```

```

inherited_olist(S,G,rpt) =
    ( joins(*,G) (-) prunes(S,G,rpt) )
    (+) ( pim_include(*,G) (-) pim_exclude(S,G) )
    (-) ( lost_assert(*,G) (+) lost_assert(S,G,rpt) )

```

```

inherited_olist(S,G) =
    inherited_olist(S,G,rpt) (+)
    joins(S,G) (+) pim_include(S,G) (-) 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(*,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 4.5.1.

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 4.5.2.

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 4.5.3.

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 4.5.6).

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.

`iif` 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 the `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 `(*,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 (*,G) state, asserts, etc.

`inherited_olist(S,G,rpt)` is the outgoing interface list for packets forwarded on (*,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 4.5.5.

`Keepalive_Period` is defined in Section 4.10.

Data-triggered PIM-Assert messages sent from the above forwarding code SHOULD be rate-limited in an 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) (-) pim_exclude(S,G)
          (+) pim_include(S,G) != 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 (*,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 (*,G) state while upstream (S,G) state is being established, during which time a router should continue to forward only on (*,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) != RPF_interface(RP(G))
              OR inheritedolist(S,G,rpt) == NULL
              OR ( ( RPF'(S,G) == RPF'(*,G) ) AND
                  ( RPF'(S,G) != 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 4.5.5 and indicates whether we have the appropriate (S,G) Join state to wish to send a Join(S,G) upstream.

Basically, Update_SPTbit(S,G,iif) 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. No One 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. $RPF'(S,G) != \text{NULL}$ check ensures that SPTbit is set only if RPF neighbor towards S is valid.

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 4.5.5) 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 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 * Register_Suppression_Time, 1.5 * Register_Suppression_Time) minus Register_Probe_Time.

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

(**) The Register-Stop Timer is set to 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). DownstreamJPState(S,G,VI) is set to Join state, causing the tunnel interface to be added to immediate_olist(S,G) and inherited_olist(S,G).

Remove Register Tunnel

VI is the Register-Tunnel virtual interface with encapsulation target of RP(G). DownstreamJPState(S,G,VI) is set to NoInfo state, causing the tunnel interface to be removed from immediate_olist(S,G) and inherited_olist(S,G). If 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 ( SPTbit(S,G) OR
            ( SwitchToSptDesired(S,G) AND
              ( inheritedolist(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
            inheritedolist(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 SHOULD NOT 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 inheritedolist(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 (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 interoperability with such older routers, but that this configuration option SHOULD NOT be enabled by default.

4.5.1. 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 4.5.3) 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 2: 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 it is RECOMMENDED that for backwards compatibility PIM Join/Prune messages with an upstream neighbor address field of all zeros also be 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.

4.5.2. 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 3: 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 it is RECOMMENDED that for backwards compatibility PIM Join/Prune messages with an upstream neighbor address field of all zeros also be 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.

4.5.3. 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 an (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 4: 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 it is RECOMMENDED that PIM Join/Prune messages with an upstream neighbor address field of all zeros also be 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) that 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 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.

4.5.4. 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 5: 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)  
        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.

4.5.5. 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 an (S,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 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 6: 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:

JoinDesired(S,G) becomes True

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

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

Transitions from Joined State

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

JoinDesired(S,G) becomes False

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

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

Join Timer Expires

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

Send Join(S,G) to the appropriate upstream neighbor, which is RPF'(S,G). Restart the Join Timer (JT) to expire after t_periodic seconds.

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

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

The upstream (S,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(S,G) to RPF'(S,G)

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

The upstream (S,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.

See Prune(S,G,rpt) to RPF'(S,G)

This event is only relevant if RPF_interface(S) is a shared medium. This router sees another router on RPF_interface(S) send a Prune(S,G,rpt) to RPF'(S,G). If the upstream router is an RFC-2362-compliant PIM router, then the Prune(S,G,rpt) will cause it to stop forwarding. For backwards compatibility, this router should override the prune so that forwarding continues.

The upstream (S,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.

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

This event is only relevant if RPF_interface(S) is a shared medium. This router sees another router on RPF_interface(S) send a Prune(*,G) to RPF'(S,G). If the upstream router is an RFC-2362-compliant PIM router, then the Prune(*,G) will cause it to stop forwarding. For backwards compatibility, this router should override the prune so that forwarding continues.

The upstream (S,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.

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

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

The upstream (S,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'(S,G) changes not due to an Assert

An event occurred that caused the next hop towards S to change. Note that this transition does not occur if an Assert is active and the upstream interface does not change.

The upstream (S,G) state machine remains in Joined state. Send Join(S,G) to the new upstream neighbor, which is the new value of RPF'(S,G). Send Prune(S,G) to the old upstream neighbor, which is the old value of RPF'(S,G). Set the Join Timer (JT) to expire after t_periodic seconds.

RPF'(S,G) GenID changes

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

The upstream (S,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.

4.5.6. (S,G,rpt) Periodic Messages

(S,G,rpt) Joins and Prunes are (S,G) Joins or Prunes sent on the RP tree with the RPT bit set, either to modify the results of (*,G) Joins, or to override the behavior of other upstream LAN peers. The next section describes the rules for sending triggered messages. This section describes the rules for including a Prune(S,G,rpt) message with a Join(*,G).

When a router is going to send a Join(*,G), it should use the following pseudocode, for each (S,G) for which it has state, to decide whether to include a Prune(S,G,rpt) in the compound Join/Prune message:

```
if( SPTbit(S,G) == TRUE ) {
    # Note: If receiving (S,G) on the SPT, we only prune off the
    # shared tree if the RPF neighbors differ.
    if( RPF'(*,G) != RPF'(S,G) ) {
        add Prune(S,G,rpt) to compound message
    }
} else if ( inherited_olist(S,G,rpt) == NULL ) {
    # Note: all (*,G) olist interfaces received RPT prunes for (S,G).
    add Prune(S,G,rpt) to compound message
} else if ( RPF'(*,G) != RPF'(S,G,rpt) ) {
    # Note: we joined the shared tree, but there was an (S,G) assert
    # and the source tree RPF neighbor is different.
    add Prune(S,G,rpt) to compound message
}
```

Note that Join(S,G,rpt) is normally sent not as a periodic message, but only as a triggered message.

4.5.7. State Machine for (S,G,rpt) Triggered Messages

The state machine for (S,G,rpt) triggered messages is required per-(S,G) when there is (*,G) join state at a router, and the router or any of its upstream LAN peers wishes to prune S off the RP tree.

There are three states in the state machine. One of the states is when there is no (*,G) join state at this router. If there is (*,G) join state at the router, then the state machine must be at one of the other two states. The three states are:

Pruned(S,G,rpt)
(*,G) Joined, but (S,G,rpt) pruned

NotPruned(S,G,rpt)
(*,G) Joined, and (S,G,rpt) not pruned

RPTNotJoined(G)
(*,G) has not been joined.

In addition, there is an (S,G,rpt) Override Timer, OT(S,G,rpt), which is used to delay triggered Join(S,G,rpt) messages to prevent implosions of triggered messages.

Figure 7: Upstream (S,G,rpt) state machine for triggered messages in tabular form

Prev State	Event			
	PruneDesired (S,G,rpt) ->True	PruneDesired (S,G,rpt) ->False	RPTJoin Desired(G) ->False	inherited_ olist (S,G,rpt) ->non-NULL
RPTNotJoined (G) (NJ)	-> P state	-	-	-> NP state
Pruned (S,G,rpt) (P)	-	-> NP state Send Join (S,G,rpt)	-> NJ state	-
NotPruned (S,G,rpt) (NP)	-> P state Send Prune (S,G,rpt); Cancel OT	-	-> NJ state Cancel OT	-

Additionally, we have the following transitions within the NotPruned(S,G,rpt) state, which are all used for prune override behavior.

In NotPruned(S,G,rpt) State				
Override Timer expires	See Prune (S,G,rpt) to RPF' (S,G,rpt)	See Join (S,G,rpt) to RPF' (S,G,rpt)	See Prune (S,G) to RPF' (S,G,rpt)	RPF' (S,G,rpt) -> RPF' (*,G)
Send Join (S,G,rpt); Leave OT unset	OT = min(OT, t_override)	Cancel OT	OT = min(OT, t_override)	OT = min(OT, t_override)

Note that the min function in the above state machine considers a non-running timer to have an infinite value (e.g., min(not-running, t_override) = t_override).

This state machine uses the following macros:

```
bool RPTJoinDesired(G) {
    return (JoinDesired(*,G))
}
```

RPTJoinDesired(G) is true when the router has forwarding state that would cause it to forward traffic for G using (*,G) shared tree state.

```
bool PruneDesired(S,G,rpt) {
    return ( RPTJoinDesired(G) AND
             ( inherited_olist(S,G,rpt) == NULL
               OR (SPTbit(S,G)==TRUE
                   AND (RPF'(*,G) != RPF'(S,G)) )))
}
```

PruneDesired(S,G,rpt) can only be true if RPTJoinDesired(G) is true. If RPTJoinDesired(G) is true, then PruneDesired(S,G,rpt) is true either if there are no outgoing interfaces that S would be forwarded on, or if the router has active (S,G) forwarding state but RPF'(*,G) != RPF'(S,G).

The state machine contains the following transition events:

See Join(S,G,rpt) to RPF'(S,G,rpt)

This event is only relevant in the "Not Pruned" state.

The router sees a Join(S,G,rpt) from someone else to RPF'(S,G,rpt), which is the correct upstream neighbor. If we're in "NotPruned" state and the (S,G,rpt) Override Timer is running, then this is because we have been triggered to send our own Join(S,G,rpt) to RPF'(S,G,rpt). Someone else beat us to it, so there's no need to send our own Join.

The action is to cancel the Override Timer.

See Prune(S,G,rpt) to RPF'(S,G,rpt)

This event is only relevant in the "NotPruned" state.

The router sees a Prune(S,G,rpt) from someone else to RPF'(S,G,rpt), which is the correct upstream neighbor. If we're in the "NotPruned" state, then we want to continue to receive traffic from S destined for G, and that traffic is being supplied by RPF'(S,G,rpt). Thus, we need to override the Prune.

The action is to set the (S,G,rpt) Override Timer to the randomized prune-override interval, t_override. However, if the Override Timer is already running, we only set the timer if doing so would set it to a lower value. At the end of this interval, if no one else has sent a Join, then we will do so.

See Prune(S,G) to RPF'(S,G,rpt)

This event is only relevant in the "NotPruned" state.

This transition and action are the same as the above transition and action, except that the Prune does not have the RPT bit set. This transition is necessary to be compatible with routers implemented from RFC2362 that don't maintain separate (S,G) and (S,G,rpt) state.

The (S,G,rpt) prune Override Timer expires

This event is only relevant in the "NotPruned" state.

When the Override Timer expires, we must send a Join(S,G,rpt) to RPF'(S,G,rpt) to override the Prune message that caused the timer to be running. We only send this if RPF'(S,G,rpt) equals RPF'(*,G); if this were not the case, then the Join might be sent to a router that does not have (*,G) Join state, and so the behavior would not be well defined. If RPF'(S,G,rpt) is not the same as RPF'(*,G), then it may stop forwarding S. However, if this happens, then the router will send an AssertCancel(S,G), which would then cause RPF'(S,G,rpt) to become equal to RPF'(*,G) (see below).

RPF'(S,G,rpt) changes to become equal to RPF'(*,G)
This event is only relevant in the "NotPruned" state.

RPF'(S,G,rpt) can only be different from RPF'(*,G) if an (S,G) Assert has happened, which means that traffic from S is arriving on the SPT, and so Prune(S,G,rpt) will have been sent to RPF'(*,G). When RPF'(S,G,rpt) changes to become equal to RPF'(*,G), we need to trigger a Join(S,G,rpt) to RPF'(*,G) to cause that router to start forwarding S again.

The action is to set the (S,G,rpt) Override Timer to the randomized prune-override interval t_override. However, if the timer is already running, we only set the timer if doing so would set it to a lower value. At the end of this interval, if no one else has sent a Join, then we will do so.

PruneDesired(S,G,rpt)->TRUE
See macro above. This event is relevant in the "NotPruned" and "RPTNotJoined(G)" states.

The router wishes to receive traffic for G, but does not wish to receive traffic from S destined for G. This causes the router to transition into the Pruned state.

If the router was previously in NotPruned state, then the action is to send a Prune(S,G,rpt) to RPF'(S,G,rpt), and to cancel the Override Timer. If the router was previously in RPTNotJoined(G) state, then there is no need to trigger an action in this state machine because sending a Prune(S,G,rpt) is handled by the rules for sending the Join(*,G).

PruneDesired(S,G,rpt)->FALSE
See macro above. This transition is only relevant in the "Pruned" state.

If the router is in the Pruned(S,G,rpt) state, and PruneDesired(S,G,rpt) changes to FALSE, this could be because the router no longer has RPTJoinDesired(G) true, or it now wishes to receive traffic from S again. If it is the former, then this transition should not happen, but instead the "RPTJoinDesired(G)->FALSE" transition should happen. Thus, this transition should be interpreted as "PruneDesired(S,G,rpt)->FALSE AND RPTJoinDesired(G)==TRUE".

The action is to send a Join(S,G,rpt) to RPF'(S,G,rpt).

RPTJoinDesired(G)->FALSE
This event is relevant in the "Pruned" and "NotPruned" states.

The router no longer wishes to receive any traffic destined for G on the RP Tree. This causes a transition to the RPTNotJoined(G) state, and the Override Timer is canceled if it was running. Any further actions are handled by the appropriate upstream state machine for (*,G).

inherited_olist(S,G,rpt) becomes non-NULL

This transition is only relevant in the RPTNotJoined(G) state.

The router has joined the RP tree (handled by the (*,G) upstream state machine as appropriate) and wants to receive traffic from S.

This does not trigger any events in this state machine, but causes a transition to the NotPruned(S,G,rpt) state.

4.6. PIM Assert Messages

Where multiple PIM routers peer over a shared LAN, it is possible for more than one upstream router to have valid forwarding state for a packet, which can lead to packet duplication (see Section 3.6). PIM does not attempt to prevent this from occurring. Instead, it detects when this has happened and elects a single forwarder amongst the upstream routers to prevent further duplication. This election is performed using PIM Assert messages. Assert messages are also received by downstream routers on the LAN, and these cause subsequent Join/Prune messages to be sent to the upstream router that won the Assert.

In general, a PIM Assert 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 an Assert message from a particular IP source address and it has not seen a PIM Hello message from that source address, then the Assert message SHOULD be discarded without further processing. In addition, if the Hello message from a neighbor was authenticated using the IPsec Authentication Header (AH) (see Section 6.3), then all Assert 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 interoperability with such older routers, but that this configuration option SHOULD NOT be enabled by default.

4.6.1. (S,G) Assert Message State Machine

The (S,G) Assert state machine for interface I is shown in Figure 8. There are three states:

NoInfo (NI)

This router has no (S,G) assert state on interface I.

I am Assert Winner (W)

This router has won an (S,G) assert on interface I. It is now responsible for forwarding traffic from S destined for G out of interface I. Irrespective of whether it is the DR for I, while a router is the assert winner, it is also responsible for forwarding traffic onto I on behalf of local hosts on I that have made membership requests that specifically refer to S (and G).

I am Assert Loser (L)

This router has lost an (S,G) assert on interface I. It must not forward packets from S destined for G onto interface I. If it is the DR on I, it is no longer responsible for forwarding traffic onto I to satisfy local hosts with membership requests that specifically refer to S and G.

In addition, there is also an Assert Timer (AT) that is used to time out asserts on the assert losers and to resend asserts on the assert winner.

Figure 8: Per-interface (S,G) Assert State machine in tabular form

In NoInfo (NI) State			
Receive Inferior Assert with RPTbit clear	Receive Assert with RPTbit set and CouldAssert (S,G,I)	Data arrives from S to G on I and CouldAssert (S,G,I)	Receive Acceptable Assert with RPTbit clear and AssTrDes (S,G,I)
-> W state [Actions A1]	-> W state [Actions A1]	-> W state [Actions A1]	-> L state [Actions A6]

In I Am Assert Winner (W) State			
Assert Timer Expires	Receive Inferior Assert	Receive Preferred Assert	CouldAssert (S,G,I) -> FALSE
-> W state [Actions A3]	-> W state [Actions A3]	-> L state [Actions A2]	-> NI state [Actions A4]

In I Am Assert Loser (L) State				
Receive Preferred Assert	Receive Acceptable Assert with RPTbit clear from Current Winner	Receive Inferior Assert or Cancel from Current Winner	Assert Timer Expires	Current Winner's GenID Changes or NLT Expires
-> L state [Actions A2]	-> L state [Actions A2]	-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI state [Actions A5]

In I Am Assert Loser (L) State				
AssTrDes (S,G,I) -> FALSE	my_metric -> better than winner's metric	RPF_interface (S) stops being I	Receive Join(S,G) on interface I	
-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI State [Actions A5]	

Note that for reasons of compactness, "AssTrDes(S,G,I)" is used in the state machine table to refer to AssertTrackingDesired(S,G,I).

Terminology:

A "preferred assert" is one with a better metric than the current winner.

An "acceptable assert" is one that has a better metric than my_assert_metric(S,G,I). An assert is never considered acceptable

if its metric is infinite.

An "inferior assert" is one with a worse metric than `my_assert_metric(S,G,I)`. An assert is never considered inferior if `my_assert_metric(S,G,I)` is infinite.

The state machine uses the following macros:

```
CouldAssert(S,G,I) =
  SPTbit(S,G)==TRUE
  AND (RPF_interface(S) != I)
  AND (I in ( ( joins(*,G) (-) prunes(S,G,rpt) )
              (+) ( pim_include(*,G) (-) pim_exclude(S,G) )
              (-) lost_assert(*,G)
              (+) joins(S,G) (+) pim_include(S,G) ) ) )
```

`CouldAssert(S,G,I)` is true for downstream interfaces that would be in the `inheritedolist(S,G)` if (S,G) assert information was not taken into account.

```
AssertTrackingDesired(S,G,I) =
  (I in ( joins(*,G) (-) prunes(S,G,rpt)
          (+) ( pim_include(*,G) (-) pim_exclude(S,G) )
          (-) lost_assert(*,G)
          (+) joins(S,G) ) )
  OR (local_receiver_include(S,G,I) == TRUE
      AND (I_am_DR(I) OR (AssertWinner(S,G,I) == me)))
  OR ((RPF_interface(S) == I) AND (JoinDesired(S,G) == TRUE))
  OR ((RPF_interface(RP(G)) == I) AND (JoinDesired(*,G) == TRUE)
      AND (SPTbit(S,G) == FALSE))
```

`AssertTrackingDesired(S,G,I)` is true on any interface in which an (S,G) assert might affect our behavior.

The first three lines of `AssertTrackingDesired` account for (*,G) join and local membership information received on I that might cause the router to be interested in asserts on I.

The 4th line accounts for (S,G) join information received on I that might cause the router to be interested in asserts on I.

The 5th and 6th lines account for (S,G) local membership information on I. Note that we can't use the `pim_include(S,G)` macro since it uses `lost_assert(S,G,I)` and would result in the router forgetting that it lost an assert if the only reason it was interested was local membership. The `AssertWinner(S,G,I)` check forces an assert winner to keep on being responsible for forwarding as long as local receivers are present. Removing this check would make the assert winner give

up forwarding as soon as the information that originally caused it to forward went away, and the task of forwarding for local receivers would revert back to the DR.

The last three lines account for the fact that a router must keep track of assert information on upstream interfaces in order to send joins and prunes to the proper neighbor.

Transitions from NoInfo State

When in NoInfo state, the following events may trigger transitions:

Receive Inferior Assert with RPTbit cleared

An assert is received for (S,G) with the RPT bit cleared that is inferior to our own assert metric. The RPT bit cleared indicates that the sender of the assert had (S,G) forwarding state on this interface. If the assert is inferior to our metric, then we must also have (S,G) forwarding state (i.e., `CouldAssert(S,G,I)==TRUE`) as (S,G) asserts beat (*,G) asserts, and so we should be the assert winner. We transition to the "I am Assert Winner" state and perform Actions A1 (below).

Receive Assert with RPTbit set AND `CouldAssert(S,G,I)==TRUE`

An assert is received for (S,G) on I with the RPT bit set (it's a (*,G) assert). `CouldAssert(S,G,I)` is TRUE only if we have (S,G) forwarding state on this interface, so we should be the assert winner. We transition to the "I am Assert Winner" state and perform Actions A1 (below).

An (S,G) data packet arrives on interface I, AND

`CouldAssert(S,G,I)==TRUE`

An (S,G) data packet arrived on a downstream interface that is in our (S,G) outgoing interface list. We optimistically assume that we will be the assert winner for this (S,G), and so we transition to the "I am Assert Winner" state and perform Actions A1 (below), which will initiate the assert negotiation for (S,G).

Receive Acceptable Assert with RPT bit clear AND

`AssertTrackingDesired(S,G,I)==TRUE`

We're interested in (S,G) Asserts, either because I is a downstream interface for which we have (S,G) or (*,G) forwarding state, or because I is the upstream interface for S and we have (S,G) forwarding state. The received assert has a better metric than our own, so we do not win the Assert. We transition to "I am Assert Loser" and perform Actions A6 (below).

Transitions from "I am Assert Winner" State

When in "I am Assert Winner" state, the following events trigger transitions:

Assert Timer Expires

The (S,G) Assert Timer expires. As we're in the Winner state, we must still have (S,G) forwarding state that is actively being kept alive. We resend the (S,G) Assert and restart the Assert Timer (Actions A3 below). Note that the assert winner's Assert Timer is engineered to expire shortly before timers on assert losers; this prevents unnecessary thrashing of the forwarder and periodic flooding of duplicate packets.

Receive Inferior Assert

We receive an (S,G) assert or (*,G) assert mentioning S that has a worse metric than our own. Whoever sent the assert is in error, and so we resend an (S,G) Assert and restart the Assert Timer (Actions A3 below).

Receive Preferred Assert

We receive an (S,G) assert that has a better metric than our own. We transition to "I am Assert Loser" state and perform Actions A2 (below). Note that this may affect the value of JoinDesired(S,G) and PruneDesired(S,G,rpt), which could cause transitions in the upstream (S,G) or (S,G,rpt) state machines.

CouldAssert(S,G,I) -> FALSE

Our (S,G) forwarding state or RPF interface changed so as to make CouldAssert(S,G,I) become false. We can no longer perform the actions of the assert winner, and so we transition to NoInfo state and perform Actions A4 (below). This includes sending a "canceling assert" with an infinite metric.

Transitions from "I am Assert Loser" State

When in "I am Assert Loser" state, the following transitions can occur:

Receive Preferred Assert

We receive an assert that is better than that of the current assert winner. We stay in Loser state and perform Actions A2 below.

Receive Acceptable Assert with RPTbit clear from Current Winner
We receive an assert from the current assert winner that is better than our own metric for this (S,G) (although the metric may be worse than the winner's previous metric). We stay in Loser state and perform Actions A2 below.

Receive Inferior Assert or Assert Cancel from Current Winner
We receive an assert from the current assert winner that is worse than our own metric for this group (typically, because the winner's metric became worse or because it is an assert cancel). We transition to NoInfo state, deleting the (S,G) assert information and allowing the normal PIM Join/Prune mechanisms to operate. Usually, we will eventually re-assert and win when data packets from S have started flowing again.

Assert Timer Expires

The (S,G) Assert Timer expires. We transition to NoInfo state, deleting the (S,G) assert information (Actions A5 below).

Current Winner's GenID Changes or NLT Expires

The Neighbor Liveness Timer associated with the current winner expires or we receive a Hello message from the current winner reporting a different GenID from the one it previously reported. This indicates that the current winner's interface or router has gone down (and may have come back up), and so we must assume it no longer knows it was the winner. We transition to the NoInfo state, deleting this (S,G) assert information (Actions A5 below).

AssertTrackingDesired(S,G,I)->FALSE

AssertTrackingDesired(S,G,I) becomes FALSE. Our forwarding state has changed so that (S,G) Asserts on interface I are no longer of interest to us. We transition to the NoInfo state, deleting the (S,G) assert information.

My metric becomes better than the assert winner's metric

my_assert_metric(S,G,I) has changed so that now my assert metric for (S,G) is better than the metric we have stored for current assert winner. This might happen when the underlying routing metric changes, or when CouldAssert(S,G,I) becomes true; for example, when SPTbit(S,G) becomes true. We transition to NoInfo state, delete this (S,G) assert state (Actions A5 below), and allow the normal PIM Join/Prune mechanisms to operate. Usually, we will eventually re-assert and win when data packets from S have started flowing again.

RPF_interface(S) stops being interface I

Interface I used to be the RPF interface for S, and now it is not. We transition to NoInfo state, deleting this (S,G) assert state (Actions A5 below).

Receive Join(S,G) on Interface I

We receive a Join(S,G) that has the Upstream Neighbor Address field set to my primary IP address on interface I. The action is to transition to NoInfo state, delete this (S,G) assert state (Actions A5 below), and allow the normal PIM Join/Prune mechanisms to operate. If whoever sent the Join was in error, then the normal assert mechanism will eventually re-apply, and we will lose the assert again. However, whoever sent the assert may know that the previous assert winner has died, and so we may end up being the new forwarder.

(S,G) Assert State machine Actions

- A1: Send Assert(S,G).
Set Assert Timer to (Assert_Time - Assert_Override_Interval).
Store self as AssertWinner(S,G,I).
Store spt_assert_metric(S,I) as AssertWinnerMetric(S,G,I).
- A2: Store new assert winner as AssertWinner(S,G,I) and assert winner metric as AssertWinnerMetric(S,G,I).
Set Assert Timer to Assert_Time.
- A3: Send Assert(S,G).
Set Assert Timer to (Assert_Time - Assert_Override_Interval).
- A4: Send AssertCancel(S,G).
Delete assert info (AssertWinner(S,G,I) and AssertWinnerMetric(S,G,I) will then return to their default values).
- A5: Delete assert info (AssertWinner(S,G,I) and AssertWinnerMetric(S,G,I) will then return to their default values).
- A6: Store new assert winner as AssertWinner(S,G,I) and assert winner metric as AssertWinnerMetric(S,G,I).
Set Assert Timer to Assert_Time.
If (I is RPF_interface(S)) AND (UpstreamJPState(S,G) == Joined) set SPTbit(S,G) to TRUE.

Note that some of these actions may cause the value of JoinDesired(S,G), PruneDesired(S,G,rpt), or RPF'(S,G) to change, which could cause further transitions in other state machines.

4.6.2. (*,G) Assert Message State Machine

The (*,G) Assert state machine for interface I is shown in Figure 9. There are three states:

NoInfo (NI)

This router has no (*,G) assert state on interface I.

I am Assert Winner (W)

This router has won an (*,G) assert on interface I. It is now responsible for forwarding traffic destined for G onto interface I with the exception of traffic for which it has (S,G) "I am Assert Loser" state. Irrespective of whether it is the DR for I, it is also responsible for handling the membership requests for G from local hosts on I.

I am Assert Loser (L)

This router has lost an (*,G) assert on interface I. It must not forward packets for G onto interface I with the exception of traffic from sources for which it has (S,G) "I am Assert Winner" state. If it is the DR, it is no longer responsible for handling the membership requests for group G from local hosts on I.

In addition, there is also an Assert Timer (AT) that is used to time out asserts on the assert losers and to resend asserts on the assert winner.

When an Assert message is received with a source address other than zero, a PIM implementation must first match it against the possible events in the (S,G) assert state machine and process any transitions and actions, before considering whether the Assert message matches against the (*,G) assert state machine.

It is important to note that NO TRANSITION CAN OCCUR in the (*,G) state machine as a result of receiving an Assert message unless the (S,G) assert state machine for the relevant S and G is in the "NoInfo" state after the (S,G) state machine has processed the message. Also, NO TRANSITION CAN OCCUR in the (*,G) state machine as a result of receiving an assert message if that message triggers any change of state in the (S,G) state machine. Obviously, when the source address in the received message is set to zero, an (S,G) state machine for the S and G does not exist and can be assumed to be in the "NoInfo" state.

For example, if both the (S,G) and (*,G) assert state machines are in the NoInfo state when an Assert message arrives, and the message causes the (S,G) state machine to transition to either "W" or "L" state, then the assert will not be processed by the (*,G) assert state machine.

Another example: if the (S,G) assert state machine is in "L" state when an assert message is received, and the assert metric in the message is worse than `my_assert_metric(S,G,I)`, then the (S,G) assert state machine will transition to NoInfo state. In such a case, if the (*,G) assert state machine were in NoInfo state, it might appear that it would transition to "W" state, but this is not the case because this message already triggered a transition in the (S,G) assert state machine.

Figure 9: Per-interface (*,G) Assert State machine in tabular form

In NoInfo (NI) State			
Receive Inferior Assert with RPTbit set and <code>CouldAssert(*,G,I)</code>	Data arrives for G on I and <code>CouldAssert(*,G,I)</code>	Receive Acceptable Assert with RPTbit set and <code>AssTrDes(*,G,I)</code>	
-> W state [Actions A1]	-> W state [Actions A1]	-> L state [Actions A2]	

In I Am Assert Winner (W) State			
Assert Timer Expires	Receive Inferior Assert	Receive Preferred Assert	<code>CouldAssert(*,G,I) -> FALSE</code>
-> W state [Actions A3]	-> W state [Actions A3]	-> L state [Actions A2]	-> NI state [Actions A4]

In I Am Assert Loser (L) State				
Receive Preferred Assert with RPTbit set	Receive Acceptable Assert from Current Winner with RPTbit set	Receive Inferior Assert or Cancel from Current Winner	Assert Timer Expires	Current Winner's GenID Changes or NLT Expires
-> L state [Actions A2]	-> L state [Actions A2]	-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI state [Actions A5]

In I Am Assert Loser (L) State				
AssTrDes (*,G,I) -> FALSE	my_metric -> better than Winner's metric	RPF_interface (RP(G)) stops being I	Receive Join(*,G) on Interface I	
-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI state [Actions A5]	-> NI State [Actions A5]	

The state machine uses the following macros:

```
CouldAssert(*,G,I) =
  ( I in ( joins(*,G) (+) pim_include(*,G)) )
  AND (RPF_interface(RP(G)) != I)
```

CouldAssert(*,G,I) is true on downstream interfaces for which we have (*,G) join state, or local members that requested any traffic destined for G.

```
AssertTrackingDesired(*,G,I) =
  CouldAssert(*,G,I)
  OR (local_receiver_include(*,G,I)==TRUE
      AND (I_am_DR(I) OR AssertWinner(*,G,I) == me))
  OR (RPF_interface(RP(G)) == I AND RPTJoinDesired(G))
```

AssertTrackingDesired(*,G,I) is true on any interface on which an (*,G) assert might affect our behavior.

Note that for reasons of compactness, "AssTrDes(*,G,I)" is used in the state machine table to refer to AssertTrackingDesired(*,G,I).

Terminology:

A "preferred assert" is one with a better metric than the current winner.

An "acceptable assert" is one that has a better metric than `my_assert_metric(*,G,I)`. An assert is never considered acceptable if its metric is infinite.

An "inferior assert" is one with a worse metric than `my_assert_metric(*,G,I)`. An assert is never considered inferior if `my_assert_metric(*,G,I)` is infinite.

Transitions from NoInfo State

When in NoInfo state, the following events trigger transitions, but only if the (S,G) assert state machine is in NoInfo state before and after consideration of the received message:

Receive Inferior Assert with RPTbit set AND

`CouldAssert(*,G,I)==TRUE`

An Inferior (*,G) assert is received for G on Interface I. If `CouldAssert(*,G,I)` is TRUE, then I is our downstream interface, and we have (*,G) forwarding state on this interface, so we should be the assert winner. We transition to the "I am Assert Winner" state and perform Actions A1 (below).

A data packet destined for G arrives on interface I, AND

`CouldAssert(*,G,I)==TRUE`

A data packet destined for G arrived on a downstream interface that is in our (*,G) outgoing interface list. We therefore believe we should be the forwarder for this (*,G), and so we transition to the "I am Assert Winner" state and perform Actions A1 (below).

Receive Acceptable Assert with RPT bit set AND

`AssertTrackingDesired(*,G,I)==TRUE`

We're interested in (*,G) Asserts, either because I is a downstream interface for which we have (*,G) forwarding state, or because I is the upstream interface for RP(G) and we have (*,G) forwarding state. We get a (*,G) Assert that has a better metric than our own, so we do not win the Assert. We transition to "I am Assert Loser" and perform Actions A2 (below).

Transitions from "I am Assert Winner" State

When in "I am Assert Winner" state, the following events trigger transitions, but only if the (S,G) assert state machine is in NoInfo state before and after consideration of the received message:

Receive Inferior Assert

We receive a (*,G) assert that has a worse metric than our own. Whoever sent the assert has lost, and so we resend a (*,G) Assert and restart the Assert Timer (Actions A3 below).

Receive Preferred Assert

We receive a (*,G) assert that has a better metric than our own. We transition to "I am Assert Loser" state and perform Actions A2 (below).

When in "I am Assert Winner" state, the following events trigger transitions:

Assert Timer Expires

The (*,G) Assert Timer expires. As we're in the Winner state, then we must still have (*,G) forwarding state that is actively being kept alive. To prevent unnecessary thrashing of the forwarder and periodic flooding of duplicate packets, we resend the (*,G) Assert and restart the Assert Timer (Actions A3 below).

CouldAssert(*,G,I) -> FALSE

Our (*,G) forwarding state or RPF interface changed so as to make CouldAssert(*,G,I) become false. We can no longer perform the actions of the assert winner, and so we transition to NoInfo state and perform Actions A4 (below).

Transitions from "I am Assert Loser" State

When in "I am Assert Loser" state, the following events trigger transitions, but only if the (S,G) assert state machine is in NoInfo state before and after consideration of the received message:

Receive Preferred Assert with RPTbit set

We receive a (*,G) assert that is better than that of the current assert winner. We stay in Loser state and perform Actions A2 below.

Receive Acceptable Assert from Current Winner with RPTbit set

We receive a (*,G) assert from the current assert winner that is better than our own metric for this group (although the metric may be worse than the winner's previous metric). We stay in Loser state and perform Actions A2 below.

Receive Inferior Assert or Assert Cancel from Current Winner

We receive an assert from the current assert winner that is worse than our own metric for this group (typically because the winner's metric became worse or is now an assert cancel). We transition to NoInfo state, delete this (*,G) assert state (Actions A5), and allow the normal PIM Join/Prune mechanisms to operate. Usually, we will eventually re-assert and win when data packets for G have started flowing again.

When in "I am Assert Loser" state, the following events trigger transitions:

Assert Timer Expires

The (*,G) Assert Timer expires. We transition to NoInfo state and delete this (*,G) assert info (Actions A5).

Current Winner's GenID Changes or NLT Expires

The Neighbor Liveness Timer associated with the current winner expires or we receive a Hello message from the current winner reporting a different GenID from the one it previously reported. This indicates that the current winner's interface or router has gone down (and may have come back up), and so we must assume it no longer knows it was the winner. We transition to the NoInfo state, deleting the (*,G) assert information (Actions A5).

AssertTrackingDesired(*,G,I)->FALSE

AssertTrackingDesired(*,G,I) becomes FALSE. Our forwarding state has changed so that (*,G) Asserts on interface I are no longer of interest to us. We transition to NoInfo state and delete this (*,G) assert info (Actions A5).

My metric becomes better than the assert winner's metric

My routing metric, rpt_assert_metric(G,I), has changed so that now my assert metric for (*,G) is better than the metric we have stored for current assert winner. We transition to NoInfo state, delete this (*,G) assert state (Actions A5), and allow the normal PIM Join/Prune mechanisms to operate. Usually, we will eventually re-assert and win when data packets for G have started flowing again.

RPF_interface(RP(G)) stops being interface I

Interface I used to be the RPF interface for RP(G), and now it is not. We transition to NoInfo state and delete this (*,G) assert state (Actions A5).

Receive Join(*,G) on interface I

We receive a Join(*,G) that has the Upstream Neighbor Address

field set to my primary IP address on interface I. The action is to transition to NoInfo state, delete this (*,G) assert state (Actions A5), and allow the normal PIM Join/Prune mechanisms to operate. If whoever sent the Join was in error, then the normal assert mechanism will eventually re-apply, and we will lose the assert again. However, whoever sent the assert may know that the previous assert winner has died, so we may end up being the new forwarder.

(*,G) Assert State machine Actions

- A1: Send Assert(*,G).
Set Assert Timer to (Assert_Time - Assert_Override_Interval).
Store self as AssertWinner(*,G,I).
Store rpt_assert_metric(G,I) as AssertWinnerMetric(*,G,I).
- A2: Store new assert winner as AssertWinner(*,G,I) and assert winner metric as AssertWinnerMetric(*,G,I).
Set Assert Timer to Assert_Time.
- A3: Send Assert(*,G)
Set Assert Timer to (Assert_Time - Assert_Override_Interval).
- A4: Send AssertCancel(*,G).
Delete assert info (AssertWinner(*,G,I) and AssertWinnerMetric(*,G,I) will then return to their default values).
- A5: Delete assert info (AssertWinner(*,G,I) and AssertWinnerMetric(*,G,I) will then return to their default values).

Note that some of these actions may cause the value of JoinDesired(*,G) or RPF'(*,G)) to change, which could cause further transitions in other state machines.

4.6.3. Assert Metrics

Assert metrics are defined as:

```
struct assert_metric {  
    rpt_bit_flag;  
    metric_preference;  
    route_metric;  
    ip_address;  
};
```

When comparing assert_metrics, the rpt_bit_flag, metric_preference,

and route_metric field are compared in order, where the first lower value wins. If all fields are equal, the primary IP address of the router that sourced the Assert message is used as a tie-breaker, with the highest IP address winning.

An assert metric for (S,G) to include in (or compare against) an Assert message sent on interface I should be computed using the following pseudocode:

```
assert_metric
my_assert_metric(S,G,I) {
    if( CouldAssert(S,G,I) == TRUE ) {
        return spt_assert_metric(S,I)
    } else if( CouldAssert(*,G,I) == TRUE ) {
        return rpt_assert_metric(G,I)
    } else {
        return infinite_assert_metric()
    }
}
```

spt_assert_metric(S,I) gives the assert metric we use if we're sending an assert based on active (S,G) forwarding state:

```
assert_metric
spt_assert_metric(S,I) {
    return {0,MRIB.pref(S),MRIB.metric(S),my_ip_address(I)}
}
```

rpt_assert_metric(G,I) gives the assert metric we use if we're sending an assert based only on (*,G) forwarding state:

```
assert_metric
rpt_assert_metric(G,I) {
    return {1,MRIB.pref(RP(G)),MRIB.metric(RP(G)),my_ip_address(I)}
}
```

MRIB.pref(X) and MRIB.metric(X) are the routing preference and routing metrics associated with the route to a particular (unicast) destination X, as determined by the MRIB. my_ip_address(I) is simply the router's primary IP address that is associated with the local interface I.

infinite_assert_metric() gives the assert metric we need to send an assert but don't match either (S,G) or (*,G) forwarding state:


```
assert_metric
infinite_assert_metric() {
    return {1,infinity,infinity,0}
}
```

4.6.4. AssertCancel Messages

An AssertCancel message is simply an RPT Assert message but with infinite metric. It is sent by the assert winner when it deletes the forwarding state that had caused the assert to occur. Other routers will see this metric, and it will cause any other router that has forwarding state to send its own assert, and to take over forwarding.

An AssertCancel(S,G) is an infinite metric assert with the RPT bit set that names S as the source.

An AssertCancel(*,G) is an infinite metric assert with the RPT bit set and the source set to zero.

AssertCancel messages are simply an optimization. The original Assert timeout mechanism will allow a subnet to eventually become consistent; the AssertCancel mechanism simply causes faster convergence. No special processing is required for an AssertCancel message, since it is simply an Assert message from the current winner.

4.6.5. Assert State Macros

The macros `lost_assert(S,G,rpt,I)`, `lost_assert(S,G,I)`, and `lost_assert(*,G,I)` are used in the olist computations of Section 4.1, and are defined as:

```
bool lost_assert(S,G,rpt,I) {
    if ( RPF_interface(RP(G)) == I OR
        ( RPF_interface(S) == I AND SPTbit(S,G) == TRUE ) ) {
        return FALSE
    } else {
        return ( AssertWinner(S,G,I) != NULL AND
                 AssertWinner(S,G,I) != me )
    }
}
```

```
bool lost_assert(S,G,I) {
    if ( RPF_interface(S) == I ) {
        return FALSE
    } else {
        return ( AssertWinner(S,G,I) != NULL AND
                 AssertWinner(S,G,I) != me AND
                 (AssertWinnerMetric(S,G,I) is better
                  than spt_assert_metric(S,I) )
        )
    }
}
```

Note: the term "AssertWinnerMetric(S,G,I) is better than spt_assert_metric(S,I)" is required to correctly handle the transition phase when a router has (S,G) join state, but has not yet set the SPTbit. In this case, it needs to ignore the assert state if it will win the assert once the SPTbit is set.

```
bool lost_assert(*,G,I) {
    if ( RPF_interface(RP(G)) == I ) {
        return FALSE
    } else {
        return ( AssertWinner(*,G,I) != NULL AND
                 AssertWinner(*,G,I) != me )
    }
}
```

AssertWinner(S,G,I) is the IP source address of the Assert(S,G) packet that won an Assert.

AssertWinner(*,G,I) is the IP source address of the Assert(*,G) packet that won an Assert.

AssertWinnerMetric(S,G,I) is the Assert metric of the Assert(S,G) packet that won an Assert.

AssertWinnerMetric(*,G,I) is the Assert metric of the Assert(*,G) packet that won an Assert.

AssertWinner(S,G,I) defaults to NULL and AssertWinnerMetric(S,G,I) defaults to Infinity when in the NoInfo state.

Summary of Assert Rules and Rationale

This section summarizes the key rules for sending and reacting to asserts and the rationale for these rules. This section is not intended to be and should not be treated as a definitive specification of protocol behavior. The state machines and pseudocode should be consulted for that purpose. Rather, this section is intended to document important aspects of the Assert protocol behavior and to provide information that may prove helpful to the reader in understanding and implementing this part of the protocol.

1. Behavior: Downstream neighbors send Join(*,G) and Join(S,G) periodic messages to the appropriate RPF' neighbor, i.e., the RPF neighbor as modified by the assert process. They are not always sent to the RPF neighbor as indicated by the MRIB. Normal suppression and override rules apply.

Rationale: By sending the periodic and triggered Join messages to the RPF' neighbor instead of to the RPF neighbor, the downstream router avoids re-triggering the Assert process with every Join. A side effect of sending Joins to the Assert winner is that traffic will not switch back to the "normal" RPF neighbor until the Assert times out. This will not happen until data stops flowing, if item 8, below, is implemented.

2. Behavior: The assert winner for (*,G) acts as the local DR for (*,G) on behalf of IGMP/MLD members.

Rationale: This is required to allow a single router to merge PIM and IGMP/MLD joins and leaves. Without this, overrides don't work.

3. Behavior: The assert winner for (S,G) acts as the local DR for (S,G) on behalf of IGMPv3 members.

Rationale: Same rationale as for item 2.

4. Behavior: (S,G) and (*,G) prune overrides are sent to the RPF' neighbor and not to the regular RPF neighbor.

Rationale: Same rationale as for item 1.

5. Behavior: An (S,G,rpt) prune override is not sent (at all) if $RPF'(S,G,rpt) \neq RPF'(*,G)$.

Rationale: This avoids keeping state alive on the (S,G) tree when only (*,G) downstream members are left. Also, it avoids sending (S,G,rpt) joins to a router that is not on the (*,G) tree. This behavior might be confusing although this specification does indicate that such a join SHOULD be dropped.

6. Behavior: An assert loser that receives a Join(S,G) with an Upstream Neighbor Address that is its primary IP address on that interface expires the (S,G) Assert Timer.

Rationale: This is necessary in order to have rapid convergence in the event that the downstream router that initially sent a join to the prior Assert winner has undergone a topology change.

7. Behavior: An assert loser that receives a Join(*,G) with an Upstream Neighbor Address that is its primary IP address on that interface cancels the (*,G) Assert Timer and all (S,G) assert timers that do not have corresponding Prune(S,G,rpt) messages in the compound Join/Prune message.

Rationale: Same rationale as for item 6.

8. Behavior: An assert winner for (*,G) or (S,G) sends a canceling assert when it is about to stop forwarding on a (*,G) or an (S,G) entry. This behavior does not apply to (S,G,rpt).

Rationale: This allows switching back to the shared tree after the last SPT router on the LAN leaves. Doing this prevents downstream routers on the shared tree from keeping SPT state alive.

9. Behavior: Resend the assert messages before timing out an assert. (This behavior is optional.)

Rationale: This prevents the periodic duplicates that would otherwise occur each time that an assert times out and is then re-established.

10. Behavior: When $RPF'(S,G,rpt)$ changes to be the same as $RPF'(*,G)$ we need to trigger a Join(S,G,rpt) to $RPF'(*,G)$.

Rationale: This allows switching back to the RPT after the last SPT member leaves.

4.7. PIM Bootstrap and RP Discovery

For correct operation, every PIM router within a PIM domain must be able to map a particular multicast group address to the same RP. If this is not the case, then black holes may appear, where some receivers in the domain cannot receive some groups. A domain in this context is a contiguous set of routers that all implement PIM and are configured to operate within a common boundary.

A notable exception to this is where a PIM domain is broken up into multiple administrative scope regions; these are regions where a border has been configured so that a range of multicast groups will not be forwarded across that border. For more information on Administratively Scoped IP Multicast, see RFC 2365. The modified criteria for admin-scoped regions are that the region is convex with respect to forwarding based on the MRIB, and that all PIM routers within the scope region map scoped groups to the same RP within that region.

This specification does not mandate the use of a single mechanism to provide routers with the information to perform the group-to-RP mapping. Currently four mechanisms are possible, and all four have associated problems:

Static Configuration

A PIM router MUST support the static configuration of group-to-RP mappings. Such a mechanism is not robust to failures, but does at least provide a basic interoperability mechanism.

Embedded-RP

Embedded-RP defines an address allocation policy in which the address of the Rendezvous Point (RP) is encoded in an IPv6 multicast group address [17].

Cisco's Auto-RP

Auto-RP uses a PIM Dense-Mode multicast group to announce group-to-RP mappings from a central location. This mechanism is not useful if PIM Dense-Mode is not being run in parallel with PIM Sparse-Mode, and was only intended for use with PIM Sparse-Mode Version 1. No standard specification currently exists.

Bootstrap Router (BSR)

RFC 2362 specifies a bootstrap mechanism based on the automatic election of a bootstrap router (BSR). Any router in the domain that is configured to be a possible RP reports its candidacy to the BSR, and then a domain-wide flooding mechanism distributes the BSR's chosen set of RPs throughout the domain. As specified in RFC 2362, BSR is flawed in its handling of admin-scoped

regions that are smaller than a PIM domain, but the mechanism does work for global-scoped groups.

As far as PIM-SM is concerned, the only important requirement is that all routers in the domain (or admin scope zone for scoped regions) receive the same set of group-range-to-RP mappings. This may be achieved through the use of any of these mechanisms, or through alternative mechanisms not currently specified.

It must be operationally ensured that any RP address configured, learned, or advertised is reachable from all routers in the PIM domain.

4.7.1. Group-to-RP Mapping

Using one of the mechanisms described above, a PIM router receives one or more possible group-range-to-RP mappings. Each mapping specifies a range of multicast groups (expressed as a group and mask) and the RP to which such groups should be mapped. Each mapping may also have an associated priority. It is possible to receive multiple mappings, all of which might match the same multicast group; this is the common case with BSR. The algorithm for performing the group-to-RP mapping is as follows:

1. Perform longest match on group-range to obtain a list of RPs.
2. From this list of matching RPs, find the ones with highest priority.

Eliminate any RPs from the list that have lower priorities.

3. If only one RP remains in the list, use that RP.
4. If multiple RPs are in the list, use the PIM hash function to choose one.

Thus, if two or more group-range-to-RP mappings cover a particular group, the one with the longest mask is the mapping to use. If the mappings have the same mask length, then the one with the highest priority is chosen. If there is more than one matching entry with the same longest mask and the priorities are identical, then a hash function (see Section 4.7.2) is applied to choose the RP.

This algorithm is invoked by a DR when it needs to determine an RP for a given group, e.g., upon reception of a packet or IGMP/MLD membership indication for a group for which the DR does not know the RP.

Furthermore, the mapping function is invoked by all routers upon receiving a (*,G) Join/Prune message.

Note that if the set of possible group-range-to-RP mappings changes, each router will need to check whether any existing groups are affected. This may, for example, cause a DR or acting DR to re-join a group, or cause it to restart register encapsulation to the new RP.

Implementation note: the bootstrap mechanism described in RFC 2362 omitted step 1 above. However, of the implementations we are aware of, approximately half performed step 1 anyway. Note that implementations of BSR that omit step 1 will not correctly interoperate with implementations of this specification when used with the BSR mechanism described in [11].

4.7.2. Hash Function

The hash function is used by all routers within a domain, to map a group to one of the RPs from the matching set of group-range-to-RP mappings (this set all have the same longest mask length and same highest priority). The algorithm takes as input the group address, and the addresses of the candidate RPs from the mappings, and gives as output one RP address to be used.

The protocol requires that all routers hash to the same RP within a domain (except for transients). The following hash function must be used in each router:

1. For RP addresses in the matching group-range-to-RP mappings, compute a value:

Value(G,M,C(i))=
 $(1103515245 * ((1103515245 * (G \& M) + 12345) \text{ XOR } C(i)) + 12345) \bmod 2^{31}$

where C(i) is the RP address and M is a hash-mask. If BSR is being used, the hash-mask is given in the Bootstrap messages. If BSR is not being used, the alternative mechanism that supplies the group-range-to-RP mappings may supply the value, or else it defaults to a mask with the most significant 30 bits being one for IPv4 and the most significant 126 bits being one for IPv6. The hash-mask allows a small number of consecutive groups (e.g., 4) to always hash to the same RP. For instance, hierarchically-encoded data can be sent on consecutive group addresses to get the same delay and fate-sharing characteristics.

For address families other than IPv4, a 32-bit digest to be used as C(i) and G must first be derived from the actual RP or group address. Such a digest method must be used consistently

throughout the PIM domain. For IPv6 addresses, it is RECOMMENDED to use the equivalent IPv4 address for an IPv4-compatible address, and the exclusive-or of each 32-bit segment of the address for all other IPv6 addresses. For example, the digest of the IPv6 address 3ffe:b00:c18:1::10 would be computed as $0x3ffe0b00 \wedge 0x0c180001 \wedge 0x00000000 \wedge 0x00000010$, where \wedge represents the exclusive-or operation.

2. The candidate RP with the highest resulting hash value is then the RP chosen by this Hash Function. If more than one RP has the same highest hash value, the RP with the highest IP address is chosen.

4.8. Source-Specific Multicast

The Source-Specific Multicast (SSM) service model [6] can be implemented with a strict subset of the PIM-SM protocol mechanisms. Both regular IP Multicast and SSM semantics can coexist on a single router, and both can be implemented using the PIM-SM protocol. A range of multicast addresses, currently 232.0.0.0/8 in IPv4 and FF3x::/32 for IPv6, is reserved for SSM, and the choice of semantics is determined by the multicast group address in both data packets and PIM messages.

4.8.1. Protocol Modifications for SSM Destination Addresses

The following rules override the normal PIM-SM behavior for a multicast address G in the SSM range:

- o A router MUST NOT send a (*,G) Join/Prune message for any reason.
- o A router MUST NOT send an (S,G,rpt) Join/Prune message for any reason.
- o A router MUST NOT send a Register message for any packet that is destined to an SSM address.
- o A router MUST NOT forward packets based on (*,G) or (S,G,rpt) state. The (*,G)- and (S,G,rpt)-related state summarization macros are NULL for any SSM address, for the purposes of packet forwarding.
- o A router acting as an RP MUST NOT forward any Register-encapsulated packet that has an SSM destination address, and SHOULD respond with a Register-Stop message to such a Register message.
- o A router MAY optimize out the creation and maintenance of (S,G,rpt) and (*,G) state for SSM destination addresses -- this state is not

needed for SSM packets.

The last three rules are present to deal with SSM-unaware "legacy" routers that may be sending (*,G) and (S,G,rpt) Join/Prunes, or Register messages for SSM destination addresses. Note that this specification does not attempt to aid an SSM-unaware "legacy" router with SSM operations.

4.8.2. PIM-SSM-Only Routers

An implementer may choose to implement only the subset of PIM Sparse-Mode that provides SSM forwarding semantics.

A PIM-SSM-only router MUST implement the following portions of this specification:

- o Upstream (S,G) state machine (Section 4.5.5)
- o Downstream (S,G) state machine (Section 4.5.2)
- o (S,G) Assert state machine (Section 4.6.1)
- o Hello messages, neighbor discovery, and DR election (Section 4.3)
- o Packet forwarding rules (Section 4.2)

A PIM-SSM-only router does not need to implement the following protocol elements:

- o Register state machine (Section 4.4)
- o (*,G) and (S,G,rpt) Downstream state machines (Sections 4.5.2, 4.5.4, and 4.5.1)
- o (*,G) and (S,G,rpt) Upstream state machines (Sections 4.5.6, 4.5.8, and 4.5.5)
- o (*,G) Assert state machine (Section 4.6.2)
- o Bootstrap RP Election (Section 4.7)
- o Keepalive Timer
- o SPTbit (Section 4.2.2)

The Keepalive Timer should be treated as always running, and SPTbit should be treated as always being set for an SSM address. Additionally, the Packet forwarding rules of Section 4.2 can be

simplified in a PIM-SSM-only router:

```
if( iif == RPF_interface(S) AND UpstreamJPState(S,G) == Joined ) {  
    oiflist = inherited_oiflist(S,G)  
} else if( iif is in inherited_oiflist(S,G) ) {  
    send Assert(S,G) on iif  
}  
  
oiflist = oiflist (-) iif  
forward packet on all interfaces in oiflist
```

This is nothing more than the reduction of the normal PIM-SM forwarding rule, with all (S,G,rpt) and (*,G) clauses replaced with NULL.

4.9. PIM Packet Formats

This section describes the details of the packet formats for PIM control messages.

All PIM control messages have IP protocol number 103.

PIM messages are either unicast (e.g., Registers and Register-Stop) or multicast with TTL 1 to the 'ALL-PIM-ROUTERS' group (e.g., Join/Prune, Asserts, etc.). The source address used for unicast messages is a domain-wide reachable address; the source address used for multicast messages is the link-local address of the interface on which the message is being sent.

The IPv4 'ALL-PIM-ROUTERS' group is '224.0.0.13'. The IPv6 'ALL-PIM-ROUTERS' group is 'ff02::d'.

The PIM header common to all PIM messages is:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|PIM Ver| Type |   Reserved   |                Checksum                |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

PIM Ver

PIM Version number is 2.

Type

Types for specific PIM messages. PIM Types are:

Message Type	Destination
0 = Hello	Multicast to ALL-PIM-ROUTERS
1 = Register	Unicast to RP
2 = Register-Stop	Unicast to source of Register packet
3 = Join/Prune	Multicast to ALL-PIM-ROUTERS
4 = Bootstrap	Multicast to ALL-PIM-ROUTERS
5 = Assert	Multicast to ALL-PIM-ROUTERS
6 = Graft (used in PIM-DM only)	Unicast to RPF'(S)
7 = Graft-Ack (used in PIM-DM only)	Unicast to source of Graft packet
8 = Candidate-RP-Advertisement	Unicast to Domain's BSR

Reserved

Set to zero on transmission. Ignored upon receipt.

Checksum

The checksum is a standard IP checksum, i.e., the 16-bit one's complement of the one's complement sum of the entire PIM message, excluding the "Multicast data packet" section of the Register message. For computing the checksum, the checksum field is zeroed. If the packet's length is not an integral number of 16-bit words, the packet is padded with a trailing byte of zero before performing the checksum.

For IPv6, the checksum also includes the IPv6 "pseudo-header", as specified in RFC 2460, Section 8.1 [5]. This "pseudo-header" is prepended to the PIM header for the purposes of calculating the checksum. The "Upper-Layer Packet Length" in the pseudo-header is set to the length of the PIM message, except in Register messages where it is set to the length of the PIM register header (8). The Next Header value used in the pseudo-header is 103.

If a message is received with an unrecognized PIM Ver or Type field, or if a message's destination does not correspond to the table above, the message **MUST** be discarded, and an error message **SHOULD** be logged to the administrator in a rate-limited manner.

4.9.1. Encoded Source and Group Address Formats

Encoded-Unicast Address

An Encoded-Unicast address takes the following format:

0																1																2																3															
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9																								
Addr Family																Encoding Type																Unicast Address																															

... (The diagram shows a continuation of the address field with a trailing ellipsis.)

Addr Family

The PIM address family of the 'Unicast Address' field of this address.

Values 0-127 are as assigned by the IANA for Internet Address Families in [7]. Values 128-250 are reserved to be assigned by the IANA for PIM-specific Address Families. Values 251 through 255 are designated for private use. As there is no assignment authority for this space, collisions should be expected.

Encoding Type

The type of encoding used within a specific Address Family. The value '0' is reserved for this field and represents the native encoding of the Address Family.

Unicast Address

The unicast address as represented by the given Address Family and Encoding Type.

Encoded-Group Address

Encoded-Group addresses take the following format:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Addr Family | Encoding Type |B| Reserved |Z| Mask Len  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Group multicast Address
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+...

```

Addr Family
Described above.

Encoding Type
Described above.

[B]idirectional PIM
Indicates the group range should use Bidirectional PIM [13].
For PIM-SM defined in this specification, this bit MUST be zero.

Reserved
Transmitted as zero. Ignored upon receipt.

Admin Scope [Z]one
indicates the group range is an admin scope zone. This is used
in the Bootstrap Router Mechanism [11] only. For all other
purposes, this bit is set to zero and ignored on receipt.

Mask Len
The Mask length field is 8 bits. The value is the number of
contiguous one bits that are left justified and used as a mask;
when combined with the group address, it describes a range of
groups. It is less than or equal to the address length in bits
for the given Address Family and Encoding Type. If the message
is sent for a single group, then the Mask length must equal the
address length in bits for the given Address Family and Encoding
Type (e.g., 32 for IPv4 native encoding, 128 for IPv6 native
encoding).

Group multicast Address
Contains the group address.

Encoded-Source Address

Encoded-Source address takes the following format:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| Addr Family   | Encoding Type | Rsrvd   | S|W|R| Mask Len   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Source Address
+-----+-----+-----+-----+-----+-----+-----+-----+...
```

Addr Family
Described above.

Encoding Type
Described above.

Reserved
Transmitted as zero, ignored on receipt.

S The Sparse bit is a 1-bit value, set to 1 for PIM-SM. It is used for PIM version 1 compatibility.

W The WC (or WildCard) bit is a 1-bit value for use with PIM Join/Prune messages (see Section 4.9.5.1).

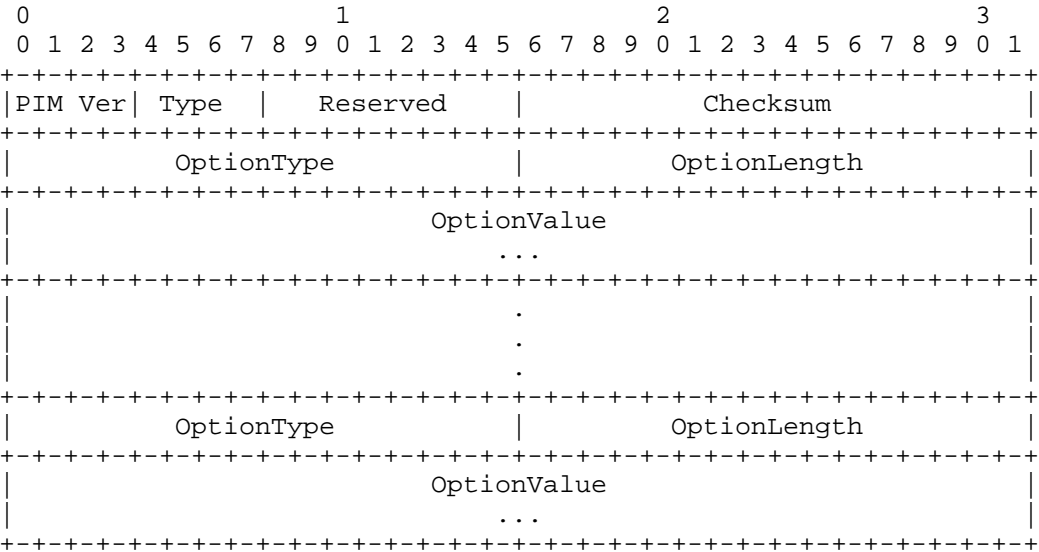
R The RPT (or Rendezvous Point Tree) bit is a 1-bit value for use with PIM Join/Prune messages (see Section 4.9.5.1). If the WC bit is 1, the RPT bit MUST be 1.

Mask Len
The mask length field is 8 bits. The value is the number of contiguous one bits left justified used as a mask which, combined with the Source Address, describes a source subnet. The mask length MUST be equal to the mask length in bits for the given Address Family and Encoding Type (32 for IPv4 native and 128 for IPv6 native). A router SHOULD ignore any messages received with any other mask length.

Source Address
The source address.

4.9.2. Hello Message Format

It is sent periodically by routers on all interfaces.



PIM Version, Type, Reserved, Checksum
Described in Section 4.9.

OptionType
The type of the option given in the following OptionValue field.

OptionLength
The length of the OptionValue field in bytes.

OptionValue
A variable length field, carrying the value of the option.

The Option fields may contain the following values:

o OptionType 1: Holdtime

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     |                                     |
|               Type = 1             |               Length = 2         |
+-----+-----+-----+-----+-----+-----+-----+-----+
|               Holdtime              |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Holdtime is the amount of time a receiver must keep the neighbor reachable, in seconds. If the Holdtime is set to '0xffff', the receiver of this message never times out the neighbor. This may be used with dial-on-demand links, to avoid keeping the link up with periodic Hello messages.

An implementation MAY provide a configuration mechanism to reject a Hello message with holdtime 0xffff, and/or provide a mechanism to remove a neighbor.

Hello messages with a Holdtime value set to '0' are also sent by a router on an interface about to go down or changing IP address (see Section 4.3.1). These are effectively goodbye messages, and the receiving routers SHOULD immediately time out the neighbor information for the sender.

o OptionType 2: LAN Prune Delay

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     |                                     |
|               Type = 2             |               Length = 4         |
+-----+-----+-----+-----+-----+-----+-----+-----+
|T|      Propagation_Delay              |      Override_Interval        |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

The LAN Prune Delay option is used to tune the prune propagation delay on multi-access LANs. The T bit specifies the ability of the sending router to disable join suppression. Propagation_Delay and Override_Interval are time intervals in units of milliseconds. A router originating a LAN Prune Delay option on interface I sets the Propagation_Delay field to the configured value of Propagation_Delay(I) and the value of the Override_Interval field to the value of Override_Interval(I). On a receiving router, the values of the fields are used to tune the value of the Effective_Override_Interval(I) and its derived timer values.

Section 4.3.3 describes how these values affect the behavior of a router.

- o OptionType 3 to 16: reserved to be defined in future versions of this document.
- o OptionType 18: deprecated and should not be used.
- o OptionType 19: DR Priority

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     |                                     |
|      Type = 19                     |      Length = 4                 |
|                                     |                                     |
|                                     |      DR Priority                 |
|                                     |                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

DR Priority is a 32-bit unsigned number and should be considered in the DR election as described in Section 4.3.2.

- o OptionType 20: Generation ID

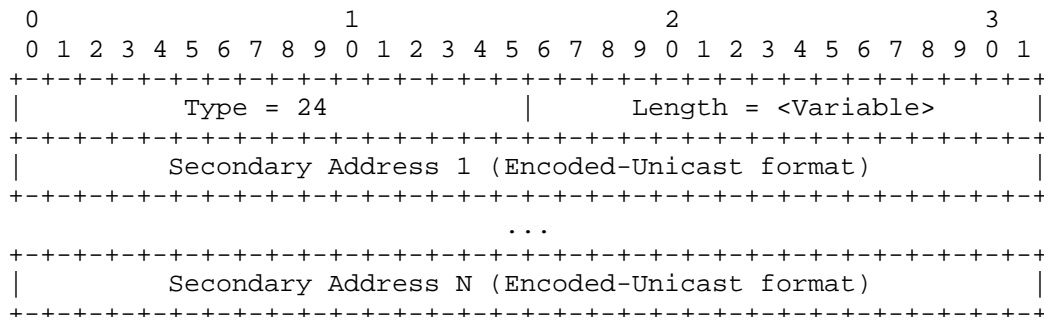
```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     |                                     |
|      Type = 20                     |      Length = 4                 |
|                                     |                                     |
|                                     |      Generation ID               |
|                                     |                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Generation ID is a random 32-bit value for the interface on which the Hello message is sent. The Generation ID is regenerated whenever PIM forwarding is started or restarted on the interface.

o OptionType 24: Address List



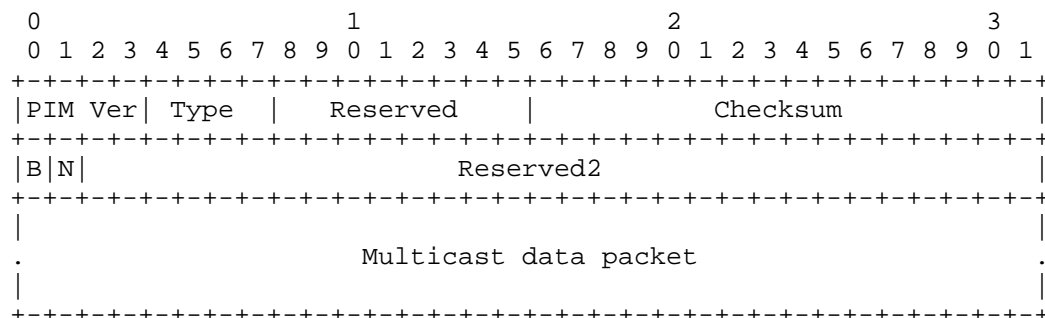
The contents of the Address List Hello option are described in Section 4.3.4. All addresses within a single Address List must belong to the same address family.

OptionTypes 17 through 65000 are assigned by the IANA. OptionTypes 65001 through 65535 are reserved for Private Use, as defined in [9].

Unknown options MUST be ignored and MUST NOT prevent a neighbor relationship from being formed. The "Holdtime" option MUST be implemented; the "DR Priority" and "Generation ID" options SHOULD be implemented. The "Address List" option MUST be implemented for IPv6.

4.9.3. Register Message Format

A Register message is sent by the DR to the RP when a multicast packet needs to be transmitted on the RP-tree. The IP source address is set to the address of the DR, the destination address to the RP's address. The IP TTL of the PIM packet is the system's normal unicast TTL.



PIM Version, Type, Reserved, Checksum

Described in Section 4.9. Note that in order to reduce encapsulation overhead, the checksum for Registers is done only on the first 8 bytes of the packet, including the PIM header and the next 4 bytes, excluding the data packet portion. For interoperability reasons, a message carrying a checksum calculated over the entire PIM Register message should also be accepted. When calculating the checksum, the IPv6 pseudoheader "Upper-Layer Packet Length" is set to 8.

- B** The Border bit. This specification deprecates the Border bit. A router **MUST** set B bit to 0 on transmission and **MUST** ignore this bit on reception.
- N** The Null-Register bit. Set to 1 by a DR that is probing the RP before expiring its local Register-Suppression Timer. Set to 0 otherwise.

Reserved2

Transmitted as zero, ignored on receipt.

Multicast data packet

The original packet sent by the source. This packet must be of the same address family as the encapsulating PIM packet, e.g., an IPv6 data packet must be encapsulated in an IPv6 PIM packet. Note that the TTL of the original packet is decremented before encapsulation, just like any other packet that is forwarded. In addition, the RP decrements the TTL after decapsulating, before forwarding the packet down the shared tree.

For (S,G) Null-Registers, the Multicast data packet portion contains a dummy IP header with S as the source address, G as the destination address. When generating an IPv4 Null-Register message, the fields in the dummy IPv4 header **SHOULD** be filled in according to the following table. Other IPv4 header fields may contain any value that is valid for that field.

Field	Value

IP Version	4
Header Length	5
Checksum	Header checksum
Fragmentation offset	0
More Fragments	0
Total Length	20
IP Protocol	103 (PIM)

On receipt of an (S,G) Null-Register, if the Header Checksum

field is non-zero, the recipient SHOULD check the checksum and discard null registers that have a bad checksum. The recipient SHOULD NOT check the value of any individual fields; a correct IP header checksum is sufficient. If the Header Checksum field is zero, the recipient MUST NOT check the checksum.

With IPv6, an implementation generates a dummy IP header followed by a dummy PIM header with values according to the following table in addition to the source and group. Other IPv6 header fields may contain any value that is valid for that field.

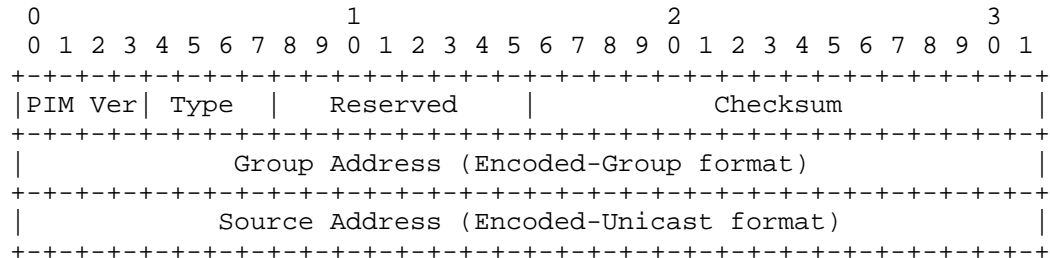
Header Field	Value

IP Version	6
Next Header	103 (PIM)
Length	4
PIM Version	0
PIM Type	0
PIM Reserved	0
PIM Checksum	PIM checksum including IPv6 "pseudo-header"; see Section 4.9

On receipt of an IPv6 (S,G) Null-Register, if the dummy PIM header is present, the recipient SHOULD check the checksum and discard Null-Registers that have a bad checksum.

4.9.4. Register-Stop Message Format

A Register-Stop is unicast from the RP to the sender of the Register message. The IP source address is the address to which the register was addressed. The IP destination address is the source address of the register message.



PIM Version, Type, Reserved, Checksum
Described in Section 4.9.

Group Address

The group address from the multicast data packet in the Register. Format described in Section 4.9.1. Note that for Register-Stops the Mask Len field contains the full address length * 8 (e.g., 32 for IPv4 native encoding), if the message is sent for a single group.

Source Address

The host address of the source from the multicast data packet in the register. The format for this address is given in the Encoded-Unicast address in Section 4.9.1. A special wild card value consisting of an address field of all zeros can be used to indicate any source.

4.9.5. Join/Prune Message Format

A Join/Prune message is sent by routers towards upstream sources and RPs. Joins are sent to build shared trees (RP trees) or source trees (SPT). Prunes are sent to prune source trees when members leave groups as well as sources that do not use the shared tree.

```

0                               1                               2                               3
0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| PIM Ver| Type  |   Reserved   |           Checksum           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Upstream Neighbor Address (Encoded-Unicast format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   Reserved   | Num groups   |           Holdtime           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Multicast Group Address 1 (Encoded-Group format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Number of Joined Sources   |   Number of Pruned Sources   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Joined Source Address 1 (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               .                               |
|                               .                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Joined Source Address n (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Pruned Source Address 1 (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               .                               |
|                               .                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Pruned Source Address n (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               .                               |
|                               .                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Multicast Group Address m (Encoded-Group format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| Number of Joined Sources   |   Number of Pruned Sources   |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Joined Source Address 1 (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               .                               |
|                               .                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Joined Source Address n (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Pruned Source Address 1 (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                               .                               |
|                               .                               |
+-----+-----+-----+-----+-----+-----+-----+-----+
|           Pruned Source Address n (Encoded-Source format)           |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

PIM Version, Type, Reserved, Checksum
Described in Section 4.9.

Unicast Upstream Neighbor Address

The address of the upstream neighbor that is the target of the message. The format for this address is given in the Encoded-Unicast address in Section 4.9.1. For IPv6 the source address used for multicast messages is the link-local address of the interface on which the message is being sent. For IPv4, the source address is the primary address associated with that interface.

Reserved

Transmitted as zero, ignored on receipt.

Holdtime

The amount of time a receiver MUST keep the Join/Prune state alive, in seconds. If the Holdtime is set to '0xffff', the receiver of this message SHOULD hold the state until canceled by the appropriate canceling Join/Prune message, or timed out according to local policy. This may be used with dial-on-demand links, to avoid keeping the link up with periodic Join/Prune messages.

Note that the HoldTime MUST be larger than the
J/P_Override_Interval(I).

Number of Groups

The number of multicast group sets contained in the message.

Multicast group address

For format description, see Section 4.9.1.

Number of Joined Sources

Number of joined source addresses listed for a given group.

Joined Source Address 1 .. n

This list contains the sources for a given group that the sending router will forward multicast datagrams from if received on the interface on which the Join/Prune message is sent.

See Encoded-Source-Address format in Section 4.9.1.

Number of Pruned Sources

Number of pruned source addresses listed for a group.

Pruned Source Address 1 .. n

This list contains the sources for a given group that the sending router does not want to forward multicast datagrams from when received on the interface on which the Join/Prune message is sent.

Within one PIM Join/Prune message, all the Multicast Group Addresses, Joined Source addresses, and Pruned Source 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. This permits maximum implementation flexibility for dual-stack IPv4/IPv6 routers. If a router receives a message with mixed family addresses, it SHOULD only process the addresses that are of the same family as the unicast upstream neighbor address.

4.9.5.1. Group Set Source List Rules

As described above, Join/Prune messages are composed of one or more group sets. Each set contains two source lists, the Joined Sources and the Pruned Sources. This section describes the different types of group sets and source list entries that can exist in a Join/Prune message.

There is one valid group set type:

Group-Specific Set

A Group-Specific Set is represented by a valid IP multicast address in the group address field and the full length of the IP address in the mask length field of the Multicast Group Address.

Each Join/Prune message SHOULD NOT contain more than one group-specific set for the same IP multicast address. Each group-specific set may contain (*,G), (S,G,rpt), and (S,G) source list entries in the Joined or Pruned lists.

(*,G)

The (*,G) source list entry is used in Join/Prune messages sent towards the RP for the specified group. It expresses interest (or lack thereof) in receiving traffic sent to the group through the Rendezvous-Point shared tree. There MUST only be one such entry in both the Joined and Pruned lists of a group-specific set.

(*,G) source list entries have the Source-Address set to the address of the RP for group G, the Source-Address Mask-Len set to the full length of the IP address, and both the WC and RPT bits of the Encoded-Source-Address set.

(S,G,rpt)

The (S,G,rpt) source list entry is used in Join/Prune messages sent towards the RP for the specified group. It expresses interest (or lack thereof) in receiving traffic through the shared tree sent by the specified source to this group. For each source address, the entry MUST exist in only one of the Joined and Pruned source lists of a group-specific set, but not both.

(S,G,rpt) source list entries have the Source-Address set to the address of the source S, the Source-Address Mask-Len set to the full length of the IP address, and the WC bit cleared and the RPT bit set in the Encoded-Source-Address.

(S,G)

The (S,G) source list entry is used in Join/Prune messages sent towards the specified source. It expresses interest (or lack thereof) in receiving traffic through the shortest path tree sent by the source to the specified group. For each source address, the entry MUST exist in only one of the Joined and Pruned source lists of a group-specific set, but not both.

(S,G) source list entries have the Source-Address set to the address of the source S, the Source-Address Mask-Len set to the full length of the IP address, and both the WC and RPT bits of the Encoded-Source-Address cleared.

The rules described above are sufficient to prevent invalid combinations of source list entries in group-specific sets. There are, however, a number of combinations that have a valid interpretation but that are not generated by the protocol as described in this specification:

- o Combining a (*,G) Join and an (S,G,rpt) Join entry in the same message is redundant as the (*,G) entry covers the information provided by the (S,G,rpt) entry.
- o The same applies for a (*,G) Prune and an (S,G,rpt) Prune.
- o The combination of a (*,G) Prune and an (S,G,rpt) Join is also not generated. (S,G,rpt) Joins are only sent when the router is receiving all traffic for a group on the shared tree and it wishes to indicate a change for the particular source. As a (*,G) prune indicates that the router no longer wishes to receive shared tree traffic, the (S,G,rpt) Join would be meaningless.
- o As Join/Prune messages are targeted to a single PIM neighbor, including both an (S,G) Join and an (S,G,rpt) Prune in the same

message is usually redundant. The (S,G) Join informs the neighbor that the sender wishes to receive the particular source on the shortest path tree. It is therefore unnecessary for the router to say that it no longer wishes to receive it on the shared tree. However, there is a valid interpretation for this combination of entries. A downstream router may have to instruct its upstream only to start forwarding a specific source once it has started receiving the source on the shortest-path tree.

- o The combination of an (S,G) Prune and an (S,G,rpt) Join could possibly be used by a router to switch from receiving a particular source on the shortest-path tree back to receiving it on the shared tree (provided that the RPF neighbor for the shortest-path and shared trees is common). However, Sparse-Mode PIM does not provide a mechanism for explicitly switching back to the shared tree.

The rules are summarized in the tables below.

	Join (* ,G)	Prune (* ,G)	Join (S,G,rpt)	Prune (S,G,rpt)	Join (S,G)	Prune (S,G)
Join (* ,G)	-	no	?	yes	yes	yes
Prune (* ,G)	no	-	?	?	yes	yes
Join (S,G,rpt)	?	?	-	no	yes	?
Prune (S,G,rpt)	yes	?	no	-	yes	?
Join (S,G)	yes	yes	yes	yes	-	no
Prune (S,G)	yes	yes	?	?	no	-

yes Allowed and expected.

no Combination is not allowed by the protocol and MUST NOT be generated by a router. A router MAY accept these messages, but the result is undefined. An error message MAY be logged to the administrator in a rate-limited manner.

? Combination not expected by the protocol, but well-defined. A router MAY accept it but SHOULD NOT generate it.

The order of source list entries in a group set source list is not important, except where limited by the packet format itself.

4.9.5.2. Group Set Fragmentation

When building a Join/Prune for a particular neighbor, a router should try to include in the message as much of the information it needs to convey to the neighbor as possible. This implies adding one group set for each multicast group that has information pending transmission and within each set including all relevant source list entries.

On a router with a large amount of multicast state, the number of

entries that must be included may result in packets that are larger than the maximum IP packet size. In most such cases, the information may be split into multiple messages.

There is an exception with group sets that contain a (*,G) Joined source list entry. The group set expresses the router's interest in receiving all traffic for the specified group on the shared tree, and it MUST include an (S,G,rpt) Pruned source list entry for every source that the router does not wish to receive. This list of (S,G,rpt) Pruned source-list entries MUST NOT be split in multiple messages.

If only N (S,G,rpt) Prune entries fit into a maximum-sized Join/Prune message, but the router has more than N (S,G,rpt) Prunes to add, then the router MUST choose to include the first N (numerically smallest in network byte order) IP addresses, and the rest are ignored (not included).

4.9.6. Assert Message Format

The Assert message is used to resolve forwarder conflicts between routers on a link. It is sent when a router receives a multicast data packet on an interface on which the router would normally have forwarded that packet. Assert messages may also be sent in response to an Assert message from another router.

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
PIM Ver										Type										Reserved										Checksum									
Group Address (Encoded-Group format)																																							
Source Address (Encoded-Unicast format)																																							
R										Metric Preference																													
Metric																																							

PIM Version, Type, Reserved, Checksum
Described in Section 4.9.

Group Address

The group address for which the router wishes to resolve the forwarding conflict. This is an Encoded-Group address, as specified in Section 4.9.1.

Source Address

Source address for which the router wishes to resolve the forwarding conflict. The source address MAY be set to zero for (*,G) asserts (see below). The format for this address is given in Encoded-Unicast-Address in Section 4.9.1.

R RPT-bit is a 1-bit value. The RPT-bit is set to 1 for Assert(*,G) messages and 0 for Assert(S,G) messages.

Metric Preference

Preference value assigned to the unicast routing protocol that provided the route to the multicast source or Rendezvous-Point.

Metric

The unicast routing table metric associated with the route used to reach the multicast source or Rendezvous-Point. The metric is in units applicable to the unicast routing protocol used.

Assert messages can be sent to resolve a forwarding conflict for all traffic to a given group or for a specific source and group.

Assert(S,G)

Source-specific asserts are sent by routers forwarding a specific source on the shortest-path tree (SPTbit is TRUE). (S,G) Asserts have the Group-Address field set to the group G and the Source-Address field set to the source S. The RPT-bit is set to 0, the Metric-Preference is set to MRIB.pref(S) and the Metric is set to MRIB.metric(S).

Assert(*,G)

Group-specific asserts are sent by routers forwarding data for the group and source(s) under contention on the shared tree. (*,G) asserts have the Group-Address field set to the group G. For data-triggered Asserts, the Source-Address field MAY be set to the IP source address of the data packet that triggered the Assert and is set to zero otherwise. The RPT-bit is set to 1, the Metric-Preference is set to MRIB.pref(RP(G)), and the Metric is set to MRIB.metric(RP(G)).

4.10. PIM Timers

PIM-SM maintains the following timers, as discussed in Section 4.1. All timers are countdown timers; they are set to a value and count down to zero, at which point they typically trigger an action. Of course they can just as easily be implemented as count-up timers, where the absolute expiry time is stored and compared against a real-time clock, but the language in this specification assumes that they count downwards to zero.

Global Timers

Per interface (I):

 Hello Timer: HT(I)

Per neighbor (N):

 Neighbor Liveness Timer: NLT(N,I)

Per Group (G):

 (*,G) Join Expiry Timer: ET(*,G,I)

 (*,G) Prune-Pending Timer: PPT(*,G,I)

 (*,G) Assert Timer: AT(*,G,I)

Per Source (S):

 (S,G) Join Expiry Timer: ET(S,G,I)

 (S,G) Prune-Pending Timer: PPT(S,G,I)

 (S,G) Assert Timer: AT(S,G,I)

 (S,G,rpt) Prune Expiry Timer: ET(S,G,rpt,I)

 (S,G,rpt) Prune-Pending Timer: PPT(S,G,rpt,I)

Per Group (G):

 (*,G) Upstream Join Timer: JT(*,G)

Per Source (S):

 (S,G) Upstream Join Timer: JT(S,G)

 (S,G) Keepalive Timer: KAT(S,G)

 (S,G,rpt) Upstream Override Timer: OT(S,G,rpt)

At the DRs or relevant Assert Winners only:

Per Source,Group pair (S,G):

 Register-Stop Timer: RST(S,G)

4.11. Timer Values

When timers are started or restarted, they are set to default values. This section summarizes those default values.

Note that protocol events or configuration may change the default value of a timer on a specific interface. When timers are initialized in this document, the value specific to the interface in context must be used.

Some of the timers listed below (Prune-Pending, Upstream Join, Upstream Override) can be set to values that depend on the settings of the Propagation_Delay and Override_Interval of the corresponding interface. The default values for these are given below.

Variable Name: Propagation_Delay(I)

Value Name	Value	Explanation
Propagation_delay_default	0.5 secs	Expected propagation delay over the local link.

The default value of the Propagation_delay_default is chosen to be relatively large to provide compatibility with older PIM implementations.

Variable Name: Override_Interval(I)

Value Name	Value	Explanation
t_override_default	2.5 secs	Default delay interval over which to randomize when scheduling a delayed Join message.

Timer Name: Hello Timer (HT(I))

Value Name	Value	Explanation
Hello_Period	30 secs	Periodic interval for Hello messages.
Triggered_Hello_Delay	5 secs	Randomized interval for initial Hello message on bootup or triggered Hello message to a rebooting neighbor.

At system power-up, the timer is initialized to `rand(0, Triggered_Hello_Delay)` to prevent synchronization. When a new or rebooting neighbor is detected, a responding Hello is sent within `rand(0, Triggered_Hello_Delay)`.

Timer Name: Neighbor Liveness Timer (NLT(N,I))

Value Name	Value	Explanation
Default_Hello_Holdtime	$3.5 * \text{Hello_Period}$	Default holdtime to keep neighbor state alive
Hello_Holdtime	from message	Holdtime from Hello Message Holdtime option.

The Holdtime in a Hello Message should be set to $(3.5 * \text{Hello_Period})$, giving a default value of 105 seconds.

Timer Names: Expiry Timer (`ET(*,G,I)`, `ET(S,G,I)`, `ET(S,G,rpt,I)`)

Value Name	Value	Explanation
J/P_HoldTime	from message	Holdtime from Join/Prune Message

See details of `JT(*,G)` for the Holdtime that is included in Join/Prune Messages.

Timer Names: Prune-Pending Timer (PPT(*,G,I), PPT(S,G,I),
PPT(S,G,rpt,I))

Value Name	Value	Explanation
J/P_Override_Interval(I)	Default: Effective_ Propagation_ Delay(I) + EffectiveOverride_ Interval(I)	Short period after a join or prune to allow other routers on the LAN to override the join or prune

Note that both the Effective_Propagation_Delay(I) and the Effective_Override_Interval(I) are interface-specific values that may change when Hello messages are received (see Section 4.3.3).

Timer Names: Assert Timer (AT(*,G,I), AT(S,G,I))

Value Name	Value	Explanation
Assert_Override_Interval	Default: 3 secs	Short interval before an assert times out where the assert winner resends an Assert message
Assert_Time	Default: 180 secs	Period after last assert before assert state is timed out

Note that for historical reasons, the Assert message lacks a Holdtime field. Thus, changing the Assert Time from the default value is not recommended.

Timer Names: Upstream Join Timer (JT(*,G), JT(S,G))

Value Name	Value	Explanation
t_periodic	Default: 60 secs	Period between Join/Prune Messages
t_suppressed	rand(1.1 * t_periodic, 1.4 * t_periodic) when Suppression_Enabled(I) is true, 0 otherwise	Suppression period when someone else sends a J/P message so we don't need to do so.
t_override	rand(0, Effective_Override_Interval(I))	Randomized delay to prevent response implosion when sending a join message to override someone else's Prune message.

t_periodic may be set to take into account such things as the configured bandwidth and expected average number of multicast route entries for the attached network or link (e.g., the period would be longer for lower-speed links, or for routers in the center of the network that expect to have a larger number of entries). If the Join/Prune-Period is modified during operation, these changes should be made relatively infrequently, and the router should continue to refresh at its previous Join/Prune-Period for at least Join/Prune-Holdtime, in order to allow the upstream router to adapt.

The holdtime specified in a Join/Prune message should be set to (3.5 * t_periodic).

t_override depends on the Effective Override Interval of the upstream interface, which may change when Hello messages are received.

t_suppressed depends on the Suppression State of the upstream interface (Section 4.3.3) and becomes zero when suppression is disabled.

Timer Name: Upstream Override Timer (OT(S,G,rpt))

Value Name	Value	Explanation
t_override	see Upstream Join Timer	see Upstream Join Timer

The upstream Override Timer is only ever set to t_override; this value is defined in the section on Upstream Join Timers.

Timer Name: Keepalive Timer (KAT(S,G))

Value Name	Value	Explanation
Keepalive_Period	Default: 210 secs	Period after last (S,G) data packet during which (S,G) Join state will be maintained even in the absence of (S,G) Join messages.
RP_Keepalive_Period	(3 * Register_Suppression_Time) + Register_Probe_Time	As Keepalive_Period, but at the RP when a Register-Stop is sent.

The normal keepalive period for the KAT(S,G) defaults to 210 seconds. However, at the RP, the keepalive period must be at least the Register_Suppression_Time, or the RP may time out the (S,G) state before the next Null-Register arrives. Thus, the KAT(S,G) is set to max(Keepalive_Period, RP_Keepalive_Period) when a Register-Stop is sent.

Timer Name: Register-Stop Timer (RST(S,G))

Value Name	Value	Explanation
Register_Suppression_Time	Default: 60 secs	Period during which a DR stops sending Register-encapsulated data to the RP after receiving a Register-Stop message.
Register_Probe_Time	Default: 5 secs	Time before RST expires when a DR may send a Null-Register to the RP to cause it to resend a Register-Stop message.

If the Register_Suppression_Time or the Register_Probe_Time are configured to values other than the defaults, it MUST be ensured that the value of the Register_Probe_Time is less than half the value of the Register_Suppression_Time to prevent a possible negative value in the setting of the Register-Stop Timer.

5. IANA Considerations

5.1. PIM Address Family

The PIM Address Family field was chosen to be 8 bits as a tradeoff between packet format and use of the IANA assigned numbers. Because when the PIM packet format was designed only 15 values were assigned for Address Families, and large numbers of new Address Family values were not envisioned, 8 bits seemed large enough. However, the IANA assigns Address Families in a 16-bit field. Therefore, the PIM Address Family is allocated as follows:

Values 0 through 127 are designated to have the same meaning as IANA-assigned Address Family Numbers [7].

Values 128 through 250 are designated to be assigned for PIM by the IANA based upon IESG Approval, as defined in [9].

Values 251 through 255 are designated for Private Use, as defined in [9].

5.2. PIM Hello Options

Values 17 through 65000 are to be assigned by the IANA. Since the space is large, they may be assigned as First Come First Served as defined in [9]. Such assignments are valid for one year and may be renewed. Permanent assignments require a specification (see "Specification Required" in [9].)

6. Security Considerations

This section describes various possible security concerns related to the PIM-SM protocol, including a description of how to use IPsec to secure the protocol. The reader is referred to [15] and [16] for further discussion of PIM-SM and multicast security. The IPsec authentication header [8] MAY be used to provide data integrity protection and groupwise data origin authentication of PIM protocol messages. Authentication of PIM messages can protect against unwanted behaviors caused by unauthorized or altered PIM messages.

Note that PIM relies upon an MRIB populated outside of PIM so securing the sources of change to the MRIB is RECOMMENDED.

6.1. Attacks Based on Forged Messages

The extent of possible damage depends on the type of counterfeit messages accepted. We next consider the impact of possible forgeries, including forged link-local (Join/Prune, Hello, and Assert) and forged unicast (Register and Register-Stop) messages.

6.1.1. Forged Link-Local Messages

Join/Prune, Hello, and Assert messages are all sent to the link-local ALL_PIM_ROUTERS multicast addresses and thus are not forwarded by a compliant router. A forged message of this type can only reach a LAN if it was sent by a local host or if it was allowed onto the LAN by a compromised or non-compliant router.

1. A forged Join/Prune message can cause multicast traffic to be delivered to links where there are no legitimate requesters, potentially wasting bandwidth on that link. A forged leave message on a multi-access LAN is generally not a significant attack in PIM, because any legitimately joined router on the LAN would override the leave with a join before the upstream router stops forwarding data to the LAN.

2. By forging a Hello message, an unauthorized router can cause itself to be elected as the designated router on a LAN. The designated router on a LAN is (in the absence of asserts) responsible for forwarding traffic to that LAN on behalf of any local members. The designated router is also responsible for register-encapsulating to the RP any packets that are originated by hosts on the LAN. Thus, the ability of local hosts to send and receive multicast traffic may be compromised by a forged Hello message.
3. By forging an Assert message on a multi-access LAN, an attacker could cause the legitimate designated forwarder to stop forwarding traffic to the LAN. Such a forgery would prevent any hosts downstream of that LAN from receiving traffic.

6.1.2. Forged Unicast Messages

Register messages and Register-Stop messages are forwarded by intermediate routers to their destination using normal IP forwarding. Without data origin authentication, an attacker who is located anywhere in the network may be able to forge a Register or Register-Stop message. We consider the effect of a forgery of each of these messages next.

1. By forging a Register message, an attacker can cause the RP to inject forged traffic onto the shared multicast tree.
2. By forging a Register-stop message, an attacker can prevent a legitimate DR from Registering packets to the RP. This can prevent local hosts on that LAN from sending multicast packets.

The above two PIM messages are not changed by intermediate routers and need only be examined by the intended receiver. Thus, these messages can be authenticated end-to-end, using AH. Attacks on Register and Register-Stop messages do not apply to a PIM-SSM-only implementation, as these messages are not required for PIM-SSM.

6.2. Non-Cryptographic Authentication Mechanisms

A PIM router SHOULD provide an option to limit the set of neighbors from which it will accept Join/Prune, Assert, and Hello messages. Either static configuration of IP addresses or an IPsec security association MAY be used. Furthermore, a PIM router SHOULD NOT accept protocol messages from a router from which it has not yet received a valid Hello message.

A Designated Router MUST NOT register-encapsulate a packet and send it to the RP unless the source address of the packet is a legal

address for the subnet on which the packet was received. Similarly, a Designated Router SHOULD NOT accept a Register-Stop packet whose IP source address is not a valid RP address for the local domain.

An implementation SHOULD provide a mechanism to allow an RP to restrict the range of source addresses from which it accepts Register-encapsulated packets.

All options that restrict the range of addresses from which packets are accepted MUST default to allowing all packets.

6.3. Authentication Using IPsec

The IPsec [8] transport mode using the Authentication Header (AH) is the recommended method to prevent the above attacks against PIM. The specific AH authentication algorithm and parameters, including the choice of authentication algorithm and the choice of key, are configured by the network administrator. When IPsec authentication is used, a PIM router should reject (drop without processing) any unauthorized PIM protocol messages.

To use IPsec, the administrator of a PIM network configures each PIM router with one or more security associations (SAs) and associated Security Parameter Indexes (SPIs) that are used by senders to authenticate PIM protocol messages and are used by receivers to authenticate received PIM protocol messages. This document does not describe protocols for establishing SAs. It assumes that manual configuration of SAs is performed, but it does not preclude the use of a negotiation protocol such as the Internet Key Exchange [14] to establish SAs.

IPsec [8] provides protection against replayed unicast and multicast messages. The anti-replay option for IPsec SHOULD be enabled on all SAs.

The following sections describe the SAs required to protect PIM protocol messages.

6.3.1. Protecting Link-Local Multicast Messages

The network administrator defines an SA and SPI that are to be used to authenticate all link-local PIM protocol messages (Hello, Join/Prune, and Assert) on each link in a PIM domain.

IPsec [8] allows (but does not require) different Security Policy Databases (SPD) for each router interface. If available, it may be desirable to configure the Security Policy Database at a PIM router such that all incoming and outgoing Join/Prune, Assert, and Hello

packets use a different SA for each incoming or outgoing interface.

6.3.2. Protecting Unicast Messages

IPsec can also be used to provide data origin authentication and data integrity protection for the Register and Register-Stop unicast messages.

6.3.2.1. Register Messages

The Security Policy Database at every PIM router is configured to select an SA to use when sending PIM Register packets to each rendezvous point.

In the most general mode of operation, the Security Policy Database at each DR is configured to select a unique SA and SPI for traffic sent to each RP. This allows each DR to have a different authentication algorithm and key to talk to the RP. However, this creates a daunting key management and distribution problem for the network administrator. Therefore, it may be preferable in PIM domains where all Designated Routers are under a single administrative control that the same authentication algorithm parameters (including the key) be used for all Registered packets in a domain, regardless of who are the RP and the DR.

In this "single shared key" mode of operation, the network administrator must choose an SPI for each DR that will be used to send the PIM protocol packets. The Security Policy Database at every DR is configured to select an SA (including the authentication algorithm, authentication parameters, and this SPI) when sending Register messages to the RP.

By using a single authentication algorithm and associated parameters, the key distribution problem is simplified. Note, however, that this method has the property that, in order to change the authentication method or authentication key used, all routers in the domain must be updated.

6.3.2.2. Register-Stop Messages

Similarly, the Security Policy Database at each Rendezvous Point should be configured to choose an SA to use when sending Register-Stop messages. Because Register-Stop messages are unicast to the destination DR, a different SA and a potentially unique SPI are required for each DR.

In order to simplify the management problem, it may be acceptable use the same authentication algorithm and authentication parameters,

regardless of the sending RP and regardless of the destination DR. Although a unique SA is needed for each DR, the same authentication algorithm and authentication algorithm parameters (secret key) can be shared by all DRs and by all RPs.

6.4. Denial-of-Service Attacks

There are a number of possible denial-of-service attacks against PIM that can be caused by generating false PIM protocol messages or even by generating false traffic. Authenticating PIM protocol traffic prevents some, but not all, of these attacks. Three of the possible attacks include:

- Sending packets to many different group addresses quickly can be a denial-of-service attack in and of itself. This will cause many register-encapsulated packets, loading the DR, the RP, and the routers between the DR and the RP.
- Forging Join messages can cause a multicast tree to get set up. A large number of forged joins can consume router resources and result in denial of service.

7. Acknowledgements

PIM-SM was designed over many years by a large group of people, including ideas, comments, and corrections from Deborah Estrin, Dino Farinacci, Ahmed Helmy, David Thaler, Steve Deering, Van Jacobson, C. Liu, Puneet Sharma, Liming Wei, Tom Pusateri, Tony Ballardie, Scott Brim, Jon Crowcroft, Paul Francis, Joel Halpern, Horst Hodel, Polly Huang, Stephen Ostrowski, Lixia Zhang, Girish Chandranmenon, Brian Haberman, Hal Sandick, Mike Mroz, Garry Kump, Pavlin Radoslavov, Mike Davison, James Huang, Christopher Thomas Brown, and James Lingard.

Thanks are due to the American Licorice Company, for its obscure but possibly essential role in the creation of this document.

8. Normative References

- [1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [2] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, October 2002.
- [3] Deering, S., "Host extensions for IP multicasting", STD 5, RFC 1112, August 1989.
- [4] Deering, S., Fenner, W., and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6", RFC 2710, October 1999.
- [5] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [6] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.
- [7] IANA, "Address Family Numbers",
<<http://www.iana.org/assignments/address-family-numbers>>.
- [8] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [9] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

9. Informative References

- [10] Bates, T., Rekhter, Y., Chandra, R., and D. Katz, "Multiprotocol Extensions for BGP-4", RFC 4760, January 2007.
- [11] Bhaskar, N., Gall, A., Lingard, J., and S. Venaas, "Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM)", RFC 5059, January 2008.
- [12] Black, D., "Differentiated Services and Tunnels", RFC 2983, October 2000.
- [13] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast" (BIDIR-PIM), RFC 5015, October 2007.
- [14] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange (IKEv2) Protocol", RFC 5996, September 2010.

- [15] Savola, P., Lehtonen, R., and D. Meyer, "Protocol Independent Multicast - Sparse Mode (PIM-SM) Multicast Routing Security Issues and Enhancements", RFC 4609, August 2006.
- [16] Savola, P. and J. Lingard, "Host Threats to Protocol Independent Multicast (PIM)", RFC 5294, August 2008.
- [17] Savola, P. and B. Haberman, "Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address", RFC 3956, November 2004.

Authors' Addresses

Bill Fenner
AT&T Labs - Research
1 River Oaks Place
San Jose, CA 95134

EMail: fenner@research.att.com

Mark Handley
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
United Kingdom

EMail: M.Handley@cs.ucl.ac.uk

Hugh Holbrook
Arastra, Inc.
P.O. Box 10905
Palo Alto, CA 94303

EMail: holbrook@arastra.com

Isidor Kouvelas
Cisco Systems, Inc.
170 W. Tasman Drive
San Jose, CA 95134

EMail: kouvelas@cisco.com

Rishabh Parekh
Cisco Systems, Inc.
170 W. Tasman Drive
San Jose, CA 95134

EMail: riparekh@cisco.com

Zhaohui (Jeffrey) Zhang
Juniper Networks
10 Technology Park Drive
Westford, MA 01886

Email: zzhang@juniper.net

Lianshu Zheng
Huawei Technologies Co., Ltd
No. 3 Xinx Road, Shang-di, Hai-dian District
Beijing 100085
China

Email: verozheng@huawei.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 14, 2011

A. Karan
C. Filsfils
D. Farinacci
Cisco Systems, Inc.
B. Decraene
France Telecom
N. Leymann
U. Joorde
Deutsche Telekom
T. Telkamp
Cariden Technologies, Inc.
March 13, 2011

Multicast only Fast Re-Route
draft-karan-mofrr-01

Abstract

As IPTV deployments grow in number and size, service providers are looking for solutions that minimize the service disruption due to faults in the IP network carrying the packets for these services. This draft describes a mechanism for minimizing packet loss in a network when node or link failures occur. Multicast only Fast Re-Route (MoFRR) works by making simple enhancements to multicast routing protocols such as PIM.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 14, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Conventions used in this document	3
1.2. Terminology	3
2. Basic Overview	4
3. Topologies for MoFRR	4
3.1. Dual-Plane Topology	4
4. Detecting Failures	7
5. ECMP-mode MoFRR	8
6. Non-ECMP-mode MoFRR	8
6.1. Variation	10
7. Keep It Simple Principle	10
8. Capacity Planning for MoFRR	10
9. Other Applications	11
10. Security Considerations	12
11. Acknowledgments	12
12. References	12
12.1. Normative References	12
12.2. Informative References	12
Authors' Addresses	12

1. Introduction

Multiple techniques have been developed and deployed to improve service guarantees, both for multicast video traffic and Video on Demand traffic. Most existing solutions are geared towards finding an alternate path around one or more failed network elements (link, node, path failures).

This draft describes a mechanism for minimizing packet loss in a network when node or link failures occur. Multicast only Fast Re-Route (MoFRR) works by making simple changes to the way selected routers use multicast protocols such as PIM. No changes to the protocols themselves are required. With MoFRR, in many cases, multicast routing protocols don't necessarily have to depend on or have to wait on unicast routing protocols to detect network failures.

MoFRR involves transmitting a multicast join message from a receiver towards a source on a primary path and transmitting a secondary multicast join message from the receiver towards the source on a backup path. Data packets are received from the primary and secondary paths. The redundant packets are discarded at topology merge points using RPF checks. When a failure is detected on the primary path, the repair occurs by changing the interface on which packets are accepted to the secondary interface. Since the repair is local, it is fast - greatly improving convergence times in the event of node or link failures on the primary path.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

MoFRR : Multicast only Fast Re-Route.

ECMP : Equal Cost Multi-Path.

Primary Join : Multicast join message sent from receiver towards the source on the primary path.

Secondary Join : Multicast join message sent from receiver towards the source on the secondary path.

2. Basic Overview

MoFRR uses standard PIM JOIN/PRUNE messages to set up a primary and a secondary multicast forwarding path by establishing a primary and a secondary RPF interface on each router that receives a PIM join. The outgoing interface list remains the same.

Data packets are received from the primary and backup paths. Redundant packets received on the secondary RPF interface are discarded because of an RPF failure. When the router detects a forwarding failure in the primary path, it changes RPF to the secondary path and immediately has packets available to forward out each outgoing interface.

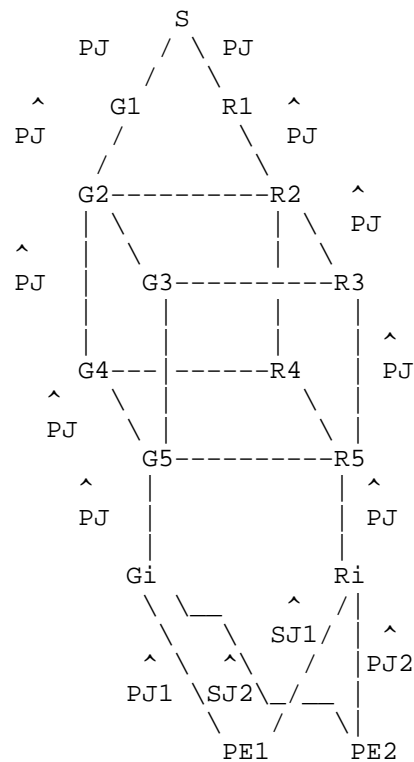
The primary and secondary MoFRR forwarding paths should not use the same nodes or links. This may be configured or determined by computations described in this document.

Note, the impact of additional amount of data on the network is mitigated when group membership is densely populated. When a part of the network has redundant data flowing, join latency for new joining members is reduced because joins don't have to propagate far to get to on-tree routers.

3. Topologies for MoFRR

MoFRR works best in topologies illustrated in the figure below. MoFRR may be enabled on any router in the network. In the figures below, MoFRR is shown enabled on the Provider Edge (PE) routers to illustrate one way in which the technology may be deployed.

3.1. Dual-Plane Topology



PJ = Primary Join
 SJ = Secondary Join

FIG1. Two-Plane Network Design

The topology has two planes, a primary plane and a secondary plane that are fully disjoint from each other all the way into the POPs. This two plane design is common in service provider networks as it eliminates single point of failures in their core network. The links marked PJ indicate the normal path of how the PIM joins flow from the POPs towards the source of the network. Multicast streams, especially for the densely watched channels, typically flow along both the planes in the network anyways.

The only change MoFRR adds to this is on the links marked SJ where the PE routers send a secondary PIM joins to their ECMP neighbor towards the source. As a result of this, each PE router receives two copies of the same stream, one from the primary plane and the other from the secondary plane. As a result of normal multicast RPF checks the multicast stream received over the primary path is accepted and forwarded to the downstream links. The copy of the stream received

on the secondary path is discarded.

When a router detects a routing failure on its primary RPF interface, it will switch to the secondary RPF interface and accept packets on that stream. If the failure is repaired the router may switch back. The primary and secondary path have only local context and not end-to-end context.

As one can see, MoFRR achieves the faster convergence by pre-building the secondary multicast tree and receiving the traffic on that secondary path. The example discussed above is a simple case where there are two ECMP paths from each PE device towards the source, one along the primary plane and one along the secondary. In cases where the topology is asymmetric or is a ring, this ECMP nature does not hold, and additional rules have to be taken into account to choose when and where to send the secondary PIM joins.

MoFRR is appealing in such topologies for the following reasons:

1. Ease of deployment and simplicity: the functionality is only required on the PE devices although it may be configured on all routers in the topology. Furthermore, each PE device can be enabled separately. PEs not enabled for MoFRR do not see any change or degradation. Inter-operability testing is not required as there is no PIM protocol change.
2. End-to-end failure detection and recovery: any failure along the path from the source to the PE can be detected and repaired with the secondary disjoint stream.
3. Capacity Efficiency: as illustrated in the previous example, the PIM trees corresponding to IPTV channels cover the backbone and distribution topology in a very dense manner. As a consequence, the secondary joins graft into the normal PIM trees (ie. trees signaled by PIM without MoFRR extension) at the aggregation level and hence do not demand any extra capacity either on the distribution links or in the backbone. They simply use the capacity that is normally used, without any duplication. This is different from conventional FRR mechanisms which often duplicate the capacity requirements (the backup path crosses links/nodes which already carry the primary/normal tree and hence twice as much capacity is required).
4. Loop free: the secondary PIM join is sent on an ECMP disjoint path. By definition, the neighbor receiving this secondary PIM join is closer to the source and hence will not send a PIM join back.

The topology we just analyzed is very frequent and can be modelled as per Fig2. The PE has two ECMP disjoint paths to the source. Each ECMP path uses a disjoint plane of the network.

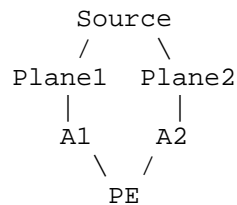


FIG2. PE is dual-homed to Dual-Plane Backbone

Another frequent topology is described in Fig 3. PEs are grouped by pairs. In each pair, each PE is connected to a different plane. Each PE has one single shortest-path to a source (via its connected plane). There is no ECMP like in Fig 2. However, there is clearly a way to provide MoFRR benefits as each PE can offer a disjoint secondary path to the other plane PE (via the disjoint path).

MoFRR secondary neighbor selection process needs to be extended in this case as one cannot simply rely on using an ECMP path as secondary neighbor. This extension is referred to as non-ecmp extension and is described later in the document.

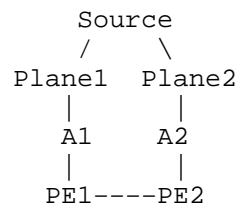


FIG3. PEs are connected in pairs to Dual-Plane Backbone

4. Detecting Failures

Once the two paths are established, the next step is detecting a failure on the primary path to know when to switch to the backup path.

A first option consists of comparing the packets received on the primary and secondary streams but only forwarding one of them -- the first one received, no matter which interface it is received on. Zero packet loss is possible for RTP-based streams.

A second option assumes a minimum known packet rate for a given data stream. If a packet is not received on the primary RPF within this time frame, the router assumes primary path failure and switches to the secondary RPF interface. 50msec switchover is possible.

A third option leverages the significant improvements of the IGP convergence speed. When the primary path to the source is withdrawn by the IGP, the MoFRR-enabled router switches over to the backup path, the RPF interface is changed to the secondary RPF interface. Since the secondary path is already in place, and assuming it is disjoint from the primary path, convergence times would not include the time required to build a new tree and hence are smaller. Realistic availability requirements (sub-second to sub-200msec) should be possible.

A fourth option consists in leveraging connected link failure. This option makes sense when MoFRR is deployed across the network (not only at PE).

5. ECMP-mode MoFRR

If the IGP installs two ECMP paths to the source and if the (S, G) PIM state is enabled for ECMP-Mode MoFRR, the router installs them as primary RPF and secondary RPF. It sends a PIM join to both RPF entries. Only packets received from the primary RPF entry are processed. Packets received from the secondary RPF are dropped (equivalent to an RPF failure).

The selected primary RPF interface should be the same as if MoFRR extension was not enabled.

If more than two ECMP paths exist, two are selected as primary and secondary RPF interfaces. Information from the IGP link-state topology could be leveraged to optimize this selection.

Note, MoFRR does not restrict the number of paths on which joins are sent. Implementations may use as many paths as are configured.

6. Non-ECMP-mode MoFRR

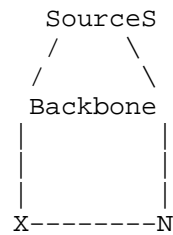


Fig5. Non-ECMP-Mode MoFRR

X is configured for MoFRR for state (S, G)
 R(X) is Xs RPF to S
 N is a neighbor of X
 R(N) is Ns RPF to S
 xs represents the IGP metric from X to S
 ns represents the IGP metric from N to S
 xn represents the IGP metric from X to N

A router X configured for non-ECMP-mode MoFRR for (S, G) sends a primary PIM join to its primary RPF R(X) and a secondary PIM Join to a neighbor N if the following three conditions are met.

C1: $xs < xn + ns$
 C2: $ns < nx + xs$
 C3: X cannot send a secondary join to N if N is the only member of the OIF list

The first condition ensures that N is not on the primary branch from X to S.

The second condition ensures that X is not on the primary branch from N to S.

These two conditions ensure that at least locally the two paths are disjoint.

The third condition is required to break control-plane loops which could occur in some scenarios.

For example in FIG3, if PE1 and PE2 have received an igmp request for (S, G), they will both send a primary PIM join on their plane and a secondary PIM join to the neighbor PE. If their receivers would leave at the same time, it could be possible for the (S, G) states on PE1 and PE2 to never get deleted as each PE refresh each other via the secondary PIM joins (remember that a secondary PIM join is not distinguishable from a primary PIM join. MoFRR does not require any PIM protocol modification).

A control-plane loop occurs when two nodes keep a state forever due to the secondary joins they send to each other. This forever condition is not acceptable as no real receiver is connected to the nodes (directly via IGMP or indirectly via PIM). Rule 3 prevents this case as it prevents the mutual refresh of secondary joins and it applies it in the specific case where there is no real receiver connected.

6.1. Variation

Rule R3 can be removed if Rule 2 is restricted as follows:

R2p: $ns < xs$

This ensures that X only sends a secondary join to a neighbor N who is strictly closer to the source than X is. By reciprocity, N will thus never be able to send an secondary join to the same source via X. The strictly smaller than is key here.

Note that this non-ECMP-mode MoFRR variation does not support the square topology and hence is less preferred.

7. Keep It Simple Principle

Many Service Providers devise their topology such that PEs have disjoint paths to the multicast sources. MoFRR leverages the existence of these disjoint paths without any PIM protocol modification. Interoperability testing is thus not required. In such topologies, MoFRR only needs to be deployed on the PE devices. Each PE device can be enabled one by one. PEs not enabled for MoFRR do not see any change or degradation.

Multicast streams with Tight SLA requirements are often characterized by a continuous high packet rate (SD video has a continuous interpacket gap of ~ 3msec). MoFRR simply leverages the stream characteristic to detect any failures along the primary branch and switch-over on the secondary branch in a few 10s of msec.

8. Capacity Planning for MoFRR

As for LFA FRR (draft-ietf-rtgwg-lfa-applicability-00), MoFRR applicability is topology dependent.

In this document, we have described two very frequent designs (Fig 2 and Fig 3) which provide maximum MoFRR benefits.

Designers with topologies different than Fig2 and 3 can still benefit from MoFRR benefits thanks to the use of capacity planning tools.

Such tools are able to simulate the ability of each PE to build two disjoint branches of the same tree. This for hundreds of PEs and hundreds of sources.

This allows to assess the MoFRR protection coverage of a given network, for a set of sources.

If the protection coverage is deemed insufficient, the designer can use such tool to optimize the topology (add links, change igp metrics).

9. Other Applications

While all the examples in this document show the MoFRR applicability on PE devices, it is clear that MoFRR could be enabled on aggregation or core routers.

MoFRR can be popular in Data Center network configurations. With the advent of lower cost ethernet and increasing port density in routers, there is more meshed connectivity than ever before. When using a 3-level access, distribution, and core layers in a Data Center, there is a lot of inexpensive bandwidth connecting the layers. This will lend itself to more opportunities for ECMP paths at multiple layers. This allows for multiple layers of redundancy protecting link and node failure at each layer with minimal redundancy cost.

Redundancy costs are reduced because only one packet is forwarded at every link along the primary and secondary data paths so there is no duplication of data on any link thereby providing make-before-break protection at a very small cost.

The MoFRR behavior described for PIM are immediately applicable to MLDP. Alternate methods to detect failures such as MPLS-OAM or BFD may be considered.

The MoFRR principle may be applied to MVPNs.

The MoFRR principle may be applied to mLDP [I-D.ietf-mpls-ldp-p2mp]. The reader may simply switch the term secondary-PIM-Join by secondary-Label-Map message.

10. Security Considerations

There are no security considerations for this design other than what is already in the main PIM specification [RFC4601].

11. Acknowledgments

The authors would like to thank John Zwiebel, Greg Shepherd and Dave Oran for their review of the draft.

12. References

12.1. Normative References

- [RFC5036] Andersson, L., Minei, I., and B. Thomas, "LDP Specification", RFC 5036, October 2007.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

12.2. Informative References

- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [I-D.ietf-mpls-ldp-p2mp] Minei, I., Wijnands, I., Kompella, K., and B. Thomas, "Label Distribution Protocol Extensions for Point-to-Multipoint and Multipoint-to-Multipoint Label Switched Paths", draft-ietf-mpls-ldp-p2mp-12 (work in progress), February 2011.

Authors' Addresses

Apoorva Karan
Cisco Systems, Inc.
3750 Cisco Way
San Jose CA, 95134
USA

Email: apoorva@cisco.com

Clarence Filsfils
Cisco Systems, Inc.
De kleetlaan 6a
Diegem BRABANT 1831
Belgium

Email: cfilsfil@cisco.com

Dino Farinacci
Cisco Systems, Inc.
425 East Tasman Drive
San Jose CA, 95134
USA

Email: dino@cisco.com

Bruno Decraene
France Telecom
38-40 rue du General Leclerc
Issy Moulineaux cedex 9, 92794
FR

Email: bruno.decraene@orange-ftgroup.com

Nicolai Leymann
Deutsche Telekom
Winterfeldtstrasse 21
Berlin 10781
DE

Email: N.Leymann@telekom.de

Uwe Joorde
Deutsche Telekom
Winterfeldtstrasse 21
Berlin 10781
DE

Email: N.Leymann@telekom.de

Thomas Telkamp
Cariden Technologies, Inc.
888 Villa Street, Suite 500
Mountain View CA, 94041
USA

Email: telkamp@cariden.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 22, 2012

IJ. Wijnands, Ed.
S. Venaas
Cisco Systems, Inc.
M. Brig
Aegis BMD Program Office
October 20, 2011

PIM source discovery via BSR
draft-wijnands-pim-source-discovery-bsr-00

Abstract

PIM Sparse-Mode use a Rendezvous Point (RP) and shared trees to forward multicast packets to Last Hop Routers (LHR). After the first packet is received by the LHR, the source of the multicast stream is learned and the Shortest Path Tree (SPT) can be joined. This draft proposes a solution to support PIM Sparse Mode (SM) without the need for PIM registers, RPs or shared trees. Multicast source information is distributed via Bootstrap Router [RFC5059] messages and flooded throughout the Multicast domain. By removing the need for RPs and shared trees, the PIM-SM procedures are simplified, improving router operations, management and making the protocol more robust.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 22, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
1.1. Conventions used in this document	4
1.2. Terminology	4
2. Source to Group Mappings via BSR	5
2.1. Originating SG BSR messages	5
2.2. Forwarding SG BSR messages	5
2.3. Processing SG BSR messages	5
3. The first packets and bursty sources	5
4. Resiliency to network partitioning	6
5. Fragmentation of BSR messages	7
6. Bootstrap Source message format	7
7. Security Considerations	9
8. IANA considerations	9
9. Acknowledgments	10
10. References	10
10.1. Normative References	10
10.2. Informative References	10
Authors' Addresses	10

1. Introduction

PIM Sparse-Mode uses a Rendezvous Point (RP) and shared trees to forward multicast packets to Last Hop Routers (LHR). After the first packet is received by the LHR, the source of the multicast stream is learned and the Shortest Path Tree (SPT) can be joined. This draft proposes a solution to support PIM Sparse Mode (SM) without the need for PIM registers, RPs or shared trees. Multicast source information is distributed via Bootstrap Router [RFC5059] messages and flooded throughout the Multicast domain. By removing the need for RPs and shared trees, the PIM-SM procedures are simplified, improving router operations, management and making the protocol more robust.

BSR provides an infrastructure to advertise 'RP mappings' to all the routers in the Multicast network via Reverse Path Forwarding (RPF). This document proposes to use BSR to advertise Source Group (SG) mappings from the First Hop Router (FHR) to all the LHRs. BSR seems like a good fit since its already designed to distribute 'mappings' through out the multicast network. The requirements for distributing SG mappings seems very close to RP mappings, for that reason there seems no need to invent a new protocol.

1.1. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

RP: Rendezvous Point.

BSR: Bootstrap Router.

RPF: Reverse Path Forwarding

SPT: Shortest Path Tree.

FHR: First Hop Router, directly connected to the Source.

LHR: Last Hop Router, directly connected to the receiver.

SG Mapping: Multicast source to group mapping.

SG BSR message: A BSR message containing a source to group mapping.

2. Source to Group Mappings via BSR

A Candidate Bootstrap (C-BSR) router is typically a router which is configured to be a BSR. Multiple C-BSR may be configured in the network and an election procedure is applied to select the Elected BSR (E-BSR) router. The E-BSR router is the router that is responsible for distributing the Group to RP mappings. See [RFC5059] section 3.1 for more details. In order to distribute Source to Group (SG) mappings, there is no need to elect an E-BSR router.

2.1. Originating SG BSR messages

Each FHR that is directly connected to an active multicast source becomes the E-BSR for that SG mapping. How a multicast router discovers the source of the multicast packet and when it considers it self the FHR follows the same procedures as the registering process described in [RFC4601]. After it is decided that a register needs to be sent, the SG is not registered via the PIM SM register procedures, but the SG mapping is distributed via a BSR message. Note, only the SG mapping is distributed in the BSR message, not the entire packet as would have been done with a PIM register. The router originating the BSR messages includes its own address in the BSR message. The BSR messages are periodically sent for as long as the multicast source is active, similar to how PIM registers are periodically sent. The timer and timeout values described in [RFC5059] apply here as well.

2.2. Forwarding SG BSR messages

The forwarding of BSR messages follows the same procedures as documented in [RFC5059] section 3.4 and 3.5.

2.3. Processing SG BSR messages

A router that receives a SG BSR messages should parse the SG BSR message and store the SG mappings with a holdtimer started with the advertised holdtime for that group. If there are directly connected receivers for that group this router should send PIM (S,G) joins for all the SG mappings advertised in the BSR message. The SG BSR mappings is kept alive for as long as the holdtimer for the source is running. Once the holdtimer expired a PIM (S,G) prune must be sent to remove itself from the tree.

3. The first packets and bursty sources

The PIM register procedure is designed to deliver Multicast packets to the RP in the absence of a native SPT tree from the RP to the

source. The register packets received on the RP are decapsulated and forwarded down the shared tree to the LHRs. As soon as an SPT tree is built, multicast packets would flow natively over the SPT to the RP or LHR and the register process would stop. The PIM register process bridges the gap between how long it takes to build the SPT tree to the FHR. If the packets would not be unicast encapsulated to the RP they would be dropped by the FHR until the SPT is setup. This functionality is important for applications where the first packet(s) must be received for the application to work correctly. An other reason would be for bursty sources. If the application sends out a multicast packet every 4 minutes (or longer), the SPT is torn down (typically after 3:30 minutes of inactivity) before the next packet is forwarded down the tree. This will cause no multicast packet to ever be forwarded. A well behaved application should really be able to deal with packet loss since IP is a best effort based packet delivery system. But in reality this is not always the case.

With the procedures proposed in this draft the packet(s) received by the FHR will be dropped until the LHR has learned about the source and the SPT tree is built. That means for bursty sources or applications sensitive for the delivery of the first packet this proposal would not be very applicable. This proposal is mostly useful for applications that don't have strong dependency on the first packet(s) and have a constant data rate, like video distribution for example. For applications with strong dependency on the first packet(s) we recommend using PIM Bidir [RFC5015] or SSM [RFC4607]. The protocol operations are much simpler compared to PIM SM, it will cause less churn in the network and both guarantee best effort delivery for the first packet(s).

An other solution to address the problems described above is documented in [I-D.ietf-magma-msnip]. This proposal allows for a host to tell the FHR its willingness to act as Source for a certain Group before sending the data packets. LHRs have time to join the SPT tree before the host starts sending which would avoid packet loss. The SG mappings announced by [I-D.ietf-magma-msnip] can be advertised directly into BSR, allowing a very nice integration of both proposals. The life time of the SPT is not driven by the liveliness of Multicast data packets (which is the case with PIM SM), but by the announcements driven via [I-D.ietf-magma-msnip]. This will also prevent packet loss due to bursty sources.

4. Resiliency to network partitioning

In a PIM SM deployment where the network becomes partitioned, due to link or node failure, it is possible that the RP becomes unreachable to a certain part of the network. New sources that become active in

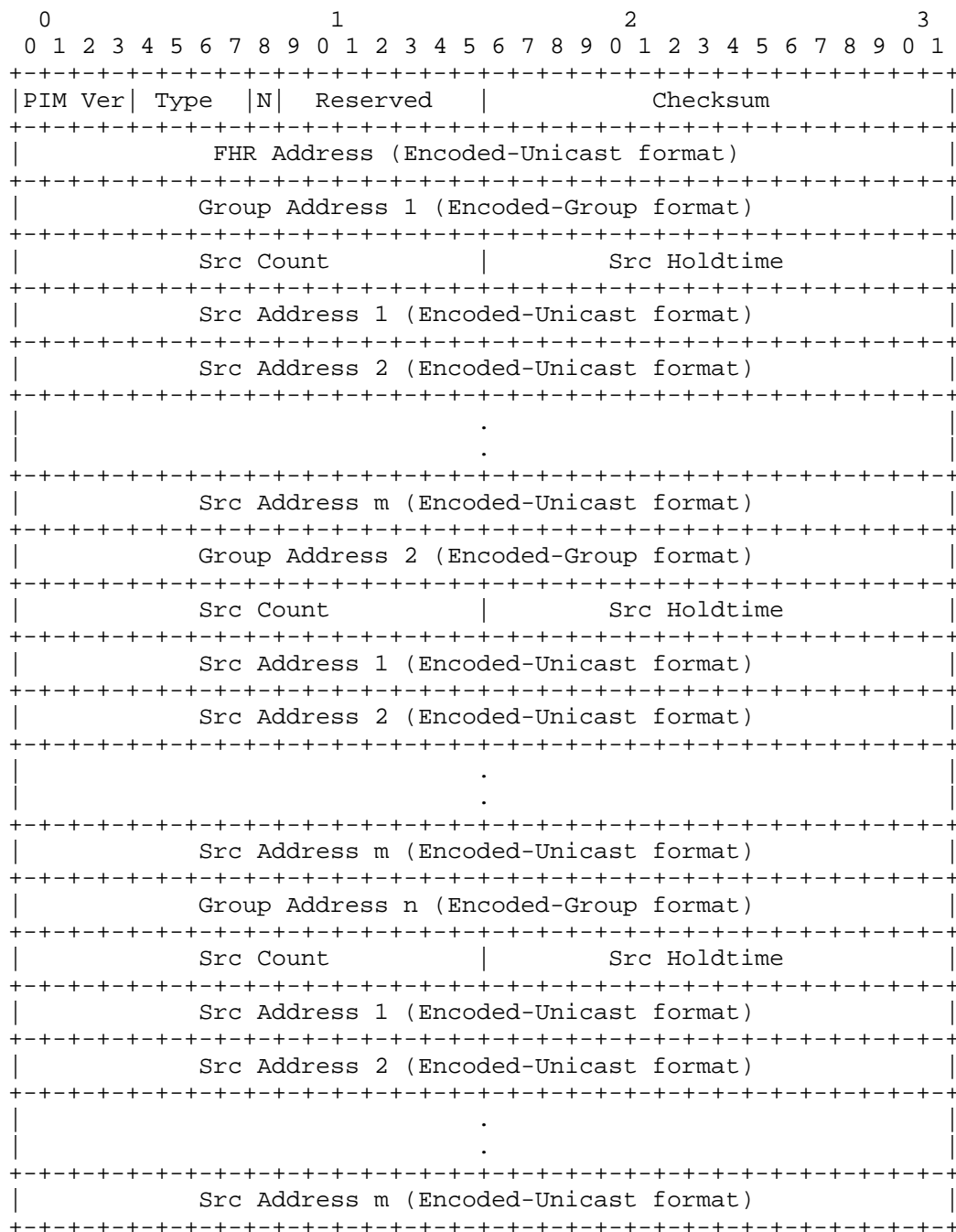
that partition will not be able to register to the RP and receivers within that partition are not able to receive the traffic. Ideally you would want to have a candidate RP in each partition, but you never know in advance which routers will form a partitioned network. In order to be fully resilient, each router in the network may end up being a candidate RP. This would increase the operational complexity of the network.

The solution described in this document does not suffer from that problem. If a network becomes partitioned and new sources become active, the receivers in that partitioned will receive the BSR SG Mappings and join the source tree. Each partition works independently of the other partition(s) and will continue to have access to sources within that partition. As soon as the network heals, the SG Mappings are re-flooded into the other partition(s) and other receives can join to the newly learned sources.

5. Fragmentation of BSR messages

[RFC5059] defines procedures to fragment BSR messages if the number of group to RP mappings is too large and the packet exceeds the MTU size. Its important for PIM to have all the RP mappings before it applies the selection process. Missing mappings may cause the wrong RP to be selected. Using BSR to distribute SG mappings we don't have this problem. There is no reason to have all the source before joining the tree. There is no selection process applied to the SG mappings, all the known SG mappings should be joined by PIM. For that reason there is no special fragmentation support defined for SG mappings.

6. Bootstrap Source message format



PIM Version: Reserved, Checksum Described in [RFC4601].

Type: PIM Message Type. Value (pending IANA) for a Bootstrap Source message

[N]o-Forward bit: When set, this bit means that the Bootstrap message fragment is not to be forwarded.

FHR Address: The address of the FHR router for the domain. This can be any address assigned to this router, but MUST be routable in the domain to allow successful forwarding (just like BSR address). The format for this address is given in the Encoded-Unicast address in [RFC4601].

Group Address 1..n: The address of the bootstrap router for the domain. The format for this address is given in the Encoded-Unicast address in [RFC4601].

Src Count How many unicast encoded sources address encodings follow.

Src Holdtime: The Holdtime (in seconds) for the corresponding source(s).

Src Address: The source address for the corresponding group range. The format for these addresses is given in the Encoded-Unicast address in [RFC4601].

7. Security Considerations

The security considerations are no different from what is documented in [RFC5059].

8. IANA considerations

This document requires the assignment of a new code point from the IANA managed registry "PIM Message Types" called "Bootstrap Source Mapping".

9. Acknowledgments

The authors would like to thank Arjen Boers for contributing to the initial idea and Yiqun Cai for his comments on the draft.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5059] Bhaskar, N., Gall, A., Lingard, J., and S. Venaas, "Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM)", RFC 5059, January 2008.

10.2. Informative References

- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.
- [RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.
- [RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.
- [I-D.ietf-magma-msnip] Fenner, B., "Multicast Source Notification of Interest Protocol (MSNIP)", draft-ietf-magma-msnip-05 (work in progress), March 2004.

Authors' Addresses

IJsbrand Wijnands (editor)
Cisco Systems, Inc.
De kleetlaan 6a
Diegem 1831
Belgium

Email: ice@cisco.com

Stig Venaas
Cisco Systems, Inc.
Tasman Drive
San Jose CA 95134
USA

Email: stig@cisco.com

Michael Brig
Aegis BMD Program Office
17211 Avenue D, Suite 160
Dahlgren VA 22448-5148
USA

Email: michael.brig@mda.mil

INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: 2012-04-24

Zhaohui (Jeffrey) Zhang
WeeSan Lee
Juniper Networks, Inc.
October 24, 2011

Avoid Unnecessary Upstream Traffic in BIDIR-PIM
draft-zzhang-pim-bidir-rp-prune-graft-00

Abstract

In a BIDIR-PIM network, data packets could be forwarded from the source to the RP and then get dropped there because there is no receiver on the other side of the distribution tree. This document specifies a way to prune the unnecessary traffic with minimum additional states and procedures.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1	Introduction	3
1.1	Terminology	8
2.	Protocol Specification	8
2.1.	RP-Prune state	8
2.2	RP-Prune/Graft message	8
2.2.1	Sending of RP-Prune message	9
2.2.2	Receiving of RP-Prune message	9
2.2.3	Sending of RP-Graft message	10
2.2.4	Receiving of RP-Graft message	10
2.3	RP-Prune-Keep/Cancel message	10
2.3.1	Sending of RP-Prune-Keep message	10
2.3.2	Receiving of RP-Prune-Keep message	10
2.3.3	Sending of RP-Prune-Cancel message	11
2.3.4	Receiving of RP-Prune-Cancel message	11
2.4	Virtual RPs	11
2.5	(*,G) RP-Prune state maintenance	12
2.6	Packet format	13
3	Security Considerations	14
4	IANA Considerations	14
5.	Acknowledgements	14
6	References	14
6.1	Normative References	14
6.2	Informative References	15
	Authors' Addresses	15

1 Introduction

[RFC4601] defines Protocol Independent Multicast - Sparse mode (PIM-SM), an efficient multicast routing protocol; however, the use of both the shortest-path tree and shared tree make the protocol extremely complex. [RFC5015] defines Bidirectional PIM (BIDIR-PIM), which distributes the packets using only bidirectional shared trees. Comparing to PIM-SM, BIDIR-PIM is a much simpler protocol; however, it is not without drawbacks. One drawback is that the data packets could be forwarded unnecessarily to the RP, Rendezvous Point, and then get dropped there when there is no receiver at all on the shared tree.

Particularly, there are two such scenarios:

- (i) There is no receiver for the group G at all;
- (ii) The senders and receivers for the group G share the same branch towards the RP.

For simplicity, we assume explicit tracking is enabled. Additionally, we use a RPA that belongs to a real router's loopback interface in our examples. The same procedures are applicable to more general situations (see section 2.4) with some changes.

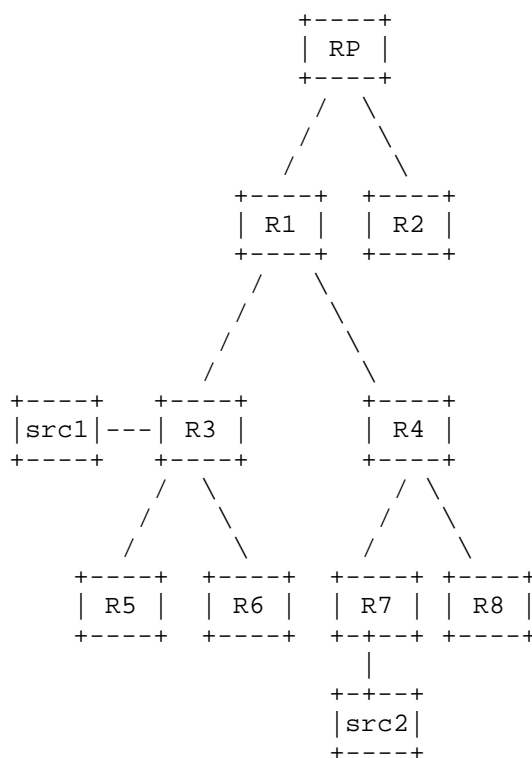


Figure 1

Scenario (i) can be illustrated with Figure 1. There is no receiver for group G at all. In this example, the source, src1, starts sending traffic, followed by another source, src2, later.

We assume each BIDIR-PIM router except the RP has installed at least a (*, G-prefix) route, which is responsible for forwarding the traffic for groups belongs to the G-prefix from the senders to the RP. On the RP itself, a special (*,G-prefix) route, called Resolve Route in this document, is installed to trigger (throttled) notifications to the control plane when packets match this route. This notification is similar to PIM-SM's way of notification of new flows, but here it serves the purpose of indicating that traffic is being unnecessarily received (if there were receivers, the traffic would have matched a more specific (*,G) forwarding route triggered by joins).

The procedures work as follows: when RP receives a data packet from src1, the (*,G-prefix) resolve route (vs. a more specific (*,G) route) is hit and the control plane is notified (that traffic is

being received unnecessarily). RP then triggers a (*,G) RP-Prune to ALL_PIM_ROUTERS on the interface on which the packet arrived. Upon receiving the RP-Prune, R1 will install a similar (*,G) resolve route.

In a similar fashion, a subsequent packet from src1 would trigger R1 to send a (*,G) RP-Prune out of the interface that connects to R3, and eventually stop the data packets from src1 traveling all the way to the RP and get dropped there.

When src2 starts sending packets toward the RP, the packets arrive at R1. The resolve route on R1 will prompt R1 to trigger a RP-Prune out to the interface connecting to R4, which results a (*,G) resolve route gets installed on R4. Similarly, a subsequent packet from src2 will trigger R4 to send a RP-Prune out of the interface connecting to R7. This effectively stops src2 from sending more traffic towards the RP. Note that the traffic from src2 does not need to get to the RP before it is pruned back.

The RP-Prune message is purposely named with the "RP-" prefix to indicate that it is triggered from the RP. Unlike the original BIDIR-PIM prune message, which travels upstream towards the RP, the RP-Prune message travels downstream.

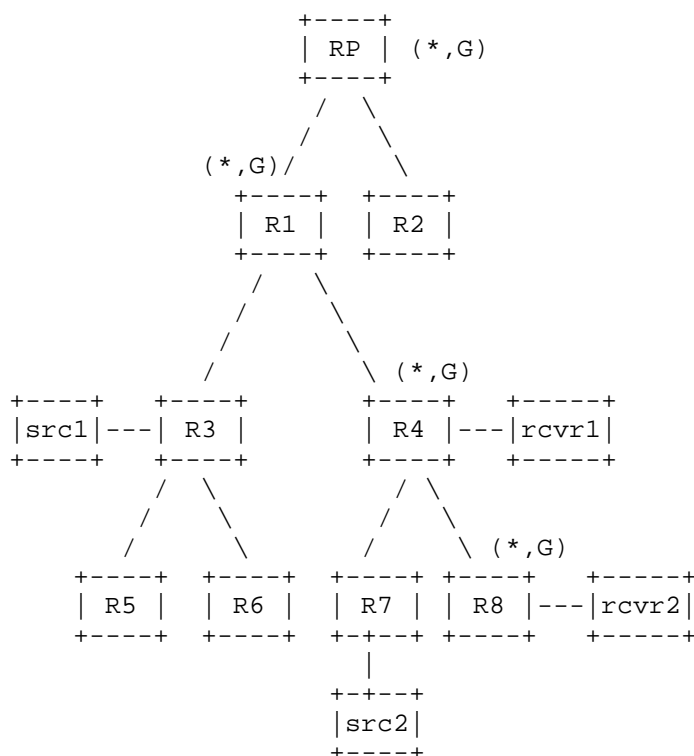


Figure 2

Scenerio (ii) can be illustrated with Figure 2. There are two receivers for group G, off R4 and R8 respectively. As a result, there are (*,G) join states on R8, R4, R1, and RP. Additionally, there are two senders for group G, off R7 and R3 respectively.

In this scenerio, the traffic from src1 would go up to R1, and then up to RP. At the same time, R1 would send the traffic down to R4 and R8 where the receivers are. The problem here is that the traffic from src1 should stop at R1 without going all the way to RP and gets dropped there.

Similarly, the traffic from src2 should make an U-turn on R4 to R8, without the need to go to R1 and RP for there are no other receivers on the shared tree.

To achieve the purpose of pruning the upstream traffic wherever it is appropriate, the RP initiates (*,G) RP-Prune messages for (*,G) join states that have only one downstream neighbor, towards that neighbor. When receiving the RP-Prune message, the downstream router does the

same by checking its corresponding (*,G) join state. If the number of the downstream interface is one and the number of downstream neighbors is one, the router further propagates the RP-Prune message downstream to that only neighbor; otherwise, the router terminates the RP-Prune message.

After receiving a RP-Prune message, the downstream router removes the upstream interface from its outgoing interface list so that traffic will not be forwarded upstream. It also creates/updates a (*,G) RP-Prune state to remember the upstream neighbor from which the RP-Prune arrived. Additionally, before sending out a RP-Prune message, it also creates/updates a (*,G) RP-Prune state to remember the interface and/or neighbor the RP-Prune is sent to.

In Figure 2, the RP-Prune message will be sent from the RP to R1, from R1 to R4, and terminated at R4 whose (*,G) state has two downstream interfaces: one to rcvr1, another to R8 where rcvr2 is attached. The reason R4 does not propagate the RP-Prune further down to R8 is because if another source off R8 start sending traffic, the traffic needs to travel upwards toward R4 to be forwarded to rcvr1.

Now, let's say src1 becomes a receiver as well and sends a (*,G) join toward the RP via R3 and R1. Now the number of downstream interface/neighbor of the (*,G) state on R1 becomes two. That will trigger a RP-Graft message being sent out of/to all the interfaces/neighbors on which the RP-Prune was sent out (the interface that the new (*,G) join arrived on should not be in that list). The RP-Graft message is to cancel out the RP-Prune message so that the traffic can be grafted back. Notice that R1 will still not send traffic to RP because RP still only has one downstream interface/neighbor so it does not send a RP-Graft to R1.

Note that for scenario (i), the RP-Prune is data driven hop-by-hop: RP has the (*,G-prefix) resolve route pre-installed while downstream router will install the (*,G) resolve route when a RP-Prune is received, and the RP-Prune is not propagated on downstream routers but re-triggered by the next packet hitting the newly installed resolve route. For scenario (ii), the RP-Prune is control driven: initiated by RP and propagated by downstream routers when appropriate.

It is worth to note that the extra (*,G) RP-Prune states are kept only on the relevant routers - where traffic is being or may be received unnecessarily.

The maintenance of the (*,G) RP-Prune states is specified in the protocol specification sections.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Protocol Specification

2.1. RP-Prune state

Like other PIM states, (*,G) RP-Prune state is soft-state; it relies on the periodic refresh to keep the state alive. The (*,G) RP-Prune state includes the following information:

- Upstream interface and neighbor from which the RP-Prune is received
 - a timer to time out the RP-Prune state after the upstream stops refreshing the RP-Prune messages (this is for scenario (i)).

This is to in case the RP-Graft from the upstream is lost.

- List of downstream interfaces. For each of them:
 - a flag indicating whether an interface-wide RP-Prune was sent

An interface-wide RP-Prune is sent in response to a resolve notification, which indicates that unnecessary traffic is being received on an interface. In contrast, a neighbor-specific RP-Prune is sent when the (*,G) join state only has one downstream interface.

- a list of downstream neighbors. For each of them:
 - if a RP-Prune was sent to the neighbor, a timer for its refresh.

This is the case where an neighbor-specific RP-Prune was sent when there is only one downstream neighbor for a (*,G) join state. <should we just propagate the RP-Prune initiated from the RP, w/o hop-by-hop refreshing independently using the timer?>

- if a RP-Prune-Keep (see Section 2.3) was received from the neighbor, a timer to timeout the neighbor after it stops sending RP-Prune-Keep.

This is the case where an interface-wide RP-Prune was sent.

2.2 RP-Prune/Graft message

RP-Prune/Graft messages are always sent downstream away from the RPA.

An interface-wide RP-Prune message is multicast to ALL-PIM-ROUTERS (sent in response to the resolve notification that it is receiving unnecessary traffic on an interface). A neighbor-specific RP-Prune message is unicast to a particular downstream neighbor (it is originated from the RP or propagated by a downstream router when there is only one downstream neighbor for a (*,G) join state).

Similarly, if a RP-Graft message is sent out of an interface where an interface-wide RP-Prune was sent out of before, ALL-PIM-ROUTERS address is used; if it is sent because a neighbor-specific RP-Prune was sent before, then the address of that particular neighbor is used.

To make sure that a RP-Graft message is received by all appropriate downstream neighbors, it is retransmitted until it has received RP-Prune-Cancel message (see below) from all the downstream neighbors listed in the RP-Prune state.

2.2.1 Sending of RP-Prune message

<to be expanded>

2.2.2 Receiving of RP-Prune message

When a RP-Prune message is received, it is first validated with the following criteria:

- It must be from the DF on the RPF interface
 - If it is addressed to ALL_PIM_ROUTERS then there must not be a (*,G) Join state.
 - If it is addressed to this router, then there must be a corresponding (*,G) join state.

If the validation fails, the message is discarded.

Otherwise, a corresponding RP-Prune state is looked up or created, with the upstream interface and neighbor recorded.

If it does not have corresponding (*,G) Join state, a (*,G) resolve route is created and the RP-Prune message is terminated.

If it does have corresponding (*,G) Join state, the RPF interface is removed from the (*,G) forwarding route's OIF list, and the upstream timer in the RP-Prune state is restarted.

If the (*,G) join state has more than one downstream interface/neighbor, the RP-Prune is terminated. If it only has one downstream interface and only one downstream neighbor on it, the RP-Prune is further propagated to that neighbor.

2.2.3 Sending of RP-Graft message

<to be expanded>

2.2.4 Receiving of RP-Graft message

2.3 RP-Prune-Keep/Cancel message

RP-Prune-Keep/Cancel messages are always sent upstream towards the RPA.

For a RP-Prune state triggered in case of a single downstream neighbor for a (*,G) join state, it is maintained by periodical refresh from RP downwards.

For a RP-Prune state that is triggered in response to resolve notification for traffic being received unnecessarily, it is maintained from the FHR upstream towards the RP. In scenario (i), the resolve notification will keep coming on R3 and R7 as long as traffic continues. That will maintain the downstream interface in the RP-Prune state (and the RP-Prune state itself). As a result, R3 and R7 will send RP-Prune-Keep messages upstream periodically, which in turn will maintain the downstream interface in the upstream's RP-Prune state (and the RP-Prune state itself). The process goes on.

If the source stop sending traffic, resolve notifications will stop happening on the FHR and it will timeout the corresponding downstream interface in the corresponding RP-Prune state. When all the downstream interfaces time out, the RP-Prune state will time out, and the router sends RP-Prune-Cancel upstream so that the upstream router can immediately remove the corresponding downstream interface.

The RP-Prune-Cancel also serves as a acknowledgement for RP-Graft message (see above section 2.2).

2.3.1 Sending of RP-Prune-Keep message

<to be expanded>

2.3.2 Receiving of RP-Prune-Keep message

The router looks up the corresponding (*,G) RP-Prune state when the message is received, and the corresponding downstream interface with

the interface-wide RP-Prune flag set. If either one is not found, an interface-wide RP-Graft message is immediately sent. Most likely, a previous RP-Graft is lost or there has been a topology change (see 2.5).

Otherwise, the timer for the downstream interface is restarted.

2.3.3 Sending of RP-Prune-Cancel message

<to be expanded>

2.3.4 Receiving of RP-Prune-Cancel message

<to be expanded>

2.4 Virtual RPs

In this document, we refer all the routers connected to the RPL, Rendezvous Point Link, as the virtual RPs. The virtual RPs perform the enhancement procedures described earlier, with some small changes.

Before the necessary changes are discussed, the following are the current standard behaviors for the routers on the RPL:

- (*,G) joins are terminated by the virtual RPs and not forwarded onto the RPL.
- traffic is always forwarded to the RPL - the RPL is added to the outgoing interface list of either (*,G) and (*,G-prefix) routes on the virtual RPs.

For the RP-Prune/Graft procedure to work, the following changes are necessary:

- (*,G) joins/prunes are propagated on the RPL to ALL-PIM-ROUTERS (with Upstream Neighbor Address set to 0), and processed by the virtual RPs. The RPL is added to the outgoing list of a (*,G) route if and only if a (*,G) join has been received on the RPL.

The (*,G) join/prunes are never sent further downstream, as in the base BIDIR-PIM case.

- The virtual RPs install a (*,G-prefix) resolve route instead of a regular forwarding route that forwards traffic onto the RPL.

With the above changes, traffic for group G will be forwarded to the RPL by a virtual RP only when at least one other virtual RP has sent

a (*,G) join to the RPL. With the enhancement procedures described earlier:

- A virtual RP's control plane will receive resolve notifications for traffic arriving on non-RPL interfaces and trigger RP-Prunes out of those interfaces, unless the traffic matches specific (*,G) forwarding routes.
- with respect to triggering RP-Prunes when there is only one downstream neighbor for a (*,G) join state, other virtual RPs are also counted as downstream neighbors if they are in the (*,G) join states on this virtual RP. Those RP-Prunes would only be triggered away from the RPL, as a result of only having one downstream neighbors on non-RPLs, as illustrated in Figure 3 below.

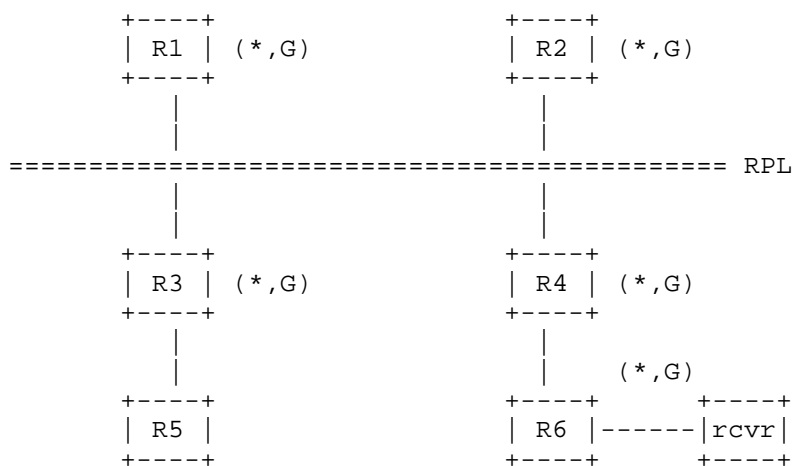


Figure 3

There is a receiver off R6, so there are (*,G) join states on R6 and R4 per normal BIDIR-PIM behavior. With the enhancement procedures, R4 will propagate the joins onto the RPL, so all R1 ~ R4 have the (*,G) state, each with only one downstream neighbor (R1~R3 has R4 as the downstream neighbor and R4 has R6 as the downstream neighbor), but only R4 will trigger RP-Prune towards R6 and R1 ~ R3 will not trigger RP-Prune to the RPL.

As soon as another receiver joins on R5, R4 will have two downstream neighbors for the (*,G) join state - one being R3 on the RPL and one being the existing R6 - so it will send RP-Graft to R6.

2.5 (*,G) RP-Prune state maintenance

In a steady environment, the (*,G) RP-Prune states are maintained by the RP-Prune refresh from upstream (for scenario (ii)), or by the RP-Prune-Keep refresh messages from downstream (for scenario (i)). In the former case, if an upstream stops refreshing RP-Prune messages, downstream will time out the corresponding states (normally the upstream should have sent RP-Graft in that case). In the latter case, the (*,G) RP-Prune states are no longer needed if relevant sources stops sending - the removal of unnecessary (*,G) RP-Prune states is achieved by the RP-Prune-Keep/Cancel procedures triggered from FHRs (see above section 2.3).

With topology/configuration changes, if the upstream interface changes, or the DF on the upstream interface changes, all RP-Prune states with that upstream interface and neighbor are deleted, with the following actions:

- if a (*,G) Resolve route was installed (scenario (i)), it is deleted.
- if the upstream interface was removed from the (*,G) forwarding route (scenario (ii)), it is re-evaluated to see if needs to be added back (w/o considering RP-Prune state)
- RP-Graft messages are sent out of downstream interfaces or to a neighbor where RP-Prune was sent before (with retransmission if necessary)

2.6 Packet format

The RP-Prune/Graft, RP-Prune-Keep/Cancel messages use the PIM-SM PIM Join/Prune format, with the following differences:

0										1										2										3																			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9										
PIM Ver										Type										Subtype										Rsvd										Checksum									
.....																												

Type

The type is X, with a 4-bit Subtype field carved out of the previous Reserved field (similar to the DF Election messages).

The four messages have the following Subtype values:

- | | |
|---|---------------|
| 1 | RP-Prune |
| 2 | RP-Graft |
| 3 | RP-Prune-Keep |

4 RP-Prune-Cancel

Upstream Neighbor Address

For RP-Prune/Graft messages, it is the address of the downstream neighbor, or zero in case of interface-wide RP-Prune/Graft messages.

For RP-Prune-Keep/Cancel messages, it is the address of the upstream DF.

Holdtime

For RP-Prune-Cancel, it is 0.

For RP-Graft, it is the amount of time within which the receiver message must respond with a RP-Prune-Cancel as an acknowledgement.

For RP-Prune, it is the amount of time that a receiver can keep the RP-Prune state alive.

For RP-Prune-Keep, it is the amount of time that a receiver can keep the downstream neighbor alive.

Number of Joined Sources, Number of Pruned Sources

Always 0.

3 Security Considerations

<Security considerations text>

4 IANA Considerations

<IANA considerations text>

5. Acknowledgements

Thanks for review comments and suggestions from Kurt Windisch and Rober Kebler.

6 References

6.1 Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC5015] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, October 2007.
- [RFC4601] Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", RFC 4601, August 2006.

6.2 Informative References

Authors' Addresses

Zhaohui (Jeffrey) Zhang
Juniper Networks, Inc.
10 Technology Park Drive
Westford, MA 01886
EMail: zzhang@juniper.net

WeeSan Lee
Juniper Networks, Inc.
1194 North Mathilda Avenue
Sunnyvale, CA 94089-1206
EMail: weesan@juniper.net