Internet Engineering Task Force (IETF)
Request for Comments: 8220
Category: Informational
ISSN: 2070-1721

O. Dornon
J. Kotalwar
V. Hemige
Nokia
R. Qiu
mistnet.io
Z. Zhang
Juniper Networks, Inc.

September 2017

Protocol Independent Multicast (PIM) over Virtual Private LAN Service (VPLS)

Abstract

This document describes the procedures and recommendations for Virtual Private LAN Service (VPLS) Provider Edges (PEs) to facilitate replication of multicast traffic to only certain ports (behind which there are interested Protocol Independent Multicast (PIM) routers and/or Internet Group Management Protocol (IGMP) hosts) via PIM snooping and proxying.

With PIM snooping, PEs passively listen to certain PIM control messages to build control and forwarding states while transparently flooding those messages. With PIM proxying, PEs do not flood PIM Join/Prune messages but only generate their own and send them out of certain ports, based on the control states built from downstream Join/Prune messages. PIM proxying is required when PIM Join suppression is enabled on the Customer Edge (CE) devices and is useful for reducing PIM control traffic in a VPLS domain.

This document also describes PIM relay, which can be viewed as lightweight proxying, where all downstream Join/Prune messages are simply forwarded out of certain ports and are not flooded, thereby avoiding the triggering of PIM Join suppression on CE devices.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8220.

Copyright Notice

Table of Contents

1. Introduction

In the Virtual Private LAN Service (VPLS), the Provider Edge (PE) devices provide a logical interconnect such that Customer Edge (CE) devices belonging to a specific VPLS instance appear to be connected by a single LAN. The Forwarding Information Base (FIB) for a VPLS instance is populated dynamically by Media Access Control (MAC) address learning. Once a unicast MAC address is learned and associated with a particular Attachment Circuit (AC) or pseudowire (PW), a frame destined to that MAC address only needs to be sent on that AC or PW.

For a frame not addressed to a known unicast MAC address, flooding has to be used. This happens with the following so-called "BUM" (Broadcast, Unknown Unicast, and Multicast) traffic:

- B: The destination MAC address is a broadcast address.

- U: The destination MAC address is unknown (has not been learned).

- M: The destination MAC address is a multicast address.

Multicast frames are flooded because a PE cannot know where corresponding multicast group members reside. VPLS solutions (RFC 4762 [VPLS-LDP] and RFC 4761 [VPLS-BGP]) perform replication for multicast traffic at the ingress PE devices. As stated in the VPLS Multicast Requirements document (RFC 5501 [VPLS-MCAST-REQ]), there are two issues with VPLS multicast today:

1. Multicast traffic is replicated to non-member sites.

2. Multicast traffic may be replicated when several PWs share a physical path.

Issue 1 can be solved by multicast snooping -- PEs learn sites with multicast group members by snooping multicast protocol control messages on ACs and forward IP multicast traffic only to member sites. This document describes the procedures to achieve this when CE devices are PIM adjacencies of each other. Issue 2 is outside the scope of this document and is discussed in RFC 7117 [VPLS-MCAST].

While descriptions in this document are in the context of the VPLS, the procedures also apply to regular Layer 2 switches interconnected by physical connections, except that the PW-related concepts and procedures do not apply in that case.

1.1. Multicast Snooping in VPLS

IGMP snooping procedures described in RFC 4541 [IGMP-SNOOP] make sure that IP multicast traffic is only sent on the following:

- ACs connecting to hosts that report related group membership

- ACs connecting to routers that join related multicast groups

- PWs connecting to remote PEs that have the above-described ACs

Note that traffic is always sent on ports that have point-to-point connections to routers that are attached to a LAN on which there is at least one other router. Because IGMP snooping alone cannot determine if there are interested receivers beyond those routers, we always need to send traffic to these ports, even if there are no snooped group memberships. To further restrict traffic sent to those routers, PIM snooping can be used. This document describes the procedures for PIM snooping, including rules for when both IGMP and PIM snooping are enabled in a VPLS instance; see Sections 2.8 and 2.11 for details.

Note that for both IGMP and PIM, the term "snooping" is used loosely, referring to the fact that a Layer 2 device peeks into Layer 3 routing protocol messages to build relevant control and forwarding states. Depending on whether the control messages are transparently flooded, selectively forwarded, or aggregated, the processing may be called "snooping" or "proxying" in different contexts.

We will use the term "PIM snooping" in this document; however, unless explicitly noted otherwise, the procedures apply equally to PIM snooping and PIM proxying. The procedures specific to PIM proxying are described in Section 2.6.6. Differences that need to be observed while implementing one or the other and recommendations on which method to employ in different scenarios are noted in Section 2.4.

This document also describes PIM relay, which can be viewed as lightweight PIM proxying. Unless explicitly noted otherwise, in the rest of this document proxying implicitly includes relay as well. Please refer to Section 2.4.1 for an overview of the differences between snooping, proxying, and relay.

1.2. Assumptions

This document assumes that the reader has a good understanding of the PIM protocols. To help correlate the concepts and make the text easier to follow, this document is written in the same style as the following PIM RFCs:

- RFC 3973 [PIM-DM]
- RFC 4607 [PIM-SSM]
- RFC 5015 [BIDIR-PIM]
- RFC 5384 [JOIN-ATTR]
- RFC 7761 [PIM-SM]

In order to avoid replicating text related to PIM protocol handling from the PIM RFCs, this document cross-references corresponding definitions and procedures in those RFCs. Deviations in protocol handling specific to PIM snooping are specified in this document.

1.3. Definitions

There are several definitions referenced in this document that are well described in the following PIM RFCs: RFC 3973 [PIM-DM], RFC 5015 [BIDIR-PIM], and RFC 7761 [PIM-SM]. The following definitions and abbreviations are used throughout this document:

- A port is defined as either an AC or a PW.
- When we say that a PIM message is received on a PE port, it means that the PE is processing the message for snooping/proxying or relaying.

Abbreviations used in this document:

- S: IP address of the multicast source.
- G: IP address of the multicast group.
- N: Upstream Neighbor field in a Join/Prune/Graft message.
- Port(N): Port on which neighbor N is learned, i.e., the port on which N’s Hellos are received.
- rpt: Rendezvous Point Tree.

- PIM-DM: Protocol Independent Multicast - Dense Mode.

- PIM-SM: Protocol Independent Multicast - Sparse Mode.

- PIM-SSM: Protocol Independent Multicast - Source-Specific Multicast.

1.4. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. PIM Snooping for VPLS

2.1. PIM Protocol Background

PIM is a multicast routing protocol running between routers, which are CE devices in a VPLS. It uses the unicast routing table to provide reverse-path information for building multicast trees. There are a few variants of PIM. As described in RFC 3973 [PIM-DM], multicast datagrams are pushed towards downstream neighbors, similar to a broadcast mechanism, but in areas of the network where there are no group members, routers prune back branches of the multicast tree towards the source. Unlike PIM-DM, other PIM flavors (RFC 7761 [PIM-SM], RFC 4607 [PIM-SSM], and RFC 5015 [BIDIR-PIM]) employ a pull methodology via explicit Joins instead of the push-and-prune technique.

PIM routers periodically exchange Hello messages to discover and maintain stateful sessions with neighbors. After neighbors are discovered, PIM routers can signal their intentions to join or prune specific multicast groups. This is accomplished by having downstream routers send an explicit Join/Prune message (for the sake of generalization, consider Graft messages for PIM-DM as Join messages) to their corresponding upstream router. The Join/Prune message can be group specific (*,G) or group and source specific (S,G).

2.2. General Rules for PIM Snooping in VPLS

The following rules for the correct operation of PIM snooping MUST be followed.

- PIM snooping MUST NOT affect the operation of customer Layer 2 protocols or Layer 3 protocols.
- PIM messages and multicast data traffic forwarded by PEs MUST follow the split-horizon rule for mesh PWs, as defined in RFC 4762 [VPLS-LDP].
- PIM states in a PE MUST be per VPLS instance.
- PIM Assert triggers MUST be preserved to the extent necessary to avoid sending duplicate traffic to the same PE (see Section 2.2.1).

2.2.1. Preserving Assert Triggers

In PIM-SM / PIM-DM, there are scenarios where multiple routers could be forwarding the same multicast traffic on a LAN. When this happens, these routers start the PIM Assert election process by sending PIM Assert messages, to ensure that only the Assert winner forwards multicast traffic on the LAN. The Assert election is a data-driven event and happens only if a router sees traffic on the interface to which it should be forwarding the traffic. In the case of a VPLS with PIM snooping, two routers may forward the same multicast datagrams at the same time, but each copy may reach a different set of PEs; this is acceptable from the point of view of avoiding duplicate traffic. If the two copies may reach the same PE, then the sending routers must be able to see each other’s traffic, in order to trigger Assert election and stop duplicate traffic. To achieve that, PEs enabled with PIM-SSM / PIM-SM snooping MUST forward multicast traffic for an (S,G) / (*,G) not only on the ports on which they snooped Join(S,G) / Join(*,G) but also towards the upstream neighbor(s). In other words, the ports on which the upstream neighbors are learned must be added to the outgoing port list, along with the ports on which Joins are snooped. Please refer to Section 2.6.1 for the rules that determine the set of upstream neighbors for a particular (x,G).

Similarly, PIM-DM snooping SHOULD make sure that Asserts can be triggered (Section 2.9.3).

The above logic needs to be facilitated without breaking VPLS split-horizon forwarding rules. That is, traffic should not be forwarded on the port on which it was received, and traffic arriving on a PW MUST NOT be forwarded onto other PW(s).

2.3. Some Considerations for PIM Snooping

The PIM snooping solution described here requires a PE to examine and operate on only PIM Hello and PIM Join/Prune packets. The PE does not need to examine any other PIM packets.

Most of the PIM snooping procedures for handling Hello/Join/Prune messages are very similar to those executed in a PIM router. However, the PE does not need to have any routing tables like those required in PIM routing. It knows how to forward Join/Prune messages only by looking at the Upstream Neighbor field in the Join/Prune packets, as described in Section 2.12.

The PE does not need to know about Rendezvous Points (RPs) and does not have to maintain any RP Set. All of that is transparent to a PIM snooping PE.

In the following subsections, we list some considerations and observations for the implementation of PIM snooping in the VPLS.

2.3.1. Scaling

PIM snooping needs to be employed on ACs at the downstream PEs (PEs receiving multicast traffic across the VPLS core) to prevent traffic from being sent out of ACs unnecessarily. PIM snooping techniques can also be employed on PWs at the upstream PEs (PEs receiving traffic from local ACs in a hierarchical VPLS) to prevent traffic from being sent to PEs unnecessarily. This may work well for small-scale or medium-scale deployments. However, if there are a large number of VPLS instances with a large number of PEs per instance, then the amount of snooping required at the upstream PEs can overwhelm the upstream PEs.

There are two methods to reduce the burden on the upstream PEs. One is to use PIM proxying, as described in Section 2.6.6, to reduce the control messages forwarded by a PE. The other is not to snoop on the PWs at all but to have PEs signal the snooped states to other PEs out of band via BGP, as described in RFC 7117 [VPLS-MCAST]. In this document, it is assumed that snooping is performed on PWs.

2.3.2. IPv4 and IPv6

In the VPLS, PEs forward Ethernet frames received from CEs and as such are agnostic of the Layer 3 protocol used by the CEs. However, as a PIM snooping PE, the PE would have to look deeper into the IP and PIM packets and build snooping state based on that. The PIM protocol specifications handle both IPv4 and IPv6. The specification for PIM snooping in this document can be applied to both IPv4 and IPv6 payloads.

2.3.3. PIM-SM (*,*,RP)

This document does not address (*,*,RP) states in the VPLS network, as they have been removed from the PIM protocol as described in RFC 7761 [PIM-SM].

2.4. PIM Snooping vs. PIM Proxying

This document has previously alluded to PIM snooping/relay/proxying. Details on the PIM relay/proxying solution are discussed in Section 2.6.6. In this section, a brief description and comparison are given.

2.4.1. Differences between PIM Snooping, Relay, and Proxying

Differences between PIM snooping and relay/proxying can be summarized as follows:

PIM snooping	PIM relay	PIM proxying
Join/Prune messages snooped and flooded according to VPLS flooding procedures	Join/Prune messages snooped; forwarded as is out of certain upstream ports	Join/Prune messages consumed. Regenerated ones sent out of certain upstream ports
Hello messages snooped and flooded according to VPLS flooding procedures	Hello messages snooped and flooded according to VPLS flooding procedures	Hello messages snooped and flooded according to VPLS flooding procedures
No PIM packets generated	No PIM packets generated	New Join/Prune messages generated
CE Join suppression not allowed	CE Join suppression allowed	CE Join suppression allowed

Other than the above differences, most of the procedures are common to PIM snooping and PIM relay/proxying, unless specifically stated otherwise.

Pure PIM snooping PEs simply snoop on PIM packets as they are being forwarded in the VPLS. As such, they truly provide transparent LAN services, since no customer packets are modified or consumed nor are new packets introduced in the VPLS. It is also simpler to implement than PIM proxying. However, for PIM snooping to work correctly, it is a requirement that CE routers MUST disable Join suppression in the VPLS. Otherwise, most of the CE routers with interest in a given multicast data stream will fail to send Join/Prune messages for that stream, and the PEs will not be able to tell which ACs and/or PWs have listeners for that stream.

Given that a large number of existing CE deployments do not support the disabling of Join suppression and given the operational complexity for a provider to manage the disabling of Join suppression in the VPLS, it becomes a difficult solution to deploy. Another disadvantage of PIM snooping is that it does not scale as well as PIM proxying. If there are a large number of CEs in a VPLS, then every CE will see every other CE’s Join/Prune messages.

PIM relay/proxying has the advantage that it does not require Join suppression to be disabled in the VPLS. Multicast as part of a VPLS can be very easily provided without requiring any changes on the CE routers. PIM relay/proxying helps scale VPLS multicast, since Join/Prune messages are only sent to certain upstream ports instead of flooded, and in cases of full proxying (vs. relay), the PEs intelligently generate only one Join/Prune message for a given multicast stream.

PIM proxying, however, loses the transparency argument, since Join/Prune packets could get modified or even consumed at a PE. Also, new packets could get introduced in the VPLS. However, this loss of transparency is limited to PIM Join/Prune packets. It is in the interest of optimizing multicast in the VPLS and helping a VPLS network scale much better, for both the provider and the customer. Data traffic will still be completely transparent.

2.4.2. PIM Control Message Latency

A PIM snooping/relay/proxying PE snoops on PIM Hello packets while transparently flooding them in the VPLS. As such, there is no latency introduced by the VPLS in the delivery of PIM Hello packets to remote CEs in the VPLS.

A PIM snooping PE snoops on PIM Join/Prune packets while transparently flooding them in the VPLS. There is no latency introduced by the VPLS in the delivery of PIM Join/Prune packets when PIM snooping is employed.

A PIM relay/proxying PE does not simply flood PIM Join/Prune packets. This can result in additional latency for a downstream CE to receive multicast traffic after it has sent a Join. When a downstream CE prunes a multicast stream, the traffic SHOULD stop flowing to the CE with no additional latency introduced by the VPLS.

Performing only proxying of Join/Prune and not Hello messages keeps the PE’s behavior very similar to that of a PIM router, without introducing too much additional complexity. It keeps the PIM proxying solution fairly simple. Since Join/Prune messages are forwarded by a PE along the slow path and all other PIM packet types are forwarded along the fast path, it is very likely that packets forwarded along the fast path will arrive "ahead" of Join/Prune packets at a CE router (note the stress on the fact that fast-path messages will never arrive after Join/Prune packets). Of particular importance are Hello packets sent along the fast path. We can construct a variety of scenarios resulting in out-of-order delivery of Hellos and Join/Prune messages. However, there should be no deviation from normal expected behavior observed at the CE router receiving these messages out of order.

2.4.3. When to Snoop and When to Proxy

From the above descriptions, factors that affect the choice of snooping/relay/proxying include:

- Whether CEs do Join suppression or not

- Whether Join/Prune latency is critical or not

- Whether the scale of PIM protocol messages/states in a VPLS requires the scaling benefit of proxying

Of the above factors, Join suppression is the hard one -- pure snooping can only be used when Join suppression is disabled on all CEs. The latency associated with relay/proxying is implementation dependent and may not be a concern at all with a particular implementation. The scaling benefit may not be important either, in that on a real LAN with Explicit Tracking (ET) a PIM router will need to receive and process all PIM Join/Prune messages as well.

A PIM router indicates that Join suppression is disabled if the T-bit is set in the LAN Prune Delay option of its Hello message. If all PIM routers on a LAN set the T-bit, ET is possible, allowing an upstream router to track all the downstream neighbors that have Join states for any (S,G) or (*,G). This has two benefits:

- No need for the Prune-Pending process -- the upstream router may immediately stop forwarding data when it receives a Prune from the last downstream neighbor and immediately prune to its upstream neighbor.

- For management purposes, the upstream router knows exactly which downstream routers exist for a particular Join state.

While full proxying can be used with or without Join suppression on CEs and does not interfere with an upstream CE’s bypass of the Prune-Pending process, it does proxy all its downstream CEs as a single one to the upstream neighbors, removing the second benefit mentioned above.

Therefore, the general rule is that if Join suppression is enabled on one or more CEs, then proxying or relay MUST be used, but if Join suppression is known to be disabled on all CEs, then snooping, relay, or proxying MAY be used, while snooping or relay SHOULD be used.

An implementation MAY choose to dynamically determine which mode to use, through the tracking of the above-mentioned T-bit in all snooped PIM Hello messages, or MAY simply require static provisioning.

2.5. Discovering PIM Routers

A PIM snooping PE MUST snoop on PIM Hellos received on ACs and PWs. That is, the PE transparently floods the PIM Hello while snooping on it. PIM Hellos are used by the snooping PE to discover PIM routers and their characteristics.

For each neighbor discovered by a PE, it includes an entry in the PIM Neighbor Database with the following fields:

- Layer 2 encapsulation for the router sending the PIM Hello.

- IP address and address family of the router sending the PIM Hello.

- Port (AC/PW) on which the PIM Hello was received.

- Hello Option fields.

The PE should be able to interpret and act on Hello Option fields as currently defined in RFC 7761 [PIM-SM]. The Option fields of particular interest in this document are:

- Hello-Hold-Time

- Tracking Support

- Designated Router (DR) Priority

Please refer to RFC 7761 [PIM-SM] for a list of the Hello Option fields. When a PIM Hello is received, the PE MUST reset the neighbor-expiry-timer to Hello-Hold-Time. If a PE does not receive a Hello message from a router within Hello-Hold-Time, the PE MUST remove that neighbor from its PIM Neighbor Database. If a PE receives a Hello message from a router with the Hello-Hold-Time value set to zero, the PE MUST remove that router from the PIM snooping state immediately.

From the PIM Neighbor Database, a PE MUST be able to use the procedures defined in RFC 7761 [PIM-SM] to identify the PIM DR in the VPLS instance. It should also be able to determine if tracking support is active in the VPLS instance.

2.6. PIM-SM and PIM-SSM

The key characteristic of PIM-SM and PIM-SSM is explicit Join behavior. In this model, multicast traffic is only forwarded to locations that specifically request it. All the procedures described in this section apply to both PIM-SM and PIM-SSM, except for the fact that there is no (*,G) state in PIM-SSM.

2.6.1. Building PIM-SM States

PIM-SM and PIM-SSM states are built by snooping on the PIM-SM Join/Prune messages received on ACs/PWs.

The downstream state machine of a PIM-SM snooping PE very closely resembles the downstream state machine of PIM-SM routers. The downstream state consists of:

Per downstream (Port,*,G):

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

Per downstream (Port,*,G,N):

- Prune-Pending Timer (PPT(N))

- Join Expiry Timer (ET(N))

Per downstream (Port,S,G):

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

Per downstream (Port,S,G,N):

- Prune-Pending Timer (PPT(N))

- Join Expiry Timer (ET(N))

Per downstream (Port,S,G,rpt):

- DownstreamJPRptState: One of {"NoInfo" (NI), "Pruned" (P), "Prune-Pending" (PP)}

Per downstream (Port,S,G,rpt,N):

- Prune-Pending Timer (PPT(N))

- Join Expiry Timer (ET(N))

where S is the address of the multicast source, G is the group address, and N is the Upstream Neighbor field in the Join/Prune message.

Note that unlike the case of PIM-SM routers, where the PPT and ET are per (Interface,S,G), PIM snooping PEs have to maintain the PPT and ET per (Port,S,G,N). The reasons for this are explained in Section 2.6.2.

Apart from the above states, we define the following state summarization macros:

UpstreamNeighbors(*,G): If there are one or more Join(*,G)s received on any port with upstream neighbor N and ET(N) is active, then N is added to UpstreamNeighbors(*,G). This set is used to determine if a Join(*,G) or a Prune(*,G) with upstream neighbor N needs to be sent upstream.

UpstreamNeighbors(S,G): If there are one or more Join(S,G)s received on any port with upstream neighbor N and ET(N) is active, then N is added to UpstreamNeighbors(S,G). This set is used to determine if a Join(S,G) or a Prune(S,G) with upstream neighbor N needs to be sent upstream.

UpstreamPorts(*,G): This is the set of all Port(N) ports where N is in the set UpstreamNeighbors(*,G). Multicast streams forwarded using a (*,G) match MUST be forwarded to these ports. So, UpstreamPorts(*,G) MUST be added to OutgoingPortList(*,G).

UpstreamPorts(S,G): This is the set of all Port(N) ports where N is in the set UpstreamNeighbors(S,G). UpstreamPorts(S,G) MUST be added to OutgoingPortList(S,G).

InheritedUpstreamPorts(S,G): This is the union of UpstreamPorts(S,G) and UpstreamPorts(*,G).

UpstreamPorts(S,G,rpt): If PruneDesired(S,G,rpt) becomes TRUE, then this set is set to UpstreamPorts(*,G). Otherwise, this set is empty. UpstreamPorts(*,G) (-) UpstreamPorts(S,G,rpt) MUST be added to OutgoingPortList(S,G).

UpstreamPorts(G): This set is the union of all the UpstreamPorts(S,G) and UpstreamPorts(*,G) for a given G. Proxy (S,G) Join/Prune and (*,G) Join/Prune messages MUST be sent to a subset of UpstreamPorts(G) as specified in Section 2.6.6.1.

PWPorts: This is the set of all PWs.

OutgoingPortList(*,G): This is the set of all ports to which traffic needs to be forwarded on a (*,G) match.

OutgoingPortList(S,G): This is the set of all ports to which traffic needs to be forwarded on an (S,G) match.

See Section 2.12 ("Data-Forwarding Rules") for the specification on how OutgoingPortList is calculated.

NumETsActive(Port,*,G): This is the number of (Port,*,G,N) entries that have the Expiry Timer running. This macro keeps track of the number of Join(*,G)s that are received on this Port with different upstream neighbors.

NumETsActive(Port,S,G): This is the number of (Port,S,G,N) entries that have the Expiry Timer running. This macro keeps track of the number of Join(S,G)s that are received on this Port with different upstream neighbors.

JoinAttributeTlvs(*,G): Join Attributes (RFC 5384 [JOIN-ATTR]) are TLVs that may be present in received Join(*,G) messages. An example would be Reverse Path Forwarding (RPF) Vectors (RFC 5496 [RPF-VECTOR]). If present, they must be copied to JoinAttributeTlvs(*,G).

JoinAttributeTlvs(S,G): Join Attributes (RFC 5384 [JOIN-ATTR]) are TLVs that may be present in received Join(S,G) messages. If present, they must be copied to JoinAttributeTlvs(S,G).

Since there are a few differences between the downstream state machines of PIM-SM routers and PIM-SM snooping PEs, we specify the details of the downstream state machine of PIM-SM snooping PEs, at the risk of repeating most of the text documented in RFC 7761 [PIM-SM].

2.6.2. Explanation for Per-(S,G,N) States

In PIM routing protocols, states are built per (S,G). On a router, an (S,G) has only one RPF-Neighbor. However, a PIM snooping PE does not have the Layer 3 routing information available to the routers in order to determine the RPF-Neighbor for a multicast flow. It merely discovers it by snooping the Join/Prune message. A PE could have snooped on two or more different Join/Prune messages for the same (S,G) that could have carried different Upstream Neighbor fields. This could happen during transient network conditions or due to dual-homed sources. A PE cannot make assumptions on which one to pick but instead must allow the CE routers to decide which upstream neighbor gets elected as the RPF-Neighbor. And for this purpose, the PE will have to track downstream and upstream Joins and Prunes per (S,G,N).

2.6.3. Receiving (*,G) PIM-SM Join/Prune Messages

A Join(*,G) or Prune(*,G) is considered "received" if one of the following conditions is met:

- o The port on which it arrived is not Port(N) where N is the upstream neighbor N of the Join/Prune(*,G).

- o If both Port(N) and the arrival port are PWs, then there exists at least one other (*,G,Nx) or (Sx,G,Nx) state with an AC UpstreamPort.

For simplicity, the case where both Port(N) and the arrival port are PWs is referred to as "PW-only Join/Prune" in this document. The PW-only Join/Prune handling is so that the Port(N) PW can be added to the related forwarding entries’ OutgoingPortList to trigger an Assert, but that is only needed for those states with AC UpstreamPorts. Note that in the PW-only case, it is OK for the arrival port and Port(N) to be the same. See Appendix B for examples.

When a router receives a Join(*,G) or a Prune(*,G) with upstream neighbor N, it must process the message as defined in the state machine below. Note that the macro computations of the various macros resulting from this state machine transition are exactly as specified in RFC 7761 [PIM-SM].

We define the following per-port (*,G,N) macro to help with the state machine below.

	Previous State		
Event	NoInfo (NI)	Join (J)	Prune-Pending (PP)
Receive Join(*,G)	-> J state Action RxJoin(N)	-> J state Action RxJoin(N)	-> J state Action RxJoin(N)
Receive Prune(*,G) and NumETsActive<=1	-	-> PP state Start PPT(N)	-> PP state
Receive Prune(*,G) and NumETsActive>1	-	-> J state Start PPT(N)	-
PPT(N) expires	-	-> J state Action PPTExpiry(N)	-> NI state Action PPTExpiry(N)
ET(N) expires and NumETsActive<=1	-	-> NI state Action ETExpiry(N)	-> NI state Action ETExpiry(N)
ET(N) expires and NumETsActive>1	-	-> J state Action ETExpiry(N)	-

Figure 1: Downstream Per-Port (*,G) State Machine in Tabular Form

Action RxJoin(N):

If ET(N) is not already running, then start ET(N). Otherwise, restart ET(N). If N is not already in UpstreamNeighbors(*,G), then add N to UpstreamNeighbors(*,G) and trigger a Join(*,G) with upstream neighbor N to be forwarded upstream. If there are Join Attribute TLVs in the received (*,G) message and if they are different from the recorded JoinAttributeTlvs(*,G), then copy them into JoinAttributeTlvs(*,G). In the case of conflicting attributes, the PE will need to perform conflict resolution per (N) as described in RFC 5384 [JOIN-ATTR].

Action PPTExpiry(N):

Same as Action ETExpiry(N) below, plus send a Prune-Echo(*,G) with upstream neighbor N on the downstream port.

Action ETExpiry(N):

Disable timers ET(N) and PPT(N). Delete Neighbor state (Port,*,G,N). If there are no other (Port,*,G) states with NumETsActive(Port,*,G) > 0, transition DownstreamJPState (RFC 7761 [PIM-SM]) to NoInfo. If there are no other (Port,*,G,N) states (different ports but for the same N), remove N from UpstreamPorts(*,G) -- this will also trigger the Upstream Finite State Machine (FSM) with "JoinDesired(*,G,N) to FALSE".

2.6.4. Receiving (S,G) PIM-SM Join/Prune Messages

A Join(S,G) or Prune(S,G) is considered "received" if one of the following conditions is met:

- The port on which it arrived is not Port(N) where N is the upstream neighbor N of the Join/Prune(S,G).

- If both Port(N) and the arrival port are PWs, then there exists at least one other (*,G,Nx) or (S,G,Nx) state with an AC UpstreamPort.

For simplicity, the case where both Port(N) and the arrival port are PWs is referred to as "PW-only Join/Prune" in this document. The PW-only Join/Prune handling is so that the Port(N) PW can be added to the related forwarding entries’ OutgoingPortList to trigger an Assert, but that is only needed for those states with AC UpstreamPorts. Note that in the PW-only case, it is OK for the arrival port and Port(N) to be the same. See Appendix B for examples.

When a router receives a Join(S,G) or a Prune(S,G) with upstream neighbor N, it must process the message as defined in the state machine below. Note that the macro computations of the various macros resulting from this state machine transition are exactly as specified in RFC 7761 [PIM-SM].

	Previous State		
Event	NoInfo (NI)	Join (J)	Prune-Pending (PP)
Receive Join(S,G)	-> J state Action RxJoin(N)	-> J state Action RxJoin(N)	-> J state Action RxJoin(N)
Receive Prune(S,G) and NumETsActive<=1	-	-> PP state Start PPT(N)	-
Receive Prune(S,G) and NumETsActive>1	-	-> J state Start PPT(N)	-
PPT(N) expires	-	-> J state Action PPTExpiry(N)	-> NI state Action PPTExpiry(N)
ET(N) expires and NumETsActive<=1	-	-> NI state Action ETExpiry(N)	-> NI state Action ETExpiry(N)
ET(N) expires and NumETsActive>1	-	-> J state Action ETExpiry(N)	-

Figure 2: Downstream Per-Port (S,G) State Machine in Tabular Form

Action RxJoin(N):

If ET(N) is not already running, then start ET(N). Otherwise, restart ET(N).

If N is not already in UpstreamNeighbors(S,G), then add N to UpstreamNeighbors(S,G) and trigger a Join(S,G) with upstream neighbor N to be forwarded upstream. If there are Join Attribute TLVs in the received (S,G) message and if they are different from the recorded JoinAttributeTlvs(S,G), then copy them into JoinAttributeTlvs(S,G). In cases of conflicting attributes, the PE will need to perform conflict resolution per (N) as described in RFC 5384 [JOIN-ATTR].

Action PPTExpiry(N):

Same as Action ETExpiry(N) below, plus send a Prune-Echo(S,G) with upstream neighbor N on the downstream port.

Action ETExpiry(N):

Disable timers ET(N) and PPT(N). Delete Neighbor state (Port,S,G,N). If there are no other (Port,S,G) states with NumETsActive(Port,S,G) > 0, transition DownstreamJPState to NoInfo. If there are no other (Port,S,G,N) states (different ports but for the same N), remove N from UpstreamPorts(S,G) -- this will also trigger the Upstream FSM with "JoinDesired(S,G,N) to FALSE".

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

A Join(S,G,rpt) or Prune(S,G,rpt) is "received" when the port on which it was received is not also the port on which the upstream neighbor N of the Join/Prune(S,G,rpt) was learned.

While it is important to ensure that the (S,G) and (*,G) state machines allow for handling per-(S,G,N) states, it is not as important for (S,G,rpt) states. It suffices to say that the downstream (S,G,rpt) state machine is the same as what is defined in Section 4.5.3 of RFC 7761 [PIM-SM].

2.6.6. Sending Join/Prune Messages Upstream

This section applies only to a PIM relay/proxying PE and not to a PIM snooping PE.

A full PIM proxying (not relay) PE MUST implement the Upstream FSM along the lines of the procedure described in Section 4.5.4 of RFC 7761 [PIM-SM].

For the purposes of the Upstream FSM, a Join or Prune message with upstream neighbor N is "seen" on a PIM relay/proxying PE if the port on which the message was received is also Port(N) and the port is an AC. The AC requirement is needed because a Join received on the Port(N) PW must not suppress this PE’s Join on that PW.

A PIM relay PE does not implement the Upstream FSM. It simply forwards received Join/Prune messages out of the same set of upstream ports as in the PIM proxying case.

In order to correctly facilitate Asserts among the CE routers, such Join/Prune messages need to send not only towards the upstream neighbor but also on certain PWs, as described below.

If JoinAttributeTlvs(*,G) is not empty, then it must be encoded in a Join(*,G) message sent upstream.

If JoinAttributeTlvs(S,G) is not empty, then it must be encoded in a Join(S,G) message sent upstream.

2.6.6.1. Where to Send Join/Prune Messages

The following rules apply to both (1) forwarded (in the case of PIM relay) and (2) refreshed and triggered (in the case of PIM proxying) (S,G) / (*,G) Join/Prune messages.

- The Upstream Neighbor field in the Join/Prune to be sent is set to the N in the corresponding Upstream FSM.

- If Port(N) is an AC, send the message to Port(N).

- Additionally, if OutgoingPortList(x,G,N) contains at least one AC, then the message MUST be sent to at least all the PWs in UpstreamPorts(G) (for (*,G)) or InheritedUpstreamPorts(S,G) (for (S,G)). Alternatively, the message MAY be sent to all PWs.

Sending to a subset of PWs as described above guarantees that if traffic (of the same flow) from two upstream routers were to reach this PE, then the two routers will receive from each other, triggering an Assert.

Sending to all PWs guarantees that if two upstream routers both send traffic for the same flow (even if it is to different sets of downstream PEs), then the two routers will receive from each other, triggering an Assert.

2.7. Bidirectional PIM (BIDIR-PIM)

Bidirectional PIM (BIDIR-PIM) is a variation of PIM-SM. The main differences between PIM-SM and BIDIR-PIM are as follows:

- There are no source-based trees, and SSM is not supported (i.e., no (S,G) states) in BIDIR-PIM.

- Multicast traffic can flow up the shared tree in BIDIR-PIM.

- To avoid forwarding loops, one router on each link is elected as the Designated Forwarder (DF) for each RP in BIDIR-PIM.

The main advantage of BIDIR-PIM is that it scales well for many-to-many applications. However, the lack of source-based trees means that multicast traffic is forced to remain on the shared tree.

As described in RFC 5015 [BIDIR-PIM], parts of a BIDIR-PIM-enabled network may forward traffic without exchanging Join/Prune messages -- for instance, between DFs and the Rendezvous Point Link (RPL).

As the described procedures for PIM snooping rely on the presence of Join/Prune messages, enabling PIM snooping on BIDIR-PIM networks could break the BIDIR-PIM functionality. Deploying PIM snooping on BIDIR-PIM-enabled networks will require some further study. Some thoughts on this topic are discussed in Appendix A.

2.8. Interaction with IGMP Snooping

Whenever IGMP snooping is enabled in conjunction with PIM snooping in the same VPLS instance, the PE SHOULD follow these rules:

- To maintain the list of multicast routers and ports on which they are attached, the PE SHOULD NOT use the rules described in RFC 4541 [IGMP-SNOOP] but SHOULD rely on the neighbors discovered by PIM snooping. This list SHOULD then be used to apply the first forwarding rule (rule 1) listed in Section 2.1.1 of RFC 4541 [IGMP-SNOOP].

- If the PE supports proxy reporting, an IGMP membership learned only on a port to which a PIM neighbor is attached (i.e., not learned elsewhere) SHOULD NOT be included in the summarized upstream report sent to that port.

2.9. PIM-DM

The key characteristic of PIM-DM is flood-and-prune behavior. Shortest-path trees are built as a multicast source starts transmitting.

2.9.1. Building PIM-DM States

PIM-DM states are built by snooping on the PIM-DM Join, Prune, Graft, and State Refresh messages received on ACs/PWs and State Refresh messages sent on ACs/PWs. By snooping on these PIM-DM messages, a PE builds the following states per (S,G,N) where S is the address of the multicast source, G is the group address, and N is the upstream neighbor to which Prunes/Grafts are sent by downstream CEs:

Per PIM(S,G,N):

Port PIM(S,G,N) Prune State:

* DownstreamPState(S,G,N,Port): One of {"NoInfo" (NI), "Pruned" (P), "Prune-Pending" (PP)}

* Prune-Pending Timer (PPT)

* Prune Timer (PT)

* Upstream Port (valid if the PIM(S,G,N) Prune state is "Pruned")

2.9.2. PIM-DM Downstream Per-Port PIM(S,G,N) State Machine

The downstream per-port PIM(S,G,N) state machine is as defined in Section 4.4.2 of RFC 3973 [PIM-DM], with a few changes relevant to PIM snooping. When reading Section 4.4.2 of RFC 3973 [PIM-DM], please be aware that, for the purposes of PIM snooping, the downstream states are built per (S,G,N,Downstream-Port) in PIM snooping and not per (Downstream-Interface,S,G) as in a PIM-DM router. As noted in Section 2.9.1, the states (DownstreamPState) and timers (PPT and PT) are per (S,G,N,Port).

2.9.3. Triggering Assert Election in PIM-DM

Since PIM-DM is a flood-and-prune protocol, traffic is flooded to all routers unless explicitly pruned. Since PIM-DM routers do not prune on non-RPF interfaces, PEs should typically not receive Prunes on Port(RPF-Neighbor). So, the asserting routers should typically be in pim_oiflist(S,G). In most cases, Assert election should occur naturally without any special handling, since data traffic will be forwarded to the asserting routers.

However, there are some scenarios where a Prune might be received on a port that is also an upstream port. If we prune the port from pim_oiflist(S,G), then it would not be possible for the asserting routers to determine if traffic arrived on their downstream port. This can be fixed by adding pim_iifs(S,G) to pim_oiflist(S,G) so that data traffic flows to the upstream ports.

2.10. PIM Proxy

As noted earlier, PIM snooping will work correctly only if Join suppression is disabled in the VPLS. If Join suppression is enabled in the VPLS, then PEs MUST do PIM relay/proxying for VPLS multicast to work correctly. This section applies specifically to full proxying and not to relay.

2.10.1. Upstream PIM Proxy Behavior

A PIM proxying PE consumes Join/Prune messages and regenerates PIM Join/Prune messages to be sent upstream by implementing the Upstream FSM as specified in Section 4.5.4 of RFC 7761 [PIM-SM]. This is the only difference from PIM relay.

The source IP address in PIM packets sent upstream SHOULD be the address of a PIM downstream neighbor in the corresponding Join/Prune state. The chosen address MUST NOT be the Upstream Neighbor field to be encoded in the packet. The Layer 2 encapsulation for the selected source IP address MUST be the encapsulation recorded in the PIM Neighbor Database for that IP address.

2.11. Directly Connected Multicast Source

PIM snooping/relay/proxying could be enabled on a LAN that connects a multicast source and a PIM First-Hop Router (FHR). As the FHR will not send any downstream Join/Prune messages, we will not be able to establish any forwarding states for that source. Therefore, if there is a source in the CE network that connects directly into the VPLS instance, then multicast traffic from that source MUST be sent to all PIM routers on the VPLS instance in addition to the IGMP receivers in the VPLS. If there is already (S,G) or (*,G) snooping state that is formed on any PE, this will not happen per the current forwarding rules and guidelines. So, in order to determine if traffic needs to be flooded to all routers, a PE must be able to determine if the traffic came from a host on that LAN. There are three ways to address this problem:

- The PE would have to do IPv4 ARP snooping and/or IPv6 Neighbor Discovery snooping to determine if a source is directly connected.

- Another option is to configure all PEs to indicate that there are CE sources that are directly connected to the VPLS instance and disallow snooping for the groups for which the source is going to send traffic. This way, traffic from that source to those groups will always be flooded within the provider network.

- A third option is to require that sources of CE multicast traffic must be behind a router.

This document recommends the third option -- sources of traffic must be behind a router.

2.12. Data-Forwarding Rules

First, we define the rules that are common to PIM-SM and PIM-DM PEs. Forwarding rules for each protocol type are specified in the subsections below.

If there is no matching forwarding state, then the PE SHOULD discard the packet, i.e., the UserDefinedPortList (Sections 2.12.1 and 2.12.2) SHOULD be empty.

The following general rules MUST be followed when forwarding multicast traffic in a VPLS:

- Traffic arriving on a port MUST NOT be forwarded back onto the same port.

- Due to VPLS split-horizon rules, traffic ingressing on a PW MUST NOT be forwarded to any other PW.

2.12.1. PIM-SM Data-Forwarding Rules

Per the rules in RFC 7761 [PIM-SM] and per the additional rules specified in this document,

```
OutgoingPortList(*,G) = immediate_olist(*,G) (+)
                        UpstreamPorts(*,G) (+)
                        Port(PimDR)

OutgoingPortList(S,G) = inherited_olist(S,G) (+)
                        UpstreamPorts(S,G) (+)
                        (UpstreamPorts(*,G) (-)
                        UpstreamPorts(S,G,rpt)) (+)
                        Port(PimDR)
```

RFC 7761 [PIM-SM] specifies how immediate_olist(*,G) and inherited_olist(S,G) are built. PimDR is the IP address of the PIM DR in the VPLS.

The PIM-SM snooping data-forwarding rules are defined below in pseudocode:

```
BEGIN
    iif is the incoming port of the multicast packet.
    S is the source IP address of the multicast packet.
    G is the destination IP address of the multicast packet.

    If there is (S,G) state on the PE
    Then
        OutgoingPortList = OutgoingPortList(S,G)
    Else if there is (*,G) state on the PE
    Then
        OutgoingPortList = OutgoingPortList(*,G)
    Else
        OutgoingPortList = UserDefinedPortList
    Endif

    If iif is an AC
    Then
        OutgoingPortList = OutgoingPortList (-) iif
    Else
        ## iif is a PW
        OutgoingPortList = OutgoingPortList (-) PWPorts
    Endif

    Forward the packet to OutgoingPortList.
END
```

First, if there is (S,G) state on the PE, then the set of outgoing ports is OutgoingPortList(S,G).

Otherwise, if there is (*,G) state on the PE, then the set of outgoing ports is OutgoingPortList(*,G).

The packet is forwarded to the selected set of outgoing ports while observing the general rules above in Section 2.12.

2.12.2. PIM-DM Data-Forwarding Rules

The PIM-DM snooping data-forwarding rules are defined below in pseudocode:

```
BEGIN
    iif is the incoming port of the multicast packet.
    S is the source IP address of the multicast packet.
    G is the destination IP address of the multicast packet.

    If there is (S,G) state on the PE
    Then
        OutgoingPortList = olist(S,G)
    Else
        OutgoingPortList = UserDefinedPortList
    Endif

    If iif is an AC
    Then
        OutgoingPortList = OutgoingPortList (-) iif
    Else
        ## iif is a PW
        OutgoingPortList = OutgoingPortList (-) PWPorts
    Endif

    Forward the packet to OutgoingPortList.
END
```

If there is forwarding state for (S,G), then forward the packet to olist(S,G) while observing the general rules above in Section 2.12.

RFC 3973 [PIM-DM] specifies how olist(S,G) is constructed.

3. IANA Considerations

This document does not require any IANA actions.

4. Security Considerations

Security considerations provided in the VPLS solution documents (i.e., RFC 4762 [VPLS-LDP] and RFC 4761 [VPLS-BGP]) apply to this document as well.

5. References

5.1. Normative References

[BIDIR-PIM] Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano, "Bidirectional Protocol Independent Multicast (BIDIR-PIM)", RFC 5015, DOI 10.17487/RFC5015, October 2007, <https://www.rfc-editor.org/info/rfc5015>.

[JOIN-ATTR] Boers, A., Wijnands, I., and E. Rosen, "The Protocol Independent Multicast (PIM) Join Attribute Format", RFC 5384, DOI 10.17487/RFC5384, November 2008, <https://www.rfc-editor.org/info/rfc5384>.

[PIM-DM] Adams, A., Nicholas, J., and W. Siadak, "Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)", RFC 3973, DOI 10.17487/RFC3973, January 2005, <https://www.rfc-editor.org/info/rfc3973>.

[PIM-SM] Fenner, B., Handley, M., Holbrook, H., Kouvelas, I., Parekh, R., Zhang, Z., and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)", STD 83, RFC 7761, DOI 10.17487/RFC7761, March 2016, <https://www.rfc-editor.org/info/rfc7761>.

[PIM-SSM] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, DOI 10.17487/RFC4607, August 2006, <https://www.rfc-editor.org/info/rfc4607>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RPF-VECTOR]
Wijnands, IJ., Boers, A., and E. Rosen, "The Reverse Path Forwarding (RPF) Vector TLV", RFC 5496, DOI 10.17487/RFC5496, March 2009, <https://www.rfc-editor.org/info/rfc5496>.

5.2. Informative References

[IGMP-SNOOP]
Christensen, M., Kimball, K., and F. Solensky, "Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches", RFC 4541, DOI 10.17487/RFC4541, May 2006, <https://www.rfc-editor.org/info/rfc4541>.

[VPLS-BGP]
Kompella, K., Ed., and Y. Rekhter, Ed., "Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling", RFC 4761, DOI 10.17487/RFC4761, January 2007, <https://www.rfc-editor.org/info/rfc4761>.

[VPLS-LDP]
Lasserre, M., Ed., and V. Kompella, Ed., "Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling", RFC 4762, DOI 10.17487/RFC4762, January 2007, <https://www.rfc-editor.org/info/rfc4762>.

[VPLS-MCAST]
Aggarwal, R., Ed., Kamite, Y., Fang, L., Rekhter, Y., and C. Kodeboniya, "Multicast in Virtual Private LAN Service (VPLS)", RFC 7117, DOI 10.17487/RFC7117, February 2014, <https://www.rfc-editor.org/info/rfc7117>.

[VPLS-MCAST-REQ]
Kamite, Y., Ed., Wada, Y., Serbest, Y., Morin, T., and L. Fang, "Requirements for Multicast Support in Virtual Private LAN Services", RFC 5501, DOI 10.17487/RFC5501, March 2009, <https://www.rfc-editor.org/info/rfc5501>.

Appendix A. BIDIR-PIM Considerations

This appendix describes some guidelines that may be used to preserve BIDIR-PIM functionality in combination with PIM snooping.

In order to preserve BIDIR-PIM snooping, routers need to set up forwarding states so that:

- on the RPL, all traffic is forwarded to all Port(N) ports.

- on any other interface, traffic is always forwarded to the DF.

The information needed to set up these states may be obtained by:

- determining the mapping between the group (range) and the RP.

- snooping and storing DF election information.

- determining where the RPL is. This could be achieved by static configuration or by combining the information mentioned in the two bullet items above.

A.1. BIDIR-PIM Data-Forwarding Rules

The BIDIR-PIM snooping data-forwarding rules are defined below in pseudocode:

```
BEGIN
    iif is the incoming port of the multicast packet.
    G is the destination IP address of the multicast packet.

    If there is forwarding state for G
    Then
        OutgoingPortList = olist(G)
    Else
        OutgoingPortList = UserDefinedPortList
    Endif

    If iif is an AC
    Then
        OutgoingPortList = OutgoingPortList (-) iif
    Else
        ## iif is a PW
        OutgoingPortList = OutgoingPortList (-) PWPorts
    Endif

    Forward the packet to OutgoingPortList.
END
```

If there is forwarding state for G, then forward the packet to olist(G) while observing the general rules above in Section 2.12.

RFC 5015 [BIDIR-PIM] specifies how olist(G) is constructed.

Appendix B. Example Network Scenario

Let us consider the scenario in Figure 3.

```
                                            +------+ AC3 +------+
                                            |  PE2 |-----| CE3  |
                                           /|      |     +------+
                                          / +------+         |
                                         /     |             |
                                        /      |             |
                                       /PW12   |             |
                                      /        |           /---\
                                     /         |PW23       | S |
                                    /          |           \---/
                                   /           |             |
                                  /            |             |
                                 /             |             |
                       +------+ /           +------+         |
          +------+     |  PE1 |/   PW13     |  PE3 |     +------+
          | CE1  |-----|      |-------------|      |-----| CE4  |
          +------+ AC1 +------+             +------+ AC4 +------+
                           |
                           |AC2
                       +------+
                       | CE2  |
                       +------+
```

Figure 3: An Example Network for Triggering an Assert

In the examples below, JT(Port,S,G,N) is the downstream Join Expiry Timer on the specified Port for the (S,G) with upstream neighbor N.

B.1. PIM Snooping Example

In the network depicted in Figure 3, S is the source of a multicast stream (S,G). CE1 and CE2 both have two ECMP routes to reach the source.

1. CE1 sends a Join(S,G) with UpstreamNeighbors(S,G) = CE3.

2. PE1 snoops on the Join(S,G) and builds forwarding state, since it is received on an AC. It also floods the Join(S,G) in the VPLS. PE2 snoops on the Join(S,G) and builds forwarding state, since

the Join(S,G)is targeting a neighbor residing on an AC. PE3 does not create forwarding state for (S,G) because this is a PW-only Join and there is neither an existing (*,G) state with an AC in UpstreamPorts(*,G) nor an existing (S,G) state with an AC in UpstreamPorts(S,G). Both PE2 and PE3 will also flood the Join(S,G) in the VPLS.

The resulting states at the PEs are as follows:

```
PE1 states:
   JT(AC1,S,G,CE3)        = JP_HoldTime
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW12 }
   OutgoingPortList(S,G)  = { AC1, PW12 }

PE2 states:
   JT(PW12,S,G,CE3)       = JP_HoldTime
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { AC3 }
   OutgoingPortList(S,G)  = { PW12, AC3 }

PE3 states:
   No (S,G) state
```

3. The multicast stream (S,G) flows along CE3 -> PE2 -> PE1 -> CE1.

4. Now CE2 sends a Join(S,G) with UpstreamNeighbors(S,G) = CE4.

5. All PEs snoop on the Join(S,G), build forwarding state, and flood the Join(S,G) in the VPLS. Note that for PE2, even though this is a PW-only Join, forwarding state is built on this Join(S,G), since PE2 has an existing (S,G) state with an AC in UpstreamPorts(S,G).

   The resulting states at the PEs are as follows:

```
PE1 states:
   JT(AC1,S,G,CE3)        = active
   JT(AC2,S,G,CE4)        = JP_HoldTime
   UpstreamNeighbors(S,G) = { CE3, CE4 }
   UpstreamPorts(S,G)     = { PW12, PW13 }
   OutgoingPortList(S,G)  = { AC1, PW12, AC2, PW13 }
```

```
PE2 states:
   JT(PW12,S,G,CE4)       = JP_HoldTime
   JT(PW12,S,G,CE3)       = active
   UpstreamNeighbors(S,G) = { CE3, CE4 }
   UpstreamPorts(S,G)     = { AC3, PW23 }
   OutgoingPortList(S,G)  = { PW12, AC3, PW23 }

PE3 states:
   JT(PW13,S,G,CE4)       = JP_HoldTime
   UpstreamNeighbors(S,G) = { CE4 }
   UpstreamPorts(S,G)     = { AC4 }
   OutgoingPortList(S,G)  = { PW13, AC4 }
```

6. The multicast stream (S,G) flows into the VPLS from two of the CEs -- CE3 and CE4. PE2 forwards the stream received from CE3 to PW23, and PE3 forwards the stream to AC4. This helps the CE routers to trigger Assert election. Let us say that CE3 becomes the Assert winner.

7. CE3 sends an Assert message to the VPLS. The PEs flood the Assert message without examining it.

8. CE4 stops sending the multicast stream to the VPLS.

9. CE2 notices an RPF change due to the Assert and sends a Prune(S,G) with upstream neighbor = CE4. CE2 also sends a Join(S,G) with upstream neighbor = CE3.

10. All the PEs start a Prune-Pending timer on the ports on which they received the Prune(S,G). When the Prune-Pending timer expires, all PEs will remove the downstream (S,G,CE4) states.

    The resulting states at the PEs are as follows:

```
PE1 states:
   JT(AC1,S,G,CE3)        = active
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW12 }
   OutgoingPortList(S,G)  = { AC1, AC2, PW12 }

PE2 states:
   JT(PW12,S,G,CE3)       = active
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { AC3 }
   OutgoingPortList(S,G)  = { PW12, AC3 }
```

```
PE3 states:
   JT(PW13,S,G,CE3)       = JP_HoldTime
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW23 }
   OutgoingPortList(S,G)  = { PW13, PW23 }
```

Note that at this point at PE3, since there is no AC in OutgoingPortList(S,G) and no (*,G) or (S,G) state with an AC in UpstreamPorts(*,G) or UpstreamPorts(S,G), respectively, the existing (S,G) state at PE3 can also be removed. So, finally:

```
PE3 states:
   No (S,G) state
```

Note that at the end of the Assert election, there should be no duplicate traffic forwarded downstream, and traffic should flow only on the desired path. Also note that there are no unnecessary (S,G) states on PE3 after the Assert election.

B.2. PIM Proxy Example with (S,G) / (*,G) Interaction

In the same network, let us assume that CE4 is the upstream neighbor towards the RP for G.

JPST(S,G,N) is the JP sending timer for the (S,G) with upstream neighbor N.

1. CE1 sends a Join(S,G) with UpstreamNeighbors(S,G) = CE3.

2. PE1 consumes the Join(S,G) and builds forwarding state, since the Join(S,G) is received on an AC.

   PE2 consumes the Join(S,G) and builds forwarding state, since the Join(S,G) is targeting a neighbor residing on an AC.

   PE3 consumes the Join(S,G) but does not create forwarding state for (S,G), since this is a PW-only Join and there is neither an existing (*,G) state with an AC in UpstreamPorts(*,G) nor an existing (S,G) state with an AC in UpstreamPorts(S,G).

The resulting states at the PEs are as follows:

```
PE1 states:
   JT(AC1,S,G,CE3)        = JP_HoldTime
   JPST(S,G,CE3)          = t_periodic
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW12 }
   OutgoingPortList(S,G)  = { AC1, PW12 }

PE2 states:
   JT(PW12,S,G,CE3)       = JP_HoldTime
   JPST(S,G,CE3)          = t_periodic
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { AC3 }
   OutgoingPortList(S,G)  = { PW12, AC3 }

PE3 states:
   No (S,G) state
```

Joins are triggered as follows:
PE1 triggers a Join(S,G) targeting CE3. Since the Join(S,G) was received on an AC and is targeting a neighbor that is residing across a PW, the triggered Join(S,G) is sent on all PWs.

PE2 triggers a Join(S,G) targeting CE3. Since the Join(S,G) is targeting a neighbor residing on an AC, it only sends the Join on AC3.

PE3 ignores the Join(S,G), since this is a PW-only Join and there is neither an existing (*,G) state with an AC in UpstreamPorts(*,G) nor an existing (S,G) state with an AC in UpstreamPorts(S,G).

3. The multicast stream (S,G) flows along CE3 -> PE2 -> PE1 -> CE1.

4. Now let us say that CE2 sends a Join(*,G) with UpstreamNeighbors(*,G) = CE4.

5. PE1 consumes the Join(*,G) and builds forwarding state, since the Join(*,G) is received on an AC.

   PE2 consumes the Join(*,G); although this is a PW-only Join, forwarding state is built on this Join(*,G), since PE2 has an existing (S,G) state with an AC in UpstreamPorts(S,G). However, since this is a PW-only Join, PE2 only adds the PW towards PE3 (PW23) into UpstreamPorts(*,G) and hence into OutgoingPortList(*,G). It does not add the PW towards PE1 (PW12) into OutgoingPortList(*,G).

PE3 consumes the Join(*,G) and builds forwarding state, since the Join(*,G) is targeting a neighbor residing on an AC.

The resulting states at the PEs are as follows:

```
PE1 states:
   JT(AC1,*,G,CE4)        = JP_HoldTime
   JPST(*,G,CE4)          = t_periodic
   UpstreamNeighbors(*,G) = { CE4 }
   UpstreamPorts(*,G)     = { PW13 }
   OutgoingPortList(*,G)  = { AC2, PW13 }

   JT(AC1,S,G,CE3)        = active
   JPST(S,G,CE3)          = active
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW12 }
   OutgoingPortList(S,G)  = { AC1, PW12, PW13 }

PE2 states:
   JT(PW12,*,G,CE4)       = JP_HoldTime
   UpstreamNeighbors(*,G) = { CE4 }
   UpstreamPorts(G)       = { PW23 }
   OutgoingPortList(*,G)  = { PW23 }

   JT(PW12,S,G,CE3)       = active
   JPST(S,G,CE3)          = active
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { AC3 }
   OutgoingPortList(S,G)  = { PW12, AC3, PW23 }

PE3 states:
   JT(PW13,*,G,CE4)       = JP_HoldTime
   JPST(*,G,CE4)          = t_periodic
   UpstreamNeighbors(*,G) = { CE4 }
   UpstreamPorts(*,G)     = { AC4 }
   OutgoingPortList(*,G)  = { PW13, AC4 }
```

Joins are triggered as follows:
PE1 triggers a Join(*,G) targeting CE4. Since the Join(*,G) was received on an AC and is targeting a neighbor that is residing across a PW, the triggered Join(S,G) is sent on all PWs.

PE2 does not trigger a Join(*,G) based on this Join, since this is a PW-only Join.

PE3 triggers a Join(*,G) targeting CE4. Since the Join(*,G) is targeting a neighbor residing on an AC, it only sends the Join on AC4.

6. If traffic is not flowing yet (i.e., step 3 is delayed so that it occurs after step 6) and in the interim JPST(S,G,CE3) on PE1 expires, causing it to send a refresh Join(S,G) targeting CE3, since the refresh Join(S,G) is targeting a neighbor that is residing across a PW, the refresh Join(S,G) is sent on all PWs.

7. Note that PE1 refreshes its JT based on reception of refresh Joins from CE1 and CE2.

   PE2 consumes the Join(S,G) and refreshes the JT(PW12,S,G,CE3) timer.

   PE3 consumes the Join(S,G). It also builds forwarding state on this Join(S,G), even though this is a PW-only Join, since now PE2 has an existing (*,G) state with an AC in UpstreamPorts(*,G). However, since this is a PW-only Join, PE3 only adds the PW towards PE2 (PW23) into UpstreamPorts(S,G) and hence into OutgoingPortList(S,G). It does not add the PW towards PE1 (PW13) into OutgoingPortList(S,G).

   PE3 states:

   ```
      JT(PW13,*,G,CE4)       = active
      JPST(S,G,CE4)          = active
      UpstreamNeighbors(*,G) = { CE4 }
      UpstreamPorts(*,G)     = { AC4 }
      OutgoingPortList(*,G)  = { PW13, AC4 }

      JT(PW13,S,G,CE3)       = JP_HoldTime
      UpstreamNeighbors(*,G) = { CE3 }
      UpstreamPorts(*,G)     = { PW23 }
      OutgoingPortList(*,G)  = { PW13, AC4, PW23 }
   ```

   Joins are triggered as follows:
   PE2 already has (S,G) state, so it does not trigger a Join(S,G) based on reception of this refresh Join.

   PE3 does not trigger a Join(S,G) based on this Join, since this is a PW-only Join.

8. The multicast stream (S,G) flows into the VPLS from two of the CEs -- CE3 and CE4. PE2 forwards the stream received from CE3 to PW12 and PW23. At the same time, PE3 forwards the stream received from CE4 to PW13 and PW23.

   The stream received over PW12 and PW13 is forwarded by PE1 to AC1 and AC2.

The stream received by PE3 over PW23 is forwarded to AC4. The stream received by PE2 over PW23 is forwarded to AC3. Either of these helps the CE routers to trigger Assert election.

9. CE3 and/or CE4 send(s) Assert message(s) to the VPLS. The PEs flood the Assert message(s) without examining it.

10. CE3 becomes the (S,G) Assert winner, and CE4 stops sending the multicast stream to the VPLS.

11. CE2 notices an RPF change due to the Assert and sends a Prune(S,G,rpt) with upstream neighbor = CE4.

12. PE1 consumes the Prune(S,G,rpt), and since PruneDesired(S,G,Rpt,CE4) is TRUE, it triggers a Prune(S,G,rpt) to CE4. Since the Prune is targeting a neighbor across a PW, it is sent on all PWs.

    PE2 consumes the Prune(S,G,rpt) and does not trigger any Prune based on this Prune(S,G,rpt), since this was a PW-only Prune.

    PE3 consumes the Prune(S,G,rpt), and since PruneDesired(S,G,rpt,CE4) is TRUE, it sends the Prune(S,G,rpt) on AC4.

```
PE1 states:
   JT(AC2,*,G,CE4)        = active
   JPST(*,G,CE4)          = active
   UpstreamNeighbors(*,G) = { CE4 }
   UpstreamPorts(*,G)     = { PW13 }
   OutgoingPortList(*,G)  = { AC2, PW13 }

   JT(AC2,S,G,CE4)        = JP_HoldTime with S,G,rpt prune flag
   JPST(S,G,CE4)          = none, since this is sent along
                            with the Join(*,G) to CE4 based
                            on JPST(*,G,CE4) expiry
   UpstreamPorts(S,G,rpt) = { PW13 }
   UpstreamNeighbors(S,G,rpt) = { CE4 }

   JT(AC1,S,G,CE3)        = active
   JPST(S,G,CE3)          = active
   UpstreamNeighbors(S,G) = { CE3 }
   UpstreamPorts(S,G)     = { PW12 }
   OutgoingPortList(S,G)  = { AC1, PW12, AC2 }
```

```
      PE2 states:
         JT(PW12,*,G,CE4)       = active
         UpstreamNeighbors(*,G) = { CE4 }
         UpstreamPorts(*,G)     = { PW23 }
         OutgoingPortList(*,G)  = { PW23 }

         JT(PW12,S,G,CE4)       = JP_HoldTime with S,G,rpt prune flag
         JPST(S,G,CE4)          = none, since this was created
                                  off a PW-only Prune
         UpstreamPorts(S,G,rpt) = { PW23 }
         UpstreamNeighbors(S,G,rpt) = { CE4 }

         JT(PW12,S,G,CE3)       = active
         JPST(S,G,CE3)          = active
         UpstreamNeighbors(S,G) = { CE3 }
         UpstreamPorts(S,G)     = { AC3 }
         OutgoingPortList(*,G)  = { PW12, AC3 }

      PE3 states:
         JT(PW13,*,G,CE4)       = active
         JPST(*,G,CE4)          = active
         UpstreamNeighbors(*,G) = { CE4 }
         UpstreamPorts(*,G)     = { AC4 }
         OutgoingPortList(*,G)  = { PW13, AC4 }

         JT(PW13,S,G,CE4)       = JP_HoldTime with S,G,rpt prune flag
         JPST(S,G,CE4)          = none, since this is sent along
                                  with the Join(*,G) to CE4 based
                                  on JPST(*,G,CE4) expiry
         UpstreamNeighbors(S,G,rpt) = { CE4 }
         UpstreamPorts(S,G,rpt) = { AC4 }

         JT(PW13,S,G,CE3)       = active
         JPST(S,G,CE3)          = none, since this state is
                                  created by a PW-only Join
         UpstreamNeighbors(S,G) = { CE3 }
         UpstreamPorts(S,G)     = { PW23 }
         OutgoingPortList(S,G)  = { PW23 }
```

Even in this example, at the end of the (S,G) / (*,G) Assert election, there should be no duplicate traffic forwarded downstream, and traffic should flow only to the desired CEs.

However, we don’t have duplicate traffic because one of the CEs stops sending traffic due to the Assert, not because we don’t have any forwarding state in the PEs to do this forwarding.

Acknowledgements

Many members of the former L2VPN and PIM working groups have contributed to, and provided valuable comments and feedback on, this document, including Vach Kompella, Shane Amante, Sunil Khandekar, Rob Nath, Marc Lasserre, Yuji Kamite, Yiqun Cai, Ali Sajassi, Jozef Raets, Himanshu Shah (Ciena), and Himanshu Shah (Cisco).

Contributors

Yetik Serbest and Suresh Boddapati coauthored earlier draft versions of this document.

Karl (Xiangrong) Cai and Princy Elizabeth made significant contributions to bring the specification to its current state, especially in the area of Join forwarding rules.

Authors' Addresses

Olivier Dornon
Nokia
Copernicuslaan 50
B-2018 Antwerp
Belgium

Email: olivier.dornon@nokia.com

Jayant Kotalwar
Nokia
701 East Middlefield Rd.
Mountain View, CA 94043
United States of America

Email: jayant.kotalwar@nokia.com

Venu Hemige
Nokia

Email: vhemige@gmail.com

Ray Qiu
mistnet.io

Email: ray@mistnet.io

Jeffrey Zhang
Juniper Networks, Inc.
10 Technology Park Drive
Westford, MA 01886
United States of America

Email: zzhang@juniper.net