                        PIM Snooping over VPLS
                   draft-ietf-l2vpn-vpls-pim-snooping-06

Abstract

   This document describes the procedures and recommendations for VPLS
   PEs to facilitate replication of multicast traffic to only certain
   ports (behind which there are interested PIM routers and/or IGMP
   hosts) via PIM Snooping and PIM Proxy.

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

   The document also describes PIM Relay, which can be viewed as light-
   weight proxy, where all downstream Join/Prune messages are simply
   forwarded out of certain ports but not flooded to avoid triggering
   PIM Join suppression on CE devices.

Requirements Language

   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].

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 October 31, 2014.

Copyright Notice

Table of Contents

1. Introduction

In 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. Forwarding Information Base for a VPLS instance is
populated dynamically by source 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 happen with the following so called BUM
traffic:

o B: The destination MAC address is a broadcast address,

o U: The destination MAC address is unknown (has not been learned),

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

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

o A. Multicast traffic is replicated to non-member sites.

o B. Replication of PWs on shared physical path.

Issue A can be solved by Multicast Snooping - PEs learn sites with
multicast members by snooping multicast protocol control messages and
forward IP multicast traffic only to member sites. This documents
describes the procedures to achieve that when PIM is running between
the CE devices. Issue B is outside the scope of this document and
discussed in[VPLS-MCAST-TREES].

While this document is in the context of VPLS, the procedures apply
to regular layer-2 switches interconnected by physical connections as
well. In that case, the PW related concept/procedures are not
applicable and that's all.

1.1. Multicast Snooping in VPLS

IGMP Snooping procedures described in[IGMP-SNOOP] make sure that IP
multicast traffic is only sent out of the following:

o Attachment Circuits (ACs) connecting to hosts that report related
   group membership

o ACs connecting to routers

o PseudoWires (PWs) connecting to remote PEs that have the above
   described ACs

Notice that traffic is always sent out of ports connecting to
routers, even those on which there are no snooped group memberships,
because IGMP Snooping alone can not determine if there are interested
receivers beyond those routers. To further restrict traffic sent to
those routers, PIM Snooping can be used, and this document describes
the procedures, including the rules when both IGMP and PIM are active
in a VPLS instance.

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 how the control messages are handled
(transparently flooded, selectively forwarded, or consumed and then
regenerated), the procedure/process may be called Snooping or Proxy
in different contexts.

Unless explicitly noted, the procedures in this document are used for
either PIM Snooping or PIM Proxy, and we will largely refer to PIM
"Snooping" in this document. The PIM Proxy specific procedures 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
Section 2.4.

This document also describes PIM Relay, which can be viewed as light-
weight Proxy. Unless explicitly noted, in the rest of the document
Proxy implicitly includes Relay as well.

1.2. Assumptions

The document assumes that the reader has a good understanding of the
PIM protocols. The text in this draft is written in the same style
as the PIM RFCs to help correlate the concepts and to make it easier
to follow. In order to avoid replicating the text relating to PIM
protocol handling here, this draft cross references into definitions
of macros and procedures from the PIM RFCs, and assumes that the user
will infer such detail from those PIM RFCs. Deviations in protocol
handling specific to PIM Snooping are specified in this draft.

1.3. **Definitions**

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

   o A port is defined as either an attachment circuit (AC) or a
      Pseudo-Wire (PW).

   o When we say a PIM message is 'received' on a port, it means that a
      PIM Snooping PE snooped the PIM message.

   Abbreviations used in the document:

   o S: IP Address of the Multicast Source.

   o G: IP Address of the Multicast Group.

   o N: Upstream Neighbor field in a Join/Prune/Graft message.

   o Rport(N): Port on which neighbor N is learnt.

   Other definitions are explained in the sections where they are
   introduced.

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. PIM shares many of the common
   characteristics of a routing protocol, such as discovery messages
   (e.g., neighbor discovery using Hello messages), topology information
   (e.g., multicast tree), and error detection and notification (e.g.,
   dead timer and designated router election). PIM does not participate
   in exchange of unicast routing databases, but it uses the unicast
   routing table to provide reverse path information for building
   multicast trees. There are a few variants of PIM. In [PIM-DM],
   multicast data is pushed towards the members similar to broadcast
   mechanism but routers without attached receivers will prune back
   towards the source. Unlike PIM-DM, other PIM flavors (PIM-SM
   [PIM-SM], PIM-SSM [PIM-SSM], and PIM-BIDIR [PIM-BIDIR]) employs a
   pull methodology via explicit joins instead of push 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 the upstream routers. 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.

o PIM Snooping MUST NOT affect the operation of customer layer-2
   protocols (e.g., BPDUs) or layer-3 protocols.

o PIM messages and multicast data traffic forwarded by PEs MUST
   follow the split-horizon rule for mesh PWs.

o PIM snooping states in a PE MUST be per VPLS instance.

o 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 Trigger

In PIM-SM/DM, there are scenarios where multiple routers could be
forwarding the same multicast traffic on a LAN. When this happens,
using PIM Assert Election process by sending PIM Assert Messages,
routers ensure that only the Assert Winner forwards 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 VPLS with snooping, two
routers may forward the same flow at the same time but each copy may
reach different set of PEs, and that 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, PIM-SM Snooping MUST not only forward multicast
traffic for an (S,G) on the ports on which they snooped Joins(S,G)/
Joins(*,G), but also towards the upstream neighbor(s)). In other
words, the ports on which the upstream neighbors are learnt must be
added to the outgoing port list along with the ports on which Joins
are snooped.

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 Rules. i.e. 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 procedures in PIM Snooping in the handling of PIM Hellos
and PIM Join/Prune packets are very similar to that of a PIM Router.

However, the PE does not need to have any routing tables like is
required in PIM Multicast Routing. It knows how to forward Join/
Prunes by looking at the Upstream Neighbor field in the Join/Prune
packets.

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

In the following sub-sections, we list some considerations and
observations for the implementation of PIM Snooping in VPLS.

2.3.1. Scaling

Snooping needs to be employed on ACs at the downstream PEs to prevent
traffic from being sent out of ACs unnecessarily. Snooping
techniques can also be employed on PWs at the upstream PEs to prevent
traffic from being sent to PEs unnecessarily. This may work well for
small to medium scale deployments. However, if there are a large
number of VPLS instances with a large number of PEs per instances,
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 Proxy 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 PEs signal the snooped states to other PEs out of
band via BGP, as described in [VPLS-MCAST-TREES]. In this document,
it is assumed that Snooping is performed on PWs.

2.3.2. IPv6

In VPLS, PEs forward Ethernet frames received from CEs and as such
are agnostic of the layer-3 protocol used by the CEs. However, as an
IGMP and PIM snooping PE, the PE would have to look deeper into the
IP and IGMP/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 draft can be applied to both
IPv4 and IPv6 payloads.

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

This draft does not address (*,*,RP) states in the VPLS network.
Although [PIM-SM] specifies that routers MUST support (*,*,RP)
states, there are very few implementations that actually support it
in actual deployments, and it is being removed from the PIM protocol
in its ongoing advancement process in IETF. Given that, this draft
omits the specification relating to (*,*,RP) support.

2.4. PIM Snooping vs PIM Proxy

The document has previously alluded to PIM Snooping/Relay/Proxy.
Details on the PIM Proxy/Relay 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 Proxy

Differences between PIM Snooping and Proxy/Relay can be summarized as
the following:

PIM Snooping	PIM Relay	PIM Proxy
Join/Prune messages snooped and flooded everywhere	Join/Prune messages snooped; forwarded as is out of certain upstream ports	Join/Prune messages consumed. Regenerated ones sent out of certain upstream ports
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

Note that the differences apply only to PIM Join/Prune messages. PIM

Hello messages are snooped and flooded in all cases.

Other than the above differences, most of the procedures are common
to PIM Snooping and PIM Proxy/Relay, 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 or new
packets introduced in the VPLS. It is also simpler to implement than
PIM Proxy. However for PIM Snooping to work correctly, it is a
requirement that CE routers MUST disable Join suppression in the
VPLS.

Given that a large number of existing CE deployments do not support
disabling of Join suppression and given the operational complexity
for a provider to manage 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 Proxy. If
there are a large number of CEs in a VPLS, then every CE will see
every other CE's Join/Prune messages.

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

PIM Proxy however loses the transparency argument since Join/Prunes
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. Data traffic will still be completely transparent.

2.4.2. PIM Control Message Latency

A PIM Snooping/Proxy/Relay 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 Proxy/Relay 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 proxy of Join/Prune and not Hello messages keeps the
PE behavior very similar to that of a PIM router without introducing
too much additional complexity. It keeps the PIM Proxy solution
fairly simple. Since Join/Prunes 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/Prunes). 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/Proxy include:

o Whether CEs do Join Suppression or not

o Whether Join/Prune latency is critical or not

o Whether the scale of PIM protocol message/states in a VPLS
   requires the scaling benefit of Proxy

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/Proxy 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, Explicit Tracking is possible,
allowing an upstream router to track all the downstream neighbors
that have Join states for any (S,G) or (*,G). That has two benefits:

o No need for PrunePending 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 if
   that's for the last downstream interface.

o For management purpose, the upstream router knows exactly which
   downstream routers exist for a particular Join State.

While full Proxy can be used with or without Join Suppression on CEs
and does not interfere with an upstream CE's bypass of PrunePending
process, it does proxy all its downstream CEs as a single one to the
upstream, removing the second benefit mentioned above.

Therefore, the general rule is that if Join Suppression is enabled on
CEs then Proxy or Relay MUST be used and if Suppression is known to
be disabled on all CEs then either Snooping, Relay, or Proxy MAY be
used while Snooping or Relay SHOULD be used.

An implementation MAY choose dynamic determination of 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.
i.e. 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:

o Layer 2 encapsulation for the Router sending the PIM Hello.

o IP Address and address family of the Router sending the PIM Hello.

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

o Hello TLVs

The PE should be able to interpret and act on Hello TLVs currently
defined in the PIM RFCs. The TLVs of particular interest in this
document are:

o Hello-Hold-Time

o Tracking Support

   o DR Priority

   Please refer to [PIM-SM] for a list of the Hello TLVs. 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 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 [PIM-SM] to identify the PIM Designated Router
   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. The root node of a tree is
   the Rendezvous Point (RP) in case of a shared tree (PIM-SM only) or
   the first hop router that is directly connected to the multicast
   source in the case of a shortest path tree. 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 Snooping States

   PIM-SM and PIM-SSM Snooping states are built by snooping on the
   PIM-SM Join/Prune messages received on AC/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):

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

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

   o Prune Pending Timer (PPT(N))

   o Join Expiry Timer (ET(N))

   Per downstream (Port, S, G):

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

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

o Prune Pending Timer (PPT(N))

o Join Expiry Timer (ET(N))

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

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

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

o Prune Pending Timer (PPT(N))

o 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. Notice that unlike on PIM-SM routers where PPT and ET are
per (Interface, S, G), PIM Snooping PEs have to maintain 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 is one or more Join(*,G) 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 is one or more Join(S,G) 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 Rport(N) ports where N is
in the set UpstreamNeighbors(*,G). Multicast Streams forwarded using
a (*,G) match MUST be forwarded to these ports in addition to
downstream ports. So UpstreamPorts(*,G) MUST be added to
OutgoingPortList(*,G).

UpstreamPorts(S,G): This is the set of all Rport(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 on Data Forwarding Rules for the specification on
how OutgoingPortList is calculated.

NumETsActive(Port,*,G): Number of (Port,*,G,N) entries that have
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): Number of (Port,S,G,N) entries that have
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.

RpfVectorTlvs(*,G): RPF Vectors [RPF-VECTOR] are TLVs that may be
present in received Join(*,G) messages. If present, they must be
copied to RpfVectorTlvs(*,G).

RpfVectorTlvs(S,G): RPF Vectors [RPF-VECTOR] are TLVs that may be
present in received Join(S,G) messages. If present, they must be
copied to RpfVectorTlvs(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 [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 facilitate the CE routers decide which Upstream
Neighbor gets elected the RPF-Neighbor. And for this purpose, the PE
will have to track downstream and upstream Join/Prune per (S,G,N).

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

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

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

o if both RPort(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 RPort(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 RPort(N) PW can be added to
the related forwarding entries' OutgoingPortList to trigger Assert,
but that is only needed for those states with AC UpstreamPort. Note
that in PW-only case, it is OK for the arrival port and RPort(N) to
be the same. See Appendix 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 is exactly as
specified in the PIM-SM RFC [PIM-SM].

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

Figure 1 : Downstream per-port (*,G) state machine in tabular form

		Previous State		
Event		NoInfo (NI)	Join (J)	Prune-Pend
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)	-

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 RPF
   Vector TLVs in the received (*,G) message and if they are
   different from the recorded RpfVectorTlvs(*,G), then copy them
   into RpfVectorTlvs(*,G).

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 to
      NoInfo. If there are no other (Port,*,G,N) state (different ports
      but for the same N), remove N from UpstreamPorts(*,G) - this also
      serves as a trigger for US FSM (JoinDesired(*,G,N) becomes FALSE).

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

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

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

   o if both RPort(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 RPort(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 RPort(N) PW can be added to
   the related forwarding entries' OutgoingPortList to trigger Assert,
   but that is only needed for those states with AC UpstreamPort. See
   Appendix 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 is exactly as
   specified in the PIM-SM RFC [PIM-SM].

Figure 2: Downstream per-port (S,G) state machine in tabular form

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

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 RPF Vector TLVs
   in the received (S,G) message and if they are different from the
   recorded RpfVectorTlvs(S,G), then copy them into
   RpfVectorTlvs(S,G).

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) state (different ports
   but for the same N), remove N from UpstreamPorts(S,G) - this also
   serves as a trigger for US FSM (JoinDesired(S,G,N) becomes 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 learnt.

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 4.5.4 of the PIM-SM RFC [PIM-SM].

2.6.6. Sending Join/Prune Messages Upstream

This section applies only to a PIM Proxy/Relay PE and not to a PIM
Snooping PE.

A full PIM Proxy (not Relay) PE MUST implement the Upstream FSM for
which the procedures are similar to what is defined in section 4.5.6
of [PIM-SM].

For the purposes of the Upstream FSM, a Join or Prune message with
upstream neighbor N is "seen" on a PIM Snooping PE if the port on
which the message was received is also Rport(N), and the port is an
AC. The AC requirement is needed because a Join received on the
Rport(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 Proxy case.

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

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

If RpfVectorTlvs(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 forwarded (in case of PIM Relay),
refresh and triggered (in case of PIM Proxy) (S,G)/(*,G) Join/Prune
messages.

o The upstream neighbor field in the Join/Prune to be sent is set to
   the N in the corresponding Upstream FSM.

o if Rport(N) is an AC, send the message to Rport(N).

o Additionally, if OutgoingPortList(X,G,N) contains at lease 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 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 they'll receive from each other, triggering
assert.

2.7. Bidirectional-PIM (PIM-BIDIR)

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

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

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

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

The main advantage of PIM-BIDIR 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 [PIM-BIDIR], parts of a PIM-BIDIR enabled network may
forward traffic without exchanging Join/Prune messages, for instance
between DF's and the RPL.

As the described procedures for Pim snooping rely on the presence of
Join/Prune messages, enabling Pim snooping on PIM-BIDIR networks
could break the PIM-BIDIR functionality. Deploying Pim snooping on
PIM-BIDIR enabled networks will require some further study. Some
thoughts are gathered 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:

o To maintain the list of multicast routers and ports on which they
   are attached, the PE SHOULD NOT use the rules as described in
   RFC4541 [IGMP-SNOOP] but SHOULD rely on the neighbors discovered
   by PIM Snooping . This list SHOULD then be used to apply the
   forwarding rule as described in 2.1.1.(1) of RFC4541 [IGMP-SNOOP].

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

2.9. PIM-DM

The characteristics 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 Snooping States

PIM-DM Snooping states are built by snooping on the PIM-DM Join,
Prune, Graft and State Refresh messages received on AC/PWs and State-
Refresh Messages sent on AC/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), "PrunePending" (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 [PIM-DM] with a few changes relevant to PIM
Snooping. When reading section 4.4.2 of [PIM-DM] for the purposes of
PIM-Snooping please be aware that 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 the previous
Section 2.9.1, the states (DownstreamPState) and timers (PPT and PT)
are per (S,G,N,P).

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
Rport(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 which is also an upstream port (UP). 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 UP 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 Proxy/Relay for VPLS Multicast to
work correctly. This section applies specifically to the full Proxy
case and not Relay.

2.10.1. Upstream PIM Proxy behavior

A PIM Proxy PE consumes Join/Prune messages and regenerates PIM Join/
Prune messages to be sent upstream by implementing Upstream FSM as
specified in the PIM RFC. 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 address picked 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

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 apart from 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:

o The PE would have to do ARP snooping to determine if a source is
   directly connected.

o Another option is to have configuration on all PEs to say 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.

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

This document recommends the third option - sources 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 is specified in the sub-
sections.

If there is no matching forwarding state, then the PE SHOULD discard
the packet, i.e., the UserDefinedPortList below SHOULD be empty.

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

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

    o 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 [PIM-SM] and per the additional rules specified in
   this document,

   OutgoingPortList(*,G) = immediate_olist(*,G) (+)
                           UpstreamPorts(*,G) (+)
                           Rport(PimDR)

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

   [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 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, 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
    Section 2.12

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

3. IANA Considerations

    This document makes no request of IANA.

    Note to RFC Editor: this section may be removed on publication as an
    RFC.

4. Security Considerations

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

5. Contributors

    Yetik Serbest, Suresh Boddapati co-authored earlier versions.

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.

6. Acknowledgements

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

7. References

7.1. Normative References

[PIM-BIDIR]
          Handley, M., Kouvelas, I., Speakman, T., and L. Vicisano,
          "Bidirectional Protocol Independent Multicast (BIDIR-
          PIM)", RFC 5015, 2007.

[PIM-DM]   Adams, A., Nicholas, J., and W. Siadak, "Protocol
           Independent Multicast Version 2 - Dense Mode
           Specification", RFC 3973, 2005.

[PIM-SM]   Fenner, B., Handley, M., Holbrook, H., and I. Kouvelas,
           "Protocol Independent Multicast- Sparse Mode (PIM-SM):
           Protocol Specification (Revised)", RFC 4601, 2006.

[PIM-SSM] Holbrook, H. and B. Cain, "Source-Specific Multicast for
           IP", RFC 4607, 2006.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119, 1997.

[RPF-VECTOR]
          Wijnands, I., Boers, A., and E. Rosen, "The Reverse Path
          Forwarding (RPF) Vector TLV", RFC 5496, 2009.

7.2. Informative References

[IGMP-SNOOP]
          Christensen, M., Kimball, K., and F. Solensky,
          "Considerations for IGMP and MLD Snooping PEs", RFC 4541,
          2006.

[VPLS-BGP]
          Kompella, K. and Y. Rekhter, "Virtual Private LAN Service
          using BGP for Auto-Discovery and Signaling", RFC 4761,
          2007.

[VPLS-LDP]
          Lasserre, M. and V. Kompella, "Virtual Private LAN
          Services using LDP Signaling", RFC 4762, 2007.

[VPLS-MCAST-REQ]
          Kamite, Y., Wada, Y., Serbest, Y., Morin, T., and L. Fang,
          "Requirements for Multicast Support in Virtual Private LAN
          Services", RFC 5501, 2009.

[VPLS-MCAST-TREES]
          Aggarwal, R., Kamite, Y., Fang, L., and Y. Rekhter,
          "Multicast in VPLS", draft-ietf-l2vpn-vpls-mcast-11, Work
          in Progress.

Appendix A. PIM-BIDIR Thoughts

This section describes some guidelines that may be used to preserve
PIM-BIDIR functionality in combination with Pim Snooping.

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

o on the RPL all traffic is forwarded to all Rport(N)

o on any other interface traffic is always forwarded to the DF

The information needed to setup these states may be obtained by :

o determining the mapping between group(range) and RP

o snooping and storing DF election information

o determining where the RPL is, this could be achieved by static
   configuration, or by combining the information mentioned in
   previous bullets.

A.1. PIM-BIDIR Data Forwarding Rules

The PIM-BIDIR Snooping 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
```
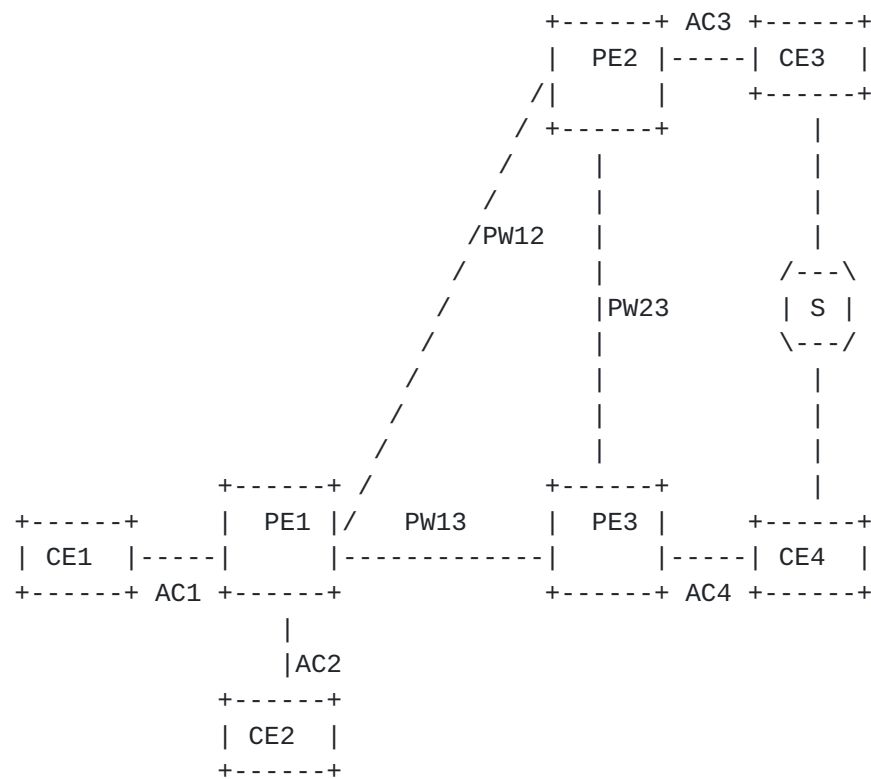
If there is forwarding state for G, then forward the packet to
olist(G) while observing the general rules above in [Section 2.12](#)

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

[Appendix B](#). **Example Network Scenario**

Let us consider the scenario in Figure 3.

      An Example Network for Triggering Assert

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

   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 Upstream Neighbor(S,G) = CE3.
   2. PE1 snoops on the Join(S,G) and builds forwarding states 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 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 is as follows:

       At PE1:
             JT(AC1,S,G,CE3)   = JP_HoldTime
             UpstreamNeighbors(S,G) = { CE3 }

```
            UpstreamPorts(S,G)      = { PW12 }
            OutgoingPortList(S,G)   = { AC1, PW12 }

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

        At PE3:
            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 Upstream Neighbor(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 existing
   (S,G) state with an AC in UpstreamPorts(S,G)

   The resulting states at the PEs:

```
        At PE1:
            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 }

        At PE2:
            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 }

        At PE3:
            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 the two CEs
   CE3 and CE4. PE2 forwards the stream received from CE3 to PW23
   and PE3 forwards the stream to AC4. This facilitates the CE
   routers to trigger assert election. Let us say 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 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-pend timer on the ports on which
         they received the Prune(S,G). When the prune-pend timer expires,
         all PEs will remove the downstream (S,G,CE4) states.

         Resulting states at the PEs:

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

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

            At PE3:
               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:

            At PE3:
               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 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 Upstream Neighbor(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 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 is 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 Joins(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 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 CE2 sends a Join(*,G) with UpstreamNeighbor(*,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) and though this is a PW-only join, forwarding
      state is build on this Join(*,G) since PE2 has 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 OutgoingPortsList(*,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 is 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. In case traffic is not flowing yet (i.e. step 3 is delayed to come 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 timer 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 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 the two

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 facilitates 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 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 FLAG sgrpt prune
    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 }

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

           JT(PW12,S,G,CE4)       = JP_Holdtime with FLAG sgrpt prune
           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 }

       At PE3:
           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 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, the reason we don't have duplicate traffic is because one of
the CEs stops sending traffic due to assert, not because we don't

have any forwarding state in the PEs to do this forwarding.

Authors' Addresses

   Olivier Dornon
   Alcatel-Lucent
   50 Copernicuslaan
   Antwerp, B2018

   Email: olivier.dornon@alcatel-lucent.com

   Jayant Kotalwar
   Alcatel-Lucent
   701 East Middlefield Rd.
   Mountain View, CA 94043

   Email: jayant.kotalwar@alcatel-lucent.com

   Venu Hemige

   Email: vhemige@gmail.com

   Ray Qiu
   Juniper Networks, Inc.
   1194 North Mathilda Avenue
   Sunnyvale,   CA 94089

   Email: rqiu@juniper.net

   Jeffrey Zhang
   Juniper Networks, Inc.
   10 Technology Park Drive
   Westford, MA 01886

   Email: zzhang@juniper.net