Allied Telesis

Routing Protocols Guide

Version 1.0

Compiled by

Andrew Riddell

Understanding Routing Protocols

Compiled by:

Andrew Riddell

About Allied Telesis, Inc.

Founded in 1987, Allied Telesis is a world class leader in delivering IP/Ethernet network solutions to the global marketplace. We create innovative, standards-based IP networks that seamlessly connect users with their voice, video and data services. We are an international company headquartered in Japan with major divisions in Europe, Asia and North and South America. Our partners include the world's largest distributors, integrators, solution providers and resellers to assure you receive immediate local service and support. Allied Telesis has been designing, manufacturing and selling networking products for over 25 years.

Our philosophy of producing products of the highest quality, at affordable prices, has resulted in Allied Telesis products being deployed in networks of all types and sizes across the world. Our proven track record of providing solid technology, excellent support and full feature products has allowed Allied Telesis to become the worldwide de-facto standard in many areas of technology.

With a portfolio of products that can provide end-to-end networking for Service Provider, Enterprise and SMB customers, Allied Telesis is the natural choice for many world class organizations.

Please visit us online at **alliedtelesis**.com

Author

Andrew Riddell

Contributors

Cameron Drewett

Steve Canton

Technical Writer and Editor

Carin van Bolderen

Illustrator

Angela Devonport

Reviewers

Andrew Riddell

Miranda de Gouw

Emma Johnson

Graphic Designer

Marnie Hart

Produced in New Zealand 2014
Document Number: C613-04007-00-REV B
ISBN: 978-0-9941128-0-4

About this Guide

What is routing?

Routing, in essence, is the act of finding a path from one place to another on which a packet can travel. Individual hosts in a network have addresses assigned to them, and other hosts send off packets destined to those addresses, trusting that the network will deliver the packets to the intended recipient.

In a small network, that is not a difficult matter, forwarding devices can hold a list of all the current addresses in the network, and the port via which to reach those addresses.

But, for larger networks, this simply is not possible. Addresses need to be grouped in blocks, and forwarding nodes contain lists of these blocks of addresses, rather than lists of individual host addresses.

The analogy with a postal service is quite appropriate. In a large mail sorting facility, mail will be sorted into bags destined to whole countries or whole cities, rather than individual little containers for each single recipient. It is only at the eventual local delivery office that mail is sorted down to bundles for individual delivery addresses.

The difference between a data network and a postal service is the dynamic nature of a data network. Postal services don't need to cater for blocks of addresses moving around at a moment's notice, but data networks do need to. So, data networks need to be inherently based on a model whereby the paths to given addresses or blocks of addresses are learnt dynamically.

The path-learning processes that enable data networks to operate in this dynamic manner are routing protocols.

Routing protocols

Routing protocols enable data networks to be self-organizing systems. Routers are aware of the blocks of addresses that are directly connected to them and, with the aid of routing protocols, advertise this knowledge to other routers. The recipients of these advertisements relay the information on to yet other routers, and so on, until the knowledge of which address blocks are located where is spread through the whole network.

This concept is generic to all Layer 3 networking systems – AppleTalk, DECnet, IPX, IP and others. Over the years, of course, IP has become the dominant Layer 3 protocol throughout the world, so this book looks exclusively at the routing protocols used by IP.

Three major classes of unicast routing protocol are in widespread use on IP networks:

- Link-state routing protocols, most notably OSPF.
- Distance-vector routing protocols, most notably RIP.
- Exterior gateway protocols, most notably BGP.

Additionally, Layer 3 forwarding of multicast needs routing protocols to enable listeners in one subnet to connect to streams being generated by sources in another subnet. Multicast routing, as we will see, operates significantly differently to unicast routing.

Understanding the finer details of IP routing protocols enables network engineers to create routing schemes that are:

- Well suited to the performance capabilities of their network equipment.
- Capable of supporting multiple paths offering different types of service.
- Easy to understand, troubleshoot and maintain.

What information will you find in this Guide?

This guide to IP routing protocols is divided into three main sections:

- IPv4 unicast routing protocols: RIP, OSPF, BGP and Route Filtering.
- IPv6 unicast routing protocols: Intro to IPv6, RIPng, OSPFv3 and BGP4+.
- Multicast routing protocols: Layer 2/Layer 3 Multicasting, MLD, PIM-DM, PIM-SM, PIM-SSM, and PIMv6.

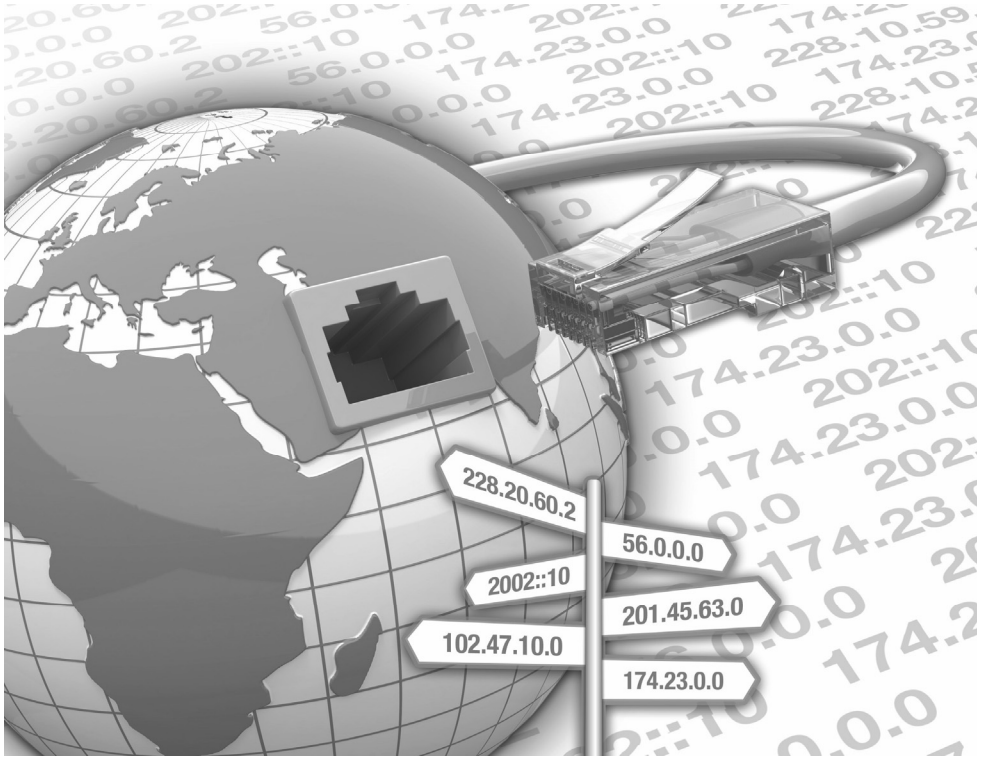
Who is this guide written for?

This guide was written for people who already know a reasonable amount about IP networking, and want to get a deeper understanding of the finer details of routing protocols.

Table of Contents

Section I

IPv4 Unicast Routing Protocols



This chapter covers the following topics:

- Introduction
- RIP operation basics
- Withdrawing routes
- Actively removing a route that has gone down
- Authentication
- Differences between RIPv1 and RIPv2
- Timers
- Redistributing routes into RIP
- Packet formats
- RIP commands

CHAPTER 1

Routing Information Protocol (RIP)

Introduction

The basic idea of RIP is very simple. Routers take their IP route table, and advertise it to other routers at regular intervals. RIP does not require sessions to be negotiated. It does not require neighbors to be configured (although, the configuration of specific neighbors is possible). It does not associate much information with each route (just the route's metric). It does not store a lot of state. Simply, routers send out their route table in RIP packets, and hope that neighbors will pick up these packets, and make use of the routing information that they contain.

Metric calculation with RIP is very simple. It is a distance vector protocol, and the metric value of a route increases by one (or more if a bigger increment is configured on a router) each time it is passed from one router to another.

This simplicity of RIP makes it quite attractive for small networks with a few routes, where highly responsive recovery from lost links or nodes is not essential. It is simple to understand, simple to configure, and has very little that can go wrong.

However, for larger networks, RIP does not really "*cut the mustard*".

The CPU overhead of advertising the whole of a large route table at regular intervals can noticeably impair the performance of routers. The slow propagation of new routes through a network is not acceptable in a dynamic environment. The simple metric calculation takes little account of path costs. RIP is not the routing protocol of choice in large and/or high performance networks. But it does have its place in small, simple networks.

In this explanation of RIP, we will proceed as follows:

- Describe the basics of how RIP advertises routes, how metrics are calculated, etc.
- Discuss the difference between RIPv1 and RIPv2.
- Consider the problems that can occur in a very simple implementation of RIP, and look at the extra features that have been added to RIP to resolve those problems.
- Present the details of the format of RIP packets.

RIP Operation Basics

Which version of RIP?

The first thing we need to establish is which version of RIP we are talking about. There are two versions – version 1 and version 2. Whilst version 2 is superior to version 1, and version 1 is little used these days, there are still networks using version 1. Hence, people are interested in understanding version 1, so we cannot completely ignore it. However, throughout this section, we will be primarily focusing on RIP version 2. Any statements we make about the protocol will definitely apply to version 2, and might also apply to version 1. However, do be aware that some of the description definitely will **not** apply to version 1.

The section Differences between RIPv1 and RIPv2 on page 12, does describe the differences between version 1 and version 2. At places in the text you will be referred to this section when we are describing aspects of the protocol that differ between version 1 and version 2.

Protocol operation

RIP packets are forwarded over UDP to port **520**; the source port of the packets is also 520. The packets can be multicast, broadcast, or unicast. The typical mode is multicast to the address 224.0.0.9. For more information on this, see page 12, Differences between RIPv1 and RIPv2. This is the default behavior in the AlliedWare Plus™ operating system. If you wish to send the packets by unicast, then you must configure specific neighbor addresses for RIP to be sent to.

Routes sent in a RIP update are not necessarily all the routes in the router's route table. Rather, the router sends those routes that have been allocated to the RIP table by network commands or by redistribution. The route Update packets can hold up to 25 routes each. The routes that are being advertised are split across as many packets as a required to carry all the routes. The information that is advertised with each route is:

- the subnet address
- the subnet mask (in RIPv1, the mask was not sent, (see page 12, for more on this)
- the next hop address
- the metric
- a tag and an address-family identifier. These are described in the section Identifiers in the update packets on page 185.

To attach RIP to one or more interfaces using AlliedWare Plus, configure as follows:

```
awplus# conf t
awplus(config)# router RIP
awplus(config-router)# network a.b.c.d/x
```

Then, all the interfaces whose IP address falls within the range covered by a.b.c.d/x will become active RIP interfaces. The connected routes on all those interfaces will be automatically installed into the RIP route table.

By default, the RIP packets sent out the active interfaces are RIP version 2 packets to the multicast address 224.0.0.9.

To also send unicast updates to specific hosts, configure as follows:

```
awplus# conf t
awplus(config)# router RIP
awplus(config-router)# neighbor <neighbour address>
```

Configuring a specific neighbor will not stop the sending of multicast updates out of the interface to which that neighbor is attached. The unicast and multicast updates will both be sent out the interface.

To stop the multicast updates, and send just the unicast updates, the following extra configuration is required:

```
awplus# conf t
awplus(config)# Router RIP
awplus(config-router)# passive-interface <interface from which to stop
sending multicasts>
```

Interestingly, the meaning of `passive-interface` is rather different in RIP to OSPF or PIM. In those more complex protocols, a passive interface is one on which no protocol packets are sent at all, and no neighbor relationships are created. However, as RIP is rather simpler, and has no neighbor negotiation process, so the concept of a Passive interface is reassigned to just turning off the sending of general (as against neighbor-specific) updates from the interface.

RIP sends route update packets in three circumstances:

- regular periodic updates
- in response to update requests
- as triggered updates when a route change occurs

Periodic updates

The default period for the periodic updates is 30 seconds. A different value for the period can be configured – see the section Timers on page 16 for a description of changing the timers.

The periodic updates are sent on all active RIP interfaces, and are sent to the multicast and/or unicast addresses, as described above.

Responding to update requests

There are two types of update requests – requests for specific sets of routes and requests for full updates.

A request for a **specific set of routes** contains the requested routes, with undefined values for the routes' metrics. The task for the responding router is to go through the routes in the request packet, and see which of them it has a route entry for in its own route table. For those that it does have an entry in its route table, it takes the metric for that route from its RIP route table, and puts that metric with the route into the response packet. For those routes that it does not have an entry in its route table, it puts an **infinite** metric (i.e. metric=16) with the route into the response packet.

When the response packet has been filled in, it is sent to the unicast address of the requesting router.

If the request packet has just one route entry, namely the route 0.0.0.0, with metric=16, then it is a request for a **full update**.

When a router joins the RIP network, it multicasts RIP full-update requests out all its interfaces, asking all other routers to send their complete updates. This way, the new router can learn routes quickly, rather than having to wait for the neighbors' next periodic updates.

When routers receive requests for a full update, they send out a full route update, unicast to the address of the requestor.

Triggered update

When a change occurs in a router's route table – for example, when an interface goes down, causing the router to lose a number of routes, or a new redistribution is configured, which brings a number of new routes into the RIP route table – it is best for other routers to find out about this as soon as possible.

Of course, a router could just wait until its next periodic update, and tell neighbors about the route-table change then. But, it is much better if they send out this information as soon as the change occurs.

The act of sending out update messages specifically in response to a change in the RIP route table is called a **Triggered update**.

A triggered update does not need to send all the routes that would be sent in a full update. It should only send those routes that have changed.

The update is sent to all active RIP interfaces. Well, in fact, it may turn out that for some interfaces there will be no point in sending the update. For instance, if the only change is that a new neighbor has appeared on interface X, and so new routes are being learnt from this neighbor, then there is actually no point in telling other neighbors attached to interface X, they will have already found out for themselves.

This business of taking note of which interface information was learnt on, and not re-advertising it back out that same interface, is an example of **Split Horizon**, which will be discussed in more detail in the section Split horizon on page 14.

Metrics

As mentioned above, RIP has a very simple scheme for calculating metrics – it is just a hop count that can be augmented by routers adding a bit of a metric boost on some of their interfaces.

The **defaults** for calculating metrics are as follows:

- Connected routes brought into RIP are given a metric of 1.
- Routes redistributed into RIP are given a metric of 1. This can be changed by using the command `default-metric <metric>` to set the default metric for redistributed routes to be some value greater than 1.
- When a router receives a route by RIP, it adds 1 to the metric before putting the route into its RIP route table.
- When a router advertises a route by RIP, it just advertises it with the metric that the route has in the router's route table; it does not add anything to the metric at the time of advertising the route.

The default behavior with regard to metrics when sending and receiving routes can be altered by the **offset-list** command:

```
awplus(config-router)# offset-list <access-list> in <offset> [<interface>]
awplus(config-router)# offset-list <access-list> out <offset> [<interface>]
```

The effect of this command is that routes being exchanged on a given interface, and which match the specified access list, will have their metric boosted by the amount **offset**.

If the first of the two commands is used, then the metrics of the matching routes are boosted as the routes are received. As the received routes are installed into the RIP route table, their metrics are increased by the configured **offset** rather than just being increased by 1.

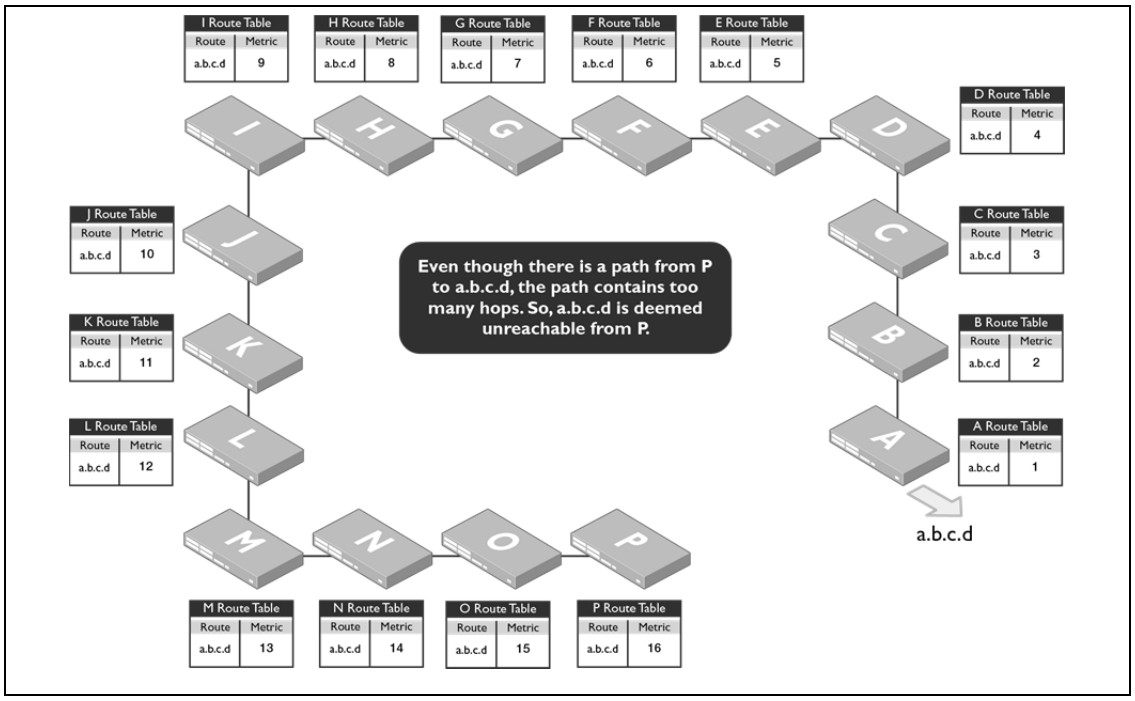
If the second of the two commands is used, then the metrics of the matching routes are boosted as the routes are being sent. As the routes are being put into a RIP packet to be advertised, the metric value they are advertised with is not simply the metric they currently have in the router's RIP route table. Instead, the metric value advertised with each route that matches the ACL (Access Control List) is:

```
<metric the route has in the RIP route table> + offset
```

What is an infinite metric?

An interesting feature of RIP is that a metric of 16 is considered to be an infinite metric. What exactly does an 'infinite' metrics mean? In short, it means unreachable. If a route is given metric 16, then it is effectively marked as dead.

Once a route has been passed through 15 routers, it is not accessible to any routers further away.



This puts a strict limit on the diameter (the number of hops between the most separated pair of routers in the network) of a RIP network. Initially, it seems rather odd to put this quite low limit on how far a RIP route can be propagated. But, we will see below, in the section: Counting to infinity on page 14, why a lowish limit is desirable.

In reality, though, the 16-hop limit is not the most significant factor in limiting the size of RIP networks. The processing overheads of RIP, as the number of routes in the network grows, are typically a much stronger limiting factor than the 16-hop limit.

As well as being used to indicate that a route has been propagated too far, the **infinite** metric is a very convenient way to flush a dead route out of the network. Once a RIP route has been assigned metric 16, it should no longer be used, and should be flushed out of the route table after a certain length of time. If a router determines that one of the routes it is advertising by RIP is no longer available, it can inform other routers not to use this route any more by sending out a triggered update that advertises this route with metric 16. There is more discussion of this in the section Timers on page 16.

Withdrawing Routes

Every routing protocol needs a way to withdraw routes when they are no longer reachable.

The quicker a dead route is removed, the better. But, a protocol does have to be careful not to be too trigger-happy in removing routes. RIP has a couple of different mechanisms for withdrawing routes.

Timing out a route that disappears from updates

RIP needs a method to deal with the case that a router that is advertising one of more routes just simply stops advertising them - maybe the router has a power failure, or its connection to the network gets unplugged for some reason, or its RIP module stops working.

In this situation, the other routers in the network need to get rid of the routes they learnt from the now-uncommunicative router, as they cannot be sure that those routes are still accessible. But, they can't just get rid of the routes the very first time that they fail to get an update from that router, as that could lead to route flapping.

Instead, RIP has a staged set of actions that are taken to remove such routes in a measured fashion.

When a router receives a route by RIP, and puts the route into its RIP route table, it starts a timer, called the Invalid timer. By default, the timeout period of the timer is 180 seconds. If the timer expires, then the router will set this route's metric to 16.

The Invalid timer is reset every time the route is seen in an Update packet. So, if the route continues to be present in the periodic updates that the router receives, then the Invalid timer will keep on being reset, and will not time out.

However, if a route just stops appearing in the periodic updates that a router is receiving, then, when the route has been absent from sufficient updates in a row, the router will set the metric for this route to 16. By default, the route must be absent from 6 consecutive updates before the router sets its metric to 16. The 180-second period of the Invalid Timer is equal to 6 of the 30-second update periods.

Once a route has been set to metric 16, it will not be used for forwarding packets. But, it is still present in the route table, and will be advertised in RIP updates. This is so that the router that has set the metric to 16 can inform other routers that the Route is no longer reachable.

The route remains in the route table, with metric 16 for a 120 second period, called the Garbage Collection time. If no-one starts advertising this route again, with a non-infinite metric, within the Garbage Collection time, then the route is removed from the route table at the end of the 120 seconds. However, if the router does receive an Update from a neighbor, that advertises a finite metric for this route, then the route will be re-instated into the route table, with the new, non-infinite metric.

There are some implementations of RIP that behave a little differently once the route has been set to Metric 16. This alternative implementation waits just 60 seconds between setting a routes' metric to 16 and flushing it out of the route table. More importantly, this alternative implementation does not allow a route to be re-instated into the route table during these 60 seconds, even if a new update is received that is advertising the route with a non-infinite metric. So, the 60 seconds become a HoldDown time, during which the route's metric value is locked at 16. The purpose of this HoldDown time is to reduce the incidence of routes flapping.

This concept of a hold down time is not specified in the RIP RFCs, and is not implemented in AlliedWare Plus.

Note – the periods of the Update, Invalid, and Garbage Collection timers are all configurable. See the Timers section, on page 16.

Actively removing a route that has gone down

If a router sees an event that causes a route to become inactive, then it can immediately inform its neighbors of this event.

The most common case is when one of the interfaces on a router goes down, and the connected route on that interface was being advertised by RIP. In this case, the router can definitely say that this route is no longer available, and can set about making the RIP network aware of this fact as soon as possible.

The router makes use of the fact that advertising a route with metric 16 indicates that it is a dead route. So, when it sees the route become unavailable, it sends out a triggered update that advertises this route with metric 16.

Neighbors that receive this triggered update will immediately set the metric of their copy of this route to 16, and send out a triggered update to their neighbors to inform them that the route is now dead.

The wave of triggered updates spreads across the RIP network quite quickly. A route can be withdrawn from a RIP network in very few seconds.

Note - even in the case where a route is set to metric 16 in this manner, the router still waits for the 120 second Garbage Collection time before flushing the route out of its route table.

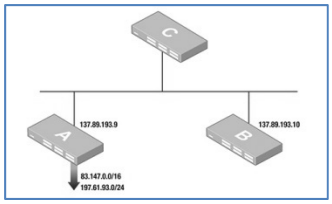
Next Hops

In RIPv2 packets, every route that is advertised is accompanied by a next hop address. RIPv1 packets do not include next hop addresses – see Differences between RIPv1 and RIPv2 on page 12.

In general, when a router advertises a bunch of routes, it is the next hop for those routes. The RIP protocol supports a shorthand whereby advertising the address 0.0.0.0 as the next hop for a route says "*the next hop is actually the source address of this RIP packet*".

To watch this video, browse to the link or scan the QR code with your smart phone:

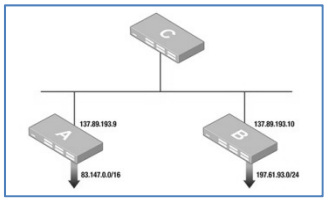
http://youtu.be/BMABiljXO60



However, there are plenty of cases where a router needs to be able to say "*don't access this route via me, go via this other router instead*". In this case, the *other* router's address is advertised as the next hop address with the route.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/n4T5t1OEYd0



Authentication

RIP can use plain text or MD5 authentication. Authentication is configured on a per-interface basis, using the commands:

```
awplus# configure terminal
awplus(config)# interface <interface-name>
awplus(config-if)# ip rip authentication mode {md5|text}
```

Having enabled authentication on an interface, you then need to tell RIP what the authentication password is. You can configure RIP to use a single static password, or a set of passwords that can change over time.

The configuration to set up a single static password is:

```
awplus# configure terminal
awplus(config)# interface <interface-name>
awplus(config-if)# ip rip authentication string <auth-string>
```

Setting up a set of changing passwords is rather more complex, involving a structure called a key-chain.

Key chains

When you want your routers to automatically change the authentication key they are using, you need to set up a set of keys that they will work through over time, and tell the routers when to change from using one key in the set to using the next key.

This combination of a set of keys and the set of times for changing keys is referred to as a **key chain**. The same key chain needs to be set up on all the routers that are neighbors on a given VLAN, so that they will all use the same keys at the same time.

Once the key chain is configured on the routers, and RIP is configured to use this key chain, then what happens is:

- Initially all the routers will use the first key in the chain as their authentication password.
- At the time that is configured for finishing with this first key, the routers will stop using this key as their authentication password.
- At the time that is configured for starting to use the next key, the routers will change to using that next key as their authentication password…and so on.

There is one complication that needs to be dealt with. If the **clocks** are slightly different on the different routers, it could be possible that one router cuts over to using a different key before the others.

Some tolerance for this situation needs to be built into the authentication process. The way this is achieved is by being able to specify the period during which a given key will be accepted as a valid password.

The idea is that you configure routers to continue to accept Key #1 as a valid password even after they have cut over to sending Key #2 as the password in the packets they send. There will be a period, at cut-over time, when the routers will accept either Key#1 or Key#2 as a valid password.

Let's look at how key chains are configured. A key chain consists of a set of keys. Associated with each key are:

- A send-lifetime—the start and end times of the period during which the key will be sent.
- An accept-lifetime— the start and end times of the period during which the key is a valid match.

To create a key chain, called rip-key-chain, proceed as follows:

```
awplus(config)# key chain rip-key-chain
awplus(config-keychain)# key 1
awplus(config-keychain-key)# key-string piano8trip
awplus(config-keychain-key)# accept-lifetime 08:30:00 Jan 09
2012 21:00:00 Feb 10 2012
awplus(config-keychain-key)# send-lifetime 09:00:00 Jan 09
2012 20:00:00 Feb 10 2012
awplus(config-keychain)# key 2
awplus(config-keychain-key)# key-string hop78run
awplus(config-keychain-key)# accept-lifetime 19:50:00 Feb 10
2012 21:00:00 Mar 10 2012
awplus(config-keychain-key)# send-lifetime 20:00:30 Feb 10
2012 20:00:00 May 10 2012
```

To configure particular interfaces to use MD5 authentication of RIP, with a changing key provided by rip-key-chain, proceed as follows:

```
awplus(config)# interface vlan1
awplus(config-if)# ip rip authentication mode md5
awplus(config-if)# ip rip authentication key-chain rip-key-chain
```

Transporting authentication information

The RIP packet header does not have a field set aside for carrying authentication information, Instead RIP authentication information is carried in the place of a route entry. When RIP updates are sent from an interface that has authentication enabled the first route entry in each Update is actually the authentication field. The field is identified as an authentication field, rather than a route entry, by the fact that is has FFFF in the first two bytes. This is discussed in more detail in the section: Packet Formats on page 20.

Because the first route entry field is used for holding the authentication information, a RIP update sent via an authentication-enabled interface can carry only 24 routes.

Differences between RIPv1 and RIPv2

RIPv1 is really a very minimal protocol. The information carried with each route is barely enough to properly identify the route. Actually, the information that RIPv1 carries with a route is not enough to properly identify the route.

RIPv1does not:

- carry subnet masks
- carry next hops with advertised routes
- support any authentication process
- send packets to the multicast address 224.0.0.9

Let's look at the details.

The place to start is the really BIG flaw in RIPv1 – it does not carry **subnet masks**. So, when a RIPv1 router receives a route from a neighbor, it has to guess the subnet mask. You might think "*How could anybody be so thoughtless as to create a routing protocol that does not carry subnet masks? What on earth were they thinking?*" But, the fact is that RIP v1 was designed back in the days when the concept of IP address classes was king. In those days, it really was expected that all IP subnets would be allocated classful addresses, so there would be no need to worry about cut up portions of classful networks being passed around.

The assumption in the design of RIPv1 was that subnet masks were irrelevant – you'd just look at the address of the route, and the mask would be obvious, based on the class that the address fell within. In retrospect, of course, this does seem awfully naive.

As we know, unfortunately for RIPv1, the practice of assigning portions of classful networks became commonplace. As a result, RIPv1's behavior of not carrying masks with the subnets it advertised became downright limiting.

- RIPv1 also did not carry next hops with advertised routes. It was not possible for a RIPv1 router to tell a neighbor, "Don't come through me to get to this route, go through that router over there". Hence, all routers would have to assume that the neighbor from whom they received a route was the next hop for that route.
- RIPv1 did not support any authentication process.
- RIPv1 does not send packets to the **multicast** address 224.0.0.9. Instead, RIPv1 broadcast packets in a subnet, or sent them to specific configured unicast neighbors.

RIP Implementation Problems and Solutions

RIP, as we have described it so far, is quite functional.

It enables routers to learn routes from each other, to choose the best route (based on a number-of-hops metric), and to withdraw dead routes. However, if the protocol operated strictly as we have described so far, there would still be a major inefficiency when a router or a link goes down.

Let's look what happens in this case. Consider the case illustrated in the animation on page 14, that A is advertising a route x.y.z.0 to B, with metric 10.

- B will put the route into its route table with metric 11, and advertise it with metric 11 to C and D.
- C and D will put the route x.y.z.0 into their route tables with metric 12.

Then, let's think about what happens if router A goes down.

After 180 seconds of not seeing any route updates from A, router B will set the metric on x.y.z.0 to 16, and advertise this metric=16 route to C and D. However, at that point, C and D still have a route with metric 12, and are both advertising that route onto the LAN that they share with B. C will say "*I have an update from B saying the metric for the route to x.y.z.0 is 16, but I just had an update from Esaying that the metric is 13. I will use the lower-metric route*". So, it will update its route to x.y.z.0 to have metric 13. Similarly, D will have recently seen an update from E saying that E's metric for the route to x.y.z.0 is 13; so D will also update its route to x.y.z.0 to have metric 13.

And, B will also receive updates from B and C. So, it will also update its metric for the route to x.y.z.0 to be 14.

Then, when the next round of updates is sent out, everybody will be telling each other that they have a metric-13 route to x.y.z.0. Hence, when each of B, C, D receives each other's updates, they will increment their metric for the x.y.z.0 route to 14. At the next round of updates, everybody's route to x.y.z.0 will end up with metric 15.

Then, finally, at the next round of updates, all of B, C, D will have a metric of 16 for the route to x.y.z.0. S

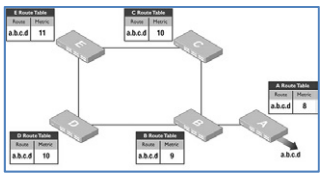
Counting to infinity

A metric of 16, of course, is the infinite metric, which means that the route is dead. Two minutes later, they will all flush this route out of their route tables. Finally, the inaccessible route x.y.z.0 will be removed from the network.

So, whilst the final outcome is the desired result, it was rather a slow painful way of getting there. All the routers clinging onto their old copies of the route to x.y.z.0, and continuing to deceive each other into thinking that the route still exists is rather inefficient. This process of a route being advertised and readvertised, with an ever incrementing metric, until it finally hits the infinite metric, is called **counting to infinity**.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/bHltWL1IwoY



To eliminate this slow process of eliminating dead routes by counting to infinity, two elements have been added to RIP – Split Horizon and Poison Reverse.

Split Horizon

The fundamental reason why the counting to infinity occurs is because routers C, D,E keep on advertising the route x.y.z.0 back onto the LAN segment from which they learnt it. This act of advertising a route back out the interface on which it was received is both pointless and error-inducing.

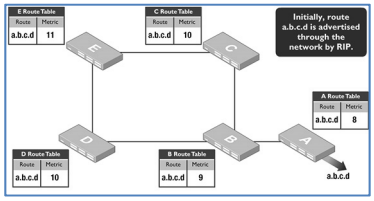
It is pointless because if you learnt a route from a neighbor attached to your VLANx interface, then the other RIP routers attached to VLANx will also have that route from the same neighbor. There is no point in re-telling them something they have already learnt. It is error inducing, as it causes the process of mutual deception that we have just walked through.

By default, RIP routers do not advertise routes back out the interface on which they learn them. This is referred to as the rule of **Split Horizon**. Let's now go back and examine what happens when router A goes down if each of B, C, D, and E is observing the rule of Split Horizon.

When router B advertises an update that indicates a metric of 16, routers C and D will receive that update, and also update their metric for x.y.z.0 to 16, as they will not have been receiving updates from each other that advertise the route with a lower metric. So, rather than go through the slow process of gradually counting the metric to 16, they will set it to 16 straight away, and flush it two minutes later.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/8cR2lunq3rw



Poison Reverse

Split Horizon is a rather passive way of avoiding deceiving other routers into thinking that you have a path to a route that you actually just learnt from them in the first place. But, it is possible to take an active approach to this. Rather than merely not advertising routes back out the interface on which they were learnt, you can actively send out messages on the interface to say "*don't even dream of trying to access this route via me*". This active approach is referred to as **Poison Reverse**.

There are two common implementations of Poison Reverse:

1. Taking all the routes learnt via each interface, and advertising metric=16 copies of those routes back out the same interfaces at every update. So, the updates that the router sends out an interface include all the routes it is actually advertising, plus metric=16 copies of all the routes it has learnt via that interface.

2. Only advertising a metric=16 route back out an interface if the router has already received a metric=16 advertisement of that route on that interface.

AlliedWare Plus implements the **first** form of Poison Reverse.

What actual advantage does Poison Reverse bring? The answer most of the time is 'not a lot'. The removal of a dead route, as described above in the discussion of Split Horizon, is not altered by Poison Reverse. The sending of the Poisoned routes does not speed the process up. Router B will still take 180 seconds before assigning metric 16 to the routes it was receiving from A, and then take a further 2 minutes to fully flush them.

The main value of Poison Reverse is as a **back-up** mechanism to sort out cases where, for some reason, a router has got the idea into its head that a particular route is accessible via a router that it is not. If the router that is mistakenly thought to be the gateway to some route is sending out metric=16 updates for that route, it will quickly clear up such mistakes.

Rather than a "*speed up the removal of dead routes*" mechanism; Poison Reverse is more a "*mop up any mistakes*" mechanism.

Timers

As with any protocol, RIP has timers to control when actions occur. A characteristic feature of RIP is that its timers are long.

There are three key RIP timers that are configurable under AlliedWare Plus, they are: Update, Invalid, and Garbage Collection timers.

Update timer

This governs the period at which periodic updates are transmitted. By default this timer is 30 seconds. It can be configured to be anything between 5 and some enormous number (`2147483647`) of seconds. Setting the timer low does not necessarily propagate routing changes through the network more quickly, as changes should be propagated by triggered updates. However, increasing the frequency of updates will reduce the time taken for discrepancies between routers' route tables to be sorted out. Increasing the frequency of the updates does add to the amount of RIP traffic on the network, and add more load to routers' CPUs.

However, given that RIP should only be used in small networks (those with small route tables to pass around), then the extra load caused by a higher update frequency should not be a major problem. On balance, the advantages of increasing the update frequency outweigh the disadvantages. While, for many protocols, it is advisable to stay with default values for timers, it should be borne in mind that RIP was defined many years ago, when link bandwidths and CPU speeds were much lower than they are today. Therefore, lowering the update time to 5 or 10 seconds is a good idea.

Invalid timer

The section Timing out a route that disappears from updates on page 7, contains a description of the Invalid timer. In short, it is the time that a router takes to set metric=16 on a route which it is no longer being seen in the updates the router is receiving. By default it is 180 seconds (6 times the default update timer). This default time is quite conservative, and is one of the major reasons why RIP is slow to react to lost routes. It is certainly advisable to set this timer to less than the default. While you do not want your network to be too trigger-happy in deciding that routes have been lost, the act of waiting for 6 full update periods before deciding that the absence of a route is not just a transient event is rather too far on the conservative side. Particularly on congestion-free Ethernet networks, there really is no need to allow for the possibility of 6 lost Update packets.

Therefore, setting the invalid timer to 3 times the update timer is advisable. For example, if the update timer has been set to, say, 10 seconds, the invalid timer would be 30 seconds. This means that Dead routes would stop being used far more quickly than the default 180 seconds, without introducing any significant risk of route flapping.

Garbage collection timer

The 2 minutes that the router keeps a metric=16 route hanging around before actually flushing it is also described in the section Timing out a route that disappears from updates on page 7. The length of this timer does not cause any delay to the convergence of the network onto using the correct set of existing routes; once the route has been set to metric=16, it stops being used for packets forwarding, so it really does not matter too much how long it hangs around in this state. The value of keeping the metric=16 route in the route table for a while is to make sure that other routers will see it in updates, and also set the route's metric to 16 in their route tables.

Changing the timers

The values of these three timers can be changed using the commands:

```
awplus# configure terminal
awplus(config)# router rip
awplus(config-router)# timers basic <update> <timeout> <garbage>
```

The three values all need to be provided when the command is entered. If you want to leave one or two of the timers at default values, then provide those default values when entering the command.

Note - the <timeout> timer represents the invalid timer.

Redistributing Routes into RIP

Routes can be redistributed into RIP from OSPF and BGP, as well as static and connected routes. The command to designate redistribution is:

```
awplus# configure terminal
awplus(config)# router rip
awplus(config-router)# redistribute {connected|static|ospf|bgp} [metric <0-
16>] [routemap <routemap>]
```

By default, redistributed routes are given a metric of 1. But, this default metric for redistributed routes can be globally changed by using the command:

```
awplus(config-router)# default-metric <metric>
```

Then, for redistribution of specific route types (OSPF or Static, etc.), the metric assigned to the redistributed routes can be overridden by specifying a value for Metric in the redistribute command.

- The routemap on the **redistribute** command filters which routes will be redistributed.
- A routemap cannot be used to configure a tag value to be carried with routes when they are advertised by RIP.

Generating a default route

If routes are being redistributed into RIP, and the router's route table contains a default route, within one of the route categories that are being redistributed, the router will automatically advertise this default route.

Viewing RIP information in AlliedWare Plus

The key piece of information that you need to see in RIP is the content of the RIP route database. This can be viewed with the command:

```
show ip rip data full
```

```
awplus# show ip rip data full
Codes: R - RIP, Rc - RIP connected, Rs - RIP static C - Connected, S - Static, O - OSPF, B - BGP
   Network            Next Hop         Metric From            If       Time
Rs 0.0.0.0/0                                1                  --
S  10.0.0.2/32        126.0.0.19            1                 vlan2
S  81.211.37.99/32    126.0.0.19            1                 vlan2
S  125.0.0.0/8        126.0.0.11            1                 vlan2
C  126.0.0.0/8                              1                 vlan2
S  169.255.255.1/32   126.0.0.19            1                 vlan2
S  172.16.0.0/24      126.0.0.11            1                 vlan2
S  172.17.0.0/24      126.0.0.11            1                 vlan2
C  192.168.0.0/24                           1                 vlan9
R  192.168.1.0/24     192.168.8.10          2 192.168.8.10    vlan8    03:00
Rc 192.168.4.0/24                           1                 vlan4
C  192.168.7.0/24                           1                 vlan7
Rc 192.168.8.0/24                           1                 vlan8
C  192.168.13.0/24                          1                 vlan3
S  192.168.50.50/32   126.0.0.19            1                 vlan2
S  192.168.56.0/24    192.168.4.10          1                 vlan4
R  192.168.78.0/24    192.168.8.10          2 192.168.8.10    vlan8    03:00
S  192.168.210.0/24   126.0.0.19            1                 vlan2
```

- The Rc routes are routes that are brought in by `Network` commands, i.e. the connected routes on interfaces whose addresses fall within the range covered by `Network` commands.
- The Rs route 0.0.0.0/0 is a route that is created by configuring `default-information originate` .
- The S and C routes are those brought into RIP by redistribution.
- The R routes are those learnt by RIP from RIP neighbors.

Information about the status of RIP interfaces is provided by the command:

`show ip rip interface`

```
awplus# sh ip rip int
vlan4 is up, line protocol is up
  Receive RIPv1 and RIPv2 packets
  Send RIPv2 packets only
  Split horizon: Enabled with Poisoned Reverse
  IP interface address:
   192.168.4.254/24
vlan8 is up, line protocol is up
  Receive RIPv1 and RIPv2 packets
  Send RIPv2 packets only
  Split horizon: Enabled with Poisoned Reverse
  IP interface address:
   192.168.8.254/24
```

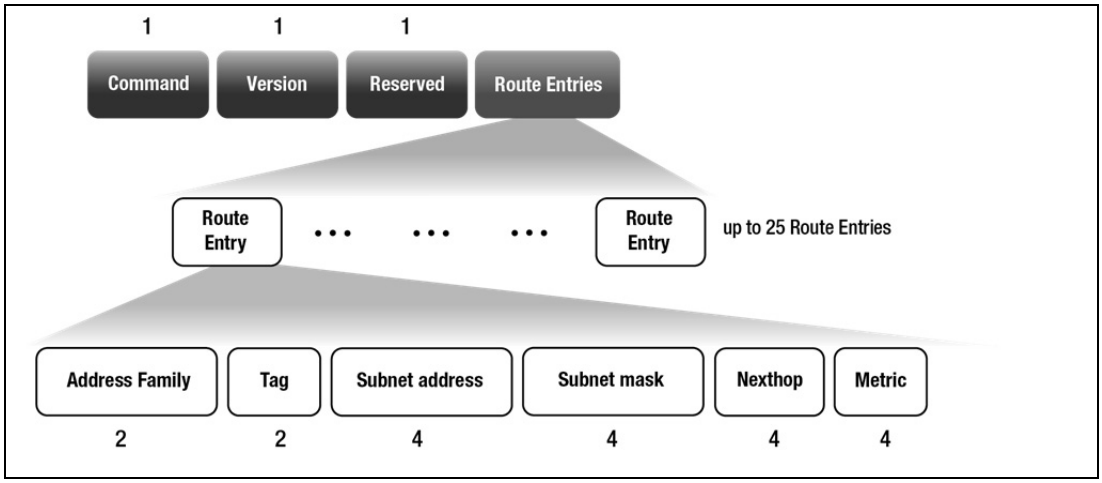
General information about the setup of RIP itself is provided by the command:

`show ip protocol`

```
awplus#sh ip protocol
Routing Protocol is "rip"
  Sending updates every 30 seconds with +/-5 seconds, next due in 18 seconds
  Timeout after 180 seconds, garbage collect after 120 seconds
  Outgoing update filter list for all interface is not set
  Incoming update filter list for all interface is not set
  Default redistribution metric is 1
  Redistributing: connected static
  Default version control: send version 2, receive version 2
    Interface        Send  Recv   Key-chain
    vlan4            2     2
    vlan8            2     2
  Routing for Networks:
    192.168.4.0/24
    192.168.8.0/24
  Routing Information Sources:
    Gateway          Distance  Last Update  Bad Packets  Bad Routes
    192.168.8.10          120  00:00:17               0           0
  Number of routes (excluding connected): 73
  Distance: (default is 120)
```

Packet Formats

There is only one packet type in RIP. It is a 4-byte header followed by up to 25 Route Table Entries (RTEs) that each contain a single route (although, the first RTE might contain authentication information instead).



RIP commands

There are only two possible commands: Request and Response.

Request

Routers send out requests when their RIP process first comes up, and they need to find out the routes that their neighbor have available to advertise. In this case, the packet contains just one Route Entry – which has address 0.0.0.0, mask 0.0.0.0 and metric 16. This Route Entry in a request packet is shorthand for "*send me all the routes that you would normally advertise to me in a periodic update*".

It is also possible to send a request that specifies a particular set of routes. The packet contains sets of Route Entries, which are the routes that the requestor wants to find out about. Typically, such a request is sent as part of a diagnostic or network monitoring process, rather than as a normal route-learning process. In response to such a request, a router creates a Response packet that contains all the same routes as were present in the request packet; the metric field in each entry is filled in with the router's metric for the route in question. If any of the routes in the request packet don't appear in the responder's route table, then the responder just puts 16 into the metric field for those routes.

Response

A response packet advertises a router's routes. Periodic updates, triggered updates, and responses to requests all have the value 2 in the command field.

Version: This will be 1 for RIPv1 packets, and 2 for RIPv2 packets.

Route tag: Information that has no significance to RIP can be carried with a route. This could enable routes that are redistributed from another protocol to carry information relevant to that protocol. For example, BGP could be advertising routes into one side of an autonomous system, and RIP carrying the routes across the autonomous system, and RIP redistributing them back to BGP on the other side of the autonomous system. In this situation, BGP could pass attributes to RIP to carry across the AS, and then hand back to BGP at the other side. The attributes would mean nothing to RIP, but would be significant to BGP.

Address Family Identifier: To keep their options open, the designers of RIP provided for the possibility of RIP carrying routes for protocols other than IPv4. So, an Address Family identifier field is provided with each Route Entry, so that the protocol to which the transported routes belong can be identified. As it has turned out, RIP has not ended up being used for non-IPv4 protocols. However, the Address Family Identifier field has been used for a couple of purposes other than identifying the protocol of the transported routes. The three possible values that can appear in this field are:

- In the case of normal route entries, the value in the field is 2, which is the value for the IPv4 Address Family Identifier (referred to as AF_INET).
- When authentication is being used, so that the first Route Entry in the Update packet is carrying Authentication information, then the Address Family Identifier value in this Route Entry is 0xffff.
- A request packet that is requesting neighbors' full updates contains just one Route Entry. The address family identifier in this one entry has the value zero.

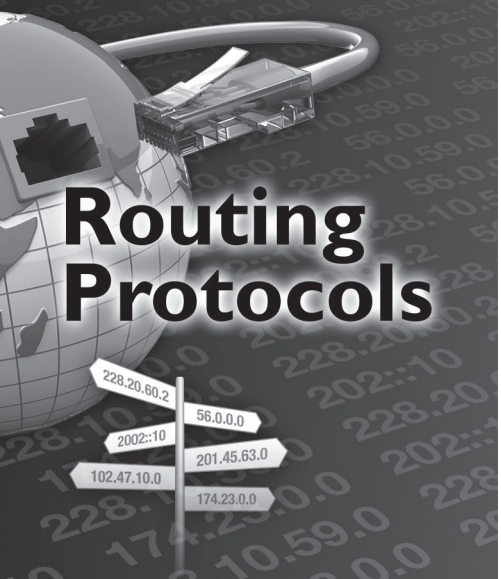
IP address: The network address of the route being advertised

Subnet Mask: The mask for the route being advertised

Next Hop: The next hop for this route, if the next hop is not the sender of the update. As explained in the section Next Hops on page 9, a next hop of 0.0.0.0 means that the sender of the update is the next hop for the route. The section also explains the circumstances in which the next hop from the router is something other than the sender of the update.

Metric: The way that routes' metrics are calculated is described in the sections Metrics on page 5 and Redistributing Routes into RIP on page 17 .

Authentication Type: Two possible values: 2=Plain Text, 3 = Keyed Message Digest.



Routing Protocols

This chapter covers the following topics:

- What is OSPF?
- Advantages and Disadvantages of OSPF
- Structure of this chapter
- Understanding the OSPF Route Calculation Process
- The Dijkstra Algorithm
- The Link State Database
- Summary so Far
- Some Basic OSPF Features
- Network Types
- Neighbor Relationships
- Authentication
- Designated Router
- Content of LSAs
- Structure of LSAs
- OSPF Network Structure
- Artifacts of Areas
- Debugging
- Neighbor Finite State Machine Debugging
- OSPF Packet Types

CHAPTER 2

Open Shortest Path First (OSPF)

Introduction

This chapter describes the widely used Open Shortest Path First (OSPF) protocol. This is not a configuration guide for OSPF, but an explanation of what OPSF does, and how it does it. The OSPF protocol is very complex, which can make it seem difficult to understand. As a result, troubleshooting issues in OSPF networks can be a very challenging task.

To help simplify things, we will start by looking at the central process of the OSPF route calculation. From there, we will see how the protocol provides all the routers in the network with the data they use for their route calculations.

As much as possible, the reasons why the protocol operates in the way it does are explained, to emphasize how the common themes of reducing protocol traffic and reducing the load on router CPUs drive the operation of the protocol.

The many illustrations provided throughout will help you understand how the OSPF elements fit together.

Overall, this chapter should give you the ability to do things like:

- Look at a packet trace of an exchange of OSPF packets, work out what information is being exchanged in those packets, the role that each packet is playing in the particular exchange, and why certain flags and status indicators in the packets have been set to the values they have.
- Look at an OSPF Link State Advertisement (LSA) database, and the IP route table on the same router, and determine whether the routes have been correctly calculated from the LSA database.
- Observe the OSPF packets a router sends, and/or the changes that it makes to its LSA database when a certain event (like an interface going down) occurs, and determine if the actions the router takes are the correct actions for the event that occurred.
- Understand the effect that given OSPF configuration changes on a router will have on the LSAs and route table on that router, and on other routers in the network.

What is OSPF?

OSPF is a link state routing protocol. It enables routers in a TCP/IP network to exchange information with each other to determine the best path for routing IP traffic to any given destination in the network.

In essence, each router informs each other router about the state of its links – i.e. the interfaces via which it connects to its local subnets. The state of a link is a description of that interface and of its relationship to its neighboring routers.

A description includes its:

- IP address
- type of network it is connected to
- bandwidth-related 'cost'
- the identities of the neighboring routers

The link states are stored in a link state database. Each router in an OSPF network has a link state database, in which it holds its own link state information, and the link state information that other routers have sent to it.

The design of the OSPF protocol takes great account of its need to support network scalability. The operation of the protocol, and a number of features within the protocol, enable an OSPF network to grow very large without the amount of protocol traffic, and the amount of OSPF-related activity that routers must perform, growing uncontrollably.

OSPF responds quickly to link failure and topology changes, this is particularly valuable in large networks where fast convergence is a priority.

Background

The OSPF routing protocol was developed for Internet Protocol (IP) networks by the Interior Gateway Protocol (IGP) working group of the Internet Engineering Task Force (IETF). The working group, formed in 1988 because in the mid-1980s it had become evident that the Routing Information Protocol (RIP) was incapable of coping with large networks.

OSPF grew out of a number of different strands of research effort. A key element was Bolt, Beranek, and Newman's (BBN) SPF algorithm developed in 1978 for the ARPANET. Dr. Radia Perlman's research on fault tolerant broadcasting of routing information, and BBN's work on area routing (1986) were also important contributions. In addition, OSPF took input from an early version of OSI's Intermediate System-to-Intermediate System (IS-IS) routing protocol.

Advantages and Disadvantages of OSPF

OSPF was developed in response to the inability of the older Routing Information Protocol (RIP) to serve large networks. RIP's inherent problem is that when a change occurs, the entire routing table information is sent to all hosts in the network. In contrast, OSPF sends only the part that is changed. Because all OSPF routers have a picture of the local network topology, changes in an OSPF network are propagated quickly, and are not subject to long hold-down timers. Additionally, as mentioned above, a prime advantage of OSPF is the fact that it elegantly scales up to large networks.

The table below outlines the advantages of OSPF:

Advantage	Description
Scalability	OSPF is specifically designed to operate well with larger networks. It does not impose a hop-count restriction and permits domain subdivision for easier management. This limits the explosion of link state updates over the whole network and aggregates routes to reduce propagation of subnet information.
Full subnetting support	OSPF can fully support subnetting, including Variable Length Subnet Masks (VLSM) and non-contiguous subnets.
Minimal bandwidth consumption	OSPF uses small **Hello** packets to verify link operation without transferring large tables. In stable networks, large updates occur only once every 30 min.
Link bandwidth awareness	OSPF takes WAN link bandwidth into account, preferring high bandwidth paths. RIP has no concept of network delays and link costs.
Tagged routes	Routes can be tagged with arbitrary values, easing interoperation with EGPs, which can tag OSPF routes with autonomous system (AS) numbers or other identifiers.

The biggest disadvantage of OSPF is its complexity: it is harder to configure and takes more management time. As a shortest path first algorithm, OSPF does require a lot of CPU power and memory.

Disadvantage	Description
Complex configuration	OSPF can be complex to configure and takes more management time. Administrators that are used to the simplicity of RIP will be challenged with the amount of new information they have to learn in order to keep up with OSPF networks.
High processor requirements	OSPF is very processor intensive.
Large memory requirements	OSPF maintains multiple copies of routing information, increasing the amount of memory needed.

The distinction between RIP and OSPF in brief

RIP is a distance vector protocol that: "tells *its neighbors about the network"*

- Sends the entire route table (which may not be completely up to date) to neighbors only
- Sends dubious information a short way

OSPF is a link state protocol that: *" tells the network about its neighbors"*

- Sends link states (completely up to date) throughout the entire network
- Sends good information a long way

Chapter Structure

The subject of OSPF is multi-faceted. Explaining OSPF is not really a linear story that has an obvious place to start, and an obvious path to follow.

Therefore, the approach followed in this chapter is to start right in the heart of the protocol. This may seem to be an odd approach to follow, and one not likely to be very helpful for the reader. Surely, it would seem more sensible to start with the simpler overview-type aspects of the protocol, and then work down into the detailed aspects. However, the fact is that once one understands the calculations that occur in the heart of the OSPF process, then the rest of the activity and structure within the protocol makes a lot more sense. Hence, we will begin by studying the Shortest Path First (SPF) calculation that OSPF uses to build up the IP route table.

From there, we will follow a somewhat undulating path – picking up some of the simpler parameters and features of the protocol, before diving into some detailed matters, then coming back up to some structural overview topics, and then into detail again.

An outline of the path we will follow is:

- Begin with an explanation of the SPF calculation.
- Take a brief look at the data elements:
    - Link State Advertisements (LSAs) - that are used to build up the shortest paths
- Introduce some basic OSPF parameters:
    - Router IDs
    - Metrics
    - Reference Bandwidth

- Understand how OSPF has different optimizations for different types of networks:
  - Broadcast
  - NBMA
  - Point-to-point
- Dive into the detail of how OSPF routers form neighbor relationships with each other:
  - Neighbor states, and state transitions
  - Timers used in OSPF
  - Authentication
- Work through the special aspects of OSPF on a broadcast network:
  - The Designated Router (DR) and Backup Designated Router(BDR)
  - DR, BDR election
  - Different multicast addresses for packets to DR and non-DR routers
- Put a real focus on LSAs:
  - Types of LSAs
  - Originating and forwarding LSAs
  - Reliable transport of LSAs
  - LSA sequence numbers
- Look at withdrawing routes from the OSPF network:
  - Aging
- Understand areas:
  - The different types of areas
  - Which LSAs are passed between areas
- Learn about the artifacts of areas:
  - LSA forwarding rules
  - Virtual links
  - Split horizon for inter-area routing
  - Inter-area and intra-area route differences

- Look at route management:
  - Summarization
  - Filtering
  - Redistribution from other protocols
  - Passive interfaces
- Find out what OSPF on-demand is.
- Finally, learn about Graceful restart.

The subject of OSPF **areas** is deliberately left to a point towards the end of the chapter. This is because areas are a structure that is used to reduce the processing and data-traffic overhead that builds up as an OSPF network grows. Areas are not fundamental to the operation of the protocol – a single-area OSPF network works just fine. Areas would not be necessary if routers had unlimited memory and processing power, and Network Administrators had unlimited capacity for working through very large Link State Advertisements (LSA) databases when tracking down network issues.

However, areas are woven into almost every aspect of OSPF operation – even the formation of neighbor relationships is dependent on the neighboring routers' interfaces being in the same area. Any discussion of LSAs quickly comes to area-related matters, like Area Summary LSAs, and the rules for flooding LSAs (i.e. which LSAs may or may not cross area boundaries).

Leaving the discussion of areas towards the end of the chapter does mean that the concept of areas is mentioned quite frequently before the concept is considered in detail. This means that at certain points you will need to just take it as read that areas exist, and trust they will eventually be explained in more detail.

Understanding the OSPF Route Calculation Process

As with any routing protocol, the purpose of OSPF is to enable the routers in a network to inform each other of the best direction towards any given IP address in that network. The process that OSFP uses to achieve this is conceptually quite simple, but complex to describe.

Before we embark on the task of getting to grips with all that detailed mechanics, and mapping out how it all fits together, let us first understand the concept that lies at the heart of it all.

Turning a network into a tree

The key insight that OSPF uses is the fact that every router can treat the network as a tree, with the router itself at the root of the tree. With a tree, calculating the best path to any destination is quite simple - find the path through the branches that takes you to that destination.

To create the tree, treat the network as a set of **subnets**, **links**, and **routers**:

- Links can only exist between pairs of routers or between a router and a subnet.
- There are no direct links between one subnet and another subnet.
- Links between **routers** are real, e.g. a PPP link between pairs of routers. The links between routers and subnets are imaginary, e.g. the 'link' from a router to an Ethernet network that is directly connected to it.

An algorithm called the *Dijkstra Algorithm* converts the interconnected set of routers, subnets, and links into a tree. The links become the branches in the tree, and the routers and subnets become nodes from which sets of branches originate.

Let us now consider the Dijkstra Algorithm in some detail.

The Dijkstra Algorithm

The purpose of the Dijkstra Algorithm is not just to convert the network into a tree, but to convert it into a tree that contains the 'shortest' paths to each node. In this algorithm, the definition of shortest is not simply 'the least number of branches from the root'. It associates a length value with each branch, and aims to find the path from the root, to any given node, that involves the least aggregate branch lengths.

At this point, you might be thinking "*Dijkstra Algorithm, tree diagrams – that's all internal computational detail. Spare me that internal stuff, and get on to the things that a user configures, like Hello timers, area border summarization, not-so-stubby-areas, etc.*" However, the fact is that the Dijkstra Algorithm is not terribly complicated, and if you get a feel for how it works, then a lot of the activity in OSPF, like the exchanging of LSAs and synchronizing of databases, makes much more sense. So, let us proceed on to look at how this algorithm works. The best way to get a clear picture of how this works is to consider an example.

Example of the Dijkstra Algorithm at work

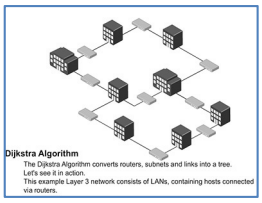
For the sake of simplicity and familiarity, in our first example we will consider a network made up of a set of Ethernet subnets, with routers routing between those subnets. Later on, as we look at how OSPF needs to be generalized to apply to a variety of network types, we will consider networks involving WAN links.

The aim of the algorithm is to find the shortest path to each node.

The process that occurs in the Dijkstra Algorithm is that a tree is built up, a node at a time, so the tree starts with just the root router, then branches are attached to that root, leading to its directly connected subnets. Then branches are connected to those subnets, leading to their directly connected routers, and so on, until the full tree is built up.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/DIOzxS42VsU



The Link State Database

The explanation above describes the central act that is at the heart of OSPF, i.e. the SPF calculation. All the other machinery of OSPF is targeted towards the goal of providing each router with the set of data from which it can perform its SPF calculation.

The description of the SPF calculation is all about building up a tree of links to all the routers and subnets in the network. This all sounds fine until you stop and think "*but, how does a router know all about all the other routers in the network, and what subnets they are connected to?*".

By itself, a router can know just its own configuration, which will tell it which subnets it is directly connected to and very little else. For a router to find out the identities of all the other routers in the network, and the sets of connected subnets, there needs to be a process whereby the routers in the network can tell each other what they know about their own immediate environment.

If everybody tells everybody else about their own immediate environment, then all the routers can build up their own complete map of the network. Once every router knows the identity of every other router and subnet in the network, and which is connected to which, then every router has all the information it needs to carry out the Dijkstra Algorithm process described above.

A key activity within the OSPF protocol is the exchange of these immediate environment descriptions. In the language of OSPF, these are called Link State Advertisements or **LSAs** for short. In essence, every router sends out an LSA that describes its own immediate environment. Routers that receive LSAs from other routers store these LSAs in their own piece of memory, called their LSA database. They then pass the LSAs on to all their neighbors. In this way, everybody's LSA is passed to everybody else, and everybody builds up an LSA database that contains all the information required to create their own shortest path tree for the whole network.

A sacred, unbreakable rule of OSPF is:

The LSA databases of all the routers in a given area must have identical contents.

For a definition of the term **area**, see the section: OSPF Network Structure on page 82.

This means that all routers in a given piece of the network must have exactly the same view of the topology of that network. They may have different rules about what information they import into their IP route tables based on the SFP calculation they perform on their LSA database, but they must all be calculating from the same LSA database. Hence, it is not possible to filter the LSAs transmitted or received by an OSPF router.

End users may on occasion think they need to filter certain LSAs out of LSA exchanges. They are wrong – what they need to do is filter out which routes they import into their IP route tables, or which interfaces they include into the OSPF network.

It cannot be emphasized enough just how sacred this rule is. In RFC 2328, there are statements like:

- "Two routers belonging to the same area have, for that area, identical area link-state databases."
- "In a link-state routing algorithm, it is very important for all routers' link-state databases to stay synchronized."
- "It is very important that the router link-state databases remain synchronized"
- "All routers belonging to the same area have identical link state databases for the area."

This is not just a statement made once in passing, as a mild recommendation. It is a central principle repeated multiple times, in different contexts.

This is because successful performance of the Dijkstra Algorithm calculations requires that a router have the **full** set of information from which to build its network tree. If any parts are missing, then the resulting tree will possibly be wrong.

Router and network LSAs

The description of the Dijkstra Algorithm involved two types of nodes: routers and subnets. To build up the shortest-path tree, a router's LSA database must contain an entry for:

1. Every router in the network, listing all the subnets it is connected to (and the cost associated with each of those connections).
2. Every subnet in the network, listing each of the routers it is connected to.

Once it has that set of information, it can go right ahead and complete the SPF calculation, and then know the best path to every subnet in the network.

The Router LSA contains a router's descriptions of links to connected networks.

The rule is that every router in an OSPF network must create its own Router LSA and send that LSA to all its neighbors, who will then pass it on to their neighbors, and so on, as described above.

We discuss the content of a Router LSA below; in the section Details of the different types of LSA on page 60. For now, we just need to know that the Router LSA contains a list of the networks that the Router connects to.

The creation of Router LSAs, then, is quite straightforward, but what about Network LSAs? How does a subnet create an LSA and send it out? A subnet is not a physical entity like a router is.

Well, as far as Network LSAs are concerned, the rule is that all the routers connected to a given subnet hold an election among themselves.

They elect a router to be the originator of the Network LSA for the subnet in question. This router, naturally enough, is referred to as the Designated Router (DR) for that subnet. It creates the necessary Network LSA, and sends it to all its neighbors. We discuss the content of a Network LSA in detail below in the section Details of the different types of LSA on page 60. For now, we just need to know that the Network LSA contains a list of the OSPF routers that connect to the subnet.

Details of the mechanics of designated router election are discussed in the section Designated Router election on page 53.

LSA flooding

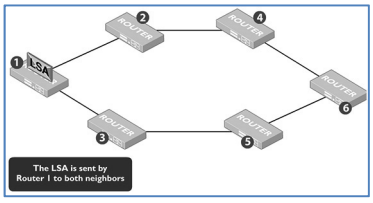
As mentioned above, the whole point of creating LSAs is to share them around. The job is complete when a given LSA has been transmitted to every router in its area. **Flooding** is the process of all the routers exchanging all these LSAs.

However, if this proceeded in an uncontrolled manner, LSAs would continue to be forwarded around in loops forever. Therefore, a simple rule is imposed. If a router has received a given LSA from one or more particular neighbors, then if it later receives that same LSA from another neighbor, it will not forward that LSA on to any of the neighbors from which it had already previously received it.

This rule is enough to prevent the endless circulation of LSAs.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/1pi5lQByXjc



Note - the term Neighbor has a specific meaning in OSPF. Not all neighbors are equal - some are more equal than others. This is discussed in the section Neighbor Relationships on page 41.

Strictly speaking, in the process of LSA flooding, a router forwards LSAs only to its 'full' neighbors (the definition of a "full" neighbor is described in the section Neighbor Relationships on page 41).

The process of LSA exchange, and how routers ensure that their neighbors have actually received the LSAs that they have sent to them, is discussed in more detail in the section: Passing around LSAs on page 68.

Also, as we will see later, the concept of splitting the OSPF network into Areas does put a limit on just how far the flooding of an LSA propagates, and adds some extra LSA types to the mix.

Summary so Far

Let us just pause for a moment here to review what we have looked at so far. In fact, the discussion up to here has effectively covered the whole of the OSPF core activity. The quite simple processes at the heart of it all are:

- All the routers in the network send out a simple description of which subnets they are connected to - this is called a Router LSA.
- Certain designated routers in the network send out a description of which routers are connected to a given subnet - this is called a Network LSA.
- Every router makes sure that all of these LSAs get passed to all the other routers in the network.
- The routers store the LSAs in an LSA database.
- By applying the Dijkstra algorithm to the contents of its LSA database, each router creates a tree representation of the network, from which it calculates the least-cost route to every subnet in the network.

And that really is it. All the other aspects of OSPF are there in order to enable the above activities to proceed in a reliable, efficient way, that can scale up to very large networks without overpowering all the routers' memory and CPU capacity.

Now let's move on to looking at all that 'other stuff'.

Some Basic OSPF Features

The discussion of OSPF throughout this chapter will make frequent reference to some basic OSPF concepts. Let's get these concepts defined here, so they make sense when referred to later.

Router ID

Every router in an OSPF network needs a unique identifier. This is referred to as the Router ID. The Router ID takes the same form as an IP address (four numbers, each between 0 and 255, divided by dots). The Router ID does not HAVE to be one of the router's actual IP addresses, but it is very, very common practice to use one of the router's IP addresses as its Router ID.

By default, the router uses its highest configured IP address as its OSPF Router ID. The Router ID can be set to a different value using the commands:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# router-id x.x.x.x
```

Metrics

The cost (also called metric) of an interface in OSPF is an indication of the overhead required to send packets across that interface. The cost of an interface is inversely proportional to the bandwidth of that interface. A higher bandwidth indicates a lower cost. There is more overhead (higher cost) and time delays involved in crossing a 56k serial line than crossing a 10M Ethernet link.

OSPF default interface cost

Every OSPF interface is assigned a cost in the range 1-65535. It represents the cost for traffic to EXIT the router via that interface. The default value of the cost is calculated from the following formula:

OSPF Cost = 10^8/<Interface Speed in bps>

Based on that formula, the following OSPF costs will be assigned to these interfaces, once OSPF is enabled on them.

Interface Type (speed)	Default OSPF Cost
Gigabit Ethernet	1
100 Mbps Ethernet	1
T3	2
T1	69

When an IP interface is assigned to a VLAN that includes physical ports with different speeds, the fastest port is considered when calculating the OSPF cost for that interface. Similarly, when OSPF is enabled on an IP interface running over a Link Aggregation, the fastest link of that Aggregation is taken into account. For ports that may have variable speed (e.g.10/100 Ethernet ports), the cost of the port is calculated from the speed of the port at the time of its addition to OSPF.

Reference bandwidth

By default, the OSPF interface cost is calculated as 10^8 /speed (in bits per second [bps]). The cost is then rounded up to the nearest whole integer. This will result in an OSPF cost of 1 for all interfaces faster than 100Mbps. In some cases that might be undesirable. The reference bandwidth feature allows flexibility in the OSPF cost assignments, without forcing you to manually assign OSPF costs to every interface. The reference bandwidth replaces the 10^8 component in the above formula. The reference bandwidth is specified in Mbps, i.e. the commands that follow specify the reference bandwidth to be 10Gbps, resulting in the calculation of OSPF interface costs:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# auto-cost reference-bandwidth 10000
```

Having set this value for the Reference Bandwidth, the OSPF costs for various speeds of interface become:

Interface Type (speed)	Default OSPF Cost
10 Gigabit Ethernet	1
1 Gigabit Ethernet	10
100 Mbps	100

The reference bandwidth is a box-wide setting, and affects the default OSPF cost calculation for all interfaces of the router.

Network Types

One of the guiding principles in the design of the OSPF protocol is to minimize the amount of data that needs to be exchanged between routers – i.e. to keep the chatter to a minimum. To that end, OSPF operates subtly differently on various types of networks, so that it can achieve an optimum level of chatter on each network type.

Some of the discussion below refers to the process of forming neighbor relationships, which is described in more detail in the section Neighbor Relationships on page 41. For now, the concept to keep in mind is that OSPF routers have neighbor relationships with each other, and the relationships can be FULL or just TWO-WAY. Routers that are FULL neighbors exchange a lot more chatter than routers that are just TWO-WAY neighbors.

Broadcast multi-access

OSPF utilizes the fact that networks like Ethernet support multicast. On such networks, the OSPF routers communicate to each other by multicast. For example, a router can send an LSA to all its neighbors by sending out a single multicast packet that all the neighbors will pick up. The fact that multicast is used on a broadcast network also facilitates automatic neighbor discovery. When a router is attached to a broadcast network segment, it sends out multicast packets announcing its presence (called, appropriately enough, *Hello* packets), secure in the knowledge that any potential neighbors out there will receive these packets and respond, and start negotiating a neighbor relationship, No pre-configuration of a router's neighbors is required, it just finds them automatically. This process is discussed in more detail in the section Neighbor Relationships on page 41.

But, because multiple routers can access the medium, there is the potential for a scaling problem as the number of routers connected to the same segment increases. If every router needs to ensure that it fully synchronizes its database with every other router on an Ethernet segment, then the amount of database synchronizing going on when N routers are attached goes up proportionally to N^2.

In accordance with the principle of chatter minimization, the OSPF protocol specifies that on a multi-access network segment there are only **two** routers that explicitly synchronize databases with all the other routers. These two routers are called the Designated Router (DR) and Backup Designated Router (BDR) for the network segment. If all the other routers synchronize databases with the DR and BDR, and the DR and BDR synchronize databases with each other, then all routers will end up with identical database contents.

In the language of neighbor relationships, the relationship that each router has with the DR and BDR is referred to as a FULL neighbor relationship. The relationship between any two routers that are not DR or BDR is referred to as a TWO-WAY relationship (denoting that they have checked that they have two-way data communication between each other).

The DR on a multi-access network is also responsible for originating the Network LSA for the network segment. The concept of the designated router is discussed in more detail below in the section Designated Router on page 52.

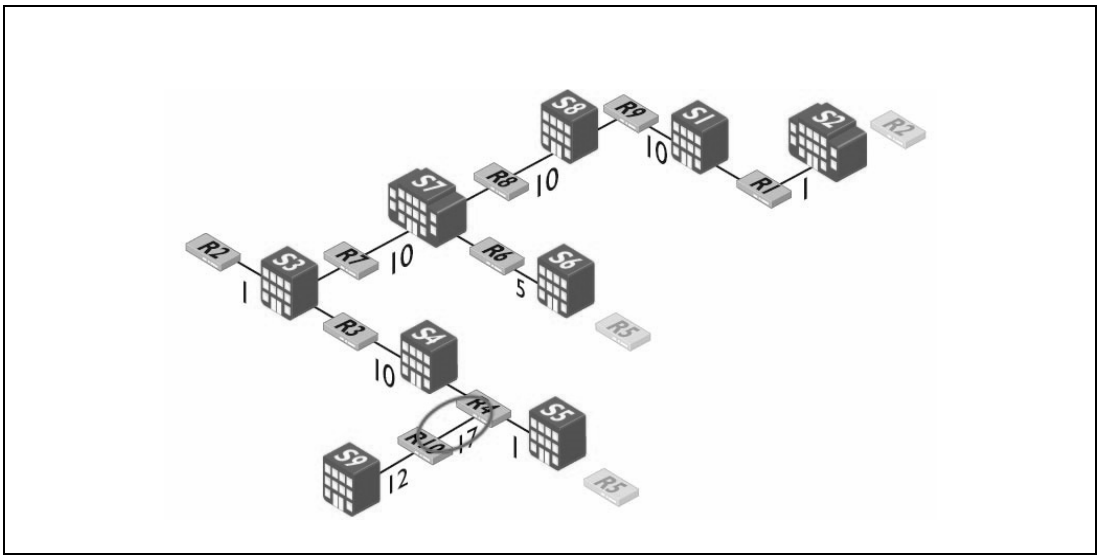
Point-to-point links

In an OSPF point-to-point network, where a direct Layer 3 connection exists between a single pair of OSPF routers, there is no need for Designated and Backup Designated Routers, as is the case in OSPF multi-access networks. Without the need for Designated and Backup Designated Routers, a point-to-point network establishes adjacency and converges faster. The neighboring routers become fully adjacent whenever they can communicate directly. In contrast, in broadcast and Non-Broadcast Multi-Access (NBMA) networks, there are rules about which routers become FULL neighbors with each other.

By convention, the routers on a point-to-point link send OSPF packets to a multicast address (rather than to each other's IP address) as much as possible. Again, this facilitates automatic neighbor discovery, as the routers can establish communication with each other without needing to know each other's IP addresses in advance. Because a point-to-point network does not really *multicast* packets, as there are not multiple recipient hosts on the network that will receive the multicast packets, the use of multicast IP on a point-to-point network is just a convenient way for the routers to be able to say "*this packet is an OSPF packet, as it is destined to the multicast address that has been reserved for use by OSPF*".

Also, in a point-to-point network, there is no Network LSA created for the network. Simply, the routers at each end of the link create Router LSAs and include connections to the subnet in use on the point-to-point link. In fact, a point-to-point link can be unnumbered, in which case, the Router LSAs make no mention of any subnet on the point-to-point link.

The fact that there is no Network LSA generated for the point-to-point link means that when the router performs the Dijkstra algorithm on a network that contains point-to-point links, it connects the routers at each end of the link directly to each other, there is no subnet between them.

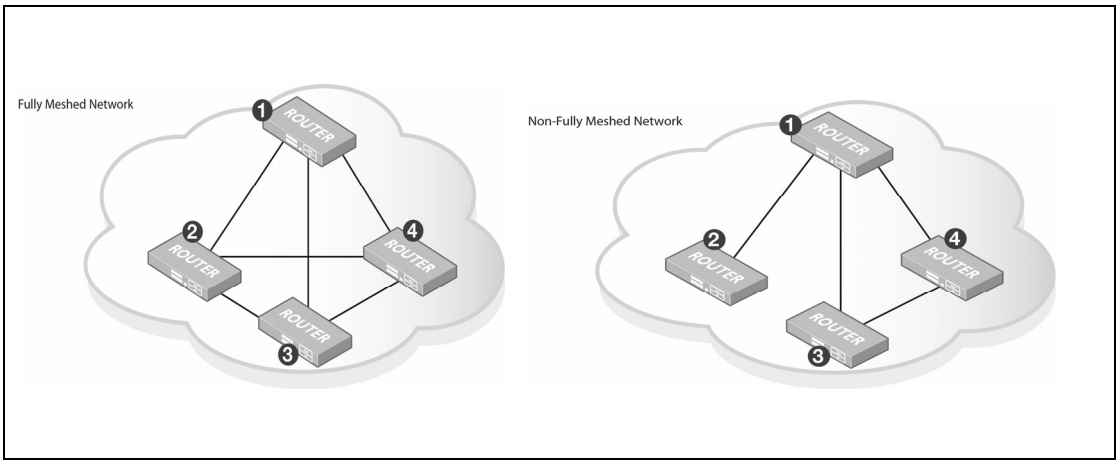


Non Broadcast Multi-Access and point-to-multipoint networks

Network providers often provide a WAN network as a 'cloud' of switching infrastructure, to which multiple sites are connected by a single physical link from each site into the cloud. Popular protocols for this type of service are Frame Relay, ATM, and X.25.

Each of these protocols supports the concept of 'logical channels' via which each router communicates to another router. From a router's point-of-view, the single physical link into the provider's cloud appears as a bundle of logical channels, with a single router at the other end of each channel.

Such a network can be fully meshed or non-fully meshed. In a fully meshed network, every router has a logical channel to every other router. In a non-fully meshed network some routers do not have logical channels to each other, and must communicate via other router(s).



OSPF can operate across these "*WAN cloud*" networks in one of two modes:

1. Non-Broadcast Multi-Access (NBMA) mode.
2. Point-to-multipoint mode.

NBMA mode is the more efficient (i.e. involves the least chatter), but can only be used on a fully meshed network. Point-to-multipoint mode can be used on a fully meshed or non-fully-meshed network.

In practice, the most common choices are to use:

- NBMA mode on fully meshed networks.
- Point-to-multipoint mode on non-fully-meshed networks.

NBMA mode

The name Non-Broadcast Multi-Access network refers to the fact that multiple routers have access to each other via the cloud, but there is no concept of a broadcast packet that will be forwarded to all the connected routers. Communication between routers must be performed in a unicast manner, via each separate logical channel.

From a neighbor-relationship point of view, OSPF treats a non-broadcast, multi-access network much like it treats a broadcast network. Since there may be many routers attached to the network, a DR and BDR are elected for the network. This Designated Router then originates a network LSA, which lists all routers attached to the non-broadcast network.

However, due to the lack of broadcast capabilities, automatic neighbor discovery is not possible. The routers must unicast packets to each other's' IP addresses. Hence, if a router is to form neighbor relationships, it must be statically configured with the IP addresses of the other routers that it needs to contact and form neighbor relationships with. Therefore, any routers that you want to be able to become DR or BDR on the network need to be configured with the IP addresses of all the other routers on the network.

NBMA mode also requires all non-DR routers to establish two-way relationships with each other. Hence this mode will only operate on a fully meshed network.

As with a broadcast network, the DR on an NBMA network:

- Originates a Network LSA for the NBMA network.
- Is responsible for ensuring that all other routers in the NBMA network receive all the LSAs they need to receive.

Point-to-multipoint mode

For a router attached to a WAN cloud network, the network appears as a bundle of point-to-point links to other routers. It is it possible for OSPF to be set up to form point-to-point links between each of the pairs that share a logical channel.

In this mode, each router has one or more point-to-point OSPF neighbor relationships established via the OSPF interface through which it connects to the 'WAN cloud'. So, it has multiple point-to-points per OSPF interface. Point-to-multipoint is a succinct way of referring to this setup.

As with normal point-to-point links:

- Multicast addresses are used, so automatic neighbor discovery is possible.
- No Network LSA is generated.
- There are no DR or BDR in the network.

This mode, requiring full neighbor relationships between every pair of routers that share a logical channel connection, generates more traffic than NBMA mode. But, it is the only reliable way to operate OSPF on a non-fully meshed network.

Note – if some routers connected to a WAN cloud are configured to operate OSPF in NBMA mode and others connected to the same WAN cloud are configured to operate OSPF in point-to-multipoint mode, then failure will ensue. The routers operating in different modes will not even be able to establish neighbor relationships with each other, as the NBMA routers will be using unicast, and the point-to-multipoint ones will be using multicast addresses.

Neighbor Relationships

When an OSPF process starts up, pretty much the first thing it needs to do is to find out if it is within reach of any other routers that are configured with OSPF. If it does find any such routers, it will form neighbor relationships with them. Neighbor relationships are required within OSPF because they enable a router to know:

- who they need to forward LSAs to
- which routers to list in a Network LSA (if it happens to be a Designated Router)

The next section describes how neighbor relationships are formed, what rules the relationships follow, and the timers that govern the maintenance of the relationships.

Neighbors

Routers that share a common segment become neighbors on that segment. Neighbors are discovered via the Hello protocol. Hello packets are sent periodically out of each OSPF interface of a router. On point-to-point and broadcast networks, these Hello packets are sent to a multicast IP address. The address is 224.0.0.5, conventionally referred to as the ALLSPFROUTERS address. On NBMA networks, Hello packets are sent to the unicast addresses of the potential neighbors. When routers receive Hello packets from other routers, they examine the contents of the Hello packets to evaluate whether they can proceed to form a neighbor relationship with these routers.

Two routers will not become neighbors unless they agree on the following:

Item	Description
Area ID	If two routers have a common segment, their interfaces on that segment must belong to the same area. (The concept of OSPF areas will be explained below in the section Areas on page 83).
Subnet and mask	The IP addresses on the interfaces by which the neighbors connect to each other must both be in the same subnet, and have the same subnet mask.
Authentication	OSPF allows for the configuration of a password for a specific area. Routers that want to become neighbors have to exchange the same password on a particular segment.

Item	Description
Hello and Dead Intervals	These intervals refer to the frequency of Hello packets and the amount of time which passes without Hello packet reception before an interface is assumed down.
Stub area flag	Two routers have to also agree on the stub area flag in the Hello packets in order to become neighbors. Stub areas are discussed in a later section. Keep in mind for now that defining stub areas will affect the neighbor election process.

A router can determine all these properties of another router by examining the Hello packets it is receiving from that router. If it decides that the other router does have matching values for all these parameters, then it will proceed to form a neighbor relationship with that other router.

Formation of adjacencies

There are several steps that a pair of routers go through in order to achieve adjacency. During the adjacency building process the interface will make a progression of state changes:

State	Description
Down	The initial state. No Hello packets have been received from the neighbor recently or at all.
Attempt	This state only applies to non-broadcast multi-access networks. The router is making a determined attempt to contact a statically configured neighbor. Hello packets are sent every HelloInterval.
Init	A Hello packet has been seen from the neighbor; however the Hello packet does not list the router as known.
Two-Way	This state is entered when the communication between to neighbors is bidirectional (i.e. the Hello packet from the neighbor lists this router as a neighbor).
Ex-Start	This is the first step at creating an adjacency between two routers. The two routers decide which is going to control the exchange between them.
Exchange	In this state, the neighbors exchange database description packets. These packets contain summaries the entries in the routers' LSA databases.
Loading	After all the database description information has been exchanged, the routers exchange any link state advertisements required to update or complete each routers' topological database and hence synchronize the two router's databases.
Full	This is the final state and the adjacency is complete. Reaching this state in itself may cause new instances of some link states.

Let us now consider some of these states in more detail.

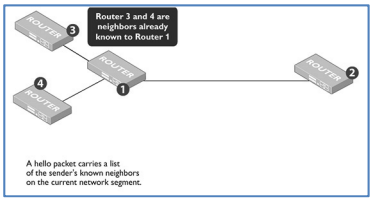
Two-Way state

One of the fields in a Hello packet is a list of the neighbor routers that the sender has discovered on the network segment into which it is sending the Hellos.

When a router sees that it is receiving Hello packets from another router on the network, and that other router satisfies the criteria to become a neighbor (the criteria listed above) then the current router will put that other router's IP address into this neighbor list in subsequent Hellos that it sends.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/K_eqXJHKvBk



Once a router sees its own Router ID in the Hello packets arriving from another router, it knows that it has established Two-Way communication with that router. It is seeing the other router's Hello packets, and the other router is seeing its Hello packets (as evidenced by its own Router ID appearing in the other router's list of discovered neighbors).

At this point, the neighbor relationship has reached the Two-Way state. In this state, the neighbors are said to be *merely adjacent*. In a multi-access network, if neither of the routers is a DR or BDR, then they take their relationship no further than this.

The neighbor relationship with a DR or BDR will not stop at this stage, but will attempt to progress on through all the other states, to achieve full adjacency. Similarly, all point-to-point and point-to-multipoint neighbors will attempt to progress to full adjacency.

Also, the sequence number in this packet is equal to the sequence number that the Master router sent in its original packet.

Ex-Start state

To achieve full adjacency, the routers must synchronize their LSA databases. This is achieved by the process of database exchange.

Before the exchange can begin, the two routers must agree on some aspects of how they will go about doing the exchange. The routers move their neighbor relationship to the Ex-Start state, and embark on this process of agreeing on how they will go about the database exchange.

The key thing they need to agree on is who will be Master of the database exchange and who will be Slave. The database exchange process uses sequence numbers to keep track of progress. The Master is the router who decides what the sequence numbers will be.

In the Ex-Start state, the routers start sending each other a new type of OSPF packet – called a Database Description (DD) packet. Initially, they each send a DD packet that has a header and no body. In the header of the DD packet there are three flags – the Initialize (I), More (M) and Master/Slave (MS) flags. The Initialize flag is set in the initial packets, to indicate that this is the start of the process, and the election of Master and Slave is not complete. The More flag indicates that there are still more DD packets to come, and the MS flag indicates whether the router considers itself to be the Master or the Slave in the Exchange process.

In the first DD packet that each router sends, they set all three flags – so they are saying:

- The Master/Slave negotiation is not complete (indicated by the I flag).
- I am Master (indicated by the MS flag).
- There will still be more DD packets to come (indicated by the M flag).

They also each generate a sequence number, and put that into the sequence number field of the DD packet header. There is no rule about what value should be used for the opening sequence number – the routers are free to choose whatever value they like.

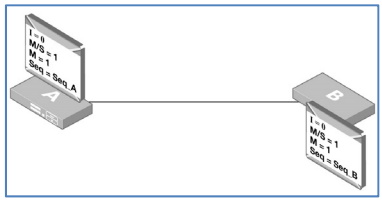
The routers receive each other's DD packets and examine them. The rule is that the router with the larger Router ID should become the Master. Therefore, the router with the lower Router ID will see a higher Router ID in its neighbor's packet, and concede that the other router should be Master.

It announces this concession by sending back a DD packet that has the:

- MS flag unset – indicating that it recognizes it is the Slave.
- I flag unset – indicating that it believes the Master/Slave negotiation is complete.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/pRJRENfxXIY



At this point, the negotiation is complete, and the routers progress from the Ex-Start state to the Exchange state.

Exchange state

In the exchange state, the two routers show each other the contents of their LSA databases. They do this by sending the header of every LSA in their database. They do not send the whole of each LSA – just enough information to uniquely establish the identity of each LSA in their databases.

The DD packets that they send each other each contain several LSA headers.

A simple sequence-number scheme is used to enable each router to make sure that it has received all the DD packets that the other router has sent.

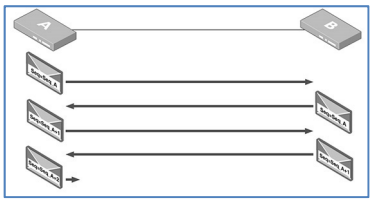
The Master increments the sequence number in each new DD packet that it sends. The next DD packet that the Slave sends must contain the sequence number of the previous DD packet it received from the Master.

The Master expects that the sequence number in each DD packet it receives from the Slave will be equal to the last sequence number it sent to the Slave.

The Slave expects that the sequence number in each DD packet it receives from the Master will be one higher than that in the previous DD packet that it received from the Master.

To watch this video, browse to the link or scan the QR code with your smart phone

http://youtu.be/pGwmzFx7cMI



If the Master does not receive the Slave's packet that acknowledges a given sequence number, then the Master will retransmit the DD packet with that sequence number.

It continues to retransmit this DD packet until it receives a DD packet from the Slave containing this sequence number. Only the Master may perform retransmissions. If the Slave notices it has missed a packet from the Master – i.e. the sequence number in a packet it has just received is more than one higher than the previous sequence number it received – then it simply does not respond to this packet. This will trigger the Master into retransmission.

When a router realizes that it is sending the packet that contains the headers of the last set of LSAs in its database, then it does not set the More flag in this packet. However, if the other router's LSA database is much larger then it may have several more DD packets to send. If so, the router that has completed sending all its LSA headers may need to continue sending further (empty) DD packets to acknowledge the DD packets it is receiving from its neighbor. All these further DD packets that are being sent for acknowledge purposes will also have the M flag unset.

When both routers have finally sent each other all their LSA headers, the Exchange phase is complete. This occurs when both routers have sent and received at least one DD packet with the M flag unset.

Loading state

The purpose of the Exchange process is to enable the routers to know what LSAs are in each other's databases. As mentioned above, in the exchange process, the routers send only the headers of their LSAs, not the full content of the LSAs. However, just the headers are enough for a router to work out whether the other router possesses:

- LSAs that it does not possess

  and/or

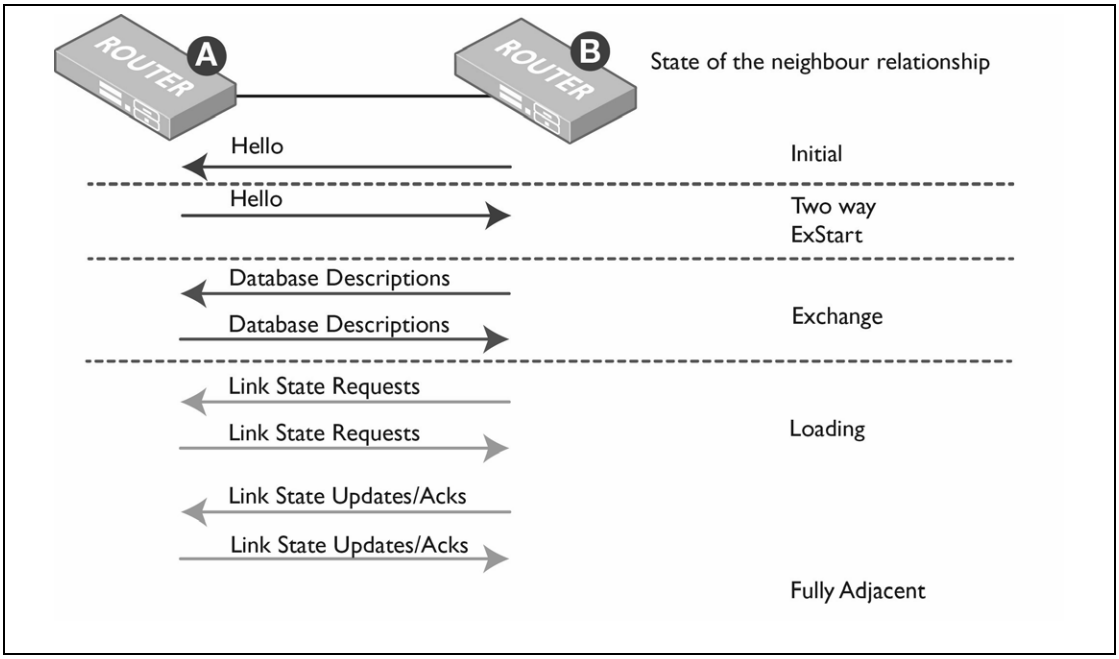
- More recent copies of LSAs that it does possess (we will discuss how it works this out in the section LSA Sequence Numbers on page 74).

If a router does determine that the other router does have any such LSAs, then it needs to request the full contents of those LSAs, so that it can get its LSA database in sync with the other router.

Note – this process works both ways – very often both routers will find they have to request LSAs from each other.

The requesting and receiving of these LSAs occurs in the Loading state.

When both routers have received all the LSAs that they need from the other, the neighbor states progress to FULL. At this point, the routers are Fully Adjacent.



Maintaining the neighbor relationship

Once a router has formed relationships with its neighbors, it needs to keep in touch with them, to check that they are all still there.

This is achieved by continuing to send Hello packets on a regular basis – one packet every Hello interval. If a router has not received any Hellos from a given neighbor for a period called the Dead interval (typically this is 4 times the Hello Interval, but you can configure it to whatever value you like) then it will assume that neighbor is dead.

When a router decides that a neighbor is dead, then:

- If the router is the DR on the segment that contained the newly-dead neighbor, then it will re-originate its Network LSA for that segment. This new Network LSA will no longer contain the newly dead router in the list of routers attached to this segment.
- If the router has a point-to-point connection to the newly dead router, then the router will re-originate its Router LSA, and the newly dead router will not be listed as a neighbor on the point-to-point link.

The originating of a new LSA will cause other routers to perform a new SPF calculation. The newly dead router will not appear in any LSAs, so the SPF calculations will result in OSPF route tables that do not include routes via that newly-dead router.

OSPF timers

Several factors are considered by OSPF enabled routers when it comes to forming adjacencies, including Hello and Dead timer settings.

Hello timer

The Hello timer is the number of seconds between a router's Hello packets.

OSPF's primary means of verifying continuing operation of the network is via its Hello Protocol. Every OSPF speaker sends small Hello packets out each of its interfaces every Hello Interval.

Hello timers are configured on a per-interface basis, using the commands:

```
awplus# configure terminal
awplus(config)# interface vlan2
awplus(config-if)# ip ospf hello-interval <1-65535>
```

By default, the Hello interval is 10 seconds.

A router can have different values of the Hello Timer on different interfaces. For routers to be neighbors, they must have the same value of the Hello Interval on the interfaces via which they communicate to each other.

Dead timer

The Dead Interval is the number of seconds before a router is declared to be down and unavailable. If the period of time that has elapsed since a router received Hello packets from a given neighbor exceeds the Dead Interval, then that neighbor is declared down.

The Dead Interval is set on a per-interface basis by the commands:

```
awplus# configure terminal
awplus(config)# interface vlan1
awplus(config-if)# ip ospf dead-interval <1-65545>
```

By default, the Dead Interval is four times the Hello Interval. As with the Hello Interval, the Dead Interval can be set to different values on different interfaces of the same router. And, also as with the Hello Interval, neighbors must agree on the value of the Dead Interval on the interfaces via which they communicate.

Poll Interval

The Poll Interval is specific to NBMA networks. In NBMA networks, the process of routers discovering their neighbors is not automatic, but is achieved by routers being pre-configured with the unicast IP addresses of the neighbors they must form relationships with. Often, this configuration is not symmetrical. A router that it has been decided will be able to become DR or BDR will be configured with a list of the IP addresses of the other routers on the subnet, so that it can neighbor up with them; but those neighbors that are not eligible to become DR or BDR will often not be configured with a list of IP addresses to neighbor up with.

As a result, any router that has been configured with a list of neighbors is obliged to make contact with those neighbors, as those other routers are very likely not to have any idea of who they are supposed to be neighboring up to. Of course, this could simply be achieved by the neighbor-list-configured routers constantly sending Hello packets to potential neighbors at the rate of one Hello packet per Hello interval. However, OSPF is not a protocol to chew up bandwidth if it can possibly avoid it. Hence, for this situation where a router on an NBMA network is trying to make contact with another router, the protocol provides the option of sending the Hello packets at a slower rate. There is a separate timer, called the Poll Interval, which defines the rate at which the Hello packets are sent in this situation.

A router, attached to an NBMA network, that has been configured with a list of IP addresses of potential neighbors, will send Hello packets to those IP addresses at the rate defined by the Poll Interval, rather than the rate defined by the Hello Interval when:

- The relationship with a neighbor is still in the Attempt state – i.e. when Hello packets have been sent to the neighbor's IP address, but no response from that neighbor has yet been received.
- The relationship with the neighbor has gone dead – i.e. there had been an active relationship with the neighbor, but the time since a Hello was last received from the neighbor has now exceeded the Dead Interval.

Typically, the Poll Interval is significantly larger than the Hello Interval – a typical value is 2 minutes. The Poll Interval is set on a per-neighbor basis by the command:

```
awplus(config-router)# neighbor <ip-address> <poll-interval>
```

Authentication

The possibility exists for a LAN-based attack using OSPF. Rouge hosts on a LAN could disrupt routing within a network by forming neighbor relationships with OSPF routers, and sending in LSAs containing deliberately incorrect information. Additionally, it is possible for OSPF to be mistakenly configured on an interface, maybe when a router is redeployed to a new location.

Hence, OSPF provides the ability to authenticate the packets that are being received.

There are two methods of authentication supported by OSPF – Plain text authentication and MD5 authentication.

Plain text authentication

This authentication takes the simple form of a password embedded in the packets. It is not very secure, as an attacker could simply sniff the OSPF packets on the network to find the password, and then start putting that password into their rouge packets.

But, plain text authentication is useful as a simple way to enable you to reconfigure OSPF on a multi-access network in a controlled fashion. If you are reconfiguring OSPF on a set of routers,

connected to a multi-access network, one by one, then there is potential for problems when routers with the new configuration are communicating with routers with the old configuration.

If you have all the routers using a plain text password, then as you change routers over to the new configuration, you change their password. Then, only routers with the new configuration can form neighbor relationships with other routers with the new configuration. Similarly, only routers with the old configuration can form neighbor relationships with other routers with the old configuration. This avoids LSAs from the old configuration getting into the databases of routers that have the new configuration, and taking a long time to age out of the network.

In AlliedWare Plus, OSPF authentication is configured on a per-interface basis.

The commands to configure plain text authentication are:

```
awplus# configure terminal
awplus(config)# interface vlan1
awplus(config-if)# ip ospf authentication
awplus(config-if)# ip ospf authentication-key <password>
```

Note - no `authentication` method is specified in the **ip ospf authentication** command. The absence of an authentication method on that command means plain text authentication.

Plain text authentication passwords do not have to be the same throughout an area, but they must be the same between neighbors.

Note, in addition to configuring authentication on an interface, it is also necessary to tell OSPF to use authentication on the area that the interface connects to:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# area 0 authentication
```

MD5 authentication

MD5 authentication is altogether more sophisticated than plan text authentication. With MD5 authentication, OSPF does the following:

- Puts a packet sequence number into each packet. The sequence number is incremented each time a packet is transmitted on a given interface.
- Calculates the MD5 hash of the packet contents (including the sequence number), and a shared key.
- Puts this hash value into the packet.
- Puts an ID number for the Key into the packet.

By doing all this, OSPF achieves:

Authentication—the receiving router performs the same hash calculation, using its copy of the shared key. If the result that it gets from the hash calculation matches that which is in the packet, then that verifies that the sender is using the correct key.

Tamper-proofing—if the contents of the packet were altered along the way, then the result of the hash calculation performed by the recipient (using the tampered packet contents) will not match the hash calculated by the sender, even if they are both using the same key.

A guard against replay attacks—the monotonically incrementing sequence number means that an attacker cannot just store previously transmitted packets and replay them, as they would have an old sequence number. Also, because the sequence number is part of the packet contents used to calculate the hash, the attacker cannot just replace the sequence numbers in its stored packets by some higher sequence numbers, because the hash value would then fail to match the packet contents.

The value of including a key ID is that it enables neighbors to change shared keys, and choose to use different keys from the set at different times. This makes the cracking of the keys that much harder.

The commands to configure MD5 authentication

```
awplus# configure terminal
awplus(config)# interface vlan1
awplus(config-if)# ip ospf authentication message-digest
awplus(config-if)# ip ospf message-digest-key <key-id> md5 <password>
```

The key-id value is the key ID that is transmitted in the OSPF packets. All neighbors that are to successfully communicate with this router must use the same key, and give it the same key ID.

Note, in addition to configuring authentication on an interface, it is also necessary to tell OSPF to use authentication on the area that the interface connects to:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# area 0 authentication
```

AlliedWare Plus assists the user in the process of changing to using a new authentication key. If you configure another MD5 key on an interface, so that there are two keys configured on the interface, then the switch will send two copies of each packet from that interface. The two copies of the packets will have different hash calculations – the hash included in one packet will be calculated using the first key, and the hash included in the other packet will be calculated using the second key. While the keys are being updated on all the routers in a subnet, there will be a period where some know the new key, but others have not been configured with it yet.

This duplicating of the packets ensures they can continue to communicate with each other during this transition period.

Once AlliedWare Plus detects that all neighbors on the subnet are using the new key, it will stop sending any packets that use the old key. At that point, you can come along and delete the old key.

Designated Router

As has been mentioned a few times already, on multi-access networks (networks supporting more than two routers); there are a couple of special routers – the Designated Router (DR) and Backup Designated Router (BDR)—which are elected and required to perform special duties.

As with a lot of aspects of OSPF, the business of having a DR and BDR adds efficiency at the cost of complexity. The complexity cost involved in having a DR and BDR is not immense – it does mean an extra parameter in OSPF configuration, some extra processing in neighbor discovery process, and some extra state (i.e. the state of each router's 'DR-ness') for the routers (and network administrators) to be aware of.

The primary efficiency gain is a reduction in the amount of LSA exchange that needs to go on in a multi-access network. Without the DR/BDR concept, every router on the subnet would have to have to explicitly synchronize its LSA database with every other router on the subnet – an N^2 amount of database synchronizing. With the DR/BDR concept, every non-DR router has to synchronize its database with just the DR and BDR – a $2N$ amount of database synchronizing. As N (the number of routers in the subnet) increases, the reduction in traffic due to just doing $2N$ worth of DB synchronizing rather than N^2 worth gets more and more significant. This is what network designers like to refer to as *scalability*. As the scale (size**)** of the network increases, the amount of traffic generated by the routing protocol does not become unmanageable**.**

The DR performs two main functions to reduce protocol traffic:

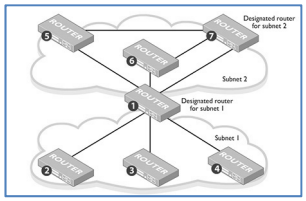
1. The DR originates the Network LSA on behalf of the subnet. There is no need for each of the routers to separately generate a Network LSA, and have multiple differently-originated copies of the Network LSA being passed around.

2. The DR becomes adjacent to all other routers in the subnet, and saves all the other routers from having to become adjacent with each other. Actually, the BDR does also form adjacencies with all the other routers in the subnet. This adds resilience (by removing the single point of failure) at the expense of some more protocol traffic.

Note that a router which is the DR for one subnet that it is connected to may not be the DR for other subnets that it is connected to**.**

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/eOmV_mnxNH0



The DR on subnet 1 is just a non-DR router on subnet 2. It forwards to the DR of subnet 2 the LSAs that it has learnt from subnet 1. The DR of subnet 2 now distributes those LSAs to the other routers in subnet 2.

Backup Designated Router

The Backup Designated Router (BDR) does not generate a Network LSA for the subnet. (If it did, the transition to a new DR would be even faster. However, this is a tradeoff between database size and speed of convergence when the DR disappears.) In some steps of the flooding procedure, the BDR plays a passive role, letting the DR do more of the work. This cuts down on the amount of local routing traffic.

Election of DR and BDR

In the discussion above, it was stated that a DR and BDR are elected on a multi-access network. Let's now take a look at how this election works.

Designated Router election

When the routers start up, and exchange Hello packets, they look at each other's Hello packets, and work out who is *king* of the network. The specific parameter in the Hello packets that governs the election is the priority value. The router with the highest priority value becomes the DR. If several routers all have the same top priority, then the router with the highest Router ID wins.

Backup Designated Router election

When an election is held, the router with the highest priority becomes the Designated Router. The router that places second becomes the BDR. The BDR is the router with the next highest priority or the next-highest Router ID (if multiple routers have the same top priority).

Priority

The default value for priority is 1. If a router has priority 0 on a particular interface, it can never become DR or BDR on the network that it connects to via that interface.

Note - a given router can have different priorities on different interfaces.

The commands to set the priority on an interface are:

```
awplus# configure terminal
awplus(config)# interface vlan2
awplus(config-if)# ip ospf priority[0..255]
```

If a router has multiple interfaces, it is recommended to configure the priorities on those interfaces in such a way that it will not be likely to become DR or BDR on several networks, as that would put significant strain on the router.

DR is not pre-empted

Once a router is elected DR, it stays DR for as long as it remains active, and connected to the network. Even if a router with better credentials (higher priority, or same priority and higher Router ID than the current DR) gets connected to the subnet, it does not knock the current DR off its perch. Whilst this means that the identity of the DR is a bit less predictable than it could be, it does mean that the OSPF network is more stable. If there was a DR re-election every time a more suitable candidate came along, that could be quite disruptive. All the tearing down of adjacencies, forming new adjacencies, originating the Network LSA from a new router, withdrawing the previous network LSA, etc. would generate busy-work in the OSPF network for no particular benefit.

If a router comes up on a network segment and there are no other routers there already, it will become a DR.

If there is an existing DR (or BDR) on a network, and a new router comes up, the new router will **not** attempt to preempt them, even if it has a higher priority than them. If the DR fails then the BDR will take over the role of DR, and a new election will be held for BDR.

Content of LSAs

LSAs are, of course, just data structures. The data structure has a header, and contents. The header is the same for each type of LSA, but the structure of the LSA content differs from one LSA type to the next.

Let us look at the various types of LSA. We will look at the LSAs as data structures, but will also consider, graphically, the physical significance of the data content.

Types of LSA

LSAs essentially fall into three categories: Topological elements, Individual routes, and Other.

Topological elements

The Router LSA and the Network LSA are the little *jigsaw pieces* that are clipped together in the **Dijkstra** calculation. These are truly the elements of a topological database, as they describe the connections between routers, interfaces and subnets.

These LSAs are only flooded within a single area, they never cross an area boundary. A router has full topological maps of only the areas that it is connected to. For other areas, it has more basic routing information.

Individual routes

The basic routing information that a router receives from beyond its own area is carried in the form of Area Summary LSAs and AS External LSAs. These LSAs are effectively just individual IP routes. So, in fact, the passing around of inter-area or external route information in OSPF is not too different to the passing around of route information in RIP. An Area Border Router sends into one area a list of the routes that it has learnt from the other area(s) that it is connected to. These are sent as Area Summary LSAs.

Note the concept of an Area Border Router – i.e. a router that has interfaces in two or more areas, is discussed in detail below in the section Router roles on page 90.

Similarly, an autonomous system Border router passes into OSPF a list of the external routes that it has learnt via non-OSPF routing. These are sent as External LSAs. Again, the concept of an Autonomous System Border Router—a router that has some interfaces that are connected to non-OSPF neighbors, and is bringing non-OSPF routes into OSPF—is discussed in detail below in the section Router roles on page 90.

Of course, OSPF does provide good facilities for summarizing routes at area borders and autonomous system borders. A border router is not obliged to separately send every individual route that is has learnt; it can be configured to summarize a set of routes in a single broader-masked route. This is a feature that can have a strong influence on reducing overhead in an OSPF network – you are well advised to make use of it. Route summarization is discussed further below in the section Route summarization on page 97.

There are two other types of LSA that fall into the Individual Route category.

- The first is the **Autonomous System Border Router** (ASBR) Summary LSA. This is a route to an Autonomous System Border Router. They are used by routers in different areas to work out how to get to the ASBR, and therefore what the next hop is on the path to the external routes learnt from the ASBR.
- The second is the special type of External LSA that is generated by AS Border Routers that reside in an NSSA area. These are referred to as **Type-7 LSAs**, and are the same as External LSAs in almost all respects, except for the fact that they are confined to NSSA areas.

Other LSAs

The types of LSAs described above are by far the most commonly used LSAs. But some other LSAs have been defined over the years. Some have effectively fallen into disuse. Some are still current, but not commonly used, and some are coming into their own.

The LSAs in this category are:

- Group Membership LSAs: These were used for Multicast OSPF (MSOPF), which is now a little-used protocol; PIM has pushed MOSPF into obscurity.
- External Attribute LSAs: These carry BGP attribute information across an OSPF network. Rarely used.
- Opaque LSAs: These are defined as a means of transporting information other than routing information within an OSPF network. The RFC that originally defined the Opaque LSAs did not define what they would be used for, it just defined the concept of three types of Opaque LSA – those that would just transfer information between one pair of neighbor routers (link-local); those that would transfer information within a single area (area-local) and those that would transfer information throughout the autonomous system. The idea was to put in place a vehicle that could be used in the future for adding features to OSPF.

  A prominent use of Opaque LSAs is in Graceful Restart (described in detail below in the section Graceful Restart on page 107), where they are used by a router to inform its neighbors that it is going into or out of the Graceful Restart state.

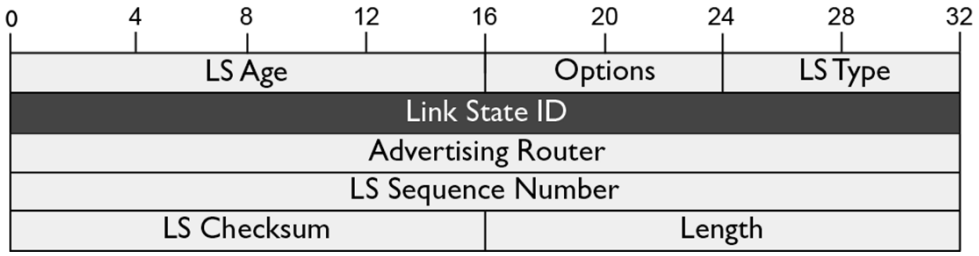
Structure of LSAs

Let's now look at the general structure of an LSA, and then we will go on to consider each of the six main types of LSA in more detail.

An LSA consists of a header and a body. The format of the header is as below.

LSA header

Let's briefly discuss each of the fields in the header:



LS Age

Normally, this is the number of seconds since this LSA was first originated. The exception to this normality is when the Age is deliberately set to MaxAge (3600) seconds, to indicate that recipients should remove this LSA from their databases.

All routers need to regularly work through their LSA databases, updating the ages of the LSAs. When an LSA's age hits MaxAge, the LSA needs to be removed. When a self-originated LSA's age hits 1800 (30 minutes), it is time to re-originate the LSA.

Options

The Options field that appears in the LSA is a universal Options field that is used in multiple different contexts within the OSPF protocol. This same Options field appears in Hello Packets and Database Description packets. The various bits in the Options field are used for different purposes. Most are used to indicate the capabilities of an originating router, and are used in neighbor relationship negotiation, so the neighbors can discover each other's capabilities and decide if they are compatible. The other bits are used to explain the origin or the intended treatment of an LSA.

The Options field has evolved quite a bit over the years. Originally, only two bits in the field were defined – the E bit and the T bit (see below for details). But the T bit has since been deprecated when it became evident that TOS-based routing was an idea that was never going to take off. The other bits in the field have been added over the years, as new features have been added to OSPF.

The full list of the bits in the Options field is:

Bit 1: DN

This bit falls into the category of defining the treatment that an LSA should receive. It was defined specifically to deal with a problem that can occur in networks where OSPF is carrying routing information between routers that both connect to the same BGP/MPLS backbone. Potentially, one router can redistribute a route from BGP into OSPF, then the other router can redistribute that same route from OSPF back into BGP, and the route can get carried around in circles. To prevent this, the router that redistributes the route from BGP into OSPF will set this DN bit; and any other router that receives this LSA will know not to redistribute the corresponding route back into BGP.

Bit 2: O

This is a router capability bit. It indicates that the originating router supports Opaque LSAs.

Bit 3: DC

This is a router capability bit. It indicates that the originating router supports OSPF over Demand Circuits. (Please see the section OSPF on Demand on page 105).

Bit 4: L

This bit is only set in Hello and DTD packets. It indicates that the packet contains a set of extra fields that carry link-local data. Note, at one time, Bit 4 of the options field was defined as the EA (Extended Attributes) bit. This was a router capability bit, which indicated that the originating router could accept Type 8 External Attribute LSAs.

Bit 5: N/P

This bit is used for two separate purposes.

- In Hello packets, it is the N bit. This is a router capability bit, indicating that originating router believes that the interface that it sent the packet out through is attached to an NSSA area. (See the section Not so stubby areas (NSSAs) on page 87).
- In Type-7 LSAs, it is the P bit. P stands for Propagate. It is used to indicate to an Area Border Router that it may translate this Type-7 LSA into a Type-5 LSA and propagate it out to other areas.

Bit 6: MC

This is a router capability bit. It indicates that the originating router supports MOSPF. It is little used, and ripe for being redefined for some other purpose.

Bit 7: E

This is a router capability bit. It indicates that the originating router is able to accept Type-5 External LSAs. It is set in all Hello packets transmitted into non-stub areas and all LSAs originated into non-stub areas. (Stub and non-stub areas are defined in the section Area types on page 85).

Bit 8: MT

This is a router capability bit. It indicates that the originating router supports a feature called Multitopology OSPF.

This bit started life as the T bit, which indicated that the originating router supported ToS-based routing. But, given that almost no-one ever implemented ToS-based routing, the bit has been redeployed for this new idea of Multitopology OSPF.

LS Type

The LSA types are:

Type	Description	Type	Description
1	Router LSA	7	NSSA External LSA
2	Network LSA	8	External Attribute LSA
3	Area Summary LSA	9	Link-local Opaque LSA
4	ASBR Summary LSA	10	Area-local Opaque LSA
5	External LSA	11	Global Opaque LSA (can be transmitted through the whole autonomous system
6	Group Membership LSA		

Link State ID

The Link State ID is the fundamental unique identifier of an LSA. This is effectively the LSA's *name*.

The Link State ID can be more than just an arbitrary name; in some LSAs it also identifies the piece of network information that the LSA is carrying. For example, for a Router LSA, the Link State ID identifies which router the LSA pertains to; for a Network LSA, the Link State ID identifies the subnet that the LSA pertains to, etc.

Link State ID is determined differently for different types of LSA:

- Router LSA – The Router ID of the originating router.
- Network LSA – The IP address on the interface that connects the DR to the Network LSA's subnet.
- Area Summary LSA – The network IP address of the route being advertised.
- ASBR summary LSA – The Router ID of the ASBR who is the subject of the LSA.
- External LSA – The network IP address of the route being advertised.

Actually, for the case of Area Summary LSAs and External LSAs, the Link State ID does not have to be the network IP address of the route being advertised; it can be any IP address within the range of IP addresses covered by that route. This is necessary in order to deal with the situation where two or more routes with the same network IP address, but different masks, are being advertised.

For example, if the following external routes are all being advertised, then there needs to be a way to give different Link State IDs to the LSAs that advertise these routes:

192.238.1.0/24, 192.238.1.0/25, 192.238.1.0/26

The preferred method is to set the Link State ID on the LSAs that advertise the longer-mask routes to be the broadcast address for those routes, rather than the network address. So, that would mean that:

Network being advertised	Link state ID
192.238.1.0/24	192.238.1.0
192.238.1.0/25	192.238.1.127
192.238.1.0/26	192.238.1.63

Advertising Router

Simply the Router ID of the router that originated this LSA.

LS Sequence Number

The sequence number indicates which *revision* of the LSA this is, so that routers can determine the latest revision when they find they have two copies of the same LSA. The sequence number is discussed in more detail in the section LSA Sequence Numbers on page 74.

LS Checksum

The checksum is calculated over the whole LSA *except* the age. The age is excluded from the checksum calculation so that the checksum does not have to be recalculated every time the age is updated.

Length

Length refers to the whole length of the LSA, including the 20 bytes of header.

Details of the different types of LSA

Router LSA

The Router LSA contains a list of the router's links that are within the area that the Router LSA is being advertised into.

So, if the router is a border router between areas A and B, then it will generate a Router LSA into area A that contains those of its links that are within area A, and a different Router LSA into area B, that contains those of its links that are within area B.

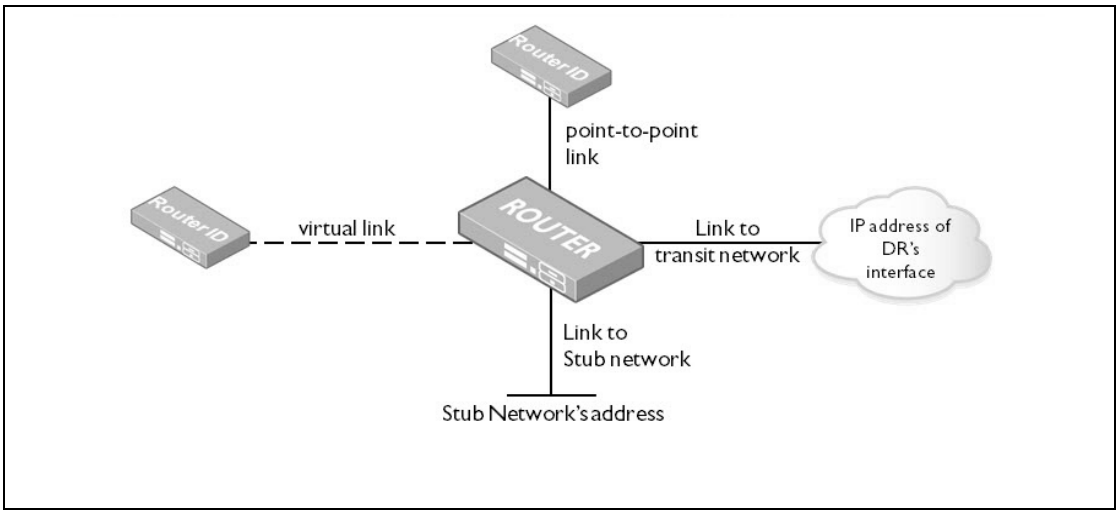
There are four types of link that a router can have:

Type Number	Description	Identifier for the link	Other data for link
1	Point-to-point (connection to another router on a point-to-point link)	Router ID of router at other end of link	The IP address of the interface at the current router's end of the link. (If the interface is unnumbered then the ifIndex of the interface)
2	Link to a transit network (a network that has other OSPF routers attached to it)	IP address of Interface by which DR connects to this network.	The IP address of the interface at the current router's end of the link.
3	Link to a Stub Network (a network that has no other OSPF routers attached to it)	The network IP address for the stub network. (Or the address of the host if this is a link to just a host)	The subnet mask on the interface at the current router's end of the link. (Or 255.255.255.255 if this is a link to just a host)
4	Virtual Link	Router ID of router at other end of the virtual link	The IP address of the interface at the current router's end of the link.

Notes

- When a router is in the process of bringing up a connection to a transit network, the link to the transit network is initially advertised as a Type-3 link. Once the router is fully adjacent to another router in the transit area, the link becomes a Type-2 link.
- When the router has a point-to-point link to another router, this link actually generates two link entries in the Router LSA – a Type-1 link, and a Type-3 link. The extra Type-3 link effectively represents the small (2-member) subnet that exists on the point-to-point link.

Importantly, each link entry in the Router LSA also has a cost associated with it. This is the value that is used in the SFP calculation to work out the least-cost route to a given destination.



One other field in the Router LSA that should be mentioned is the **rtype** field. This is a one-byte field in the LSA's header, and appears immediately after the generic header.

There are five bits in this field that have been defined:

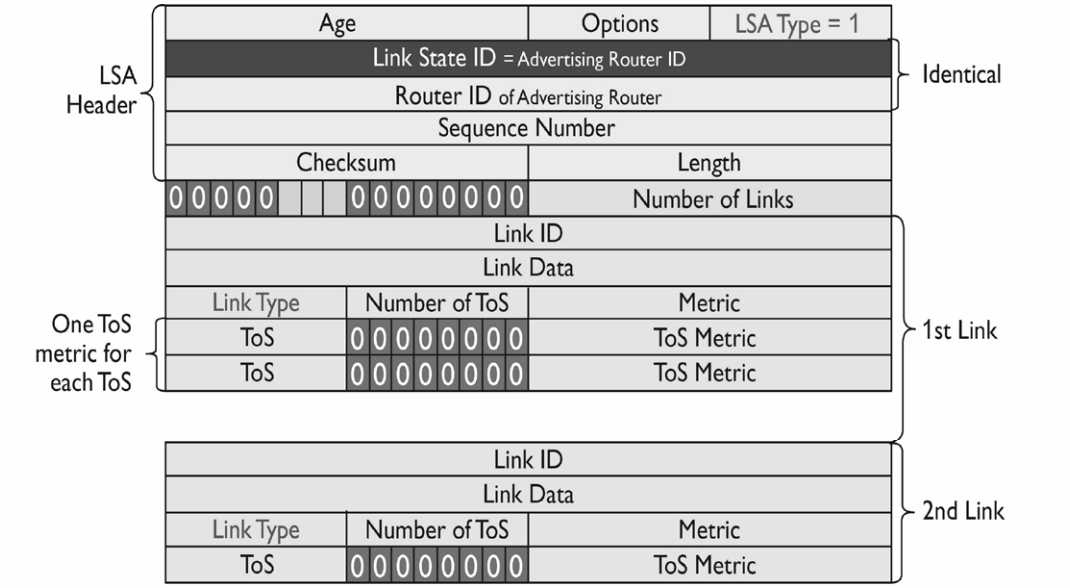
V – Indicates that the router is an endpoint of one or more virtual links that pass through the area that this LSA is being transmitted into, (see the section Virtual Links on page 94).

E – Indicates that the router is an AS Border router, or is the border router between a non-stub network and an NSSA, (see the section Types of non-stub area on page 85).

B – Indicates that the router is an Area Border router.

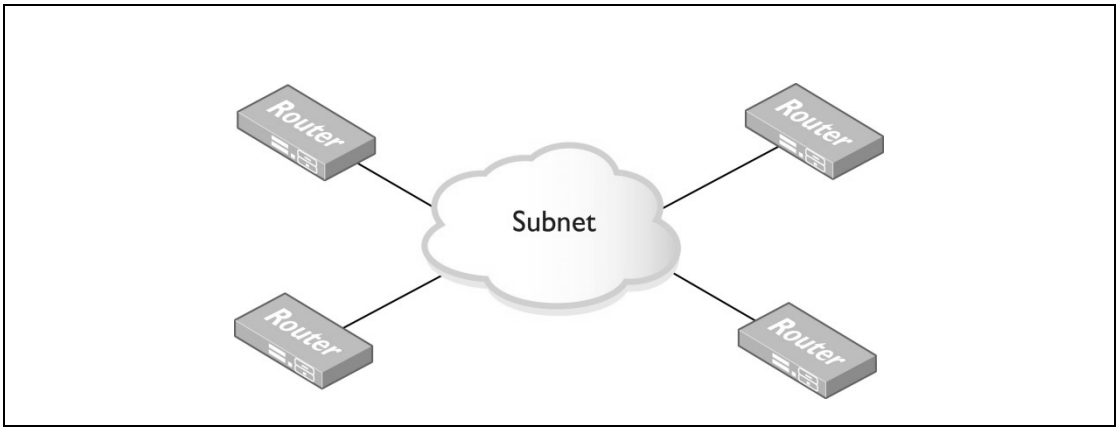
W – Used for the purposes of the now-defunct M-OSPF.

Nt – Indicates the router is an NSSA border router, translating all Type-7 LSAs to Type-5.



Network LSA

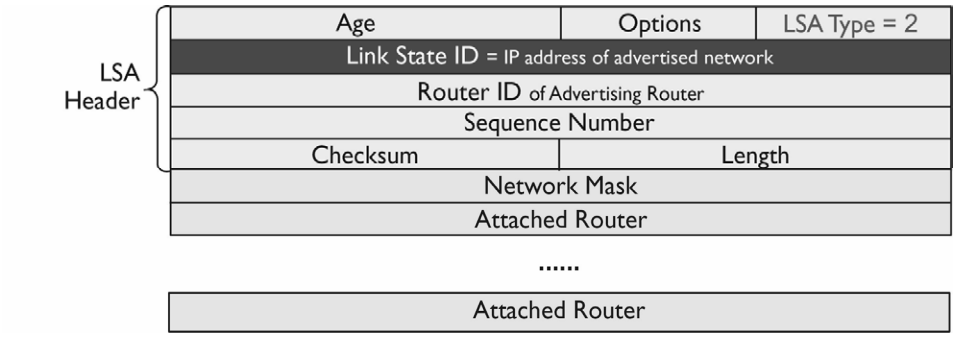
A Network LSA is a list of the routers attached to a subnet.



As is explained in the section Designated Router on page 52, a Network LSA is originated by the Designated Router in Multi-access networks.

The LinkState ID of a Network LSA is the IP address if the interface via which the DR connects to the subnet in question.

The routers listed in the LSA are represented by their Router IDs.

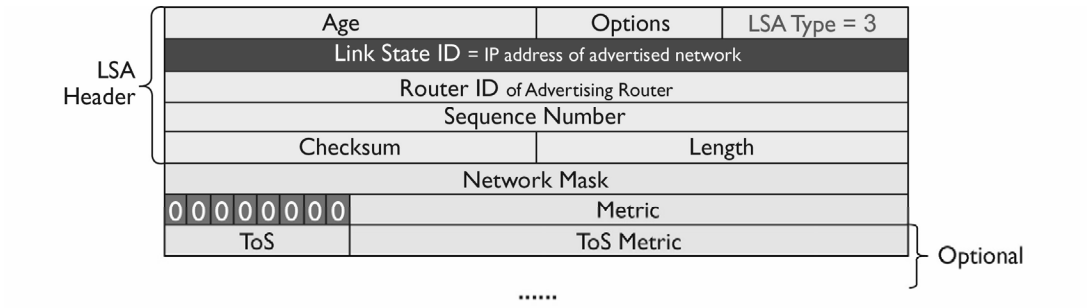


Area Summary LSA

The Summary LSA is simply the way that an Area Border router advertises the routes from one of its attached areas to the other areas that it is attached to.

The content of this type of LSA is very simple – the operative pieces of information are just the network address, mask and metric of the route being advertised.

The LSID is the network address of the route. The network mask and metric are contained in the data portion of the LSA.



ASBR Summary LSAs

To understand why such an LSA needs to exist, consider a router receiving an External LSA, and then trying work out how to actually send packets to the subnet that is advertised in that External LSA. The receiving router knows that the originator of the External LSA is a gateway to the advertised subnet, so it knows that if it is to send packets to that subnet, then it just needs to get the packets delivered to the ASBR, who will then know how to forward them to the destination.

But, how does the receiving router know how to get the packets to the ASBR. The only identifier of the originator in the External LSA is its Router ID. There is no necessary connection between a router's Router ID and its IP address(es).

If the recipient of the External LSA is in the same area as the originator of the External LSA, there is no problem – the originator's Router LSA is in the recipient's LSA database, and one of the end results of the SPF calculation is knowledge of the actual route the owner of each Router ID in the local area.

But when the recipient is in a different area from the originator, all the recipient sees is a Router ID that it has no way of finding a route to. Hence, Area Border routers need to create LSAs that inform routers how to reach ASBRs that are in other areas.

This is what happens

All Area Border routers keep a list of routes that they have learnt to ASBRs. For the ASBRs that are in an area directly connected to the ABR, the routes are generated when the ABR has to look up the route to the originator of External LSAs that it has received. For the ASBRs that are in other areas, the routes are generated from ASBR Summary LSAs.

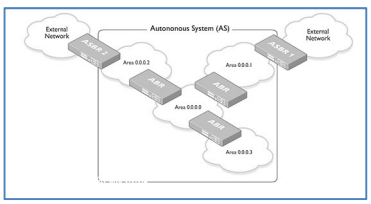
For any given area that the Area Border Router is connected to, it originates ASBR Summary LSAs for all the ASBRs that it knows of, but that routers in this other area would not yet know of.

A few points to be clear on are:

- ASBR Summary LSAs are not originated by the ASBRs themselves; they are originated by ABRs.
- ASBR Summary LSAs are not forwarded from area to area. Each Area Border router looks at the list of ASBRs it has learnt, and decides which ASBR Summary LSAs it needs to originate, to send into the areas it is connected to. So, the ASBR Summary LSAs are re-originated at each area border, they are not simply passed through the area border.
- ASBR Summary LSAs are really just a special case of Area Summary LSAs, and are essentially generated in the same way. As with Area Summary LSAs, an Area Border Router asks itself "*what routes to ASBRs do I know of, from any source of information, and which of my connected areas do I need to pass on this ASBR route information to*?"

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/dpICQU0KHx8

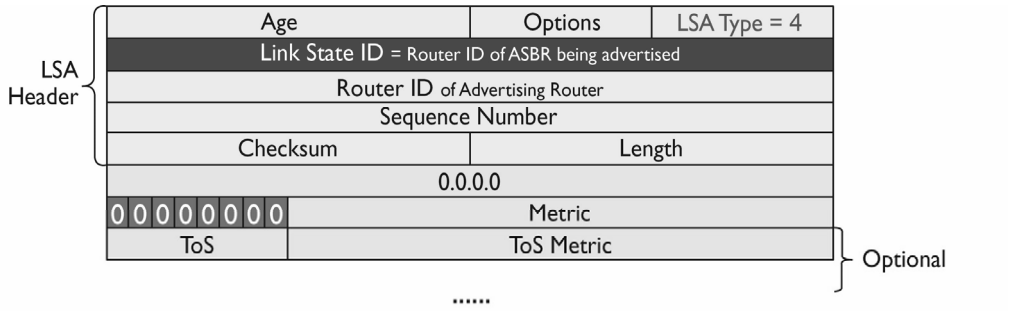


The content of the ASBR Summary LSA is effectively the same as that of an Area Summary LSA. As this LSA is just advertising a single route to a destination, it just needs to list the destination, its mask and the metric of the route.

The 'destination' advertised by the LSA is not actually an IP address; rather it is the Router ID of the ASBR that the LSA pertains to. This Router ID might be an IP address on the ASBR, but it does not have to be.

The LSID is the Router ID of the ASBR.

The mask (always 0.0.0.0, as it is not actually relevant) and the metric are contained in the data portion of the LSA.



External LSAs

An External LSA represents a route that is being redistributed into OSPF from an external source – either another routing protocol, or just a connected or static route attached to a non-OSPF interface of one of the OSPF routers in the network.

While this type of LSA is, again, just a route, it contains a bit more information than is present in the Type-3 and Type-4 LSAs.

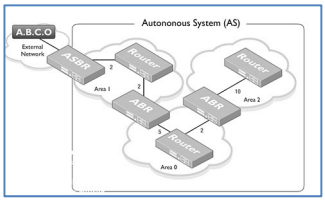
As with the Type-3 and Type-4 LSAs, it does contain the network address of the destination subnet (carried as the Link State ID), the destination subnet's mask and a metric. However, it also carries a metric Type indicator, a forwarding address, and an external route tag.
Let's understand each of these items.

The metric Type indicator determines how the route's metric is calculated. There are two choices: Type-1 and Type-2.

If the metric Type is set to Type-1, then the metric of the route calculated from the LSA is calculated by adding the metric in the LSA to the metric of the path from the current router to the ASBR that originated the LSA.

To watch this video, browse to the link or scan the QR code with your smart phone:

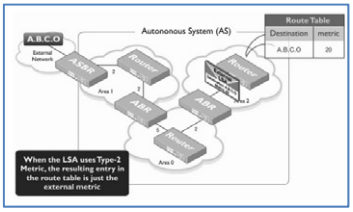
http://youtu.be/vj4_XG86PNY



If the metric Type is set to Type-2, then the metric of the route calculated from the LSA is just the metric in the LSA

To watch this video, browse to the link or scan the QR code with your smart phone:

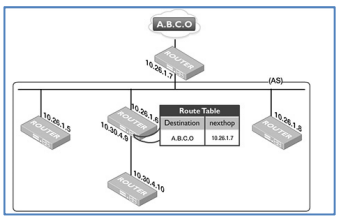
http://youtu.be/cjkSxNaxJy4



The forwarding address deals with the case where the next hop address on an external route is actually an address within a transit subnet. In this case, the other routers attached to this subnet should not use the originating router as the next hop in the routes they calculate from the LSA, but can route directly to the external next hop.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/76OMCdjk_K8

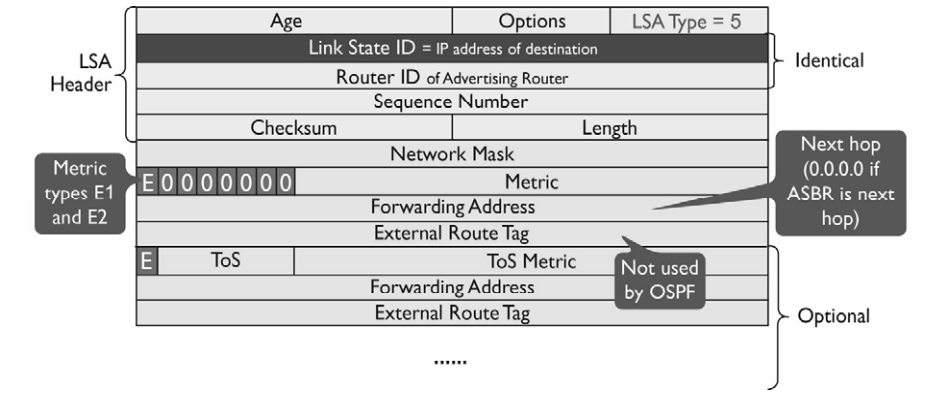


Although routers that are not directly connected to the subnet in which the forwarding address resides will not use that address as a next-hop, they do know that there is no point in creating a route based on the LSA unless they do have a route to the forwarding address.

This is because packets sent to the external route in question must go through the device at the forwarding address at some point in their journey, so if there is no route to that device, it is not clear how the packets will be able to get there.

When a router decides whether to create a route based on an External LSA, and that LSA contains a non-zero forwarding address, the router must check that it has a route to that forwarding address.

An external route tag is just a piece of information that the originating router can put into the LSA. There is no rule about the format of this information, or what it should be used for. Typically, it is used for filtering purposes.



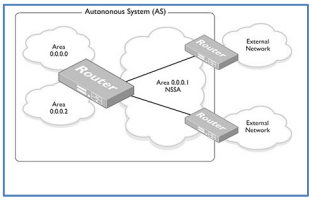
NSSA External LSAs (Type-7 LSAs)

It is not valid for routers in an NSSA area to receive normal External LSAs, but the routers may receive External routes from ASBR routers that reside in the same NSSA area. So, a special type of External LSA is required, to indicate that the LSA has been originated by a router in the NSSA area, to differentiate these from normal External LSAs that might have mistakenly made their way into the NSSA area. These special NSSA External LSAs are typically referred to as Type-7 LSAs, as they are allocated Type value 7.

Type-7 LSAs are transmitted only within the NSSA in which they were originated. They may not be transmitted out of that area. If the NSSA borders a non-stub area, you may wish that some or all of the external routes contained in the Type-7 LSAs originated in the NSSA are transmitted into the bordering non-stub area. If so, then the Area Border router at the border between an NSSA area and the non-stub area will be required to translate the Type-7 LSAs into normal External LSAs when it transmits them into the non-stub area.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/1Mebr5_-BWA



The structure of a Type-7 LSA is exactly the same as that of a Type-5 LSA. The only difference between them is the value of the LSType field.

Passing around LSAs

Reliable and efficient delivery of LSAs to all the routers they need to get to is at the heart of OSPF. In this section, we will look at:

- When the originator of an LSA will transmit the LSA.
- How LSAs are transmitted to all the routers that need to receive them.
- How a router makes sure that an LSA it has sent to a neighbor has been received by that neighbor.
- What a router does upon receipt of an LSA.

When does the originator send out its LSAs?

Every LSA has a unique originator. This is the one and only router in the network that is the source of the LSA. One of the central principles of OSPF is that only the originator of an LSA may:

- Generate the LSA.
- Send out new updates of the LSA.
- Withdraw the LSA from the network by *prematurely aging* it, i.e. by sending out a new copy of the LSA that has the age deliberately set to MaxAge (3600 seconds).

Every router is the originator of its own Router LSA. The DR on a multi-access subnet is the originator of the Network LSA for the subnet. Above, in sections ASBR Summary LSAs and External LSAs on page 65, we saw that Area Border routers are the originators of Summary LSAs, and Autonomous System Border Routers are originators of External LSAs.

So, every LSA has its unique originator, and every other router simply keeps a copy of the LSA, and passes it on to neighbors.

In RFC 2328, there is a very precise list of 10 events that will cause a router to 'originate' LSAs – i.e. to send out a copy of the LSAs that it originates. This precise definition of the events that cause LSA origination is required in a protocol specification – matters like this cannot be left to chance in a complex protocol.

Here, we will summarize these events, and discuss their context.

Event 1: The first reason listed for the origination of LSAs is one that is not strictly necessary for the passing around of topology information. It is the 1800-second refresh. This is a case where the protocol generates a little more traffic than it strictly needs to, in the interests of robustness. The rule is that every time a router notes that the age of one of its self-originated LSAs has reached 1800 seconds (half an hour), the LSA needs to be refreshed.

The act of refreshing an LSA means:

- Incrementing the LSA's sequence number (sequence numbers are discussed in LSA Sequence Numbers on page 74).
- Setting the LSA's age back to zero.
- Sending a copy of the LSA to all of the router's relevant full neighbors in the area.

This 1800-second refresh really goes hand-in-hand with the fact that the MaxAge of LSAs is one hour. That is, every time a router notices that the age of ANY LSA in its database (not just its self-originated LSAs, but ANY LSA) has reached 3600 seconds (one hour), then that LSA must be removed from the database.

The 1800-second refresh ensures that active LSAs do not reach an age of 3600 seconds. Hence, when any LSAs in a router's database do reach 3600 seconds it is clear that these are truly inactive LSAs.

Event 2: When one of the router's OSPF interfaces changes state (goes UP or DOWN). This usually requires a change to the Router LSA, as the Router LSA is effectively a list of the router's active OSPF interfaces. In particular, when a router's OSPF process first comes up, it will create a Router LSA as its OSPF interfaces become active.

Event 3: When the DR on a subnet that the router is attached to changes. In particular, if the router becomes the DR, or stops being a DR, it must start originating a Network LSA, or withdraw a Network LSA, respectively. But, even if the router's own status as a DR does not change, it needs to update its Router LSA, as the Router LSA mentions the IP address of the DR in any multi-access subnet that the router is attached to.

Event 4: If the router gains or loses a Full neighbor, it needs to update its Router LSA. Also, if it is DR on the subnet that the neighbor belongs to, it will need to update the Network LSA for that subnet.

Events 5 – 7: These are specific to the case that the router is a border router between two areas, and therefore is originating Summary LSAs into the areas. If a route in one area is

added/deleted/modified, then the Summary LSAssent to other areas will usually need to be updated.

Event 8: Is specific to the case where the router brings a virtual link up or down. Virtual links are discussed in more detail in the section Virtual Links on page 94.

Events 9 -10: Theses are specific to the case where the router is an autonomous system boundary router, and is therefore generating External LSAs. If the routes it is receiving from other routing protocols are added/deleted/modified, then the External LSAs it is advertising into OSPF will change.

LSA forwarding rules

Once LSAs have been originated, they need to get passed around from router to router, so that other routers learn them. There is no point in originating the LSAs unless they do get passed to other routers.

The process of passing LSAs around from router to router is referred to as *Flooding*. Effectively, the goal of LSA flooding is to ensure that any given LSA gets into the LSA databases of all the routers that it should. Note that the previous sentence says "*all the routers that it should*" rather than "*every router in the network*". The fact is that not every LSA goes to every router in the network. For different types of LSAs there are rules about the scope of the LSA's flooding.

For instance:

- Router and Network LSAs are flooded only to routers in the same area as the originator.
- An Area Border Router sends Summary LSAs into particular areas, and those Summary LSAs are flooded only to routers in the area that the Area Border Router sent them into.
- ASExternal LSAs are flooded into all routers in all areas except stub areas.

The precise rules for which LSAs are sent to which types of area will be explained in more detail in the section Areas on page 83. There are effectively three different circumstances in which a router transmits LSAs:

1. When the router is originating its own self-originated LSAs – due to any of the 10 events discussed above.
2. When the router receives a newer version of an LSA it already has – it will pass this new version on to all relevant full neighbors save that from which it received this LSA. This is the typical case where an LSA is in the process of being flooded – as it arrives at each router that it is being flooded to, that router needs to decide which neighbors it will forward it to. An LSA will only ever be forwarded to full neighbors.
3. When in the Loading phase of establishing a neighbor relationship. This is the phase in which the router sends to the neighbor those LSAs that the neighbor realizes it does not have (because it saw their headers in the Database Exchange Phase).

The process of transmitting LSAs

Given the fact that LSAs are the central piece of information that is exchanged by the OSPF protocol, it is very important that every router know that every LSA that it has sent to a given neighbor has actually been received by that neighbor. So, LSA exchange in OSPF implements a process of reliable transport.

The reliable transport mechanism is not terribly complicated. In broad terms, it operates as follows:

1. A router has an LSA retransmit queue for each of its full neighbors. Every time it sends a given LSA to a given neighbor, it puts a copy of that LSA into the retransmit queue for that neighbor.

2. When the neighbor receives the LSA, it needs to send an acknowledgement that the LSA has been received.

3. When the sender of the LSA receives an acknowledgement of receipt from a given neighbor, it removes that LSA from the retransmit queue it is holding for that neighbor.

4. If an LSA has been in a retransmit queue for more than a certain period (known as the retransmit time, typically 5 seconds) then the router retransmits the LSA to the neighbor to which that retransmit queue is related.

There is some interesting detail to consider at each of these 4 steps, though. So, let us look at the details of each step in turn.

Sending an LSA to a neighbor

LSAs are transmitted in Link State Update (LSU) packets. In the interests of keeping total protocol traffic to a minimum:

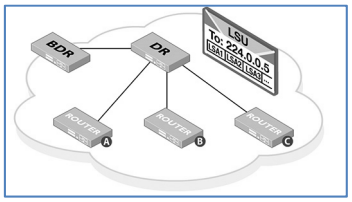
- there can be multiple LSAs in a single LSU packet
- an LSU can be multicast to multiple neighbors at once

The multicasting of LSUs is, of course, a little selective. For a start, it is not possible to multicast LSUs on NBMA networks, as such networks do not support multicast. So, in an NBMA network, LSUs must be unicast separately to each neighbor. On point-to-point networks, the LSUs are sent to the multicast address 224.0.0.5, but there is, of course, only one recipient of the packets.

The multicasting is really only fully utilized on broadcast networks. In a broadcast subnet, the DR and BDR send LSUs to ALL other routers in the subnet, so they send LSUs to the address 224.0.0.5 (referred to as the ALLSPFROUTERS address). The non-DR routers, on the other hand, only need to send their LSUs to the DR and BDR, so they send LSUs to the address 224.0.0.6 (referred to as the ALLDROUTERS address).

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/FMz8Qj6HTu0



But, when the router multicasts LSUs, which neighbor's retransmit queue does it put the LSAs into? Well, it adds the LSAs to the retransmit queues for all the neighbors that should receive and process the LSU.

Sending an acknowledgement

There are two ways that a router can acknowledge receipt of an LSA – explicit acknowledgement and implicit acknowledgement

Explicit acknowledgement involves sending a Link State Acknowledgement (LSAck) packet. An LSAck packet contains one or more LSA headers of the LSAs that the router wishes to acknowledge. I.e., the LSAck packet effectively says "*I have received the LSAs whose headers are contained in this packet*".

Implicit acknowledgement is achieved by forwarding on the received LSAs to other neighbors in such a way that the neighbor that the router received them from will also see the forwarded LSAs. This, of course, is done by sending the LSAs in an LSU packet to a multicast address. We will discuss the details of this further down.

First, let's look into the details of explicit acknowledgement.

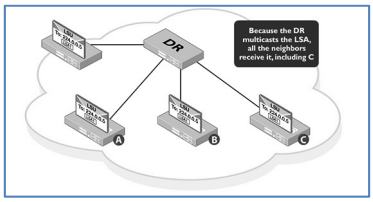
In fact, explicit acknowledgement can, itself, be done in two different ways – direct acknowledgement and delayed acknowledgement.

The *standard* way for a router to acknowledge LSAs is by delayed acknowledgement. In typical OSPF fashion, the point of delayed acknowledgement is to prevent sending too many packets. What happens is that a router keeps track of the LSAs that it receives from given neighbors over a period of time (less than 5 seconds) and then sends out a single LSAck to acknowledge those LSAs all at once.

What is more, on a broadcast network, delayed LSAcks are not simply sent back to the unicast address of the neighbor that sent the LSAs. By sending LSAcks to multicast addresses, a router can Ack multiple neighbors at once. So, if a DR on a broadcast network has received some LSAs from each of several non-DR neighbors, it can acknowledge all those LSAs at once, to the various different neighbors that sent the LSAs, by sending an LSAck to 224.0.0.5. All the other routers in the subnet receive the LSAck, and see the LSAs they sent to the DR being Acked.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/GN64R_Q1K2E



Similarly, if one of the non-DR routers receives LSAs from the DR and/or BDR, it sends its LSAck to the ALLDROUTERS address 224.0.0.6, and the DR and BDR both pick up that LSAck.

On NBMA networks, LSAcks are sent to the unicast address of the neighbor that is being Acked.

Direct acknowledgement is the act of sending an LSAack straight away, as soon as the LSU packet has been received. The acknowledgment is sent to the unicast address of the LSU sender, irrespective of the network type.

A direct acknowledgement is sent when the router wants to be sure the neighbor has got the message that it does not need to send it a particular LSA again. That is, a direct acknowledgement is sent when:

- A duplicate copy of an LSA has been received.
- An LSA has been received with age 3600 and the LSA in question has already been removed from the LSA database. (In which case, the router had quite possibly already received that LSA with MaxAge, which is why it has already been deleted from the LSA database.)

Now let's consider implicit acknowledgement:

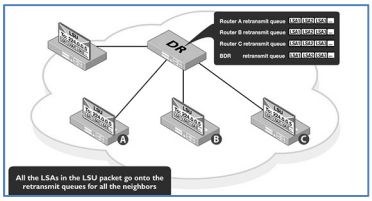
If a router sends an LSA to a neighbor, and subsequently sees the same LSA come back from that neighbor, then the act of seeing that same LSA come back from that neighbor is referred to as implicit acknowledgement The fact that this LSA arrives from that neighbor is pretty solid evidence that the neighbor received the LSA. Given that the purpose of the acknowledgement process is so that the sender of an LSA can have confirmation that the recipient received it, then this implicit acknowledgement fulfils that purpose.

You might wonder how it would come about that a router would send an LSA back to a neighbor that it received it from. In fact, we described above (in the section LSA flooding on page 33) that routers quite explicitly **avoid** sending LSAs back to the neighbors from which they received them. Well, there is a case where it is quite natural and inevitable that a router will see an LSA reflected back by a neighbor. The case is in a broadcast network, when a non-DR router sends an LSA to the DR. The DR will retransmit this LSA to all other routers on

the network that have not previously received the LSA. But, the DR multicasts the LSU to the ALLSPFROUTERS address 224.0.0.5, so the original sender of the LSA will see this retransmission of the LSA.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/FjWtTTnhlrg



LSA Sequence Numbers

One potential problem in an OSPF network is that of multiple copies of given LSAs floating around. For example, a router could originate its Router LSA, and then very shortly afterwards one of its interfaces could change state, and it would have to originate a new copy of the Router LSA.

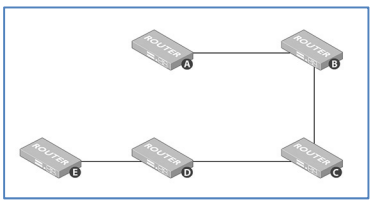
As other routers receive the LSAs, they have to decide to keep just one version of the LSA. Of course, the right version to keep is the most up-to-date one. Most of the time, it is quite easy to work out which is the most up-to-date version – it is the one that arrived most recently.

However, you cannot **guarantee** that any given router will receive LSAs in the same order as they were generated. For example, a new link could become active between the origination of the first and second versions of the LSA, and this new link could provide a short-cut to certain routers.

The second LSA would be transmitted via this short cut, and could possibly arrive at some routers before the first version of the LSA has made its way around to those routers.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/-8mQrDcgcms



Trying to rely on the order of arrival as the criterion for deciding which version of an LSA is more up-to-date is rather risky. A truly robust protocol does not take those sorts of risks. OSPF is a proudly robust protocol, so it has a more reliable, deterministic method of deciding which the more up-to-date LSA version is.

The mechanism that is used is a sequence number that appears in the header of all LSAs. The sequence number can take on a value between -2147483647 and 2147483647. When a router originates one of its self-originated LSAs for the first time (well, the first time since its last reboot), it sets the sequence number in the LSA to -2147483647 (0x80000001). Then, each time the router has to re-originate the LSA – (for any of the 10 reasons that a router originates LSAs), it increments the sequence number.

This simple mechanism makes it easy to decide which is the most up-to-date version of a given LSA – it is the version with the highest value of the sequence number.

There are a couple of situations in which the sequence number of an LSA will suddenly decrease – when a router reboots, and when the LSA sequence number wraps around from 2147483647 (0x7fffffff) back to -2147483647. As you might expect, OSPF has thought ahead, and has cooked up a plan for how to deal with those situations.

If a router reboots, it will very likely receive back copies of its own originated LSAs from its neighbors as it goes through the database exchange process with those neighbors. (The neighbors would have held onto copies of the rebooting router's LSAs for up to an hour). Typically, the sequence numbers in these received copies of its own LSAs will be higher than the sequence number value (-2147483647) it was planning to put into the LSAs as it originated them. The simple way to deal with this situation, though, is simply to make sure that it originates its LSAs with a sequence number value higher than the value it sees in the received copies of its own LSAs.

There is a general rule that if a router EVER receives a copy of one of its self-originated LSAs, and the sequence number in this LSA is higher than the sequence number in its own current copy of the LSA, then it re-originates the LS A, with a sequence number one greater than the value in the received LSA.

When the sequence number in an LSA reaches 2147483647, the rule is that the router must send out a new copy of the LSA, with the age set to MaxAge, to age the LSA out of the network. Then, once the router's neighbors have acknowledged the MaxAge LSA, the router re-originates the LSA with a sequence number of -2147483647. This does mean that the LSA does briefly disappear from the network, causing some routing disruption. But, do note that the full range of possible sequence numbers is VERY large. Even if a given LSA was re-originated once every second (which is a crazy behavior in itself) it would take over 136 years for its sequence number to reach 2147483647. Suffice to say – this is not an event that many network administrators are losing sleep over.

Hold-down timers

Further to the business of sequence numbers wrapping around (well, to sequence numbers not wrapping around), it should be pointed out that an LSA can only be updated, at most, every 5 seconds. Also, a router is not allowed to accept an LSA if the copy already in the database is less than l second old. So, if a router is configured erroneously, if its interfaces are coming up and going down in rapid succession, or otherwise experiencing problems, its neighbors will not process any burst of LSA updates that arrived within a second of each other, nor will they send those LSAs to other routers.

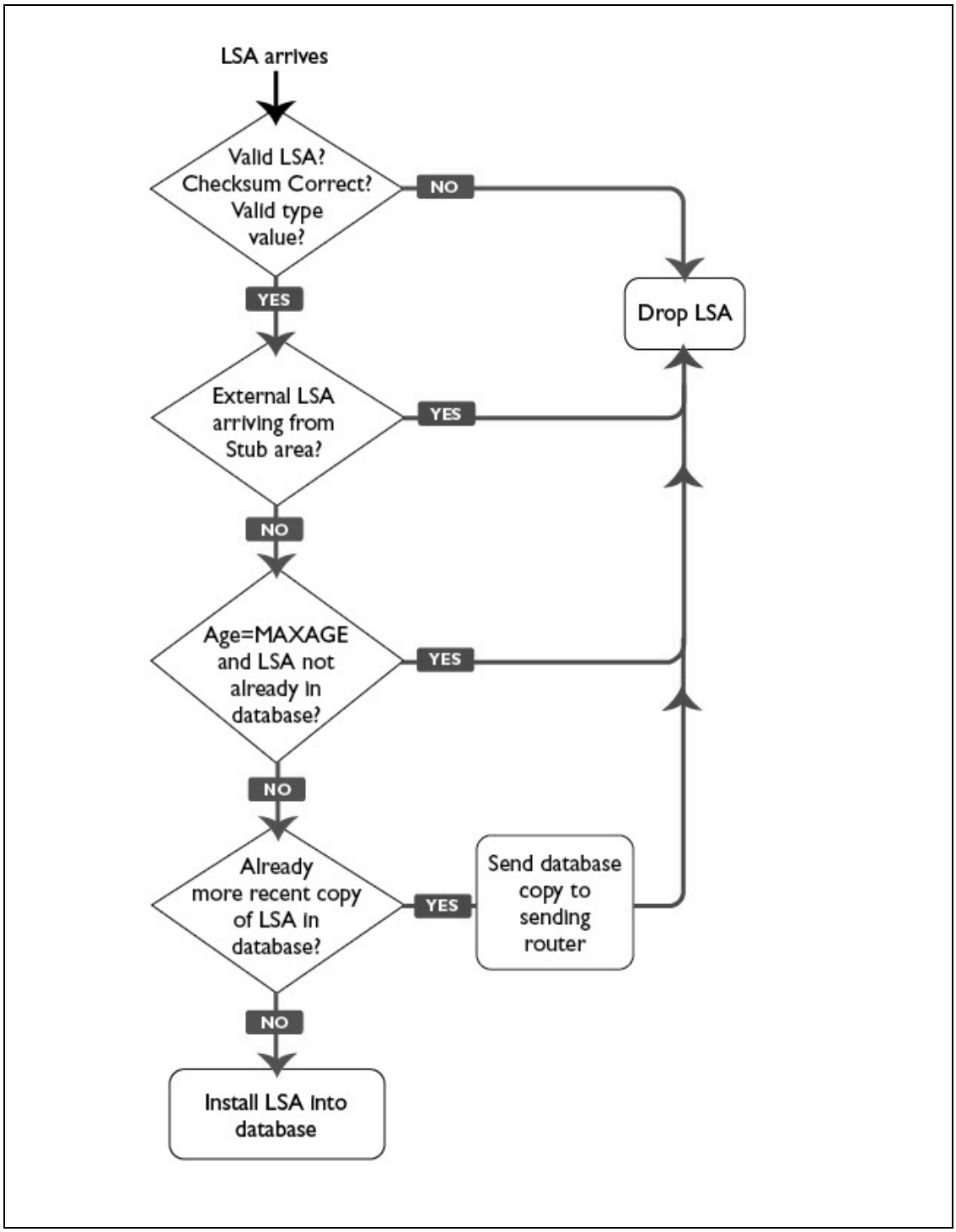
Receiving LSAs

Up until now, we have been discussing the process of sending LSAs – a router originating its own LSAs, and routers passing on LSAs that they have received from other routers. But, let us just spend a bit of time thinking about exactly what a router does when it receives LSAs. In the discussion above, it has been implicit that a router does not blindly take every LSA it receives and shove that LSA into its LSA database. Routers have to be cleverer than that – they have to make sure that only the *right* LSAs go into their LSA databases. So, the thing we need to look at here is just how a router decides which are the *right* LSAs.

When receiving an LSA, a router performs a number of checks on the LSA before deciding to put it into its LSA database.

- First it checks the validity of the LSA – is its checksum correct? Does it have a valid value in the LSA type field?
- Then, it checks whether the LSA is of a type that should be arriving on the interface that it is arriving on. Specifically, an ASExternal LSA should not arrive on an interface that is in a stub area. If this has occurred, then the LSA is discarded.
- If the age of the LSA is MaxAge, and there is no copy of the LSA already in the router's database, then the LSA is discarded.
- If there is already of a copy of the LSA in the router's LSA database, and the router's existing copy is more recent (has a higher sequence number) then the received LSA is discarded, but the router sends its more recent copy back to the router it just received this older copy of the LSA from. I.e., the receipt of the older copy indicates that the sending neighbor's LSA database is not quite up to date, so the receiving router takes the opportunity to help its neighbor get up to date.
- Finally, if the received LSA is valid, and either there is no copy of the same LSA already in the database, or there is an older copy of the LSA in the database, then the newly received LSA is installed into the database.

The diagram following illustrates what a router does when it receives LSAs:



Changing the content of the LSA database will often result in the routing table being updated. But, if the change to the database is simply that one or more LSAs are replaced by new copies that differ only in their sequence number and age, then there is no need to update the routing table.

LSA Database Maintenance

A router needs to keep an eye on the age of all the LSAs in its database. If any LSA's age reaches MaxAge, it needs to be cleaned out of the router's database, and the router needs to tell everybody else to clean it out of their databases as well.

The router needs to keep updating the ages of the LSAs in its database as time progresses. Whenever any LSA's age does reach MaxAge, the cleanout process kicks into action.

As soon as the LSA's age reaches MaxAge, the route table calculation must ignore the LSA.

The router also immediately sends the MaxAge LSA to all the neighbors that the LSA should be sent to. Once all those neighbors have acknowledged the LSA, it is removed from the LSA database.

How do routes actually get withdrawn in an OSPF network?

If a particular router that is advertising a set of LSAs gets cut off from the network, does that router's neighbors say "*ah, I know that router is no longer available, I will remove its LSAs from my database, and tell everybody else to do the same*"? Well, no, the interesting fact is that they do not.

One of the set of sacred principles of OSPF is that only the originator of a given LSA can age out that LSA. No other router is allowed to decide that an LSA is dead, and prematurely set its age to MaxAge.

That does mean that dead LSAs (LSAs originated by an unreachable router) can hang around in the LSA databases in a network for up to an hour.

Does that mean that dead routes can hang around in the network for up to an hour, waiting for LSAs' ages to reach MaxAge, and finally disappear? Fortunately, the answer to that is "no, OSPF is better than that".

So, let's look at just exactly how routes end up being removed, even if dead LSAs can hang around for up to an hour.

We can do this by looking at a few different situations in which a route is lost.

Case 1: An interface goes down

If a router has a set of interfaces, connected to different subnets, that are OSPF interfaces, then those subnets will be advertised as connections on that router's Router LSA. What is more, if the router is the only router connected to a particular subnet (and it is an NBMA or Broadcast subnet) then the router will also generate a Network LSA for that subnet.

Other routers will, therefore, calculate routes to this subnet when they perform their SPF calculations.

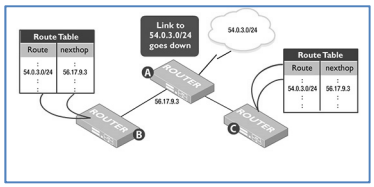
If one of the interfaces goes down, then the router will re-issue a new version of its Router LSA (with the sequence number incremented). In this new version of the Router LSA, the subnet attached to the now-down interface will no longer be listed.

Also, if the router was originating a Network LSA for this subnet, then it will prematurely age this Network LSA. I.e., it will set the age of the LSA to MaxAge, and flood it to the network.

In this way, the other routers in the same area as the router in question will end up with changes to their LSA databases. They will redo their SFP calculations and come up with a result that does not include a route to the subnet attached to the now-down interface of that originating router.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/D3olWgR-5h0



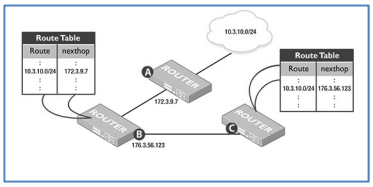
So, that is how the route to the newly lost subnet is quickly removed from the route tables of the routers in the network.

Case 2: A router goes down

If a router that has one or more neighbors, and is also the gateway to one or more subnets, goes down, then the situation is rather different to the case above, as this router is not able to prematurely age the LSAs it originates. Because it is down, it is not able to do anything much at all.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/y6qQWV8nE38



But, the routers that are the neighbors of the lost router will have experienced a change of state on the interfaces via which they connected to that router, so they will re-issue their Router LSAs. Also, any routers that are DRs on Broadcast or NBMA networks that the lost router was attached to, will re-issue their network LSAs for the relevant network(s).

Again, the other routers in the network will experience changes to their LSA databases, and will redo their SPF calculations.

When they do the SPF calculations, they will find that although they still have a Router LSA for the lost router, there are no other Router LSAs or network LSAs that attach to that router. So there will be no continuous paths to the subnets that this router was the gateway for. Hence their route tables will no longer hold routes to those subnets.

Again, the routes to the inaccessible subnets are quickly removed from the route table in the network, even though the LSAs originated by the down router are not yet flushed.

Note - if the router stays down long enough for the age of its LSAs to reach MaxAge, then they will be flushed from the network. But, at that point, the flushing of those LSAs will be a mere formality; it will not have an effect on the contents of the route tables.

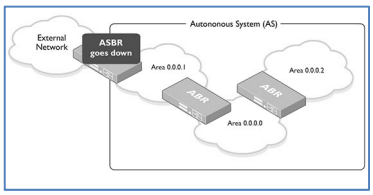
Case 3: An ASBorder router goes down

A special case to consider is that where a router goes down, and that router has been originating ASExternal LSAs.

This case is special because ASExternal LSAs are outside of the SPF calculation process. The routes that are generated from ASExternal LSAs are not found by connecting Router LSAs to Network LSAs to create a path to the destination. Rather, External LSAs are simply routes in their own right, and are installed into the route table as-is (with an appropriate metric calculation).If the router that is originating the ASExternal LSAs goes down, then it cannot prematurely age the LSAs. These LSAs will remain in all the routers' LSA databases even though the originating router is down.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/KUZYcrbutvc



What prevents these unreachable external routes from remaining in the routers' route tables? The answer is that OSPF has a simple rule relating to ASExternal routes – namely that a router must have a route to the originator of the External LSA before the external route can be put into the route table. Because the originating router is down, the route to that router will no longer be available (as its neighbors will have removed it from their Router LSAs and Network LSAs). Hence the routes corresponding to the External LSAs that it has originated will not be installed into the routers' route tables.

The effect of OSPF timers on route withdrawal scenarios

- If a link goes down for 20 seconds, then comes back up, OSPF doesn't notice, as the dead timer is 40 seconds (by default).
- If a link flaps constantly, but at least one of every four Hello packets make it across, OSPF doesn't notice. Because the dead timer, by default, is 4 times the Hello timer.
- If a link goes down for anywhere from a minute to half an hour, OSPF floods an LSA when it goes down, and another LSA when it comes back up.
- If a link stays down for more than half an hour, LSAs originated by remote routers (that have become unreachable) begin to age out, because at least some LSAs will reach MaxAge in that time. When the link comes back up, all these LSAs will be re-originated by the routers that are connected to the restored link and reflooded.
- If a link is down for more than an hour, LSAs originated by remote routers will have aged out and been flushed. When the link comes back up, it will be as if it were brand new.

OSPF Network Structure

At last, we have come to the section where we properly describe OSPF areas. Although areas have been referenced many times in previous parts of this chapter (as areas are so pervasive to the protocol) it is only now that we actually describe areas in detail.

To allow better control and management over larger internetworks, OSPF allows a large autonomous system to be structured into a hierarchical form. Contiguous routers and networks are grouped into areas that connect together using a logical backbone. These areas act as the equivalent of smaller autonomous systems within the larger AS, yielding the same benefits of localized control and traffic management that autonomous systems provide for a large internetwork between organizations.

Hierarchical structure

The components that make up the hierarchical structure include:

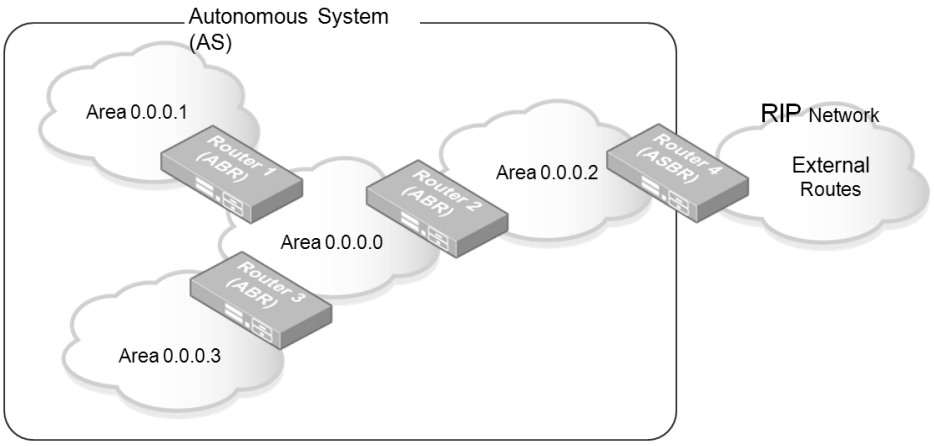
- The Autonomous System
- Areas
- Networks
- Neighbors

Let us understand each of these components.

The Autonomous System

The highest level of the hierarchy is the autonomous system (AS). An autonomous system is defined as a number of networks, all of which share the same routing and administration characteristics.

An AS is either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of an entity (such as a university, a business enterprise, or ISP). An autonomous system is also sometimes referred to as a routing domain. Each autonomous system is assigned a globally unique number called an Autonomous System Number (ASN).



Note that in the context of the OSPF protocol, an autonomous system is the set of routers, under a specific administration, that are all using OSPF. If there are other routers, under the same administration, that are using another routing protocol (e.g. RIP or BGP), then those routers are considered to be outside the OSPF autonomous system. The routers that are on the border between the OSPF autonomous system and 'the rest of the world' are referred to as Autonomous System Border Routers (ASBR).

Areas

An AS can be divided into multiple areas as shown in the diagram above. Each area represents a collection of contiguous networks and hosts. Areas limit the set of routers to which Router and Network LSAs are flooded. Imposing this limit keeps control of the total amount of OSPF packet exchange that goes on in the network. It also reduces the size of the LSA databases within individual routers - they don't need to hold the Router LSAs and Network LSAs originated by routers that are outside their area.

An area also defines the set of routers whose LSA databases must be identical. All routers within a given area must have identical contents in their LSA databases, the content will differ from the LSA databases of routers in other areas.

Each area is uniquely identified by its Area ID. The Area ID is a 32-bit number that uses the dotted decimal notation of an IP address (it is not an IP address, however). All routers within a given area must agree on the Area ID identifying the area. The OSPF protocol calls for Area IDs to be numbers, not strings, so you must enter either a decimal number or a number in IP Address format.

Under AlliedWare Plus, an area is not explicitly created on a router. Rather, it is introduced implicitly when a network is configured, as the area that the network belongs to is to be specified in that configuration command:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# network 10.0.0.0/8 area 1.1.1.1
```

Cardinal rules regarding areas

Two fundamental rules regarding OSPF areas are:

1. Every OSPF autonomous system must have a **backbone** area.
   This area has **Area ID 0**.

2. All other areas in the autonomous system must touch the backbone area. I.e., for every other area in the autonomous system, there must be at least one router that has at least one interface in that other area and at least one interface in the backbone area. This requirement can be satisfied in a rather artificial manner by means of a virtual link. Virtual links are discussed in more detail below in the section Virtual Links on page 94. But in essence, a virtual link is a tunnel through one or more non-backbone areas that enables a router that does not actually touch the backbone to have a tunneled connection to the backbone.

Areas types and terminology

Area types

Not all OSPF areas are created equal. In fact, there are a number of different categories of area. As always, this extra complexity is for the purpose of giving you more control over the amount of OSPF traffic that flows through the network and over the sizes of routers' LSA databases.

The categorization of OSPF areas rests primarily on how they deal with ASExternal LSAs. By default, OSPF handles external routes by flooding them throughout the OSPF network as ASExternal LSAs. That flooding occurs even across area borders. Therefore, splitting your OSPF autonomous system into areas provides little benefit when a large number of external routes are redistributed into OSPF. However, if some areas are designated as ASExternal-LSA free zones, then the number of routers that have to transmit and store the ASExternal LSAs can be reduced.

At the highest level of categorization, OSPF areas fall into two types:

- Stub areas – which do not originate ASExternal LSAs, and do not receive ASExternal LSAs from other areas.
- Non-stub (sometimes called transit) areas – which do take in, and do generate, ASExternal LSAs

Within these higher-level categories, there are a number of subcategories – in particular there are multiple different flavors of Stub area.

Types of non-stub area

Firstly, the non-stub areas fall into two categories. The first category is the backbone area. The other category is all other non-stub areas. The backbone area is a non-stub area with the special property that it has Area ID 0. Every OSPF autonomous system must have a backbone area.

The Autonomous System diagram on page 83 has Area 2 (labeled area 0.0.0.2) as a non-backbone, non-stub area. This area will originate ASExternal routes, as it contains an AS Border router. The backbone area will also send ASExternal LSAs into this area. So, this is an example of an "*other non-stub area*".

Types of stub area

Although we have said above that stub areas are ASExternal-LSA free zones, this is not 100% true. So, in fact, stub areas are categorized by just how zealous they are about keeping themselves clean of LSAs representing routes that are external to the area.

There are four categories of stub area.

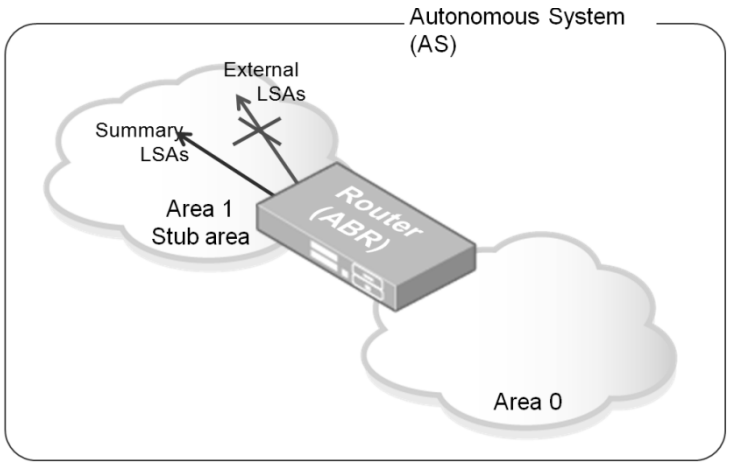
1. Normal stub areas

These areas:

- Do receive Summary LSAs from other areas.
- Do not contain any ASBRs. As a result, no External LSAs are ever generated within the area.
- Do not receive External LSAs from other areas.

The AlliedWare Plus commands to configure a normal stub area are:

```
awplus# configure terminal
awplus(config)# router ospf <process-id>
awplus(config-router)# area <area-id> stub
```



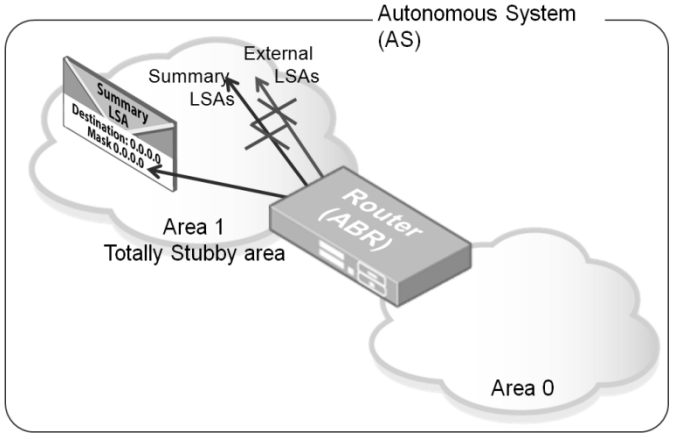
2. Totally stubby areas

These areas:

- Do not contain any ASBRs. As a result, no External LSAs are ever generated within the area.
- Do not receive External LSAs from other areas.
- Do not receive Summary LSAs from other areas. They just receive a single default route that is originated by the Area Border router.

The AlliedWare Plus commands to configure a totally stubby area are:

```
awplus# configure terminal
awplus(config)# router ospf <process-id>
awplus(config-router)# area <area-id> stub no-summary
```



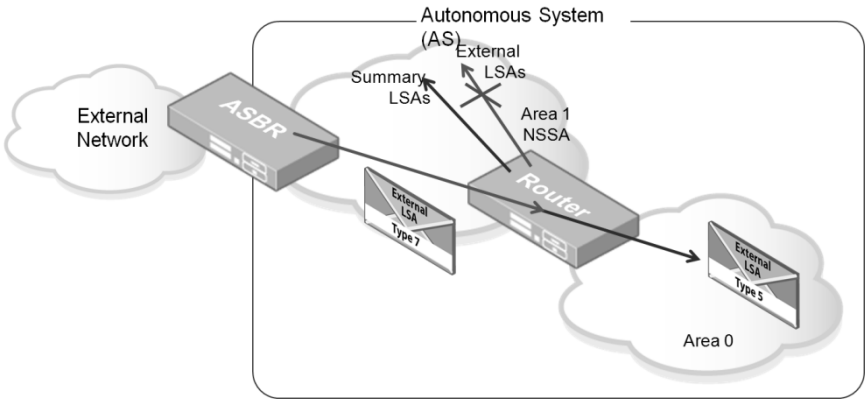
3. Not so stubby areas (NSSAs)

These areas:

- Do receive Summary LSAs from other areas.
- Do contain ASBRs. So, External LSAs are generated within the area. The External LSAs generated in an NSSA are Type-7 LSAs.
- Can advertise their External LSAs to other areas (translated to Type-5 LSAs)
- Do not receive External LSAs from other areas.

The AlliedWare Plus commands to configure an NSSA are:

```
awplus# configure terminal
awplus(config)# router ospf <process-id>
awplus(config-router)# area <area-id> nssa
```



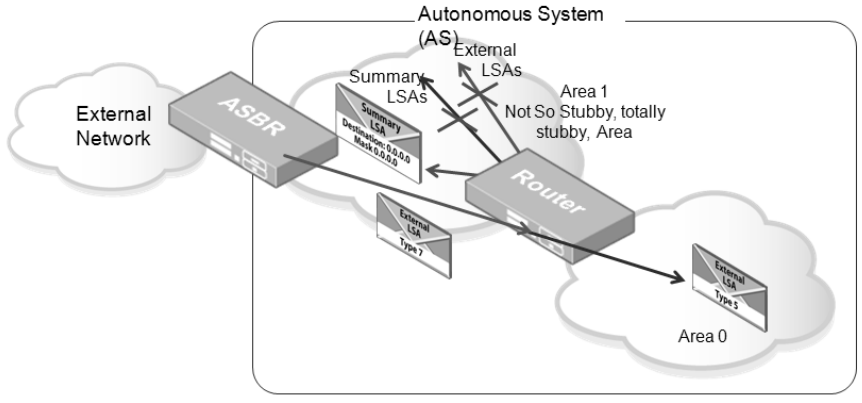
4. Not so stubby, totally stubby, areas

These areas:

- Do contain ASBRs. So, External LSAs are generated within the area. The External LSAs generated in an NSSA are Type-7 LSAs.
- Can advertise their External LSAs to other areas (translated to Type-5 LSAs)
- Do not receive External LSAs from other areas.
- Do not receive Summary LSAs from other areas. They just receive a single default route that is originated by the Area Border router.

The AlliedWare Plus commands to configure a Not-so-stubby-totally-stubby area are:

```
awplus# configure terminal
awplus(config)# router ospf <process-id>
awplus(config-router)# area <area-id> nssa no-summary
```



Artifacts of Areas

The concept of splitting the autonomous system into areas provides a structure upon which it is possible to define all manner of rules, and all manner of extra properties to assign to routes, LSAs, and routers.

These rules, properties, etc. all have sensible, useful purposes. But, they can be a challenge for those who are trying to understanding OSPF. The number of rules, and the amount of associated terminology, can be rather daunting.

In this section, let us look at the various OSPF features that arise from the existence of areas, to understand where all these items fit in, and the purpose that they play.

A number of these features are already discussed elsewhere in this section, but it is useful to have an overview of them all together here, to see how they fit together.

LSA forwarding rules

The biggest piece of value that areas deliver is the ability to control the amount of LSA passing around that goes on, and the number of LSAs that routers have to store, groom, and perform the Dijkstra algorithm on.

This is achieved by having rules about how far afield LSAs can propagate, and about how routing information contained in one area is transferred to other areas.

The rules are quite simple:

1. Router LSAs and Network LSAs are only passed to routers in the same areas as the originator. Hence, these LSAs never cross area borders.

2. At the border between areas, the routes known in one area are advertised into the adjoining area(s) in the form of Summary LSAs.

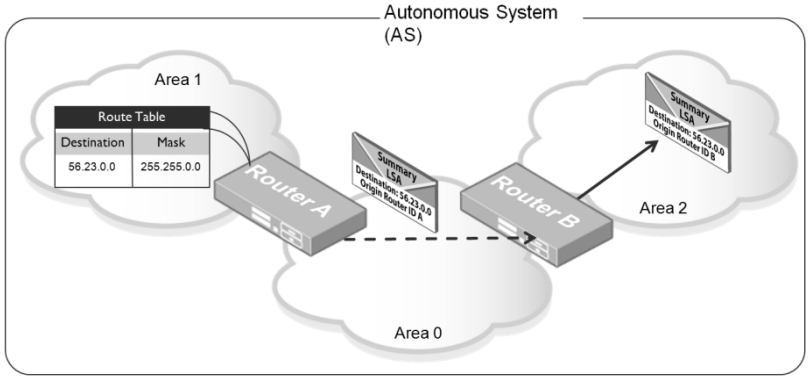
3. Summary LSAs are never transmitted out of the area into which they were originated. When the Summary LSA reaches the other side of an area, the router at the border into the next area originates a new Summary LSA into that next area. There is the special case that general Summary LSAsare not transmitted into totally stubby areas; just a special default-route Summary LSA is transmitted into those areas.

4. External LSAs are transmitted into all non-stub areas.

More information about these rules

Rule (1) means that by varying the sizes of areas, you can control the number of Router LSAs or Network LSAs any given router has to store; and forward to other routers. Without this rule, every router would receive every Router LSA and Network LSA for the whole network, and would have to perform the Dijkstra calculation across the whole lot.

That would mean that the limit on how big an OSPF network could grow would be governed by the storage and processing capacity of the lowest-spec router in the network.



Rule (2) provides the opportunity for route summarization at area borders. It is not necessary that every route in a given area generate exactly one summary route to be originated into adjoining areas. If several routes within one area can be coalesced together into a single route with a broader netmask, and the summary route contains this broader route, then that can cut down the number of summary routes that need to be generated. The summarizing of routes at the area border is discussed more in the section Route summarization on page 97.

Router roles

The creation of areas means that there are borders between areas, as well as borders between the autonomous system and the rest of the world. The routers on these borders have specific actions they must perform due to that location. So, for ease of identification, and clarity, terminology has been introduced to categorize the various roles that routers can play within a network.

Area Border Router

An OSPF router can be a member of multiple areas. Routers with membership in multiple areas are known as Area Border Routers (ABRs). Each ABR maintains a separate topological database for each area the router interfaces to. The ABR is responsible for forwarding routing information between the areas it borders. In particular, it originates Type 3 Network Summary LSAs, and Type 4 ASBR Summary LSAs, in order to transmit routing information from one area to others.

Autonomous System Border Router (ASBR)

An ASBR is a router that is connected to routers or hosts that are outside of the OSPF autonomous system. An ASBR could simply have a single connected route that lies outside the AS, or it could be configured with some static routes that point to destinations outside of the AS, or it might be exchanging routes with external routers using a non-OSPF routing protocol like RIP or BGP.

Internal Router (IR)

A router is called an Internal Router if it only has OSPF adjacencies with routers in the same area.

Backbone Router (BR)

A Backbone Router is a router with an interface in to the backbone area. An ABR would be a BR, although the reverse need not be true.

Route types

Where dynamic routing protocols such as RIP and IGRP have only one route type, a look at an OSPF routing table shows several different OSPF route types. There are three types of OSPF routes - intra-area, inter-area, and external (either type 1 or 2). What are these different categories of OSPF routes? Why have different categories of OSPF routes? Is this not just adding extra complexity?

Well, to answer the first question:

Intra-Area routes are ones that are calculated from the Router LSAs and Networks LSAs (using the Dijkstra algorithm) that have been received from routers in the same area as the current router. They are routes to destinations that are in the same area as the current router.

Inter-Area routes are those that have been created from Network Summary routes that have been received from ABRs connected to the current router's area. These represent routes to destinations in other areas of the OSPF autonomous system.

External routes are those created from External LSAs. These are routes to destinations that are outside the OSPF autonomous system. The difference between Type-1 External routes and Type-2 External routes is explained above in the section External LSAs.

Why have these different categories?

There are two prime reasons:

1. To enable choices to be made between different routes to the same destination.

A hierarchy of relative preference between different route types is defined in the OSPF RFCs. The order of preference (from most preferred to least preferred) is:

- Intra-Area
- Inter-Area

- Type-1 External
- Type-2 External
- NSSA Type-1 External
- NSSA Type-2 External

It is also possible to influence the relative priorities of the different route types by setting administrative distances.

In AlliedWare Plus, to set the administrative distances of different route types in OSPF 100 to:

- 20 for inter-area routes
- 10 for intra-area routes
- 40 for external routes

Use the commands:

```
awplus(config)# router ospf 100
awplus(config-router)# distance ospf inter-area 20 intra-area 10
external 40
```

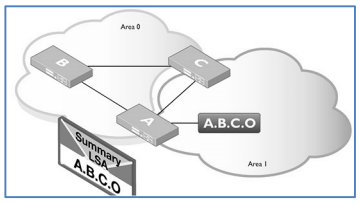
To set the administrative distance for all routes in OSPF 100 back to the default of 110, use the commands:

```
awplus(config)# router ospf 100
awplus(config-router)# no distance ospf
```

The practical application of these relative priorities of the different OSPF types is illustrated in the diagram below:

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/mfvR8ybdAzU



2. To aid network management and trouble-shooting. By splitting the routes up into categories, and labeling with their category in the route table, it makes it easier to work out which routes have come from where.

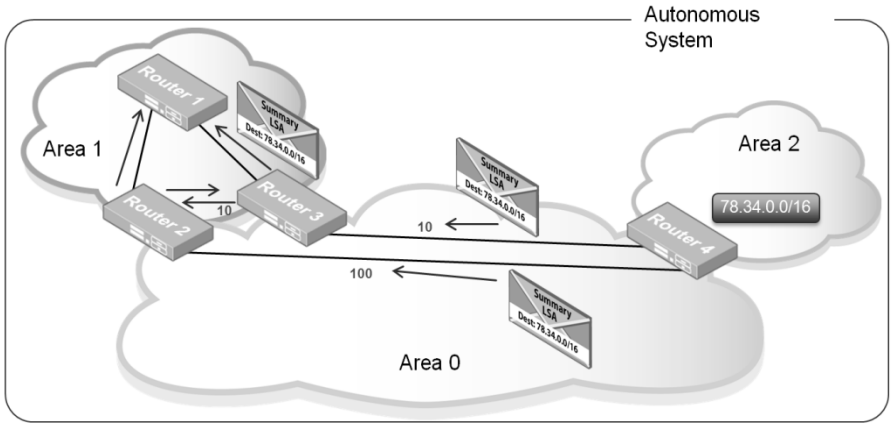
Split Horizon

Split Horizon is a term frequently used in the context of RIP, but not often used in relation to OSPF. However, OSPF does actually implement a Split Horizon.

Generically, the term Split Horizon means **not sending routing information back in the direction from which it came.**

In RIP, this is implemented on a per-interface basis – a RIP router that implements Split Horizon does not send advertisements out through interface X for routes whose egress interface is interface X. In OSPF, the implementation of Split Horizon is a little more complex (of course, everything is 'a little more complex' in OSPF).

To understand OSPF Split Horizon, consider the diagram below.



- The area-border Router R4 sends Summary LSAs for the subnet 78.34.0.0/16 into the backbone.
- Those Summary LSAs arrive at Area Border Routers R2 and R3.
- These routers Originate new Summary LSAs for the subnet 78.34.0.0/16 into Area 1.
- Because these routers have a link to each other in Area 1, they will send this summary LSA to each other.

Now, this is where the Split Horizon comes in.

Consider what happens when Routers R2 and R3 receive each other's Summary LSAsfor 78.34.0.0/16. As far as **R2** is concerned, it has received one Summary LSA for 78.34.0.0/16 from **R4**, with cost of **110** (an LSA cost of 10 + 100); and one Summary LSA for 78.34.0.0/16 from **R3**, with cost **30** (an LSA cost of 10 +10 + 10).

If it simply took the best-cost option, it would install a route to 78.34.0.0/16 with R3 as the next hop.

Then, that would be a route pointing into Area 1. So, it would create a Summary route for 78.34.0.0/16 and advertise it back into Area 0. This is exactly the sort of behavior that leads to routing **loops**.

To prevent this happening, OSPF implements the following Split Horizon rules:

- The only Summary LSAsfrom which an ABR can create route table entries are Summary LSAsthat it has received from the backbone area.
- When deciding on the Summary LSAs that it will originate into the backbone area, the only routes that an ABR will consider are intra-area routes that it has learnt in the other areas it is connected to.

Virtual Links

One of the more abstract constructs that result from partitioning the AS into areas is Virtual Links.

As mentioned above, in the section Cardinal rules regarding areas on page 84, all non-backbone areas must border the backbone area. This gives OSPF a star topology, with the backbone area being the hub, and all the other areas being the spokes.

This requirement goes hand-in-hand with the Split Horizon rules above; as those rules mean that an Area Border Router that does not connect to the backbone is not going to be able to do much with the Summary LSAs that it receives.

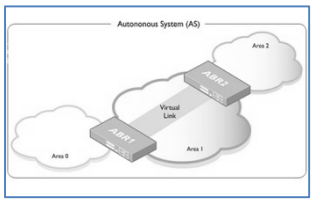
The net result is that traffic being transported from one non-backbone area to another non-backbone area must follow a path of the form:

source nonbackbone area → backbone area → *destination nonbackbone area*

What happens if, due to unfortunate circumstances, you end up with a network in which some areas don't have a connection to the backbone? In particular, this can happen if a large non-backbone area has to be divided up.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/6Y9EeNQtGlo



The solution to this dilemma is the Virtual Link. This is a mechanism that tunnels a small corridor of the backbone area across a non-backbone area, to provide a remote area a tenuous connection to the backbone.

Through this tunnel, the backbone sets up a supply line through which it can transport Summary LSAs to the ABR at the far end of the tunnel. Thereby, that ABR can learn of routes that are in the backbone, and in other areas.

If these Summary LSAs were not being tunneled in by the backbone, then the only Summary LSAs that the ABR would be receiving would be from non-backbone areas.

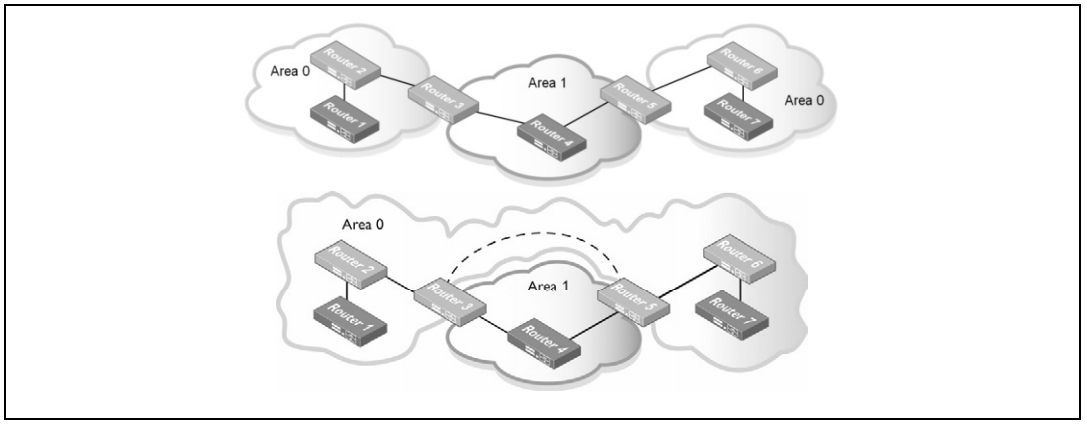
One of the Split Horizon rules above states that an ABR may not create route table entries from Summary LSAs that it has learnt from non-backbone areas. It would completely fail to create route entries for **any** inter-area routes.

The area that the virtual link tunnels through is referred to as the **transit** area.

There is another purpose that virtual links serve – namely that of keeping the backbone area contiguous. One of the requirements of OSPF is that the backbone area be a single contiguous area; it cannot split up into disconnected clumps of backbone-ness.

If the structure of the network works out in a way that causes the backbone area to be split into pieces, then virtual links are required to connect the pieces of the backbone into a contiguous whole.

In the example below, a virtual link is configured between R3 and R5. The backbone effectively becomes continuous:



Configuring a virtual link

The virtual link must be configured on both ends of the virtual link. Each router specifies the Router ID of the router at the other end of the link, and the transit area that the link will pass through.

For example, for the case illustrated above, the configuration would be:

On Router R3:

```
awplus# configure terminal
awplus(config)# router ospf 1
awplus(config-router)# area 1 virtual-link 10.10.11.5
```

On Router R5:

```
awplus# configure terminal
awplus(config)# router ospf 1
awplus(config-router)# area 1 virtual-link 10.10.11.3
```

It is possible to configure other parameters on a virtual link, like the Hello and Dead times, and authentication parameters.

Networks

The next level of organizational structure within an OSPF autonomous system is the Network. The concept of a Network within OSPF, and the way that OSPF deals with different categories of Network are discussed in detail in the section Network Types on page 36.

Neighbors

The most granular level of structure (apart from individual routers) in OSPF is the neighbor relationship. This relationship is a foundational building block that the structure of OSPF is based upon.

The forming of neighbor relationships provides the links that the SPF calculation is based upon. The rules governing the parameters that must match up before routers can be neighbors, ensures configuration consistency across an area. The processes controlling the exchange of LSAs between neighbors ensure the database consistency across an area. This is a fundamental requirement for the correct route tables to be created in all routers.

The details of neighbor relationships – how they form, and how they are maintained, are described above in the section Neighbor Relationships on page 41.

Managing the Content of Route Tables

Within OSPF, there are a number of features you can use that will affect the choice of entries that eventually go into routers' route tables.

The features are:

- Route summarization at area borders
- Route filtering at area borders
- Route redistribution
- Summarization of redistributed routes
- Generating default routes
- Route maps and filtering

Most of these are not features that are inherent in the OSPF protocol, but they are all important aspects of using OSPF in a network. Almost all implementations of OSPF will provide these features, to enable the user to control what routes end up in the network's route tables.

Let us now look at each of these features. First we will consider summarization in general, and then go into the individual features listed above.

Route summarization

Within a network, there are likely to be a number of different subnets - class C network ranges are forever being cut up into small pieces and allocated to small parts of the network where just a small set of hosts need to be put off into their own segment.

Of course, the routers that are logically close to any given subnet need to know the routes to the individual subnets. But, routers at a greater remove just need to know the route to follow to reach the whole address block that encompasses those subnets.

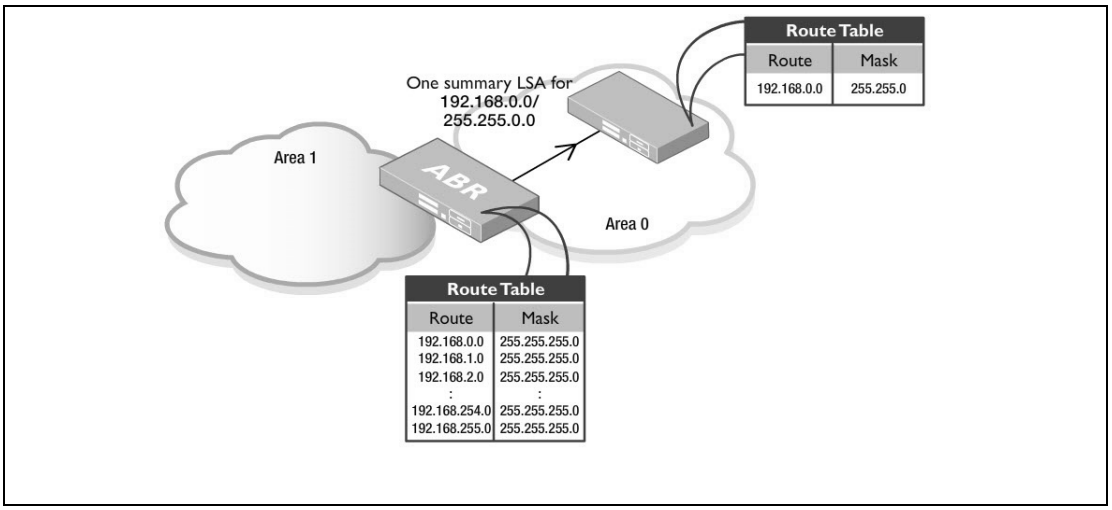
The obvious analogy is direction signs on roads. When you are driving out on the open road, the road signs point to complete cities; as you reach the edge of a city, the signs point to individual suburbs, and then when you are close to your destination, you are looking at signs that name individual streets.

In the world of routing protocols, there is a concept of route summarization, whereby routers can receive information about a number of individual subnets, but will advertise a single route that encompasses the full address range that covers all those individual subnets.

Benefits

Route summarization has two main benefits.

- The most obvious is that it is a way to reduce the size of route tables. As we have seen regularly in the discussion of OSPF, the act of not letting tables get too big is an important consideration. This reduces the amount of memory that the routers require in order to store the routing tables, and the amount of computing power they need in order to perform route lookups at an acceptable rate.

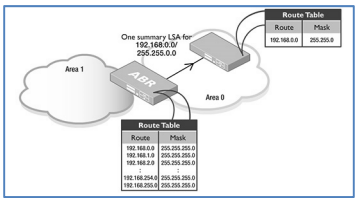


- Another, not quite so obvious, benefit is that summarization leads to less updates to route tables. Links at the edge of the network have a tendency to go up and down rather often. This results in the subnets at the edge changing between an available state and an unavailable state.

The following animation shows a network **using** route summarization.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/-Xcr6XRgk40



When a given subnet changes between available and unavailable, routers are obliged to advertise this fact. However, if a router is not advertising every individual subnet that is downstream of it, but is advertising a summarized route, then the sudden unavailability of one particular subnet does not mean the whole summarized route is unavailable. So, there is no need for the summarizing router to make any change in what it advertises to upstream routers.

This reduces the amount of OSPF traffic that is exchanged. It also reduces the amount of time the routers spend recalculating routes and updating their route tables. In particular, if there is a route flapping (and interface going up and down with unseemly frequency) in a particular portion of the network, then summarization isolates the rest of the network from the negative effects of this flapping.

Summarization in OSPF

There are two types of summarization in OSPF:

- inter-area route summarization
- external route summarization

Inter-area route summarization occurs at area boundaries. Summarization can be done at any area boundary.

External route summarization is performed by ASBRs. The ASBR can take the many external routes it learns, and convert them into a smaller number of summarized routes that are advertised in the External LSAs that it sends out.

Both these forms of summarization are discussed in more detail below.

Route summarization at area borders

In order to take advantage of summarization, the subnets assigned to networks within a given area should be assigned in a contiguous way, so they can be gathered together in a single range, or a small set of ranges.

The range(s) that encompass the subnets within an area are configured on the area.

For example, if you want to summarize all the routes in Area 1 into the two summary routes 192.16.0.0/16 and 203.18.0.0/16, then the configuration is:

```
awplus# configure terminal
awplus(config)# router ospf 1
awplus(config-router)# area 1 range 192.16.0.0/16
awplus(config-router)# area 1 range 203.18.0.0/16
```

With this configuration, the ABR will advertise these two summary routes into the other areas that it borders. It will not advertise routes that are subnets within these summary routes. Also, it will not advertise a summary route unless it has learnt, via an interface in Area 1, at least one subnet that lies within the range covered by the summary route.

If the router learns, via an interface in Area 1, other routes that are outside of these summary routes, it will also advertise those, as individual routes, into other areas.

Route filtering at area borders

The area border is also a convenient place at which to filter out routes that do not need to be known by routers outside the area.

It is possible to configure the router to not advertise a particular range, using the command:

```
awplus(config-router)# area 1 range 203.18.0.0/16 not-advertise
```

Route redistribution

The act of importing routes into OSPF from other sources, or exporting routes from OSPF to other protocols, is referred to as Route Redistribution.

When routes are imported from other sources – BGP, RIP, static routes, and connected routes on non-OSPF interfaces, they are advertised into OSPF as External LSAs. These LSAs are discussed in detail above in the section External LSAs on page 65.

Redistribution into OSPF is configured on the OSPF process. Redistribution is configured on a per-protocol basis. I.e., redistribution can be configured separately for BGP, RIP, static routes, and connected routes.

For example, to redistribute into OSPF all routes learnt by BGP, the commands are:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# redistribute bgp
```

In the **redistribute** command, it is possible to configure metric, metric-type, tag, and route-map. Configuring metric defines the external metric that will be carried in the External LSAs that advertise the redistributed routes. By default, redistributed routes are given a metric of 20, but configuring a metric on the redistribute command can overrule that default.

Configuring the metric-type on the **redistribute** command enables you to set whether the routes are redistributed as Type-1 External LSAs or Type-2 External LSAs. By default, external routes are redistributed as Type-2 External LSAs, but configuring metric-type on the 'redistribute' command can convert that metric type to Type-1 for the redistributed routes.

Configuring a tag will put a value into the tag field within the External LSAs created for the redistributed routes. The purpose of the tag field is described above in the section External LSAs on page 65.

Configuring as route-map on the **redistribute** command can filter which routes are redistributed (this is described below in the section Filtering of redistributed routes on page 103). A routemap can also set the metric, tag, and metric-type of the redistributed routes.

For example, routes learnt via interface VLAN1 can be redistributed as Type-1 External LSAs:

```
awplus# configure terminal
awplus(config)# route-map rmap1 permit 3
awplus(config-route-map)# match interface vlan1
awplus(config-route-map)# set metric-type 1
awplus(config)# router ospf 100
awplus(config-router)# redistribute bgp route-map rmap1
```

Note that if routes are redistributed from an external source, they are always advertised as External LSAs, even if the subnet address of the route falls within the address range covered by a **Network** command configured on OSPF on the ASBR.

Redistribution from OSPF into other routing protocols is configured on those other protocols, and so is not covered in this section.

External route summarization

There is an opportunity, on an ASBR, to summarize routes, and originate External LSAs for the summarized routes, rather than for each individual route that is being imported from the external protocol. If a number of different external routes are being learnt by the AS border router, and all those routes fall within a particular range, the router advertises external routes using summary addresses that have been configured as network/mask pairs.

To configure external route summarization on an AlliedWare Plus switch, the commands are:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# summary-address 202.14.0.0/16
```

The **summary-address** command also takes a couple of other parameters:

Not-advertise – This filters out the routes covered by the summary. No routes within the address range covered by the summary address/mask will be redistributed.

Tag – This specifies a string that will be put into the tag field of the External LSA that advertises the summary route.

When creating these summary addresses, it is important to take care about not creating overlapping summaries, as that would result in an ambiguity about the true path to the addresses that lie within the area of overlap.

Originating default routes

There are two circumstances in which an OSPF router originates a default route.

1. When the router is an Area Border Router connected to a Totally Stubby Area (or a Totally Stubby, Not –So-Stubby Area). In this case, the router generates a Type-3 (Area Summary) LSA, and transmits that LSA into the Totally Stubby Area. The routers in that area end up with a default route as an Inter-Area route.

2. When the router is an autonomous system Border Router. If the ASBR receives a default route from an external source, then, by default, it does not redistribute this route into OSPF. To configure an ASBR to redistribute an externally-learnt default route, use the commands:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# default-information originate
```

The **default-information originate** command can take some parameters – a **metric** and **metric-type** to advertise with the LSA; and a **route-map** that must be matched by at least one route in the route table before the default route will be redistributed.

One other significant parameter is the **always** parameter. If this parameter is specified, then the router will always originate a default route, irrespective of whether it has learnt one from an external source or not.

The following commands make the router immediately become an ASBR (whether or not it is receiving any routes from external sources) generating an External LSA for the default route:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# default-information originate always
```

Route maps and filtering

As has been pointed out previously in this chapter, a cardinal rule of OSPF is that the LSA databases of all the routers in an area must be identical.

Filtering should never be performed at the LSA level. If routers are deciding not to pass on some of the LSAs that they receive, then they will be undermining a fundamental requirement of OSPF.

Hence, the filtering that occurs in OSPF is carried out in the transfer of routes between the IP route table and the OSPF LSA database. If a router is receiving certain routes in LSAs, but you do not want the router to use those routes, then you can configure the router to decide which of the routes resulting from the SPF calculation will and will not be installed into the IP route table. Similarly, if the router has certain non-OSPF routes in its route table, and it is configured to redistribute non-OSPF routes to OSPF, you can configure filters to choose just exactly which routes will be redistributed into OSPF.

Filtering

Filtering should never be performed at the LSA level:

- OSPF filters the transfer of routes between the IP route table and the OSPF LSA database.
- An In distribute list filters routes being brought into the IP route table.

The commands to prevent OSPF from bringing 192.168.1.0/24 into the IP route table are:

```
awplus# configure terminal
awplus(config)# access-list 100 deny ip 192.168.1.0 0.0.0.255 any
awplus(config)# access-list 100 permit ip any any
awplus(config)# router ospf 1
awplus(config-router)# distribute-list 100 in
```

This example uses an ACL in the distribute list. It is also possible to use a route-map.
The commands to achieve the same filtering of 192.168.1.0/24 would be:

```
awplus# configure terminal
awplus(config)# ip prefix-list 100 seq 5 permit 192.168.1.0/24
awplus(config)# route-map 100 deny 10
awplus(config-route-map)# match ip address prefix-list 100
awplus(config)# route-map 100 permit 20
awplus(config)# router ospf 1
awplus(config-router)# distribute-list route-map 100 in
```

Filtering of redistributed routes

There are two ways to configure the filtering of redistributed routes:

1. An **out** distribute list
2. A route-map on the **redistribute** command

For method 1, it is possible to configure a distribute list per external route source.

```
distribute-list <list-name> out {bgp|connected|rip|static}
```

Where `<list-name>` is an ACL.

It is possible to have one filter for the routes redistributed from BGP and a different filter for the routes redistributed from static routes, etc.

For method 2, a routemap is configured on the command:

```
redistribute {<protocol>} route-map
```

Only routes that match permit entries in the route map will be redistributed. Of course, **set** clauses in the route map could also alter properties (metric, tag, and metric type) of the routes being redistributed.

Again, it is possible to have different route-maps for different sources of external routes.

Filtering at the Area Border

We should also recall that Area Border Routers can filter which routes they have learnt from one area they will convert into Summary LSAs to send to another area, as described above in the section Route filtering at area borders on page 103.

Actually, filtering of LSAs does occur, despite the statement above that it is not possible to filter LSAs, the truth is that there is a command which will cause some filtering of LSAs.

It is the command **ip ospf database-filter**, which can be configured on a per-interface basis, using the commands:

```
awplus# configure terminal
awplus(config)# interface vlan1
awplus(config-if# ip ospf database-filter all out
```

This command will prevent the transmitting of any LSAs out through the interface on which it is configured. This, of course, has the potential to violate the requirement that all LSA databases in an area must be identical. So, care must be taken when using this command.

The purpose of the command is to reduce the amount of OSPF traffic that is being transmitted in cases where routers have multiple interfaces connected to the same area. If a router is sending the same set of LSAs out multiple interfaces into the same area, then that does provide redundancy, in that it reduces the odds of any given LSA not getting flooded through the area; but it also causes a lot more packets to be transmitted than is strictly necessary.

This command gives you the option to not transmit LSAs on one or more of these interfaces, and thereby reduce traffic.

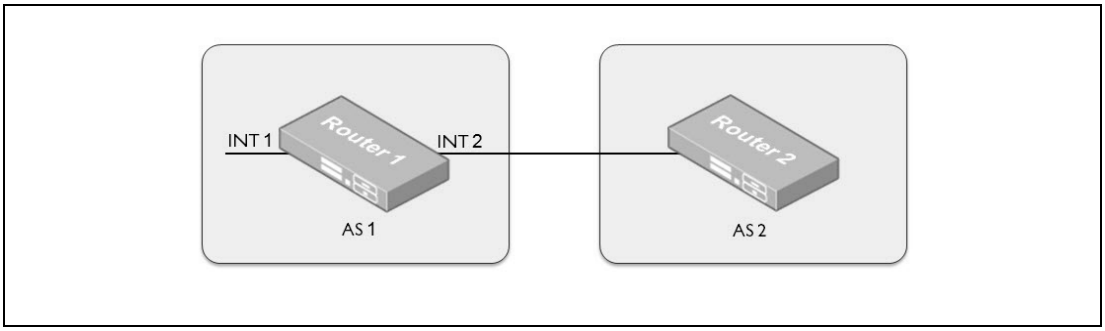
But, this command should NEVER be used on an interface that is the only connection that a router has into a given area. It must only ever be used on interfaces that are just one of the interfaces connecting the router to a given area.

The important point to understand about this command is that it is not aimed at controlling which LSAs reach other routers; it is simply aimed at reducing the number of duplicate copies of the LSAs that a router transmits into an area.

Passive Interfaces

A passive interface in OSPF is one which does not send or receive OSPF routing traffic.

In some cases it is desirable to include a subnet into the OSPF routing process (and Link State Database), without actually running OSPF on the interface of the router connected to that subnet. This is particularly useful for interfaces that are used as BGP peering links or for customer connectivity.



In the figure above, Router 1 is using port INT 2 for BGP peering with Router 2. Let's also assume that AS1 is using OSPF as its IGP. The IP subnet representing the link between the two peering routers, must be available to OSPF, as it contains the next hop value for all BGP routes that are received from AS2.

There are two choices for incorporating this subnet into OSPF:

1. redistribute it as an external into OSPF
2. include the interface of port INT 2 into OSPF

The second approach is typically preferred, as it injects the route as a *native* OSPF route, subject to address summarization at an ABR, while the first approach will inject it as a non-summarizable External. The only problem with the second approach is that it is a security *hole*, as it opens the possibility of routers establishing an OSPF adjacency with Router 2 and exchanging OSPF routes between the two, which is clearly not intended.

This is handled by defining the interface as *passive*, which injects it as an interior OSPF route, while at the same time the interface does **not** run OSPF, and the router is **not** capable of establishing adjacency over it.

In AlliedWare Plus a passive interface is configured using the following commands:

```
awplus# configure terminal
awplus(config)# router ospf 100
awplus(config-router)# passive-interface vlan2
```

OSPF on Demand

Although OSPF is a protocol that aims to keep the chatter to a minimum, it does still require regular transmission of packets – Hello packets are sent every 10 seconds, and a full LSA database update occurs every 30 minutes.

If two OSPF routers are neighbors connected by an on-demand dial-up link, their OSPF packets will keep that link up. Typically, dialup links cost money to be UP, so it is not desirable to have a routing protocol keeping the link up, and wasting money.

Hence, OSPF on demand was developed.

OSPF on demand adds two main factors to the OSPF protocol:

1. A special mode of operation for the routers on each end of an on-demand link.
2. The concept of Do-Not-Age (DNA) LSAs, which even the routers that are not directly connected to the on-demand link need to process.

Let us look at the details of each of these factors.

Special operating mode of the end-points of an on-demand link

In reality, it only really makes sense for point-to-point links to be on-demand links. While a router can have its connection to a Broadcast or NBMA network configured as an on-demand link, it is rarely the case that this will actually be a true on-demand link.

In fact, the Hello suppression mechanism described below is only implemented on Point-to-Point and Point-to-multipoint links. This is because on Broadcast and NBMA networks, it is necessary to continue exchanging Hellos, so that routers can keep track of which neighbors exist in that network.

The routers at each end of an on-demand link should set the **DC** bit in the Hello packets that they send. If both ends of the link see that the other is setting the DC bit in its Hello packets, then that constitutes a successful agreement to treat that link as an on-demand link.

Once they have agreed to treat the link as an on-demand link, then:

- They will only send Hello packets across the link up until they have completed the Database Exchange phase of their neighbor relationship negotiation. Thereafter, they do not send each other any additional Hello packets. The determination of whether the neighbor relationship is down is not governed by the expiration of the Dead timer. Instead, the router will use other mechanisms to decide if the relationship has gone down. For example, if the router attempts to bring the physical link up, but fails, then that will probably be grounds for deciding that the OSPF neighbor relationship is down.
- They are much more selective about which LSAs are transmitted across the on-demand link. If a router at one end of an on-demand link receives an LSA, then it will only transmit that LSA across the on-demand link if:
  - This is an LSA that the router did not previously have in its LSA database.

  or
  - The LSA was already in the LSA database, but the new copy is a true change of the LSA, rather than just a periodic refresh of the LSA.
- When the routers send an LSA across the on-demand link, they set a Do-not-Age bit in the LSA. This indicates to the router on the other end of the demand link, AND all other routers that it forwards that LSA to, that they should not increment the age of this LSA. So, the LSA will remain in those routers' LSA databases indefinitely, until they are explicitly told to remove it – as described next.

Do-not-Age (DNA) LSAs

The highest bit in an LSA's Age field has been designated the Do-not-Age bit. If this bit is set in an LSA's age, then any router receiving the LSA should not change the age of the LSA, but just let the LSA reside in its LSA database, with the same age as it had when it first arrived (an 'eternal youth').

The reason for this, of course, is that the Do-not-Age LSA is one that has been sent across an on-demand circuit. No further copies of that LSA will be sent unless it actually changes. So, if no changes occur to the LSA, then no refresh of the LSA will ever be received. If the LSA were being aged, it would age out of routers' databases after the first hour of it not changing.

These DNA LSAs can be removed from a router's LSA database if:

- The router receives a new copy of the LSA with the Age set to MaxAge.

or

- The originator of the router has actually been unreachable (i.e. there is no route to that router's IP address) for at least MaxAge seconds.

Graceful Restart

The job of a routing protocol is to maintain the contents of a router's routing table. The routing protocol is not involved in packet forwarding decisions, it just populates the route table with the entries upon which packet forwarding decisions are based.

So, if the routing protocol is not active for a brief period, then it is quite possible for the router to keep on forwarding packets, using the entries with which the routing protocol had populated the routing table.

Moreover, if it is given an appropriate warning, there is actually no reason why a neighbor router cannot continue using the routes it learnt from a router whose routing protocol is briefly inactive.

By default, as soon as an OSPF router detects that a neighbor's OSPF process has gone down (when the dead timer for that neighbor expires), then, if it is the Designated Router on the subnet it shares with the now-dead router, it will re-originate the Network LSA for that subnet. The now-dead router will not be in the list of routers in the new version of the Network LSA. As a result, all the routers in the network will redo their SPF calculations, and thereby remove any routes that pass through the router whose OSPF process has gone inactive (because no subnets in the network will now be connected to this router).

However, as noted above, it does not have to be this way. If the router knows that its OSPF process is going down for just a brief period, then it could say to its neighbors "*I am going quiet for a while, but please just carry on as though I was still active. I will join back into the conversation again quite soon*"

Because it is not the routing process itself that is going down on this router, it will still be able to forward packets that are sent to it. So the neighbors can still continue to use whatever routes they have which pass though the router in question.

The idea just described above is called Graceful Restart. The mechanics of Graceful Restart are defined in RFC3623. This process has been implemented in AlliedWare Plus, and in other vendors' routers.

By implementing Graceful Restart, AlliedWare Plus routers can reboot the OSPF process without having to disturb the routing tables in the network. This is made use of most particularly in VCStack Master Failover. Because the OSPF process running on a VCStack backup member is running in a 'quiet' mode, then when the backup member becomes Master, it must reboot its OSPF process, to put it into a fully active mode. So, as the backup member detects that it needs to become Master, it uses Graceful Restart to inform neighbors that its OSPF process is about to reboot. Then, when its OSPF process has started up again in fully active mode, it resumes the OSPF communication with the neighboring routers.

The requirement on neighbor routers

Graceful Restart is a cooperative activity between the router whose OSPF process is about to restart (referred to as the Restarting Router), and its full neighbors. The full neighbors must be running OSPF implementations that are Graceful Restart aware. This is because, when the Restarting Router goes into Graceful Restart, the full neighbors must go into a Graceful Restart Helper mode.

If they are not able to go into this mode, then Graceful Restart cannot work. Neighbors that are not Graceful Restart aware will simply o perate in the traditional manner – e.g. those that are designated routers will originate a new copy if the relevant Network LSA that omits the Restarting Router, when the dead timer for their connection to Restarting Router expires.

The operation of Graceful Restart

The operation of Graceful Restart is surprisingly simple.

First, the Restarting Router sends out a special LSA called a **Grace** LSA. This is to inform the neighbors that the Restarting Router is about to go into Graceful Restart. The Grace LSA specifies a Grace Period. The Grace Period is the maximum period of time for which the router expects its OSPF process to be out of action. The helper routers know that if they do not hear from the Restarting Router again by the time the Grace Period has expired, that they should act as though the OSPF process on the Restarting Router has been terminated. That is, they should omit the Restarting Router from any Network LSAs they originate.

Upon receiving the Grace LSA, the neighbors acknowledge it, and then go into Helper mode, if they are capable of doing so. We will look at the details of Helper mode below in the section Operation of Helper Mode on page 112.

You should take note that, even though the act of neighbors going into Helper mode is vital to the success of Graceful Restart, the Restarting Router will not necessarily know if its neighbors are willing and able to go into Helper mode at the time the Restarting Router goes into its restart. Even if a neighbor acknowledges a Grace LSA, that is not an indicator that it is going to go into Helper mode. Neighbors can simply acknowledge (Ack) the Grace LSA as a matter of good form. If a neighbor steadfastly refuses to acknowledge the Grace LSA, even after multiple retransmits of the Grace LSA, then that is, of course, a clear indicator that the neighbor will not be doing any helping.

But irrespective of whether the Restarting Routers knows some full neighbors will not be helping (as they did not Ack the Grace LSA), or it is just hoping they all will help (if they did all Ack the Grace LSA), the Restarting Router will just go ahead and restart anyway. As we will see below, the rules governing how the Restarting Router should behave after its restart mean that even if some full neighbors did not help, the situation can be recovered relatively tidily.

A few other points to note about Grace LSAs are:

- They have link-local scope. This means that a router which receives one should not forward it on to any of its neighbors. This is because it is only the Restarting Router's immediate neighbors that need to know that the Restarting Router is doing a Graceful Restart. Those immediate neighbors need to know, so they can take up the opportunity to go into Helper Mode. But any routers more than one hop away from the Restarting Router should be quite oblivious to the fact that it is restarting.
- As well as specifying the length of the Grace Period, the Grace LSA should specify the reason for the restart, and the IP address on the interface via which the Restarting Router is transmitting the Grace LSA.
- The reason for restart can have the following values:

  0 - Unknown

  1 - Software Restart

  2 - Software Reload/upgrade

  3 - Switching over to a backup processor

Having sent out the Grace LSA, and having received Acks from all the full neighbors (or, possibly, having decided that one or more full neighbors are simply not going to Ack the Grace LSA despite multiple retransmissions), the Restarting Router stores away, in some safe piece of memory, the fact that it is doing a graceful OSPF restart, and the length of the Grace period.

At this point, the Restarting Router is deemed to have entered the Graceful Restart state. It continues to be in this state until it specifically exits the state. In particular, the act of the OSPF process coming up again after its reboot does **not** exit the router from Graceful Restart state. The router will continue to be in Graceful Restart state for some time after the OSPF process has come up again. The criteria that must be satisfied before the Restarting Router can exit Graceful Restart state a described below in the section Exiting Graceful Restart on page 110.

Now that it has entered Graceful Restart state, the Restarting Router performs the reboot of its OSPF process.

Actions when the OSPF process comes up again

When the OSPF process comes up again, it operates almost exactly as it would if it was normally starting from cold. It establishes neighbor relationships with its neighbors by going through the normal neighbor relationship negotiation process. It rebuilds its LSA database by learning LSAs from its neighbors. It performs an SPF calculation from the contents of the LSA database.

But, there are some ways in which the startup in Graceful Restart differs from a normal cold start of the OSPF process:

- It does not originate any of the LSAs that it would normally originate - its Router LSA, Network LSAs (if it is a designated router), Summary LSAs (if it is an ABR), ASExternal LSAs (if it is an ASBR) etc. The reason that it does not originate its LSAs while it is still in Graceful Restart state is that it must examine the copies of its own self-originated LSAs that it gets back from its neighbors. If it went and re-originated its own LSAs immediately after restart, then the neighbors would install these fresh copies of those LSAs, rather than sending back the copies they had from before the restart. The examination that the Restarting Router performs on its own LSAs that are transmitted back to it by its neighbors is described below in the section Exiting Graceful Restart on page110.

- It does not make any changes to its routing table. Even though the Restarting Router performs an SPF calculation after restart, the routes resulting from that calculation are not installed into the routing table at that time. The SPF calculation is performed solely to provide the router with an OSPF route table that can be used for establishing any virtual links it might be configured with.

- If the router detects that it must have been the DR for a subnet before restart, then it will again resume the DR role. It would detect that it had been DR by seeing its own Router ID advertised as the DR in the Hello packets received from neighbors who are within the subnet in question.

Exiting Graceful Restart

After the OSPF process has come up again, the Restarting Router is on the lookout for when to officially exit Graceful Restart state. There are three different sets of criteria that will trigger the Restarting Router to exit Graceful Restart.

1. The ideal situation is that all the full neighbors successfully entered Helper mode, and no topology changes occurred while the Restarting Router was restarting. In this case, the Restarting Router should successfully re-establish neighbor relationships with all the neighbors that it had before the restart, and find that the information it receives from the neighbor is consistent with no topology changes having occurred while it was restarting.

    - This is where the process of examining the copies of its own LSAs that it gets back from its full neighbors comes in. Firstly, when it receives back a copy of its own Router LSA, it will see a list of the neighbors it knew of, prior to restart, in each area it borders. This will enable it to know the list of routers it must now re-establish neighbor relationships with. Until it has established neighbor relationships with all the routers that it had as neighbors prior to restart, it knows that the task of getting back to that pre-restart state is not complete.

- The contents of these LSAs will also enable it to detect if any topology changes have occurred, as will be discussed in item (2) below.
- If it does successfully re-establish all the neighbor relationships it had prior to restart, and detects no topology changes, then it will exit Graceful Restart, and perform the actions described below in the section Actions upon exiting Graceful Restart on page 110.

2. If the Restarting Router does detect that any topology changes occurred while it was restarting, then it will exit Graceful Restart state immediately, irrespective of how it is progressing on the task of re-establishing relationships with all its old neighbors.

   The two main ways it can detect that a topology change has occurred are:

- It sees in its own pre-restart Router LSA, that it gets sent back to it, that it was a neighbor of some router Y. But when it receives a fresh Router LSA for router Y, from some other neighbor, this Router LSA does not include the Restarting Router in the list of neighbors. That is - the Restarting Router sees that it used to be neighbor of router Y, before restart, but finds that router Y has since decided that they are no longer neighbors. This means that, at least as far as router Y is concerned, the network topology has changed.
- It forms a full neighbor relationship with a router Y, and finds that Y does not even send the Restarting Router's own pre-restart Router LSA back to it. This indicates that router Y had decided that the Restarting Router was dead, and had flushed the Restarting Router's Router LSA from its LSA database.

  The reason that a neighbor router had removed the Restarting Router from its neighbor list is that the neighbor had never gone into Helper mode, or had exited helper mode early (i.e. before the Restarting Router has announced that Graceful Restart is over).

  A neighbor would fail to go into Helper mode if it does not support Helper mode, or it failed to receive the Grace LSA or it was itself already engaged in gracefully restarting.

  The reasons a neighbor would take an early exit from Helper mode are discussed below in the section Operation of Helper Mode on page 112.

  Whichever the case, the fact is that the neighbor will have treated the Restarting Router as dead, and potentially originated Network LSAs that omit the Restarting Router. In other words, the whole point of Graceful Restart has been lost, so the Restarting Router might as well exit Graceful Restart forthwith, and get on with the business of acting a though it was doing a normal cold start of OSPF.

3. The Grace Period expires before the Restarting Router has completed forming the necessary relationships with all its neighbors.

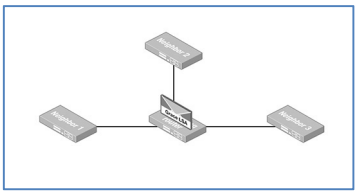
Actions upon exiting Graceful Restart

When the Restarting Router exits Graceful Restart, it does the following:

1. Sends out a Grace LSA to inform the neighbors that it has exited Graceful Restart, and that they can now exit Helper mode. This is a Grace LSA with the age set to MaxAge.
2. Re-originates all its own LSAs.
3. Deletes any copies of its own LSAs that it had got from its neighbors.
4. Performs an SPF calculation, and does install the resulting routes into the route table.
5. Clears out from the route table any routes that are no longer active.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/nmH30NyqEXg



Operation of Helper Mode

The first point that should be emphasized here is that only FULL neighbors of the Restarting Router can go into Helper Mode. This is because it is only full neighbors that exchange LSAs with the Restarting Router, so they are the only neighbors that would heed Grace LSAs sent by the Restarting Router. Moreover, only full neighbors of the Restarting Router originate LSAs that have links to the Restarting Router; so it is only the full neighbors that need to agree not to re-originate new versions of those LSAs that would omit the links to the Restarting Router.

When a full neighbor of the Restarting Router receives a Grace LSA from the Restarting Router, the neighbor decides if it is OK for it to enter Helper mode. The reasons why it might enter Helper mode are:

- The state of its neighbor relationship with the Restarting Router is not currently at Full.
- There are currently some LSUs (other than just periodic refreshes) that it has sent to the Restarting Router, which the Restarting Router has not yet acknowledged. This would mean that its LSA database is currently out of sync with that of the Restarting Router.
- The neighbor is, itself, currently in the process of going through a Graceful Restart.

If the neighbor does enter Helper Mode, then it will behave as though the Restarting Router's OSPF process is active, regardless of whether the dead timer on its neighbor relationship with the Restarting Router expires. Any LSAs it originates that advertise links to the Restarting Router will continue to advertise those links. Hence, those links will continue to be present in other routers' shortest-path trees, and therefore routes involving those links will continue to populate other routers' route tables.

But, a neighbor that is in Helper Mode will immediately drop out of Helper Mode if it receives, from anywhere, any new or altered LSA that it would also pass on to the Restarting Router. That is, if it realizes that there has been a change anywhere in the network that would affect the Restarting Router's LSA database. Because the state of the Restarting Router's route table, frozen in its pre-restart state, is then probably out of sync with the changed state of the network, a seamless Graceful Restart is no longer possible. The neighbor drops out of Helper Mode so that it can stop routing via the Restarting Router and also tell other routers to also stop routing via the Restarting Router.

Upon exiting Helper Mode, the neighbor will no longer continue to pretend that the Restarting Router's OSPF process is still alive when the dead timer on its adjacency with the Restart Router expires. Instead, when that dead timer expires, the neighbor will re-originate its LSAs, and omit any links to the Restarting Router that might have been in those LSAs. Also, it will cause a re-election of the Designated Router for the subnet that the Restarting Router belongs to, so that if the Restarting Router had been the Designated Router on that subnet, it will be no longer.

This all has the effect that when the Restarting Router's OSPF process comes up again, it will detect that a topology change has occurred, as described in part (ii) of the section Exiting Graceful Restart on page 110. The fact that one or more neighbors is now originating LSAs that do not contain a link to the Restarting Router is exactly the sort of inconsistency the Restarting Router detects that a topology change has occurred.

If no topology change occurs during the Graceful Restart, then the helping neighbors will remain in Helper Mode until they receive the MaxAge Grace LSA from the Restarting Router, which is the mechanism that the Restarting Router uses to indicate that Graceful Restart has finished.

If, for any reason, a helping neighbor has not received the MaxAge Grace LSA by the time that Grace Period has expired, then it will just exit Helper Mode at that moment.

Monitoring and Debugging Commands

AlliedWare Plus provides a number of commands that enable you to probe the state of OSPF on a router.

Interfaces

The state of all active or passive OSPF interfaces is output by the command: show ip ospf interface .Here is some example output for this command, followed by an explanation of the less obvious items:

```
awplus#show ip ospf interface
vlan20 is up, line protocol is up
  Internet Address 89.219.78.4/16, Area 0.0.0.1, MTU 1500
  Interface state Backup
  Process ID 0, Router ID 183.72.173.10, Network Type BROADCAST, Cost: 1
  Transmit Delay is 1 sec, State Backup, Priority 1
  Designated Router (ID) 150.87.20.1, Interface Address 89.219.78.3
  Backup Designated Router (ID) 183.72.173.10, Interface Address 89.219.78.4
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:10
  Neighbor Count is 1, Adjacent neighbor count is 1
  Crypt Sequence Number is 507
  Hello received 100 sent 102, DD received 4 sent 3
  LS-Req received 1 sent 1, LS-Upd received 4 sent 4
  LS-Ack received 4 sent 3, Discarded 0
```

Much of the information in this output above is easy to interpret. Some of the less obvious items are:

Item	Description
Interface state Backup	The BDR switch on the network attached to interface VLAN20.
Cost: 1	The OSPF cost for this interface, the cost value that is used in the SPF calculation.
Designated Router (ID)	The Router ID of the designated router on the network attached to this interface.
Interface Address	Interface IP address via which the DR attaches to this network.
Hello due in 00:00:10	The switch will next send a Hello packet out this interface in10 sec.
Neighbor Count	The total number of neighbors on the network connected to this interface.
Adjacent neighbor count	The number of fully adjacent neighbors on the network access via this interface.
Crypt Sequence Number	The value that is used for MD5 encryption if MD5 authentication is in use.

Neighbors

You can see the neighbors that the switch currently has neighbor relationships with. The command is: show ip ospf neighbor

```
awplus#show ip ospf neighbor
OSPF process 0:
Neighbor ID     Pri   State           Dead Time   Address         Interface
150.87.20.1       1   Full/DR         00:00:35    89.219.78.3     vlan20
183.72.173.11     1   Full/Backup     00:00:36    183.72.173.11   vlan40
```

Item	Description
Neighbor ID	The Router ID of the neighbor
Priority	The priority value in the Hello packets being sent by the neighbor.
Designated Router (ID)	The Router ID of the designated router on the network attached to this interface.
State	The current state of the neighbor relationship and whether the neighbor is a DR, Backup DR, or DR-other if the network shared with the neighbor is a broadcast or NBMA network.
Dead Time	The number of seconds until the dead time expires. It is refreshed back up to the full dead time value whenever a Hello is received from the neighbor in question.
Address	The source IP address of the Hello packets the neighbor is sending.
Interface	The interface the neighbor is accessed on.

OSPF route table

The routes in the OSPF route table are:

- routes learnt by OSPF from other routers
- connected routes on OSPF interfaces
- routes redistributed into OSPF

The command show ip ospf route shows the best route, in the OSPF route table, to given destinations. These are routes that OSPF puts up as candidates for inclusion into the switch's main RIB.

```
awplus#show ip ospf route
E2 52.68.0.0/16 [1/20] via 89.219.78.3, vlan20
C  89.219.0.0/16 [1] is directly connected, vlan20, Area 0.0.0.1
E2 150.87.20.0/24 [1/20] via 89.219.78.3, vlan20
IA 173.54.182.0/24 [2] via 183.72.173.11, vlan40, Area 0.0.0.0
C  183.72.173.0/24 [1] is directly connected, vlan40, Area 0.0.0.0
```

The entries in the first column are the route type:

E2 = Type-2 external OSPF route

C = connected route

IA = inter-area OSPF route

The values in square brackets [ ] are the metrics of the routes. For the case of external routes, both the internal and external metrics are listed. So, for example, when an external route is shown with metrics [1/20], that means that metric of the internal route to the ASBR is 1, and the external metric with which the route was redistributed into OSPF is 20.

Border routers

You can see a list of border routers in the network, both AS Border Routers and Area Border Routers, with the command: show ip ospf border-routers

```
awplus#show ip ospf border-routers
OSPF process 0 internal Routing Table
Codes: i - Intra-area route, I - Inter-area route

i 183.72.173.11 [1] via 183.72.173.11, vlan40, ABR, Area 0.0.0.0
i 150.87.20.1 [1] via 89.219.78.3, vlan20, ASBR, Area 0.0.0.1
```

LSA database

You can see the whole LSA database of the switch with the command:
show ip ospf database

```
awplus#show ip ospf database

Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
183.72.173.10   183.72.173.10    834 0x80000004 0xf27d 1
183.72.173.11   183.72.173.11    807 0x80000003 0xf27b 1

Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
183.72.173.10   183.72.173.10    834 0x80000001 0xb7c2

Summary Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Route
89.219.0.0      183.72.173.10   1025 0x80000001 0xe985 89.219.0.0/16
173.54.182.0    183.72.173.11    807 0x80000001 0x8583 173.54.182.0/24
                ASBR-Summary Link States (Area 0.0.0.0)

[Continued next page...]
```

```
Link ID         ADV Router      Age  Seq#       CkSum
150.87.20.1     183.72.173.10   1020 0x80000002 0x0d92


Router Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
150.87.20.1     150.87.20.1     1041 0x80000004 0x0932 1
183.72.173.10   183.72.173.10   1025 0x80000004 0x0bc6 1

Net Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum
89.219.78.3     150.87.20.1     1041 0x80000001 0xd342

Summary Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Route
173.54.182.0    183.72.173.10    806 0x80000001 0x9573 173.54.182.0/24
183.72.173.0    183.72.173.10   1025 0x80000001 0x9363 183.72.173.0/24

AS External Link States

Link ID         ADV Router      Age  Seq#       CkSum  Route              Tag
52.68.0.0       150.87.20.1       58 0x80000002 0x370b E2 52.68.0.0/16    0
150.87.20.0     150.87.20.1     1256 0x80000001 0x7841 E2 150.87.20.0/24  0
```

You can see that if the router is (as in this case) an Area Border Router, then the LSA databases for the areas it is connected to are each listed separately. This output is just showing the content of the LSA header for each LSA. To see more detail you need to list specific sets of LSAs.

You can list LSAs by:

- Type – `show ip ospf database router`, `show ip ospf database network`, `show ip ospf database summary`, etc.
- Advertising router – `show ip ospf router adv-router <Router ID>`
- Self-originated – all the LSAs originated by the router you are entering the command into

When you list the database in these ways, you get real detail of the LSAs.

For example:

```
awplus# show ip ospf database router
Router Link States (Area 0.0.0.0)
  LS age: 838
  Options: 0x2 (-|-|-|-|-|-|E|-)
  Flags: 0x1 : ABR
  LS Type: router-LSA
  Link State ID: 183.72.173.10
  Advertising Router: 183.72.173.10
  LS Seq Number: 80000004
  Checksum: 0xf27d
  Length: 36
   Number of Links: 1
Link connected to: a Transit Network
     (Link ID) Designated Router address: 183.72.173.10
     (Link Data) Router Interface address: 183.72.173.10
      Number of TOS metrics: 0
       TOS 0 Metric: 1
```

```
awplus#show ip ospf database network
Net Link States (Area 0.0.0.0)
  LS age: 844
  Options: 0x2 (-|-|-|-|-|-|E|-)
  LS Type: network-LSA
  Link State ID: 183.72.173.10 (address of Designated Router)
  Advertising Router: 183.72.173.10
  LS Seq Number: 80000001
  Checksum: 0xb7c2
  Length: 32
  Network Mask: /24
        Attached Router: 183.72.173.10
          Attached Router: 183.72.173.11
```

```
awplus#show ip ospf database external
 AS External Link States

  LS age: 106
  Options: 0x2 (-|-|-|-|-|-|E|-)
  LS Type: AS-external-LSA
  Link State ID: 52.68.0.0 (External Network Number)
  Advertising Router: 150.87.20.1
  LS Seq Number: 80000002
  Checksum: 0x370b
  Length: 36
  Network Mask: /16
        Metric Type: 2 (Larger than any link state path)
        TOS: 0
        Metric: 20
        Forward Address: 0.0.0.0
        External Route Tag: 0
```

Debugging

OSPF packet debug

To get a trace of the OSPF packets being sent and received by the switch, use the command: debug ospf packet.

```
awplus#debug ospf packet
awplus#term mon
% Warning: Console logging enabled
awplus#18:14:57 awplus OSPF[1217]: SEND[LS-Upd]: 1 LSAs to destination 224.0.0.5
18:14:57 awplus OSPF[1217]: SEND[LS-Upd]: To 224.0.0.5 via vlan40:183.72.173.10, length 56
18:14:58 awplus OSPF[1217]: RECV[LS-Ack]: From 183.72.173.11 via vlan40:183.72.1
73.10 (183.72.173.11 -> 224.0.0.5)
18:14:58 awplus OSPF[1217]: SEND[Hello]: To 224.0.0.5 via vlan40:183.72.173.10,
length 48
18:15:01 awplus OSPF[1217]: RECV[Hello]: From 150.87.20.1 via vlan20:89.219.78.4
 (89.219.78.3 -> 224.0.0.5)
18:15:01 awplus OSPF[1217]: RECV[Hello]: From 183.72.173.11 via vlan40:183.72.17
3.10 (183.72.173.11 -> 224.0.0.5)
term 18:15:04 awplus OSPF[1217]: SEND[Hello]: To 224.0.0.5 via vlan20:89.219.78.
```

LSA debugging

The command to show a trace of the LSAs being sent and received is: debug ospf lsa

```
awplus#debug ospf lsa
18:16:08 awplus OSPF[1217]: LSA[0.0.0.1:Type1:183.72.173.10:(self)]: Flooding vi
a interface[vlan20:89.219.78.4]
18:16:08 awplus OSPF[1217]: LSA[0.0.0.1:Type1:183.72.173.10:(self)]: Flooding to
 neighbor[150.87.20.1]
18:16:08 awplus OSPF[1217]: LSA[0.0.0.1:Type1:183.72.173.10:(self)]: Added to ne
ighbor[150.87.20.1]'s retransmit-list
18:16:08 awplus OSPF[1217]: LSA[0.0.0.1:Type1:183.72.173.10:(self)]: Sending upd
ate to interface[vlan20:89.219.78.4]
18:16:08 awplus OSPF[1217]: LSA[0.0.0.1:Type1:183.72.173.10:(self)]: router-LSA
refreshed
18:16:08 awplus OSPF[1217]:   LSA Header
18:16:08 awplus OSPF[1217]:     LS age 0
18:16:08 awplus OSPF[1217]:     Options 0x2
18:16:08 awplus OSPF[1217]:     LS type 1 (router-LSA)
18:16:08 awplus OSPF[1217]:     Link State ID 183.72.173.10
18:16:08 awplus OSPF[1217]:     Advertising Router 183.72.173.10
18:16:08 awplus OSPF[1217]:     LS sequence number 0x80000005
18:16:08 awplus OSPF[1217]:     LS checksum 0x6cb
18:16:08 awplus OSPF[1217]:     length 36
18:16:09 awplus OSPF[1217]:  NSM Message Header
18:16:09 awplus OSPF[1217]:  VR ID: 0
18:16:09 awplus OSPF[1217]:  VRF ID: 0
18:16:09 awplus OSPF[1217]:  Message type: Router ID update (36)
18:16:09 awplus OSPF[1217]:  Message length: 21
18:16:09 awplus OSPF[1217]:  Message ID: 0x00000000
[Contined next page…]
```

```
18:16:11 awplus NSM[1130]: Port up notification received for port2.0.22
18:16:11 awplus OSPF[1217]: NSM Message Header
18:16:11 awplus OSPF[1217]:  VR ID: 0
18:16:11 awplus OSPF[1217]:  VRF ID: 0
18:16:11 awplus OSPF[1217]:  Message type: Link Up (29)
18:16:11 awplus OSPF[1217]:  Message length: 80
18:16:11 awplus OSPF[1217]:  Message ID: 0x00000000
18:16:11 awplus OSPF[1217]: NSM Interface
18:16:11 awplus OSPF[1217]:  Interface index: 340
18:16:11 awplus OSPF[1217]:  Name: vlan40
18:16:11 awplus OSPF[1217]:  Flags: 4163
18:16:11 awplus NSM[1130]: Port up notification received for vlan40
18:16:11 awplus OSPF[1217]:  Status: 0x%016llx
18:16:11 awplus OSPF[1217]:  Bandwidth: 125000000.000000
18:16:11 awplus OSPF[1217]: LSA[0.0.0.0:Type3:89.219.0.0:(self)]: Install summary-LSA
18:16:11 awplus OSPF[1217]: LSA[0.0.0.0:Type3:89.219.0.0:(self)]: LSA refresh scheduled at
LS age 2435
18:16:11 awplus OSPF[1217]: LSA[0.0.0.0:Type3:89.219.0.0:(self)]: Flooding via
interface[vlan40:183.72.173.10]
18:16:11 awplus OSPF[1217]: LSA[0.0.0.0:Type3:89.219.0.0:(self)]: summary-LSA(0x10107b00)
originated
```

Neighbor finite state machine debugging

To show the negotiation with neighbors and the regular checks on the state of neighbor relationships, use the command: **debug ospf nfsm**

For example, the debug below shows a set of neighbor state changes with a neighbor `183.72.173.11` on VLAN40. The relationship goes through the following states: 2-Way, Ex-Start, Exchange, Loading, and Full.

```
awplus#debug ospf nfsm
18:18:40 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: 2-Way (AdjOK?)
18:18:40 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Status change 2-Way ->
ExStart
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: ExStart
(HelloReceived)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Neighbor is Opaque-
capable
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: ExStart
(NegotiationDone)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Status change ExStart
-> Exchange
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Exchange
(HelloReceived)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Exchange
(HelloReceived)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Exchange
(ExchangeDone)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Status change Exchange
-> Loading
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Loading
(HelloReceived)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Loading
(HelloReceived)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: nfsm_ignore called
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Loading (LoadingDone)
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Status change Loading
-> Full
18:18:41 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Full (Hello Received)
18:18:42 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Full (Hello Received)
18:18:42 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Full (Hello Received)
18:18:42 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: nfsm_ignore called
18:18:42 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Full (2 WayReceived)
18:18:42 awplus OSPF[1217]: NFSM[vlan40:183.72.173.10-183.72.173.11]: Full (AdjOK?)
```

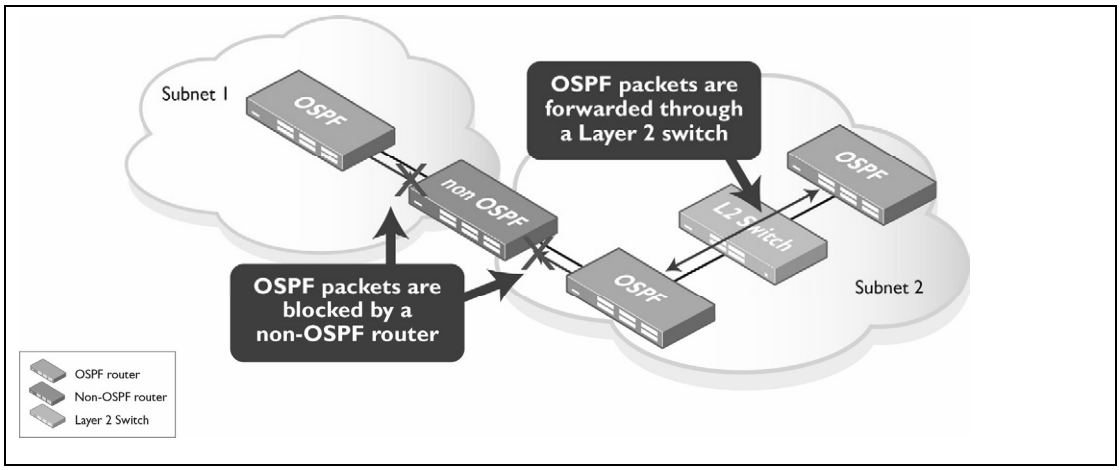
OSPF Packet Types

Being a process for communicating information between routers, OSPF needs to send packets to carry out this communication. In typical OSPF fashion, the implementation of the packets is efficient and clean. There are only five types of OSPF packets, and each packet type has a clear role, and precise definitions of what it carries and when it is sent.

Note – OSPF packets are never forwarded at Layer 3. This means that any given OSPF packet is only ever sent from a router to a neighbor on a link, they are never routed on to other routers. If there are Layer 2 switches between a pair of OSPF neighbors, then those L2 switches will forward the OSPF packets, but once the packet arrives at the OSPF neighbor, its journey ends – it is processed by the receiving neighbor, and never forwarded on.

Initially, you might think this is at odds with the fact that LSAs are flooded through an OSPF area. But, the distinction to keep clearly in mind is that it is the LSAs that are flooded, not the packets carrying the LSAs. When a router receives a packet containing a set of LSAs, it processes the content of the packet, and decides which, if any, of the LSAs in the packet it needs to forward on to its neighbors. When it does forward some or all of the LSAs on to neighbors, it creates new packets in which to transmit those LSAs. It never just simply forwards on the packet in which it received the LSAs.

Also, it is important to have clearly in mind that there cannot be Layer 3 routers located between OSPF neighbors, for the very fact that OSPF packets are never Layer 3 forwarded. Routers simply cannot become neighbors if there is a device between them that cannot forward their packets to each other.



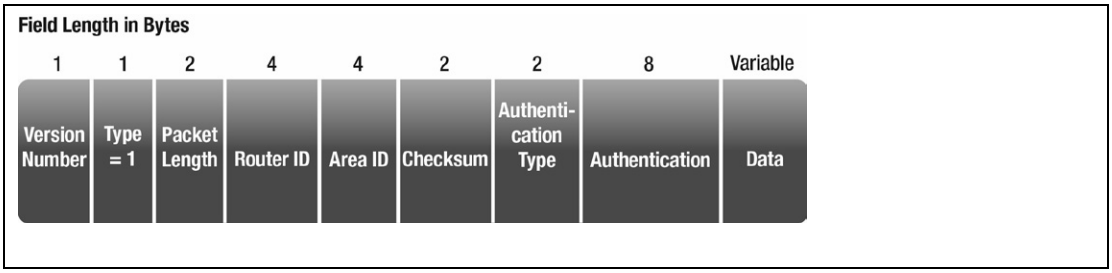
The five packet types are as follows:

1. Hello
2. Database Description
3. Link-State Request
4. Link-State Update
5. Link-State Acknowledgment

Let's look at the OSPF packet header, which all five packet types share.

OSPF packet header

Every OSPF packet starts with a standard 24 byte header. This header contains all the information necessary to determine whether the packet should be accepted for further processing.



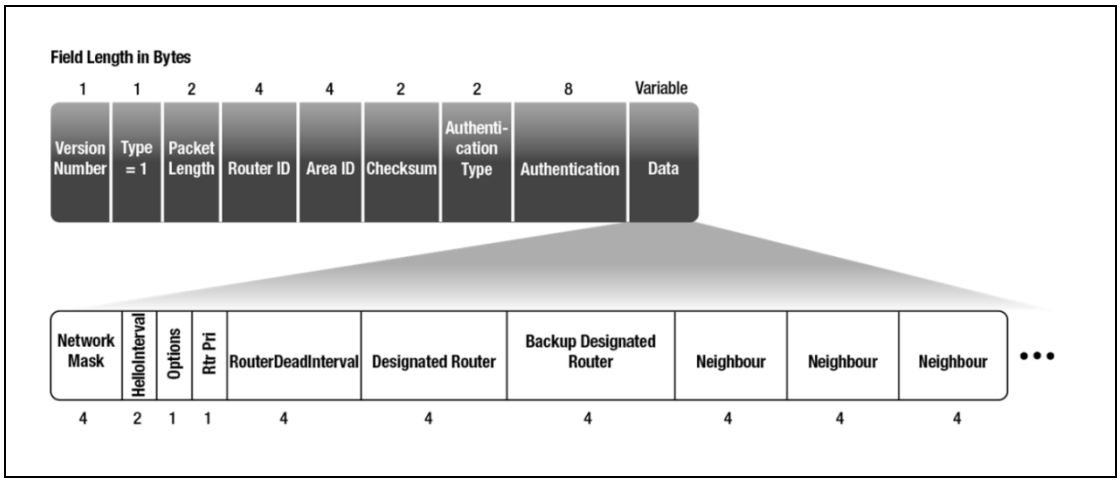
The table following describes each of the fields in the 24 byte OSPF header packet:

Packet Field	Description
Version number	Identifies the OSPF version used
Type	Identifies the OSPF packet type as one of the following: **Hello** - establishes and maintains neighbor relationships. **Database description** - describes the content of the topological database. These packets are transmitted during the Exchange phase of neighbor negotiation. **Link-state request** – requests an LSA during the Loading phase of neighbor negotiation. **Link-state update** – transmits one or more LSAs. **Link-state acknowledgment** - acknowledges link-state Update packets.
Packet Length	Specifies the packet length, including the OSPF header, in bytes.
Router ID	The Router ID of the sender of the packet.
Area ID	Identifies the area to which the packet belongs. All OSPF packets are associated with a single area. Packets travelling over a virtual link are labeled with the backbone Area ID of 0.0.0.0.
Checksum	Checks the entire packet contents for any damage suffered in transit.

Packet Field	Description
Authentication Type	Contains the authentication type – none, plaintext, MD5
Authentication	Contains authentication information

Type 1 - Hello packet

Hello packets are sent periodically on all interfaces (including virtual links) in order to establish and maintain neighbor relationships.



Network mask

The network mask on the interface that the Hello was transmitted from.

HelloInterval

The number of seconds between this router's Hello packets.

Options

The optional capabilities supported by the router, these are described above in the section Options on page 124.

Rtr Pri

This router's Router Priority. This is used in the election of Designated Router on multi-access networks.

RouterDeadInterval

If neighbors do not receive Hellos from this router for RouterDeadInterval seconds, then they will declare their OSPF neighbor relationship with this router to be down.

Designated Router

The identity of the Designated Router for this network, as far as the sending router is concerned. The identifier in this field is not the Router ID of the DR, but is the IP address on the interface via with the DR connects to this network. Set to 0.0.0.0 if there is no Designated Router.

Backup Designated Router

The identity of the Backup Designated Router for this network, as far as the sending router is concerned. The identifier in this field is not the Router ID of the BDR, but is the IP address on the interface via with the BDR connects to this network. Set to 0.0.0.0 if there is no BDR.

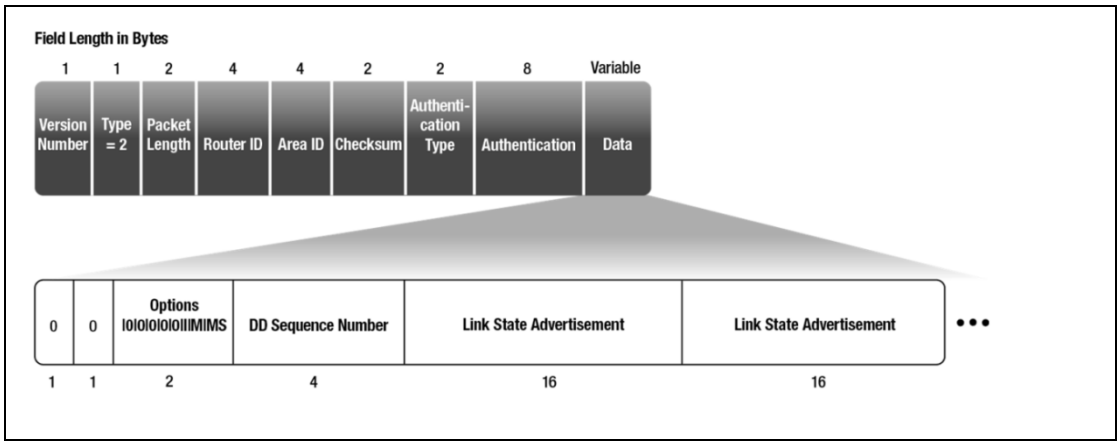
Neighbors

The Router IDs of each router from which the sender of the Hello has received valid Hello packets within the last RouterDeadInterval seconds.

Type 2 - Database Description packet

Database Description packets are transmitted during the exchange phase of neighbor relationship negotiation, as described above in the sections Ex-Start and Exchange on page 44.

The format of the Database Description packet is very similar to both the Link State Request and Link State Acknowledgment packets.



Options

The optional capabilities supported by the router:

I-bit

The Init bit. When set to 1, this packet is the first in the sequence of Database Description Packets.

M-bit

The More bit. When set to 1, it indicates that more Database Description Packets are to follow.

MS-bit

The Master/Slave bit. When set to 1, it indicates that the router is the Master during the Database Exchange process. Otherwise, the router is the Slave.

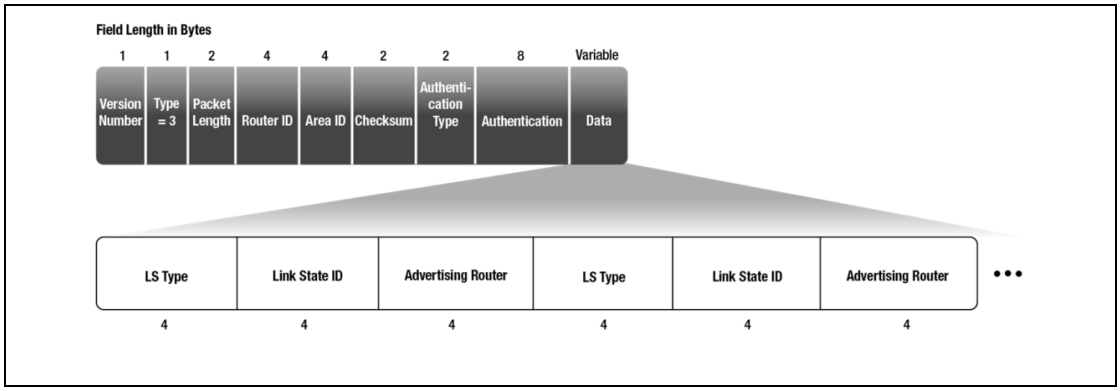
DD sequence number

This is the sequence number negotiated during the Ex-Start phase. Then, the Master increments the sequence number with each new DD packet it sends, and the Slave echoes the number in its responding DD packets. The process is described in the sections Ex-Start and Exchange on page 44.

The rest of the packet consists of a list of LSA headers.

Type 3 - Link State Request packet

Link State Request packets are used specifically in the Loading phase of neighbor relationship negotiation. These are the packets that one router sends to the other to request the LSAs that it realizes (from the Exchange process) it is missing from its LSA database. See a description of this process in the section Loading on page 46.

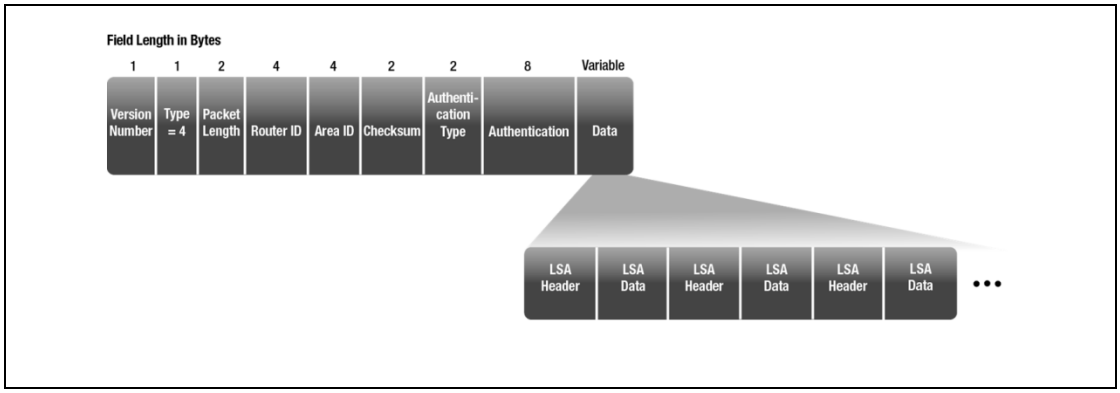


The data section consists of one or more triplets of LSType-LSID-Advertising Router. This is the list of LSAs that the Router saw in the neighbor's DD packets, and now wishes to receive the full content of.

The request is implicitly assumed to be a request for the latest version that is available of the specified LSAs.

Type 4 - Link State Update packet

This is the packet type that does the central communication task in OSPF – namely transmitting LSAs from one router to another. A Link State Update packet carries one or more LSAs. Each LSA is carried in full – all the data of the LSA is present; it is not summarized.

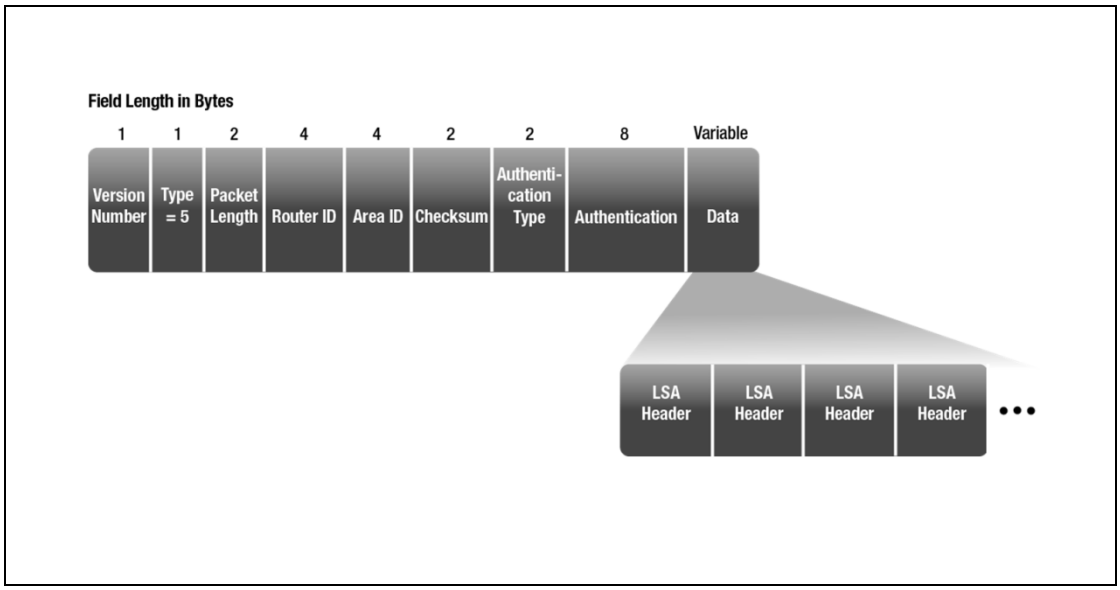


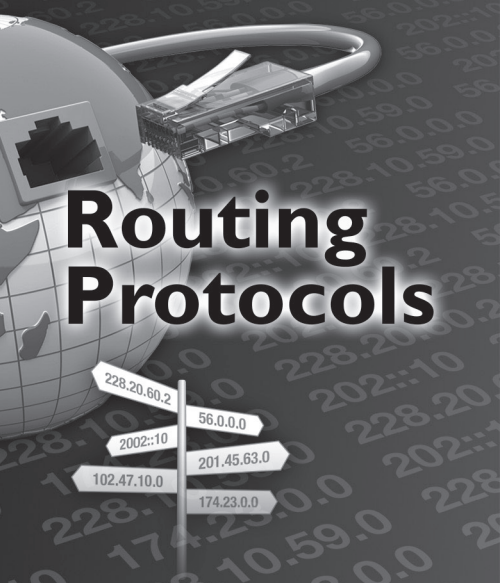
The structures of the LSAs carried in the packets are exactly as described in the section Content of LSAs on page 54.

There is no restriction on the combinations of LSAs that may reside together in the same LSUpdate packet. So, a packet can contain any combination of Router LSAs, Network LSAs, and External LSAs etc.

Type 5 - Link State Acknowledgement packet

In order for LSA transmission to be reliable, routers need to acknowledge the LSAs they receive, so that the sender knows they have been successfully received. The Link State Acknowledgement packet is used for explicit acknowledgement of received LSAs. The whole process of acknowledging LSAs is explained in the section Sending an acknowledgement on page 72.





Routing Protocols

This chapter covers the following topics:

- Advantages of Border Gateway Protocol
- BGP Basics – Forming Neighbors and Exchanging Routes
- Attributes
- Extended Communities
- Choosing the Best Route
- Route Aggregation
- Configurable Parameters in BGP
- Withdrawing Routes
- iBGP versus eBGP
- IGP Synchronization
- Next Hops
- Route Flap Dampening
- Configuring Route Dampening
- BGP and Route Maps
- 4-byte AS Numbers
- Address Families
- Packet Formats

CHAPTER 3

Border Gateway Protocol (BGP)

Introduction

Border Gateway Protocol (BGP for IPv4 and BGP4+ for IPv6) is an exterior gateway protocol (EGP). The purpose of BGP is to advertise, learn, and choose the best paths inside the Internet.

ISPs (Internet Service Providers) use BGP to exchange the Internet routing table with each other. Enterprises also use BGP to exchange routing information with ISPs, allowing the Enterprise routers to learn Internet routes.

BGP not only enables ISPs to exchange routes with each other, but also to control what data passes through their networks. ISPs need to keep fine control over the routes that they advertise out of their network, and who they advertise those routes to. They have commercial reasons for sending different traffic through different paths. In particular, ISPs whose main service is to provide bulk Internet backbone transport need to be very sure whose data they are transporting, as they do not want to be transporting data on behalf of people who have not paid for the service.

Since the paths via which Internet data is directed are subject to commercial agreements, network providers need to be able to implement policies that control the content of their route tables, and control the routes that they advertise to which neighbors. Internal routing protocols like OSPF and RIP do not have facilities for the types of policies that BGP needs. Although some filtering can be performed in OSPF and RIP, the sets of parameters that can be filtered on are rather limited.

So, instead of just using metric as the criterion for choosing the best route to a destination, BGP uses a process with path attributes, where path attributes are a variety of parameters that are associated with routes and exchanged in routing updates. BGP has an elaborate best path algorithm that is controlled by these path attributes, and allows network engineers flexibility in how routers choose the best BGP routes.

Moreover, the routing protocols used between ISPs are advertising huge numbers of routes (potentially hundreds of thousands of routes), so the routing protocol they use needs to be efficient, not a protocol that requires regular updates of all the routes (as the 30 minute refresh OSPF requires). Therefore, BGP was developed to operate quite differently from OSPF or RIP.

BGP does not send route updates to multiple neighbors in the local subnet (as is typical with IGPs), but uses TCP (port 179) to establish connections to just a specific set of peer routers

with which it will exchange routing information. BGP peer routers can be in the same subnet, or can be separated by several routers.

BGP does not send any more route updates than it absolutely has to. When routers first peer up, they exchange the route table data that they wish to inform each other of. Thereafter, they only send each other route information if anything changes.

BGP4 RFCs

The RFC at the heart of most BGP4 implementations is RFC1771 (BGP4)

Significant additions have been made to BGP4 since then, for example:

- RFC2385 : MD5 authentication of BGP peering connections
- RFC2796 : BGP Route Reflectors
- RFC3065 : BGP Confederations
- RFC2918 : BGP Route Refresh

These have all been wrapped up into RFC4271.

Other aspects of BGP have been defined in RFCs such as:

- RFC 1997 : BGP Communities Attribute
- RFC 2283 : Multiprotocol Extensions for BGP-4
- RFC 2439 : BGP Route Flap Damping
- RFC 3392 : Capabilities Advertisement with BGP-4
- RFC 4360: BGP Extended Communities Attribute
- RFC 4893 : BGP Support for Four-octet AS Number Space

BGP ASNs and AS_Path attributes

BGP uses path attributes (PAs). PAs define information about a path, or route, through a network. Some BGP PAs describe information that is used to choose the best BGP route. PAs are also used for other purposes, such as preventing routing loops.

If no BGP PAs have been explicitly set, BGP routers use the BGP autonomous system path (AS_Path) PA when choosing the best route among competing routes. The AS_Path PA itself has many subcomponents, one of which is the autonomous system number (ASN)

The integer ASN identifies one organization that considers itself autonomous from other organizations. Each company with a network that connects to the Internet can be considered to be an autonomous system and can be assigned an ASN. (IANA assigns unique ASNs.) Each ISP has an ASN. Some large ISPs have multiple ASNs.

When a router uses BGP to advertise a route, the route is associated with a set of PAs, including the AS_Path. The AS_Path PA associated with a route lists the ASNs that would be part of an end-to-end path to that destination, as learned using BGP.

BGP uses the AS_Path for two key functions:

- Choose the best route for a prefix based on the shortest AS_Path.
- Prevent routing loops.

Advantages of Border Gateway Protocol

Key features

The main features of BGP that differentiate it from other routing protocols are:

Attributes

Every route that is exchanged by BGP can have a set of attributes associated with it, such as:

- A list of the autonomous systems (AS) that the route has passed through.
- A list of the communities that the route is associated with.
- Whether the route was created by aggregating other routes.
- The Router ID of the route's originator.

The range of attributes that can be associated with a BGP route is far richer than anything that is possible in OSPF or RIP. And, the attributes stay with the route as it makes its way across the Internet. In fact, routers can add more attributes to a route as it goes along.

This provides plenty of scope for defining policies for blocking routes, or sending specific routes to specific neighbors, altering their metrics, etc., etc.

Path vector

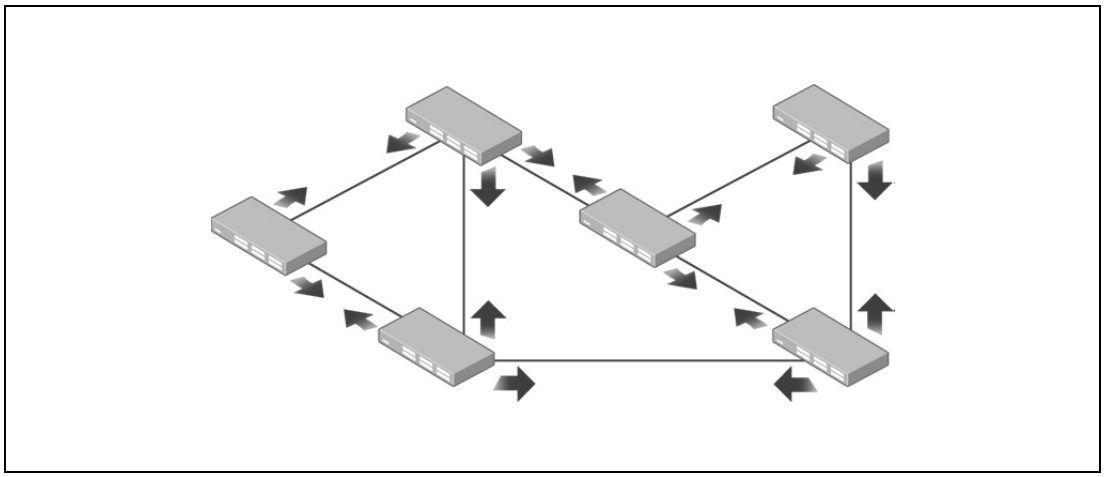
BGP is referred to as a Path Vector Protocol. What this means is that BGP is vector protocol in the way that RIP is, in that routers send a list of routes to their neighbors, rather than sending link states in the way that OSPF does. Being a vector protocol also implies that the choice between competing routes is made on the basis of some type of 'distance' value that is carried around with each route and incremented by each router that the route passes through.

The 'Path' part of the term Path Vector Protocol refers to the 'distance' value that the routes carry around with them. In the case of BGP, they carry a list of the autonomous systems that the route has passed through. This is referred to as the Path that the route has followed. The entries in the list are the AS numbers of the traversed autonomous systems. If a tie-breaker is needed between two routes to the same destination, with the same subnet mask, then the primary means of choosing between them is that the one with the shorter AS_Path (i.e. the

one that has passed through fewer autonomous systems) is preferred. Of course, if they both have AS_Paths of the same length, the protocol then needs to consider other attributes, which we will discuss later on.

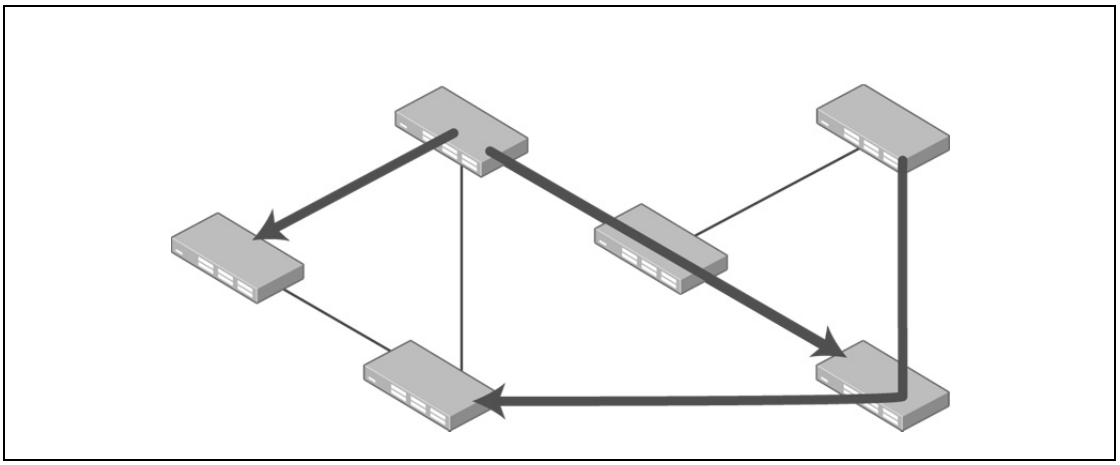
Neighbors connected by TCP sessions

In internal routing protocols like OSPF and RIP, the neighbor relationships are between routers in the same subnet. RIP and OSPF packets are never forwarded at Layer 3; they are transmitted into an IP subnet and picked up by other routers who are connected to the same subnet. In those protocols, routing information is spread step-by-step right through an IP network, in a blanket fashion.



But, BGP works quite differently. Because BGP is designed for large IP networks to exchange routing information with each other, so the neighbors that are exchanging the routing information may not necessarily be in the same subnet. There might be some other pieces of connecting IP network in between the two autonomous systems that are forming a BGP connection between each other.

Rather than just transmitting route updates into locally connected subnets for neighbors to pick up; BGP routers establish TCP connections to their neighbors, and pass the routing updates through these TCP connections.



Using TCP to transport the routing updates has the added advantage, of course, of providing reliable delivery.

Internal and external operating modes

Although BGP's primary role is that of advertising routing information between autonomous systems, it can also be used as the routing protocol within an autonomous system. BGP can be used as both an External Routing Protocol and as in Internal Routing protocol. In the former context, it is referred to as **eBGP**, and in the latter context as **iBGP**. There are some subtle differences between the ways the protocol operates in these two modes.

iBGP optimizations

When BGP is used as an internal routing protocol (i.e. as iBGP), the connection-oriented nature of BGP neighbor relationships (i.e. the fact that there is a TCP connection established between each pair of neighbors) can get rather unwieldy. There are very often large numbers of routers within an autonomous system, and it becomes rather impractical if each router needs to establish TCP connections to all the others. The optimizations of Clusters and Confederations have been developed, to enable iBGP to operate effectively without requiring every router in the AS to neighbor up with every other router in the AS.

Forming Neighbors and Exchanging Routes

Before we move on to looking at the advanced and complicated features of BGP, let's get to grips with the fundamentals – how do routers form neighbor relationships with each other, and how do they exchange routes?

Forming the neighbor relationship

Under AlliedWare Plus, configuring BGP neighbors is very easy. The commands are simply:

```
awplus# configure terminal
awplus(config)# router bgp <local AS Number>
awplus(config-router)# neighbor <neighbor IP address> remote-as
<neighbor's AS Number>
```

If the neighbor's AS number is different to the router's local AS number, then this is configuring an **e**BGP connection. If the AS numbers are both the same, then it is configuring an **i**BGP connection.

Once a router has been configured with a neighbor definition, as above, it will try to establish a TCP session to that neighbor. The **default TCP** port number for BGP is port **179**.

By default, the TCP packets sent in an eBGP connection have a time-to-live (TTL) of 1; meaning neighbors do need to be connected to the same subnet. Packets with a TTL of 1 will never be forwarded by a router, as the TTL expires the first time a router tries to forward them. For iBGP, the TTL is not set to a low value, and is typically set to 255.

The majority of eBGP connections are between routers at each end of a dedicated link between a pair of ASs; this is why eBGP defaults to a TTL of 1. In this situation, there are no routers between the BGP neighbors, so they are both in the same subnet, and the packets they exchange do not need a TTL of more than 1. Choosing a default TTL of 1 guards against unexpected connections being formed if the router is mistakenly configured with a neighbor address that is more than 1 hop away. It is actually possible to explicitly configure an eBGP connection to use a TTL greater than 1.This is described below in the section eBGP multihop on page 155.

OK, once two BGP routers have been configured to be neighbors of each other, what happens?

Check that this is a viable neighbor

When a BGP router receives TCP packets on its BGP port (port 179 by default), it checks various properties of those packets, to decide if it should accept the packets and establish a connection with the sender. The checks are:

1. Is the source IP address of the TCP packets equal to the one of the addresses that has been configured as a BGP neighbor?

2. Is **my autonomous system number** in the first actual BGP packet of the session (known as the OPEN packet) equal to the AS number that has been configured for this neighbor?

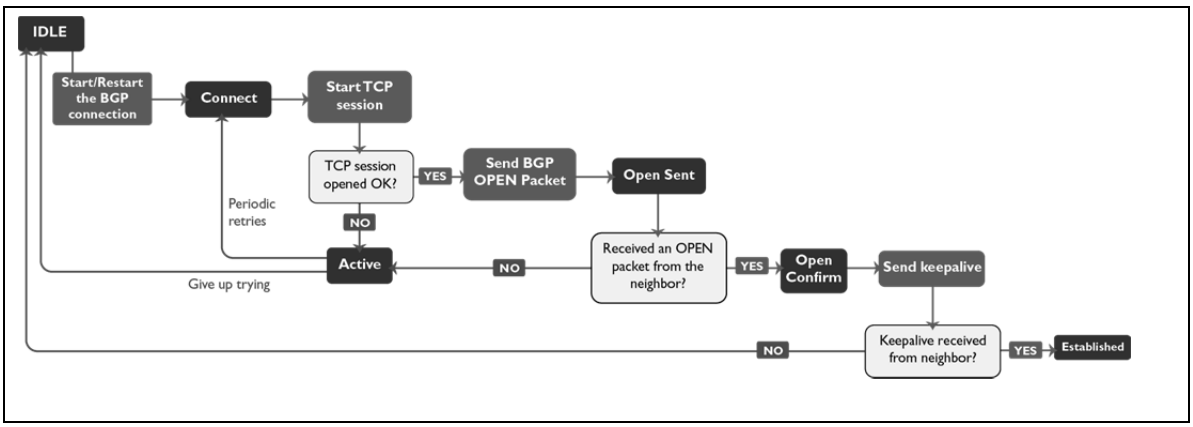
3. If the router is configured to expect the neighbor to authenticate it, then does the neighbor successfully perform the authentication? See Authentication on page 154 for a description of BGP authentication.

4. Is the Router ID in the neighbor's packets different to my own? See Router ID on page 154 for an explanation of the BGP Router ID.

Proceed through the stages of establishing the neighbor connection

The set of stages that an establishing BGP neighbor connection typically goes through are:

Idle –> Connect -> OpenSent -> OpenConfirm – Established



Let's look at each of these states, and a 6th state, Active, that the session can go into if things go wrong along the way.

1. **Idle** - If a BGP connection has been deactivated **or** has just been configured (and has not really kicked into gear yet) **or** has hit an error, and had to drop out, then it will be in the idle state. Unless it has been deactivated, it will quickly try to progress out of the Idle state, by attempting to establish a TCP connection with the configured neighbor.

2. **Connect** - When a router starts negotiating the TCP session – i.e. sends the initiating TCP SYN packet, or receives a TCP SYN packet from the neighbor – the connection enters the Connect state. It remains in the Connect state until the TCP three-way handshake is complete.

3. **OpenSent** - Once the TCP session has been established, the routers send each other BGP Open messages. These messages are the start of the actual BGP communication (as against the underlying TCP session). The OPEN messages are quite brief. The key pieces of information they contain are the sender's AS Number and Router ID. So, it is upon receiving the OPEN message that the router can check criteria (2) and (4) in the section Check that this is a viable neighbor on page 134 above. When the connection is in the OpenSent state, the router has sent its Open packet, and is waiting to receive the neighbor's Open packet.

4. **OpenConfirm** - This is a brief state that is entered when the neighbor's OPEN message has been received. At this point, the router sends a KeepAlive message to the neighbor, and takes the connection to the Established state.

5. **Established** - The BGP session is now ready to go. The routers can now start exchanging routes with each other.

6. The 6th state is **Active** - The **Active** state is a "*things have not gone well, but we are still trying*" state.

    - If the TCP session three-way handshake could not be completed (maybe because the wrong IP address has been configured for the neighbor, or the neighbor is not listening on the right TCP port, etc.), then the connection goes from Connect into the Active state.

    - If the TCP session closes while the BGP session is in the OpenSent state, then the connection goes from OpenSent into the Active state.

Whilst in the Active state, the router will accept incoming BGP attempts from the neighbor, and will periodically attempt to establish the connection to the neighbor.

Start exchanging routes

Once the BGP session has reached the Established state, the neighbors will start exchanging routes with each other. BGP exchanges routes in Update packets. Update packets can carry a list of routes to withdraw, and a list of routes to add, all in the same packet.

The general structure of an Update packet is:

- the list of routes to withdraw
- a set of attributes
- the list of routes to add

All the routes that are being added share that same set of attributes. There can be only one set of attributes per Update message; so it is not possible to say that some of the routes to be added have one set of attributes, and the others have another set of attributes. The set of attributes that are present in the Update packet are applied to all the routes in the list of routes to be added. Routes that have different sets of attributes need to be transmitted in separate Update packets.

Closing a session due to error

If a router sees an error in a BGP packet that it receives from its peer, then it will close down the session. The sorts of errors that will cause this are things like:

- An invalid value in an attribute in an Update message; like an ORIGIN attribute that has a value other than IGP, EGP or unknown.
- An invalid length for an attribute in an Update message.
- A mandatory attribute missing from an Update message.
- An Update packet with no actual routes in it.

Immediately prior to closing the session, the router that decides its neighbor has sent an errored packet will send a special packet to the neighbor. This is called a **Notification** packet. It contains an error code, and subcode, to explain to the neighbor what error it has seen. (The error code and subcode values are itemized in the section Packet Formats - *Notification* on page 186.

When the session is closed, the routers at both ends of the session must remove from their route tables any routes they have learnt from each other. Also, if they have advertised any of these routes on to other neighbors, then they must send *"I am withdrawing this route"* messages (i.e. an Update packet containing a list of routes to withdraw) to those neighbors.

Ongoing session

If neither router detects any errors in the BGP packets they receive from each other, neither router reboots, and the link between the routers stays constantly up, then the BGP session between them can stay open indefinitely.

The routers will continue to send BGP keep alive messages to each other at regular intervals.

If route table changes occur, and the neighbors have to advertise new routes to each other, or withdraw routes they had been advertising, then they will just send Update packets for these events as and when required.

Attributes

Attributes are properties that are associated with a route. Quite a variety of attributes have been defined over the years. Moreover, the attributes fall into four categories. First, let's understand the four categories that attributes can fall into, and then look at the attributes themselves. The reason for discussing the attribute categories first is because, as we look at each individual attribute, one of the properties of the attribute that will have to be mentioned is its type (i.e. the category it falls into). The discussion of the individual attributes will make more sense if we already understand what the categories are.

Attribute categories

The four categories that attributes fall into are:

1. Well-Known Mandatory
    - Well-Known means that all BGP implementations are expected to recognize and process these attributes.
    - Mandatory means that all BGP implementations are expected to include these attributes with all routes that they send out in Update packets.
    - Well-Known Mandatory Attributes are the core attributes that are in use all the time.
2. Well-Known Discretionary
    - Again, Well-Known means that all BGP implementations are expected to recognize and process these attributes.
    - Discretionary means that a BGP router can decide whether or not it includes these attributes with the routes it advertises in the Update packets that it sends.
3. Optional Transitive
    - Optional means that it is OK for a BGP implementation not to actually recognize these attributes.
    - Transitive means that, regardless of whether or not a router recognizes the attribute, it needs to be kept with the route that it arrived with, and passed on with that route when it is re-advertised to other peers.
4. Optional Non-Transitive
    - It is OK for a BGP implementation not to actually recognize these attributes.

- Non-Transitive means that the BGP router should not pass this attribute on with routes. The attribute is intended to be processed just by the current router; it is not necessarily relevant to further routers along the path.

  Of course, the router can decide that such an attribute, possibly with a new value, is relevant to one or more neighbors, and so will have this attribute in the Update message that passes the route on to those neighbors.

Description of commonly used BGP attributes

The short list below describes a few of the most commonly used BGP attributes:

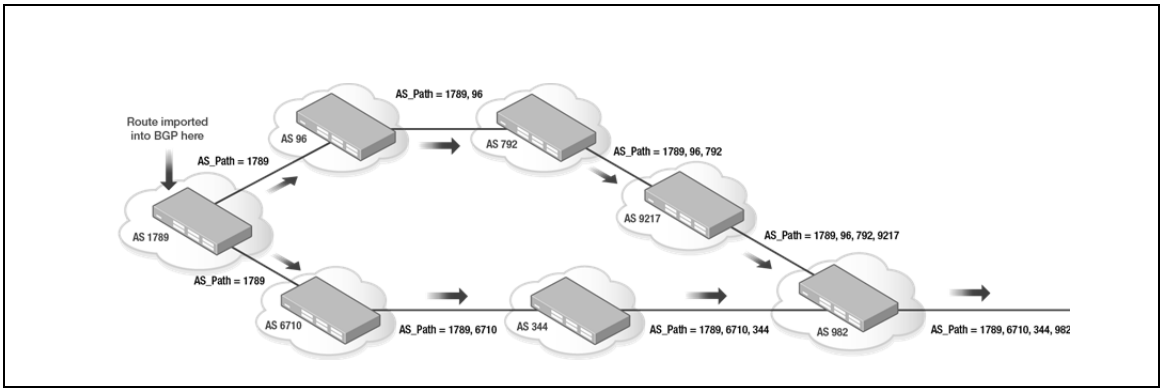
AS_Path

AS_Path is the most significant BGP attribute, as it is the primary attribute for determining the best route to a given destination. It carries the Path that characterizes BGP as a Path Vector protocol.

Not surprisingly, AS_Path is a Well-Known Mandatory attribute.

The AS_Path attribute contains a list of AS numbers - all the autonomous systems that the route has passed through. As a route is forwarded from one AS to another, the router in each of the ASs adds its own AS number into the AS_Path list.

By default, the AS_Path is a simple list of AS numbers, with the numbers appearing in the order that the route passed through those autonomous systems. But, as will be described below in the section Route Aggregation on page 150, when multiple routes, with different AS_ Paths are put together into an aggregate route, the AS_Path has to be stored rather differently.



Origin

Routes can enter BGP in different ways. The purpose of this attribute is to record, very broadly, how the route originally got pulled into BGP.

It is a Well-Known, Mandatory attribute.

The three values that the attribute can take on are:

1. **IGP** This value does not mean that the route was redistributed from an IGP (like OSPF or RIP). Rather, it means that the route was directly brought into BGP, using the **network** command. So, it is effectively a native BGP route. Also, in some circumstances, an aggregate route is given an Origin value of IGP. This will be described below in the section Route Aggregation on page 150.

2. **EGP** This value is effectively obsolete. It indicates that the route was redistributed from EGP, a now disused fore-runner to BGP.

3. **INCOMPLETE** Routes that have arrived into BGP by any other method – typically by redistribution – are given this Origin value. In some circumstances, an aggregate route is given an Origin value of Incomplete. This will be described below in the section Route Aggregation on page 150.

The Origin attribute is used as a form of preference for routes. This is discussed in more detail below in the section: Choosing the Best Route on page 147, but basically, with other factors being equal, routes with origin IGP are preferred over those with origin EGP, which, in turn, are preferred over those with Origin Incomplete.

Next Hop

The fact that a BGP router is advertising a route does not mean that the recipients of that advertisement will treat the advertising router as the next-hop for that route. Just because router B receives a route from router A does not mean that router B uses router A as the next hop in that route.

Instead, a BGP update associates an explicit next hop attribute with routes.

The Next Hop attribute is a Well-Known Mandatory attribute.

When the receiving router puts the route into its route table, the next hop for that route should be the address contained in the Next Hop attribute that arrived with the route.

There are some conventions and commands relating to the choice of the address to send as the next hop. We will discuss those more below in the section Next Hops on page 169.

Local Preference

Within the local AS, there may be specific reasons to prefer some routes over others; possibly even preferring routes with a longer AS_Path over those with a shorter AS_Path. There can be various reasons for this – maybe there is a particularly high-bandwidth path out of the AS, so routes via that path are preferred; or there could be an agreement with a particular neighboring AS to route certain blocks of addresses through that AS.

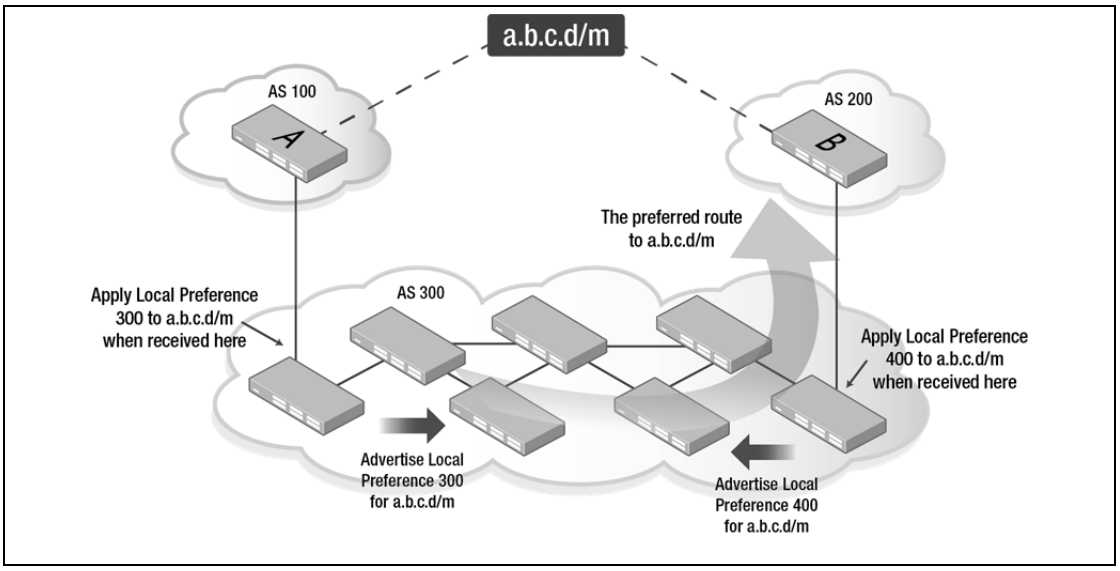
Whatever, the reason for this local policy that prefers certain routes, the network administrator applies this policy by using the Local Preference attribute to routes. Given that

this attribute is **local** it is not passed from one AS to the next. Therefore, the attribute is typically applied to routes on reception, using a route map.

The Local Preference is the first or second attribute used in choosing a best route, as described below in the section Choosing the Best Route on page 147.

The higher the Local Preference value, the more preferred the route. Local Preference is a Well-Known, Discretionary attribute.

The diagram below shows using Local Preference to choose the best egress path from the AS:



Multiple Exit Discriminator (MED)

If an autonomous system has multiple links to other autonomous systems, then it may wish to influence those neighboring autonomous systems to use some of the links in preference to others. It can include MED attributes with the routes it advertised, and give different values to the MEDs in the updates that it sends on different links.

The lower the MED, the more preferred the route (yes, this is the opposite of Local Preference, where the higher value wins).

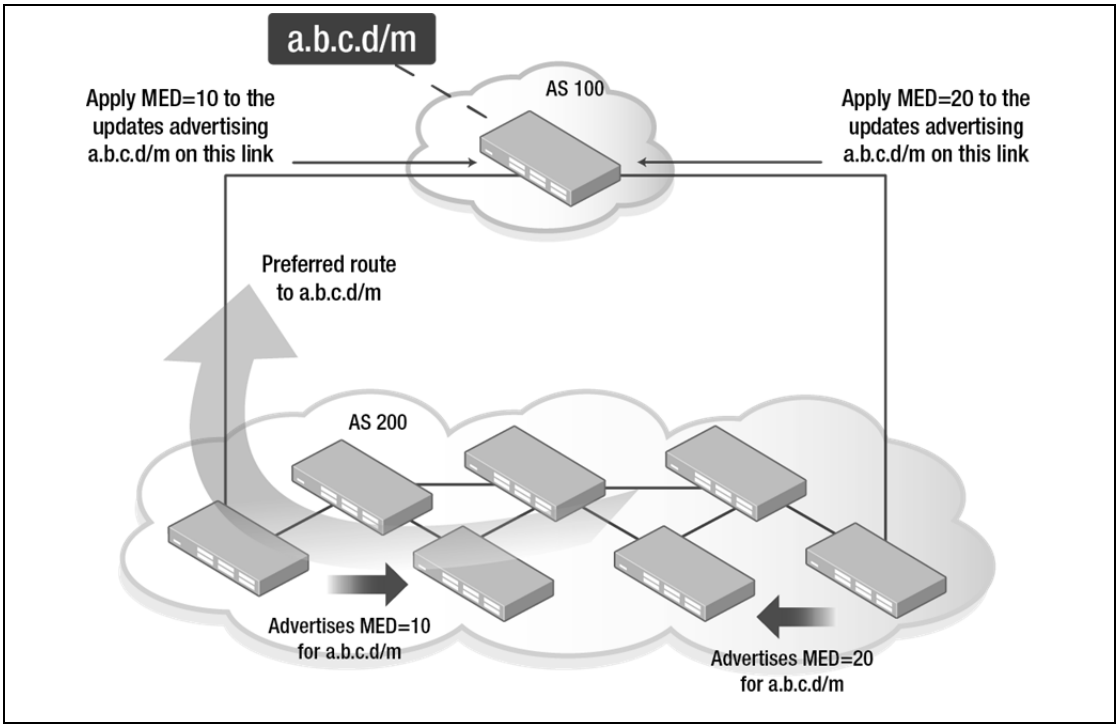
MED can be thought of as the complement to Local Preference.

- The purpose of Local Preference is to represent the choice that the local AS thinks it should make regarding the best connection to the neighboring AS.
- The purpose of MED is to represent the choice that the neighboring AS believes the local AS should make regarding the best connection to the neighboring AS.

MED is an Optional Non-Transitive attribute.

It is non-transitive as its purpose is just to influence the choice of link used between two neighboring autonomous systems. If the router that receives an MED value in an Update passes that attribute on to yet other neighbors, then the attribute would be misleading, as it does not relate to the connections to those neighbors.

The diagram below shows using MED to influence choice of path to a neighbor AS:



Communities

The name Communities conjures an image of sets of routes that are closely aligned, and maybe travel around as a group, and possibly all originate from a similar location.

To a certain extent, the Community attribute is aimed at embodying such a concept of a closely-related set of routes. But, to a great extent, it is just another attribute. It is just another tag that can be applied to routes to give you options about how to treat those routes.

The Community attribute is an Optional Transitive attribute.

What is the Community attribute typically used for?

It is very uncommon to use the Community attribute directly to choose the best route for a destination. More likely, the Community attribute is used to:

- Filter out certain routes on ingress or egress
- Set a route-preference attribute on certain routes

For instance, an ISP could say to its clients, "*If you apply community value XYZ on some of your routes, then these will be not be advertised beyond the ISP's AS*" or something similar. Or, if there are two connections between a pair of ASs, and a higher Local Preference is being applied to the routes arriving via one of those connections, there could be an agreement that this higher Local Preference is not applied to routes that have a particular Community attribute.

In general, the Community attribute is another tag to use for manipulating how routes are treated.

Community attributes can be specific to a single AS, or can be globally known.

The attribute is just a 4-byte number.

- The first two bytes are used in Community attributes that are defined just within a given autonomous system. Those two bytes hold the AS number of that autonomous system.
- The second two bytes contain the Community identifier. If the attribute is specific to an autonomous system, then the value in these bytes is simply a value that the administrator of the AS would have dreamed up; there are no rules about what values to use.

Community values are invariably written in the form XX:YY, where XX is the AS number and YY is the locally-assigned second half of the attribute value.

The values ranging from 0x0000000 through 0x0000FFFF and 0xFFFF0000 through 0xFFFFFFFF are reserved.

As mentioned above, some Community values are globally known. These predefined Community values which should be equally interpreted by all BGP routers, are:

- NO_EXPORT (0xFFFFFF01)

  Routes that have this value for the Community attribute should not be advertised out of the local AS.

- NO_ADVERTISE (0xFFFFFF02)

  If a BGP router receives a route that has this value for the Community attribute should not advertise the route to any neighbors.

- NO_EXPORT_SUBCONFED (0xFFFFFF03)

  This is a somewhat more restrictive version of NO_EXPORT. In this case, the requirement is that the route cannot be advertised to other sub-autonomous systems that form a Confederation within an autonomous system (the concept of Confederations is introduced in the section Confederations on page 164).

- NOPEER (0xFFFFFF04)

  The purpose of this Community value is to cut down on the extent to which broader-mask routes are advertised and narrower-mask routes that are subsets of those broader-mask routes are also advertised.

  If both the broader-mask route and the narrower-mask routes are pointing in the same direction, then the narrower-mask routes are completely covered by the broader-mask route, and do not need to be advertised. And, in fact, the unnecessary advertising of these routes is just adding extra entries into the Internet's already very large route tables.

  It is quite possible that when a small client AS is advertising routes to a larger provider, then it is possibly sensible to advertise broader-mask routes and narrower-mask routes that are subsets of those broader-mask routes.

It is possible that the narrower-mask routes need to be treated differently to the broader-mask route. But when the provider AS goes to advertise routes to another provider AS (a peer), then it should only advertise the broader-mask route, and not go filling up the Internet at large with the narrower-mask routes.

If the narrower-mask routes are given the NOPEER community, then it is saying "*It is OK to advertise this route to other client ASs, but when it reaches a point where it is going to be advertised from one provider to a peer provider, do not advertise it*".

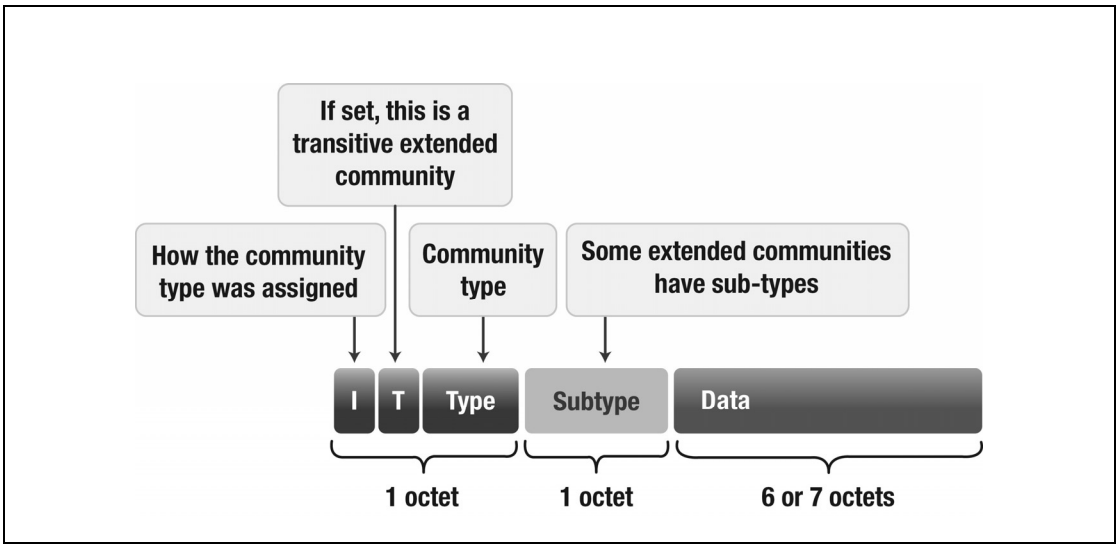
So, it is a bit different from the NO-EXPORT community value, which tells the receiving AS to not advertise the route to ANY other AS. In the case of the NOPEER community value, it is an instruction to just avoid advertising the route to other large provider ASs.

Extended Communities

Communities were not enough. Users wanted even more fine-grained control over what happened to routes, so a whole new set of attribute values, called Extended Communities, were introduced. The definition of the Community attribute had all been a bit ad-hoc: it was just a number, with no particular structure imposed on the number, and not much control over who used what value of the number.

With Extended Communities, the attribute value includes a Type field so that multiple values can be identified as belonging to the same Type. Then, rules can be applied to whole Types of attributes, rather than just having rules that have to be applied to a whole lot of individual attribute values.

The structure of an Extended Community value is as shown below:



- The I bit and T bit are really part of the Type field. They indicate properties of the Type.

- The I bit indicates which of IANA's Assigned Number allocation schemes has been used to allocate this Type value. There are a number of schemes defined in RFC2434:
    - If the I bit in the Type field is set to 0, then the value was assigned using the 'First Come First Served' method of RFC2434.
    - If the I bit in the Type field is set to 1, then the value was either assigned using the Standards Action method of RFC2434, or is an experimental value.
- The T bit indicates whether or not this type of Extended Community attribute is transitive:
    - If the T bit is set to 0, then this type of attribute can be transmitted to other ASs.
    - If the T bit is set to 1, then this type of attribute can be transmitted between ASs in a Confederation, but not to other external ASs.

Then, the actual value of the Type may be just the remaining six bits of the first byte, or may be those bits plus the whole of the second byte.

The types that just use the first byte for the Type value are called Regular Types. Those that use two bytes are called Extended Types. (Yes, Extended Types of Extended Community attributes – plenty of extension going here).

There is no indicator within a Type value as to whether it is a Regular Type or an Extended Type. Rather, a set of rules have been defined as to which are which:

Type	Standards Action	First Come First Served
Regular, transitive	**0x90-0xbf**	**0x00-x3f**
Regular, non-transitive	**0xdo-0xff**	**0x40-0x7f**
Extended, transitive	**0x9000-0xffff**	**0x0000-0x3fff**
Extended, non-transitive	**0xd000-0xffff**	**0x4000-0x7fff**

Extended Community Attributes are typically written in the form:

`type_name:<value part 1>:<value part 2>:<value part 3>....`

For example, a type of Extended Community Attribute called Route Target carries a value that consists of an AS number, and an identifier number that identifies a set of routes. So, the value section of the attribute has two parts to it. Route Target extended attribute values, therefore, are written in the form: RT: AS number: Routes ID

For example, RT:100:27

Extended Community Attributes are Optional, and may be transitive or non-transitive, depending on the Type in the attribute value.

Atomic aggregate

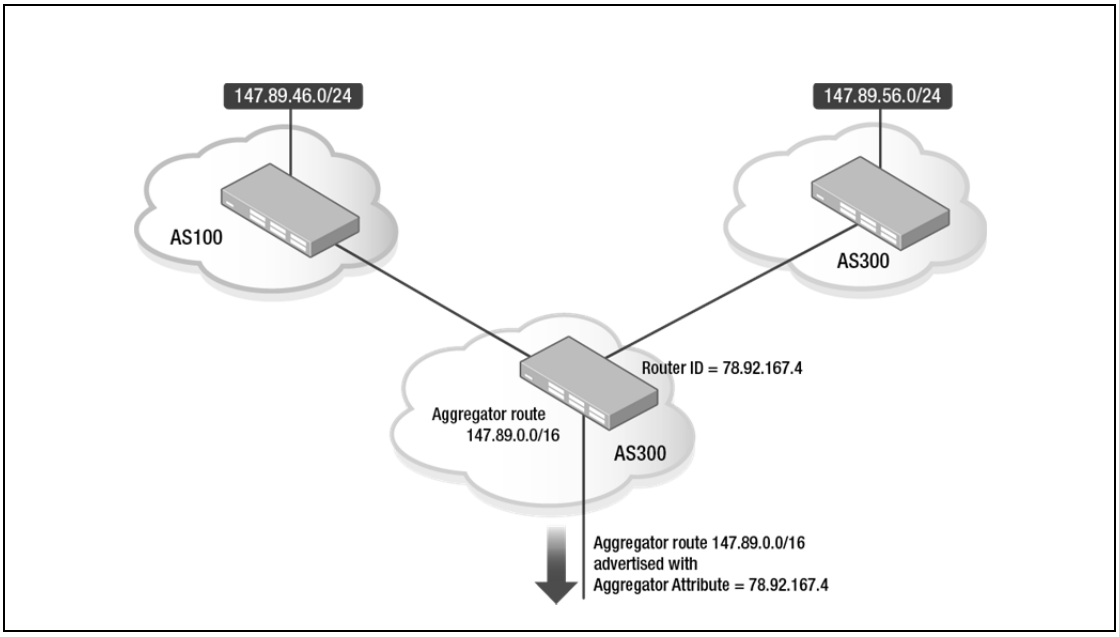
When routes are aggregated, the routes that make up the aggregate could well have different AS_Paths. The aggregate route can carry around an AS attribute that is the gathering together of all the AS numbers that are contained in the different AS_Paths. This gathering of all the different AS numbers is called the AS Set; this is discussed in more detail in the section Route Aggregation on page 150.

However, sometimes a router is not configured to gather the aggregated routes' AS_Path entries into an AS set, but instead the AS_Path attribute associated with the aggregate route will just be the AS_Path that was brought in by one of the constituent routes of the aggregate.

In this case, the recipients of the aggregated route need to be warned that the set of AS numbers in the route's AS_Path are not the full story on all the ASs that all the constituents of the aggregate have passed through. This warning is achieved by associating an Atomic Aggregate attribute with the aggregate route. This is a Well-Known Discretionary attribute.

Aggregator

When a router creates an aggregate route, it can attach to this route an Aggregator attribute. The purpose of this attribute is to identify the router that created the aggregate. As such, it contains the AS number and IP address (typically, Router ID) of the router that created the aggregate. It is an Optional Transitive attribute.



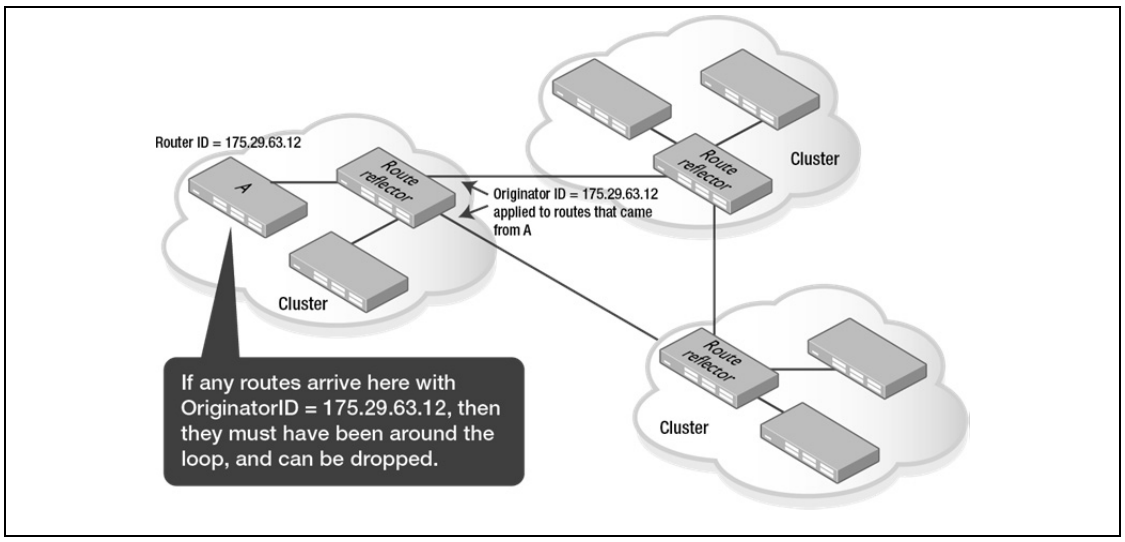
Originator ID

When using iBGP in a network that contains multiple clusters of routers attached to Route Reflectors (the concept of Route Reflectors is described in the section Route Reflectors on page 160) it is possible for routes to end up going around in a loop if you are not careful. We need a way for a router to see if a route it had originated is coming back to it.

The Originator ID, an Optional, Non Transitive attribute, has been defined. The content of the attribute is the Router ID of the router that originally advertised the route that the attribute is attached to.

Interestingly, though, it is not the originating router itself that attaches the attribute. Instead, the attribute is attached by a Route Reflector.

This is simply because it is the Route Reflection process that introduces the risk of routes going around in a loop. But, individual non-Route-Reflector routers do not necessarily know that there is a Route Reflector in the network. They do not know if they need to add the Originator ID attribute. But, a Route Reflector obviously DOES know that there is a Route Reflector in the network (itself), so it knows that it is introducing the risk of looped routes. Hence, the Route Reflector adds the Originator ID attribute.



Cluster List

The Cluster List is another mechanism for preventing loops in an iBGP network with multiple Route Reflectors. In this case, it enables the Router Reflector to detect the loops, whereas the Originator ID is aimed at enabling individual originating routers to detect loops.

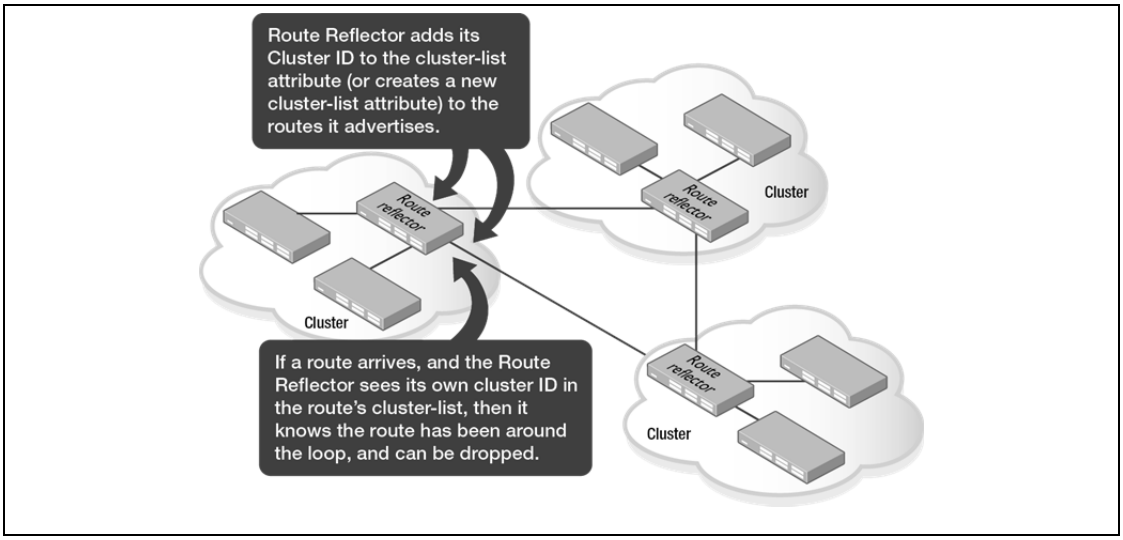
Each Router Reflector has a Cluster ID. As it forwards a route, it adds its Cluster ID to the route's Cluster List attribute. Then, as Route Reflectors receive routes, they can check whether their own Cluster ID is in the incoming routes' Cluster List attributes. If a Route Reflector's own Cluster ID is seen in an incoming route's Cluster List, then that route has been around a loop.

The Cluster List and the Originator ID catch slightly different situations.

The Originator ID catches the case where a route has come back to an originating router via a different Route Reflector from that which first associated the Originator ID with the route.

The Cluster List catches the case where the route comes back to the Route Reflector before it can be re-advertised to the originating router.

Cluster List is an Optional, Non Transitive attribute.



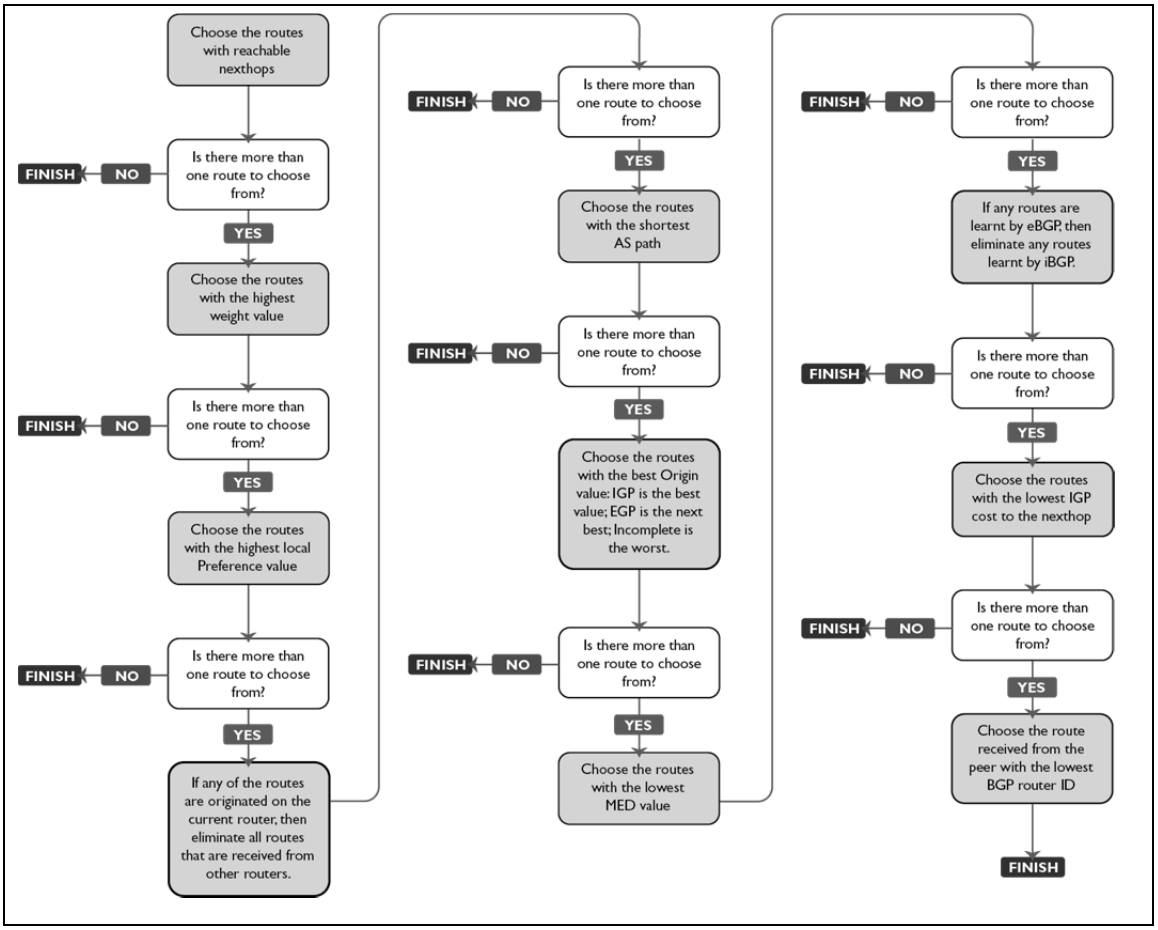
Choosing the Best Route

Having discussed all these attributes, many of which are directly aimed at the task of choosing the best route to given destinations, let's now look at how that best route decision is made.

The route decision is made as follows:

1. Any routes whose next hop is not reachable are instantly disqualified.
2. Select the path with the highest Weight. (Cisco only)
3. If path weights are the same, select the path with the highest Local Preference value.
4. Prefer locally originated routes (network routes, redistributed routes, or aggregated routes) over received routes.
5. Select the route with the shortest AS_Path length.
6. If all paths have the same AS_Path length, select the path based on Origin: IGP is preferred over EGP; EGP is preferred over Incomplete.
7. If the origins are the same, select the path with lowest MED value.
8. If the paths have the same MED values, select the path learned via **e**BGP over one learned via iBGP.
9. Select the route with the lowest IGP cost to the next hop.
10. Select the route received from the peer with the lowest BGP Router ID.

The diagram following illustrates how the best route decision is made:



Bringing Routes into BGP

A routing protocol needs routes to advertise

Once we have configured a router to establish a BGP session with one or more peers, we need to tell the router what routes to advertise to those peers.

In BGP, there are no immediately 'natural' routes to advertise. In contrast, RIP and OSPF have the concept of the routing protocol being configured on an interface. It is natural that the connected routes on those interfaces are immediately advertised by RIP or OSPF. That is the nature of an IGP (Interior Gateway Protocol) routing protocol – it is designed to share around the routes within the local network. However, BGP is fundamentally an **exterior** routing protocol, aimed at taking a selection of routes from one region and advertising them to another region. There are not necessarily any routes that are clearly the 'natural' routes for BGP to advertise.

So, how do we tell BGP which routes to advertise?

The network command

The primary method of telling BGP which routes to advertise is the **network** command.

```
network {<ip-prefix/length>|<ip-network-addr>} [mask <network-mask>]
[route-map <route-map-name>]
```

This specifies a route that will be advertised by BGP if it appears in the route table. It does not matter how the route gets into the route table – whether it is a static route or a connected route, or learnt by OSPF, or whatever. If the route is in the route table, then BGP can begin to advertise it.

Routes that come into BGP via the **network** command have the value **IGP** in their Origin attribute.

There are a number of aspects of this command that need to be explained:

Masks

If the mask (or prefix length) is explicitly specified in the command, then it is nice and clear what netmask the advertised route will have. But, if the mask is not specified, then there are rules about the way the mask is calculated:

- The first rule applies to the case that the network address is a **classful** network address. A classful network address is one in which the combination of zero and non-zero bytes conforms to the mask of the network class that the address falls within.(For example, 23.0.0.0 is an example of a classful network address in the Class A range, and 143.245.0.0 is an example of a classful network address in the Class B range.) If the address specified in the command is a classful network address, then the command will decide to use the subnet mask for that class (i.e. 255.0.0.0 for class A, 255.255.0.0 for Class B, 255.255.255.0 for class C).
- The second rule is for the case where the network address is **not** a **classful** address, but is any old address. In this case, the command will decide to treat the address as a host address, and so use the mask 255.255.255.255.

Route maps

The route map that may be configured on this command is not for the purpose of filtering routes, but rather is for the purpose of applying **attribute** values to the route as it is brought into BGP.

Redistribution

Routes can also be explicitly redistributed into BGP. Static, connected, and routes learnt by other routing protocols, can all be redistributed into BGP.

The command is:

```
redistribute {ospf|rip|connected|static} [route-map <route-map-
entrypointer>]
```

This time the route map can filter routes as well as set attributes. You can imagine that OSPF holds a large number of routes, but you only want to redistribute a subset of those routes into BGP. In that case, you can use the route map to select which routes are redistributed.

Routes that are redistributed into BGP arrive with the value *Incomplete* in their Origin attribute.

BGP route table

The network command and Redistribution tell BGP which routes to advertise. The mechanics of the process is that the BGP process maintains its own BGP route table. The routes that are imported into BGP by the network command and redistribution are stored in this BGP route table. Additionally, routes that the router receives by BGP from its BGP peer routes are also stored into the BGP route table.

The BGP route table, then, becomes the pool of routes that the router can advertise by BGP to its peers. It does not necessarily advertise all BGP routes to all peers. For example, it does not advertise back to Peer X the routes it learnt from Peer X. Also, route-maps and filters can be applied to different peering connections, to control which routes can and can't be advertised over that connection.

The other major purpose of the BGP route table is that this is the pool of candidate routes that BGP can offer to be installed into the switch's main IP route table.

The content of the BGP route table is displayed by the command: `show ip BGP.`

Route Aggregation

When you know that all the routes within a given address block are reachable via your router, then it makes sense to advertise this entire address block as a single route, rather than advertising all the separate routes that lie within the block. This act of route aggregation reduces the number of routes that are being advertised, and so keeps route tables smaller, thereby generally reducing overheads and improving efficiency.

Configuring aggregated routes

To configure a router to advertise an aggregated route, use the command:

```
aggregate-address <ipaddr/m> {summary-only|as-set}
```

The effect of this command is that whenever any route within the address range covered by ipaddr/m arrives into the BGP route table (note, it must arrive into the BGP route table, not just into the IP route table), the route ipaddr/m will also be added to the BGP route table.

The operation of the *summary-only* and *as-set* parameters is as follows:

summary-only

If the *summary-only* parameter is specified in the command, then only the aggregate route itself will be advertised, none of the component routes within the address range covered by the aggregate will be advertised.

By default, the component routes are advertised, but once the **summary-only** parameter is specified, they are not advertised.

as-set

Typically, when a router advertises an aggregate route, it will have learnt multiple routes that are within the address block covered by the aggregate route. It is highly possible that the AS_Path attributes of those various component routes will not all be the same. In fact, it is quite likely that AS numbers that appear in some component routes' AS_Paths will not appear in the AS_Paths of other component routes.

What should be put into the AS_Path attribute of the aggregate route? Do you choose the AS_Path of the first component route in the BGP route table? Do you choose the AS_Path of a randomly selected component route? Do you look for the AS_Path that is shared by the greatest number of component routes? What do you do?

The solution that has been chosen is that the AS_Path attribute actually consists of two segments:

- A **sequence** segment, in which the AS numbers are strictly lined up in the order that the route has passed through the corresponding autonomous systems.
- A **set** segment, in which AS numbers are gathered, but in no particular order.

Normally, the AS_Path attribute associated with a route only contains a sequence segment, providing an ordered list of the autonomous systems that the route has passed through.

But, when an aggregate route is created, and the ***as-set*** parameter is specified on the command that creates the aggregate, then the AS_Path attribute attached to the route has an AS Set segment. The content of this AS Set is the aggregate of all the AS numbers in all the AS_Path attributes of the component routes that fall within the address range covered by the aggregate route. In other words, the content of the AS Set is the set of all the AS numbers that appear at least once somewhere in the AS_Path attributes of all the component routes that the aggregate route covers.

None of the AS numbers in the AS Set is duplicated. If a given AS number appears in more than one component route's AS_Path, then that AS number will appear only once in the aggregate route's AS Set.

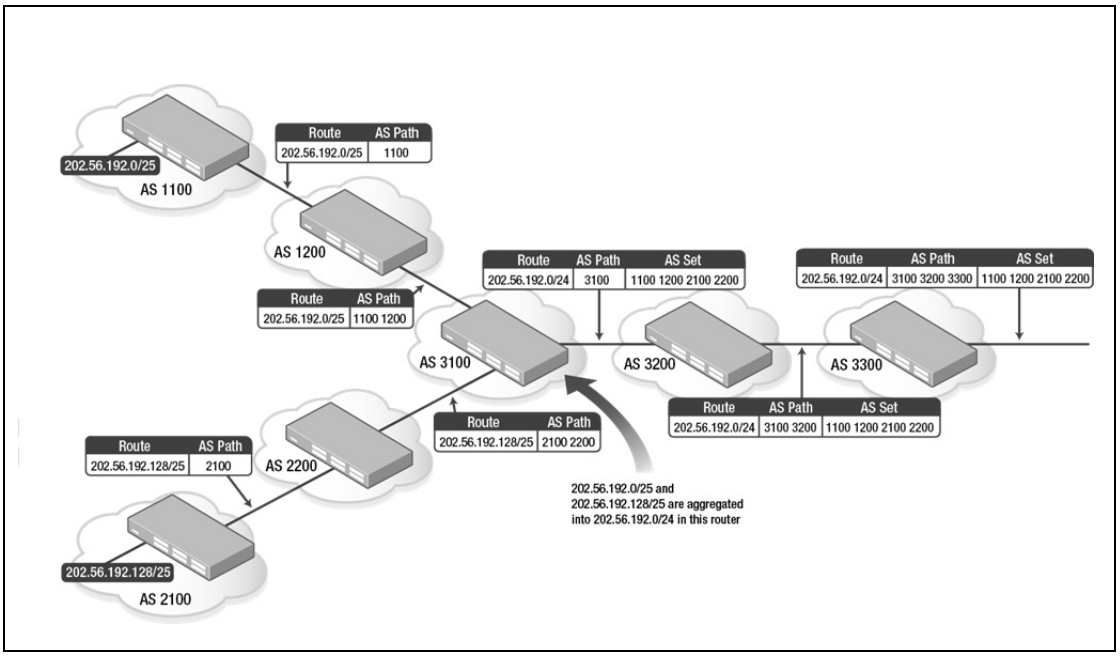
The AS Set segment of the aggregate route's AS_Path attribute at least lets routers see the full set of autonomous systems that the aggregate route's component routes have been through.

While the AS Set cannot be used to make best-route decisions based on AS_Path length, it at least allows routers to **detect** routing **loops**. It is still valid for routers to detect loops by looking into the AS Set, and finding its own AS number.

If the ***as-set*** parameter is not specified in the **aggregate address** command then the aggregate route ends up with an empty AS_Path attribute.

Once an aggregate starts being passed from router to router, the sequence segment of its AS_Path attribute continues to be updated by each router inserting its own AS number, as per usual.

If an aggregate route was created with the ***as-set*** parameter specified, then after it has been passed on from the router that originally created it, it will have contents in both the AS Set and AS Sequence segments of its AS_Path attribute. The AS Set segment of the AS_Path attribute will be an unordered collection of all the autonomous systems that the aggregate's component routes passed through before the aggregate was created. The AS Sequence segment of the AS_Path attribute will be an ordered list of the autonomous systems that the aggregate route has passed through since it was created.



Attributes on aggregate routes

There are four attributes that we should discuss in relation to aggregates: AS_Path, Atomic Aggregate, Aggregator, and Origin.

AS_Path

As discussed just above, the AS_Path attribute of an aggregate will often end up containing an AS Set segment.

Atomic aggregate

As described above, in the section Atomic aggregate on page 153, if the AS Set is not created in the aggregate route's AS_Path attribute, then it needs to be given an Atomic Aggregate attribute.

Aggregator

As described above in the section Aggregator on page145, a router that creates an aggregate route can put its own Router ID into an Aggregator Attribute associated with the route, to identify who created the aggregate.

Origin

There are a set of conventions that govern the value of the Origin attribute that is applied to aggregated routes.

If the **as-set** parameter is:

- **Not** specified in the **aggregate address** command, then the Origin attribute is set to IGP. In effect, the aggregate route is treated as a newly created BGP route, as though it had been brought in using the **network** command.
- Specified in the **aggregate address** command, AND ALL of the component routes in the BGP route table that fall within the address range of the aggregate route have Origin IGP, then the aggregate route has Origin IGP.
- Specified in the **aggregate address** command, and even one of the component routes in the BGP route table that fall within the address range of the aggregate route has Origin Incomplete, then the aggregate route has Origin Incomplete.

Configurable Parameters in BGP

There are a number of parameters that are required for the general mechanics of the BGP protocol. We need to run through these parameters, to understand what they do, and how to configure them.

Router ID

Every router needs an ID. The ID is used in various circumstances:

- As described above in the section Choosing the Best Route on page 147, the advertising neighbor's Router ID is a last-resort tie-breaker when all the other aspects of competing routes all match.
- As described above in the section Aggregator on page 145, the Router ID is the value entered into the Aggregator attribute by a router that creates an aggregate route.
- If a session collision occurs (both ends of the connection establish BGP TCP sessions to each other at the same time), then the session that was initiated by the peer with the higher Router ID is the session that is retained. The other session is closed.

The Router ID takes the form of an IPv4 address. It does not necessarily have to be one of the IP addresses on the router itself, but it is customary to choose one of the router's IP addresses.

By default, the router will automatically choose the Router ID as follows:

- If a loopback IP address has been configured on the router, using the **IP address** command under **interface lo**, then this IP address will be used as the Router ID.
- If the loopback IP address has not been configured, then the router will choose the highest IP address that it has.

The Router ID can also be explicitly configured in router BGP mode, using the command:

```
bgp router-id <routerid>
```

Authentication

An authentication password can be configured, on a per-neighbor basis, with the command:

```
neighbor {<ip-address>|<peer-group-name>} password <password>
```

A pair of neighbors must have matching passwords in order for a BGP session between them to come open successfully. The password is sent in an Optional Parameter field in the Open packet.

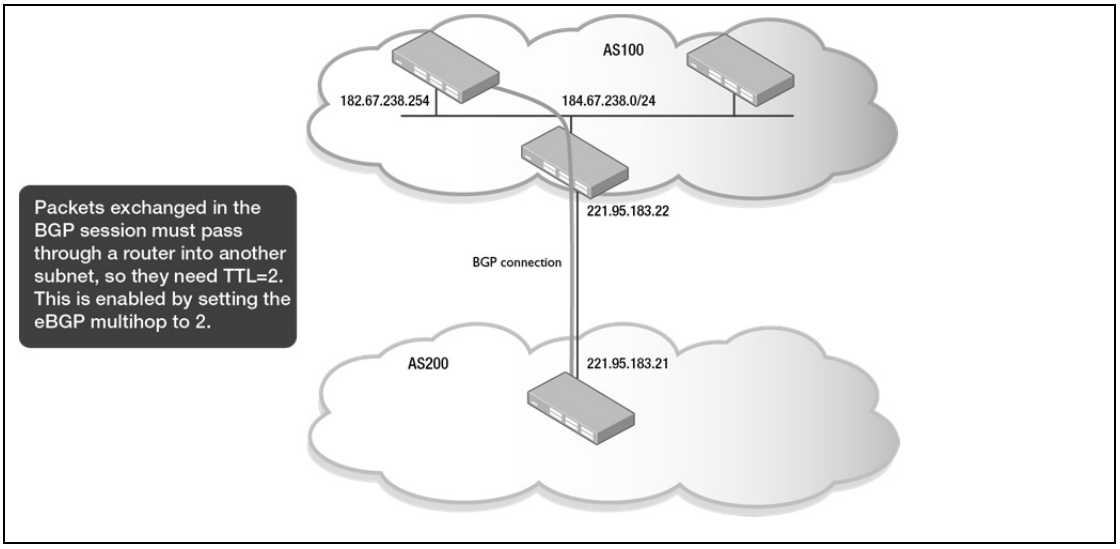
eBGP multihop

By default, eBGP sessions (i.e. sessions between routers in different autonomous systems) are assumed to be connected across a common LAN that is the "*exchange LAN*" between the autonomous systems. Therefore, there is an assumption that the packets in the BGP session will not need to be routed, i.e. they will just pass within this one subnet. Hence, the TTL value in eBGP packets is 1.

If you do want to create an eBGP connection between two neighbors that are in different subnets, you need to set a TTL value **higher** than 1 in the packets, or else they will just be dropped at the first router they encounter.

The command to set the TTL to a value other than 1 is:

neighbor <*neighborid*> ebgp-multihop [<*count*>]

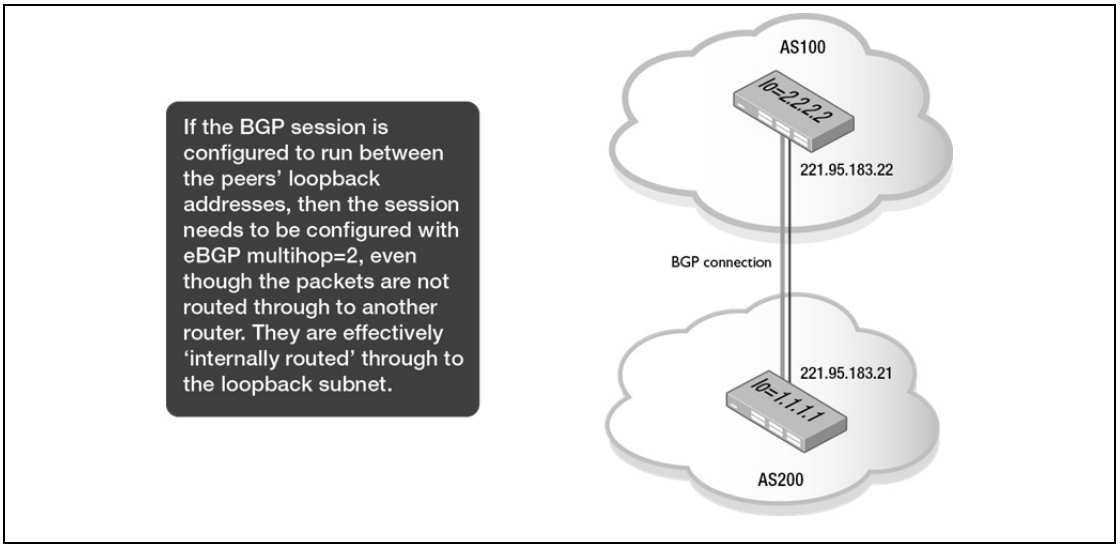


Even if two routers do happen to be directly connected on the same subnet, **but** the IP addresses they are using for their BGP connection are not the addresses in that shared subnet, then the **ebpg-multihop** parameter will need to be set to a non-default value. The specific case where this most frequently arises is when routers are using their loopback IP address for their BGP connection.

If one router has IP address 1.1.1.1 on its loopback interface, and another has 2.2.2.2 on its loopback interface, and the neighbor relationship is set up to use the loopback interfaces, with a command like:

```
neighbor 2.2.2.2 remoteas 2
neighbor 2.2.2.2 interface lo
```

Then, it is necessary to also configure: `neighbor 2.2.2.2 ebgp-multihop 2`



Passive

The term '*passive*' has a rather different meaning in BGP to that which it has in other routing protocols.

In OSPF, a passive interface is an interface that is considered to be part of the OSPF domain, but which does not transmit any OSPF packets.

In OSPF:

- The connected route on a passive interface is automatically brought into the routing protocol's route table.
- No neighbor relationships will be formed with other routers connected to the passive interface.
- No routes are advertised or learnt via the passive interface.

By contrast, BGP has the concept of a passive neighbor definition. This is quite a more active sort of passivity, in that when you configure a neighbor definition as passive, it just means that the current router will not initiate BGP sessions to the neighbor in question. However, if that neighbor initiates the session, then the current router will still respond, and negotiate a normal, active BGP connection with the neighbor.

Therefore, a router will exchange routes with a 'passive' neighbor, provided it is the neighbor that initiates the BGP connection.

Auto-summary

Auto-summary affects routes that are redistributed into BGP and those brought into BGP by the **network** command. Fundamentally, the purpose of **auto-summary** is to replace routes that are subnets of a classful route by just a single route entry for the whole of the classful route.

The rules relating to autosummary are:

- For routes **redistributed** into BGP:
  - If the route is not a classful route, it will be replaced by the classful route of which it is a subset. For example, if redistribution is bringing the route 15.23.4.0/24 into BGP, then if **auto-summary** is configured, then the actual route brought into BGP will be 15.0.0.0/8.
- For routes brought into BGP using the **network** command:
  - If a network command has specified a classful network address, with no mask, then if any subnet of this classful network appears in the IP route table, then bring the classful network into BGP. For example, if the router is configured with the command: `network 23.0.0.0` and **auto-summary** is configured. Then, if the route 23.178.29.0/24, say, appears in the IP route table, then the route 23.0.0.0/8 will be brought into the BGP route table.

Note - in the case of the **network** command, **auto-summary** does not cause the subnet route to be suppressed. In the above example, the router was configured with **network 23.0.0.0** and network **23.178.29.0 mask 255.255.255.0** then both 23.0.0.0/8 and 23.178.29.0/24 are brought into BGP if the route 23.178.29.0/24 appears in the IP route table.

If no **auto-summary** is specified, then a classful network like 23.0.0.0/8 would only be imported into BGP if the exact route 23.0.0.0/8 appears in the IP route table.

Timers

There are two configurable timers in BGP – the **keepalive** timer and the **hold** timer.

The keepalive timer governs **how often** a BGP router sends keepalive messages to neighbors. The hold timer governs **how long**, without receiving keepalive messages from a given neighbor; the router waits before deciding the neighbor is dead.

- The keepalive timer defaults to 30 seconds.
- The hold down timer defaults to 90 seconds.

The timer values can be changed by using the command:

```
timers bgp <keepalive> <holdtime>
```

This command is a little odd, in that the configuring of a hold down time requires that the keepalive time also be configured. But, a keepalive time can be configured on its own.

For example, the command: **`timers bgp 40`**

configures the keepalive time to 40.

The command: **`timers bgp 40 120`**

configures the keepalive time to 40 and the hold time to 120

But, there is no way to just set the hold time to 120.

Withdrawing Routes

Routing protocols enable routers to tell each other about the routes that they have available. It is also necessary for routers to be able to tell each other when previously available routes are no longer available, i.e. to withdraw routes.

In BGP, the process for withdrawing routes is effectively the same as the process for advertising available routes. As described above in the section Start exchanging routes on page 135, a BGP Update packet can carry a list of routes to withdraw as well as a list of routes that are available.

As soon as a BGP router realizes that a given route is no longer available, and it had been advertising that route by BGP, then it can send an Update message to withdraw that route.

When the neighbor receives this withdrawal message, it will remove the route in question from its BGP route table, and will also send withdraw-this-route Updates to any neighbors to whom it had been advertising the route. In this way, the route is removed from the network, one hop at a time.

iBGP versus eBGP

In BGP, a clear distinction is drawn between BGP connections that are formed between routers in different autonomous systems and those connections that are formed between routers that are both in the same autonomous system.

- When BGP is communicating between routers in different autonomous systems, they are referred to as eBGP (external BGP) connections.
- When BGP is communicating between routers that are both in the same autonomous system, they are referred to as iBGP (internal BGP) connections.

On the face of it, whether the routers are in different autonomous systems, or both in the same system, shouldn't really make a lot of difference. Why is such a strong distinction made between these two types of BGP connections?

The key reason for the distinction revolves around the fact that the AS_Path is the primary means by which BGP detects routing loops.

Loop detection

When routes are being advertised from one AS to another, the router will append its AS number to the AS _Path before sending the route. Then, if a router sees its own AS number on a route it receives, then it will know that route has been around a loop, and it will not accept that route.

But, when routes are being advertised within an AS, this method of loop detection is no good. If Router A and Router B are two routers in the same AS, and each appended its own AS number to the AS_Path attribute of the routes it sent, then both routers would see their own AS number in the AS_Path attribute of all the routes they received from each other (because both routers are putting the same AS number in their AS_Path attributes). As a result, the routers would drop all the routes that they received from each other. This is not going to be a very successful way of advertising routes.

In iBGP, the routers do not append their own AS number to the AS_Path attribute of the routes they send. But, that then leaves iBGP with NO method for detecting routing loops, which is yet another dilemma.

Therefore, to stop routing loops within an iBGP network, a simple, but strict, rule is applied – a route that is learnt by iBGP cannot be advertised to iBGP peers.

That, of course, leads to yet another little dilemma – if a router cannot learn a route from one iBGP peer, and pass it on to other iBGP peers, how are routes ever going to be passed around in an iBGP network?

Well, the solution to this problem has been to insist that the BGP peering connections in an iBGP network be fully meshed. Every router in an iBGP network must have a peering connection with every other router in the network. Because it is OK for a router to learn a route by eBGP and pass it on to iBGP peers, then in a fully meshed network, all routes learnt from outside the AS will be passed to all routers within the AS. Similarly all routes originated within the AS will be passed out of the AS.

Hence, by imposing a couple of rather rigid ad hoc rules, we end up with a viable solution that enables iBGP to distribute routes to all routers without causing any routing loops.

- By forbidding routers to forward iBGP-learnt routes on to iBGP peers, we achieve the aim of preventing routing loops in the iBGP network.
- By insisting that there be a full mesh of peering connections between routers in an iBGP network, we achieve the goal of getting routes distributed to all routers.

But, this solution comes at a cost. The cost is that scaling up an iBGP network is difficult. As the number of routers in an iBGP network grows, the number of peering connections required within the network grows much faster. This adds a lot of overhead in a large iBGP network.

Two BGP extensions have been devised to deal with this scaling problem:

- Route Reflectors (and clusters)
- Confederations

Route Reflectors

As explained above, the purpose of Route Reflectors is to reduce the number of peering connections that are required in an iBGP network.

The approach that is taken is to designate one or more iBGP routers in the network who **are** allowed to forward on routes from one iBGP connection to another.

This enables an iBGP network to be broken up into a set of clusters, with each cluster consisting of a Route Reflector and a set of 'client' routers.

Each Route Reflector is explicitly configured with the list of routers that are its clients. A Route Reflector has certain rules about how it interacts with clients.

The rules are:

1. If the Route Reflector learns a route from a client neighbor, it may advertise that route to other iBGP neighbors, both clients and non-clients.
2. If the Route Reflector learns a route from a non-client iBGP neighbor, it may advertise this route to its clients, but not to other non-clients.

The following table provides a good summary of where a Route Reflector may forward the routes it learns:

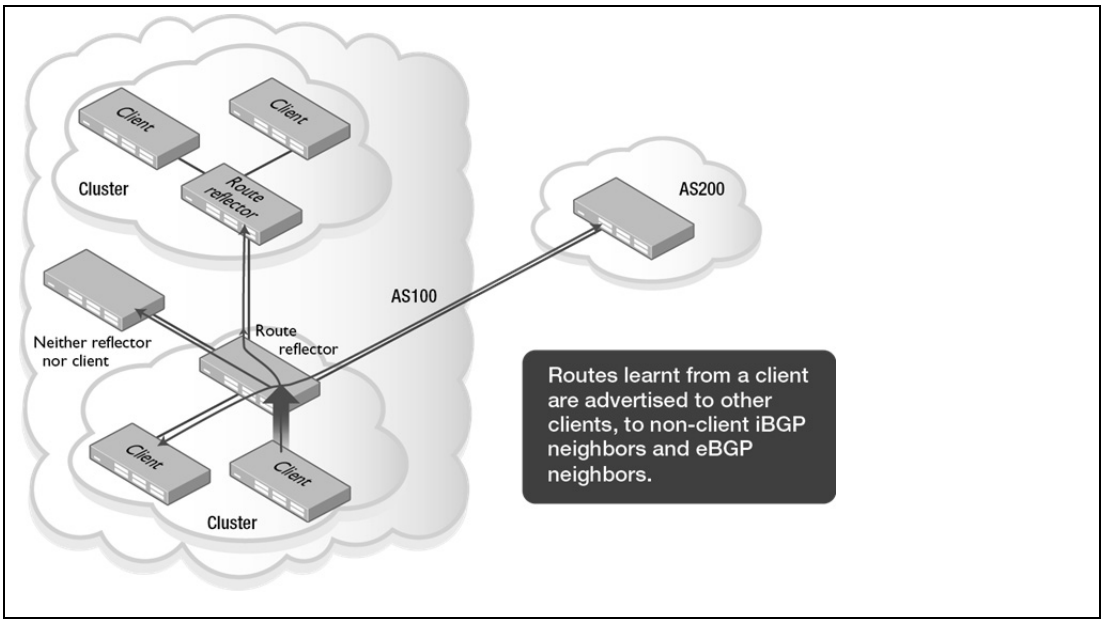
Type of neighbor route is learnt from	Can the route be advertised to clients?	Can the route be advertised to non-clients iBGP neighbors?
Client	**Yes**	**Yes**
Non-client	**Yes**	**No**
eBGP	**Yes**	**Yes**

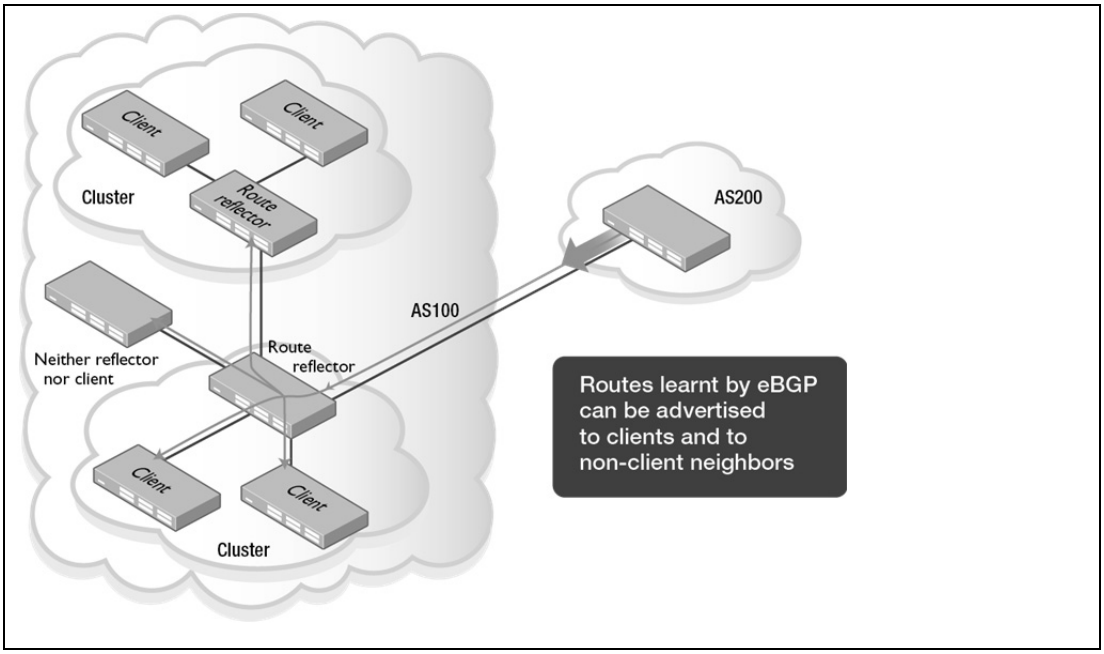
In effect, the Route Reflector treats its client neighbors as though they were eBGP neighbors.

The structure within the iBGP network effectively becomes 2-tiered:

- Each Route Reflector feeds routes to and from its clients.
- The Route Reflectors exchange routes with each other, via a fully meshed set of inter-Route Reflector connections.

One Route Reflector does not need a connection to another Route Reflector's clients; it can send routes to those clients via their Route Reflector.





Configuration of Route Reflectors

A really interesting aspect of this route reflection business is that the clients are quite unaware that they are clients of a Route Reflector. The Route Reflector needs to know who its clients are, but the clients do not need to know who their Route Reflector is.

Similarly, Route Reflectors do not need to know the identities of any other Route Reflectors in the network. This is all because the rules described above apply only to how the Route Reflectors treat the advertisement of routes to/from their clients. There are no special rules about what clients do, and there are no special rules about how Route Reflectors treat non-clients (they just apply the standard iBGP rules to their interactions with non-clients).

The only configuration required is on the Route Reflectors.

They need to be told who their clients are, using the command:

```
neighbor <neighborid> route-reflector-client
```

Note, as soon as a router is configured with an instance of this command, it realizes that it is a Route Reflector. There is no special command to put a router into Route Reflector mode. The simple act of having clients is enough to indicate to a router that it is a Route Reflector.

A Route Reflector also needs a Cluster ID.

```
bgp cluster-id {<ip-address>|<cluster-id>}
```

Route Reflector redundancy

A Route Reflector represents a bit of a single point of failure. Good network design always aims to eliminate single points of failure. It is not unusual to double-up Route Reflectors to achieve redundancy.

This is achieved by each client router in the network being configured into the client lists of two or more Route Reflectors.

If one of the Route Reflectors goes down, the clients will still have another Route Reflector that will be advertising routes to/from those clients.

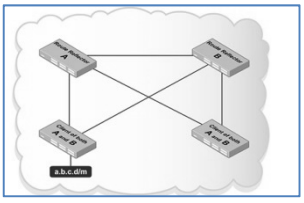
Of course, the Route Reflectors are quite unaware that they are sharing clients. Each Route Reflector knows nothing more than the list of clients it is configured with; it has no idea what client lists are configured on other Route Reflectors in the network. In fact, it does not even know which other routers in the network are Route Reflectors.

This ignorance on the part of the Route Reflectors, though, will easily lead to routing loops.

Consider the simple example below of a client and two Route Reflectors.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/gT2QXWHfYu0



The client will advertise routes by iBGP to Route Reflector A.

The Route Reflector will apply the rule that any route learnt from the client can be advertised to other iBGP neighbors, so it will advertise the route on to Route Reflector B.

Route Reflector B will apply the rule that any route learnt from another iBGP neighbor can be advertised to a client, so it will advertise the route back to the client.

A routing loop has occurred.

Fortunately, BGP has a way to prevent this type of routing loop from occurring. As with most things in BGP, the prevention of this routing loop is achieved with the aid of **attributes**.

Route Reflector loop prevention

When a Route Reflector takes a route that it has learnt from an iBGP peer, and advertises this route to another iBGP peer, there are two special Route Reflection related attributes that it can associate with the route. They are:

1. The **Originator ID**. This is the Router ID of the iBGP peer (the client) from which the Route Reflector learnt the route.

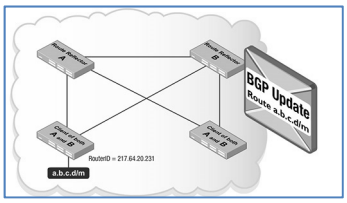
2. The **Cluster List**. This is a list of the Cluster IDs of all the Route Reflectors that the route has passed through.

Let's look at how each of these attributes is used for preventing routing loops.

The Originator ID is only added to a route if it does not already have one. The Originator ID is added to the route by the first Route Reflector that the route passes through, and then it is never changed. It always holds the Router ID of the original client router that first sent the route into the iBGP network. If that route ever makes its way back to that original router, the router will see its own Router ID in the Originator ID field, and know that this route has been around a loop. Having seen that, it will drop the route.

To watch this video, browse to the link or scan the QR code with your smart phone:

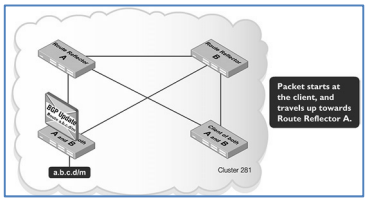
http://youtu.be/YxM0ot84Wek



As with the Originator ID, the Cluster List is added to a route by the first Route Reflector that the route passes through. If a Route Reflector receives a route from a client and the route does not have a Cluster ID attribute, the Route Reflector creates the attribute, and puts its own Cluster ID as the first entry in the list. Subsequent Route Reflectors that the route passes through each add their own Cluster ID to the list.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/pemle_2jkTw



The idea is that if there are Route Reflectors in a redundant pair, or a redundant group, then they all have the same Cluster ID. The set of redundant Route Reflectors, along with their clients, form a group, known as a Cluster.

Then, if a Route Reflector receives a route, and finds its own Cluster ID in the route's cluster List, then it knows that route will have been reflected by a Route Reflector in its Cluster. Therefore it knows that all the clients in the current Cluster should already have received this route, so it does not need to advertise the route to them.

Confederations

The Confederation is an alternative approach to the iBGP scaling problem. As with the Route Reflector approach, it collects the routers within an iBGP network into separate groups that exchange routes with each other, but only have one or two BGP connections to any of the other groups.

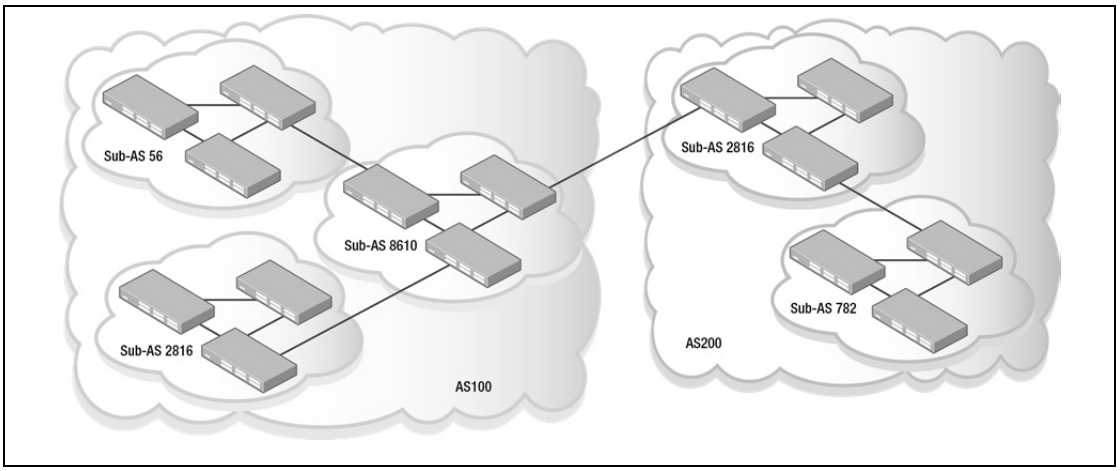
The structure that the Confederation approach adds to an iBGP network is altogether more elaborate than that of the Route Reflector approach. The approach that is taken is to divide the autonomous system into a set of sub-autonomous systems. So, the iBGP network becomes a Confederation of sub-autonomous systems (sub-ASs).

This Confederation of sub-ASs is like a mini Internet, with iBGP operating within each sub-AS, and a slight variation on eBGP operating between the sub-ASs.

Within each sub-AS, routers must be fully meshed, as with a standard iBGP network.

The connections between the sub-ASs do not need to be fully meshed. Sub-ASs can pass routes on from one to another, just like autonomous systems in the actual Internet.

Each sub-AS has an AS number. These AS numbers must be unique within the Confederation, but may overlap with AS numbers used on autonomous systems elsewhere in the Internet, because the AS numbers used inside the Confederation never leave the Confederation.



Configuring Confederations

When a router is within a sub-AS of a Confederation, then the AS number used by its BGP process is the AS number of the sub-autonomous system.

Hence, the AS number in the command: `router BGP <AS Number>` is the AS number of the sub-autonomous system. But, the router does also need to know the AS number of the overall AS that its sub-AS is located within. This is because, if the router has any eBGP neighbors, then the AS number it puts into the AS_Path attribute of any routes it sends to eBGP neighbors must be the AS number of the overall AS, not the AS number of the sub-AS. Routers external to the Confederation should be kept completely ignorant of the existence of the sub-ASs in the Confederation.

The router needs a command to tell it the AS number of the overall AS that its sub-AS resides within:

```
bgp confederation identifier <AS Number of the overall AS>
```

The router also needs to know who all the other sub-ASs within its Confederation are. This is because, as we will see a bit further on, the way a router updates the AS_Path attribute of routes it sends by eBGP to a fellow Confederation sub-AS is slightly different to what is does when sending a route to a truly external AS. The command to inform the router of the other sub-ASs within the Confederation is:

```
BGP confederation peers <list of the AS Numbers of the other sub-ASs in the
confederation>
```

Attribute usage in Confederations

As with Route Reflectors, the Confederation approach to operating iBGP networks needs a way to prevent routing loops. And, as always, the solution lies with attributes.

In the case of Confederations, the attribute that is used for loop prevention is the *AS_Path* attribute. This is quite natural, as the AS_Path is the attribute that is used for prevention of routing loops between autonomous systems in the wider Internet.

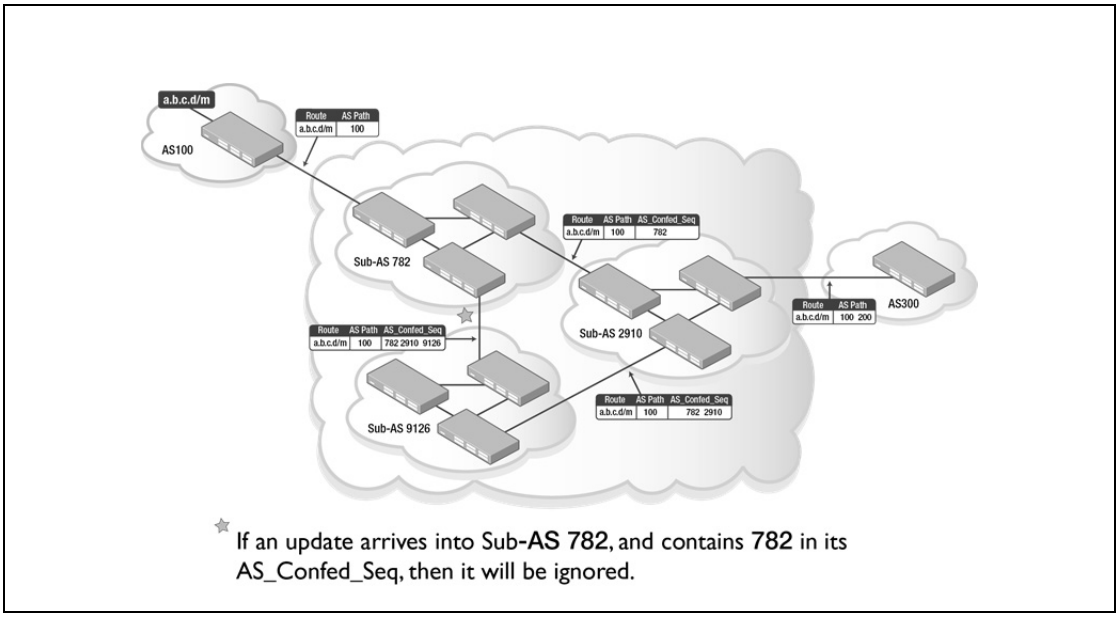
However, the use of the AS_Path attribute for prevention of loops within a Confederation does require a bit of modification to the structure of the attribute.

Specifically, Confederations introduce two new segments to the AS_Path attribute:

1. the Confederation **Sequence** segment, referred to as the AS_CONFED_SEQUENCE
2. the Confederation **Set** segment, referred to as the AS_CONFED_SET

These two new segments are used in absolutely the same way as the existing Sequence and Set segments within the AS_Path attribute, but they are reserved for exclusive use by the AS numbers of Confederation sub-ASs. The rules are:

- When a router advertises a route within its own sub-AS, it does nothing to the AS_Path attribute.
- When a router advertises a route to a router in another sub-AS of the same Confederation, it puts its sub-AS number into the AS_CONFED_SEQUENCE segment of the AS_Path attribute.
- If a router in a Confederation sub-AS aggregates together routes that already have values in the AS_CONFED_SEQUENCE segments of their AS_Path attribute, then it adds an AS_CONFED_SET segment to the AS_Path attribute. It puts the collected set of values from the aggregated routes' AS_CONFED_SEQUENCE segments into this AS_CONFED_SET segment.
- If a router receives a route, and its own sub-AS number appears in the AS_CONFED_SEQUENCE or AS_CONFED_SET segment of the route's AS_Path attribute, then it knows the route has been around a loop, and drops it.
- If a router is sending a route to a router that is in a completely external AS, then any vestige of Confederateness is removed from the route's AS_Path attribute. If AS_CONFED_SEQUENCE and/or AS_CONFED_SET segments are present in the AS_Path attribute, they are removed before the route is advertised to the external peer. The AS number of the overall Confederation AS is added to the standard Sequence segment of the AS_Path attribute, as per normal. Therefore, to routers in external ASs, there is no sign whatever of the whole Confederation business going on inside the current AS.



BGP peering between sub-ASs

As has been alluded to above, the eBGP peering between routers in different sub-ASs within a Confederation is **almost** like a normal eBGP peering. The differences are:

- The AS_CONFED_SEQUENCE and/or AS_CONFED_SET segments of the routes' AS_Path attributes are used to track the sub-ASs that the routes have passed through; rather than using the standard Sequence and Set segments of the AS_Path attribute.
- The Next Hop attribute is handled differently, for more on this see the section below.
- But, apart from that, the connections are just like eBGP connections. Even the TTL of the packets defaults to 1.This type of peering is often referred to as **c**BGP (Confederation BGP) peering, or sometimes **ei**BGP peering.

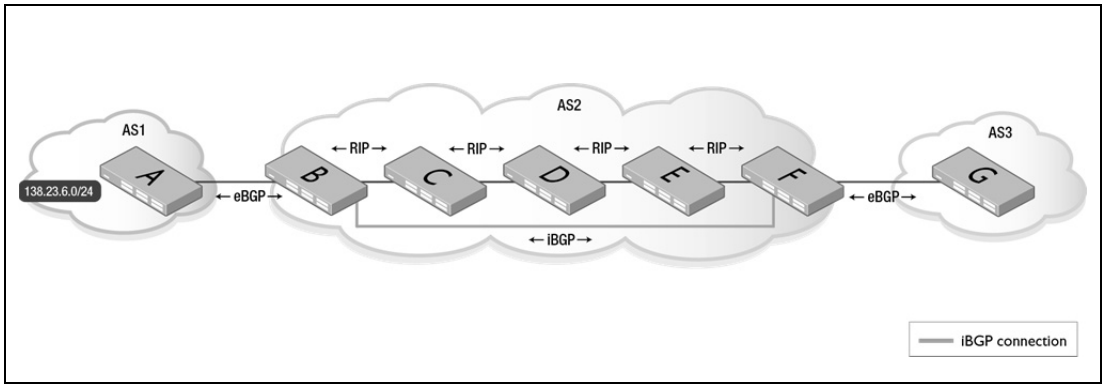
IGP Synchronization

Another feature that is specific to iBGP is IGP synchronization.

This is a feature that is specifically required in the case where routing information is 'transiting' through an AS. That is, when a BGP router on one side of the AS learns routes from external ASs by eBGP, this router passes the routes across the AS by iBGP to a BGP router on the other side of the network and the BGP router on the other side of the network then sends the routes by eBGP to other external ASs.

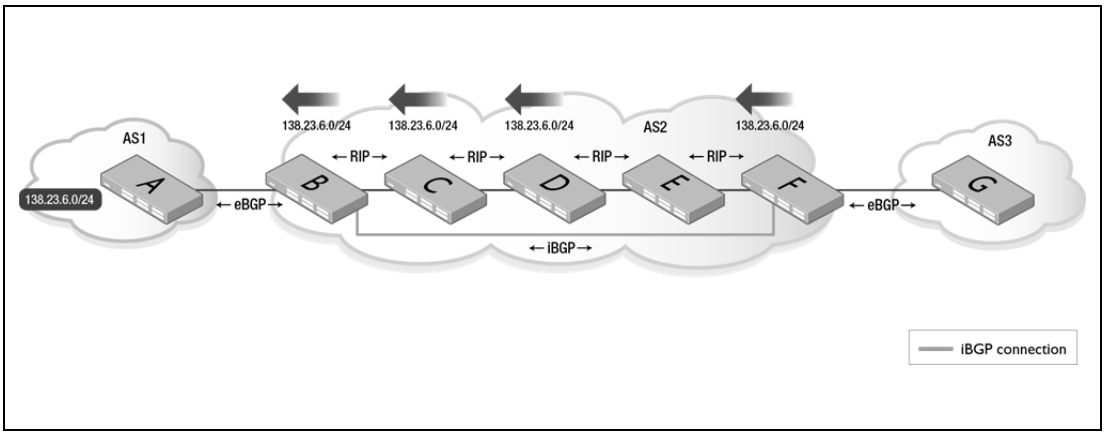
A problem can arise in this situation. Because the iBGP session effectively tunnels route information across the AS, but packets destined to any of those routes have to cross the AS hop-by-hop.

Consider the simple network illustrated below:



1. Router B will learn the route 138.23.6.0/24 from Router A.
2. Then, it will advertise this route to Router F by iBGP, and at the same time, advertise it to Router C by RIP.
3. Once Router F receives the route, it could pass it on, by eBGP, to Router G.
4. Also, Router C will be passing the route by RIP to Router D, who, in turn, will pass it on to Router E, and then to Router F.

But, if the BGP advertising is **faster** than the RIP advertising, we could, for a while, have a situation like that illustrated below:



Routers A, B, C, D, F, and G all have routes to 138.23.6.0/24, but Router E does not have the route yet.

If, at this point, Router G transmitted a packet towards a host in the subnet 138.23.6.0/24, that packet would forward to router F. Router F would then send it to Router E.

But, then we hit a problem. Router E does not have a route to 138.23.6.0/24. It might drop the packet, or might even send the packet back to Router F (if, for example, its default route is via Router F).

This is an undesirable state of affairs.

To avoid this, we need to tell Router F that even if it receives the route for 138.23.6.0/24 via the iBGP 'express tunnel', it must wait until it has ALSO received the route via the more pedestrian IGP hop-by-hop method before it can go passing that route on to eBGP neighbors.

This process of forcing a router to wait until a route has been learnt by IGP as well as iBGP is call **IGP synchronization**. In other words, the routes that the router has leant by iBGP must match up with the routes that it has learnt by IGP. Once a route has achieved this sync-up, it is then OK to advertise it on to eBGP neighbors.

The command to turn on IGP synchronization on a router is simply **`synchronization`** in Router BGP mode. IGP synchronization is disabled by default. It does not need to be enabled in an AS where there are no iBGP connections, or in an AS where all the internal route advertising is done by iBGP (i.e. when there is no other IGP in use in the AS).

Next Hops

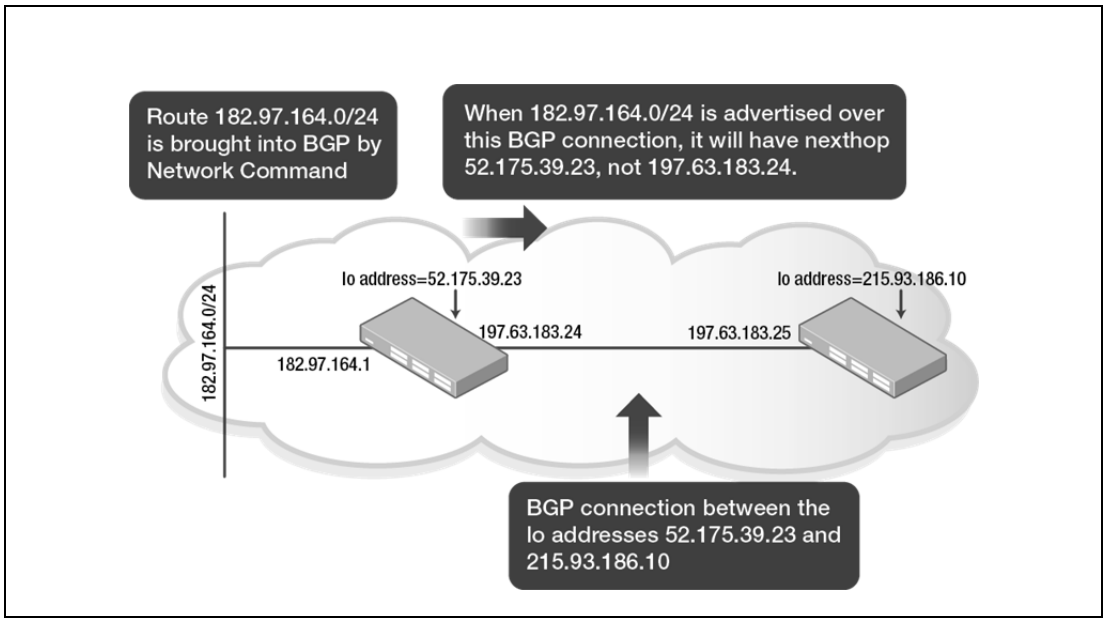
In most routing protocols, working out the next hop for a route is a simple matter. But, in BGP, because the device advertising the route to you is not necessarily directly connected, the determination of a route's next hop is not quite so straightforward.

The next hop address for a route is carried with the route in a **Next Hop** attribute. There are conventions for how the content of this attribute should be managed in various circumstances.

The conventions are:

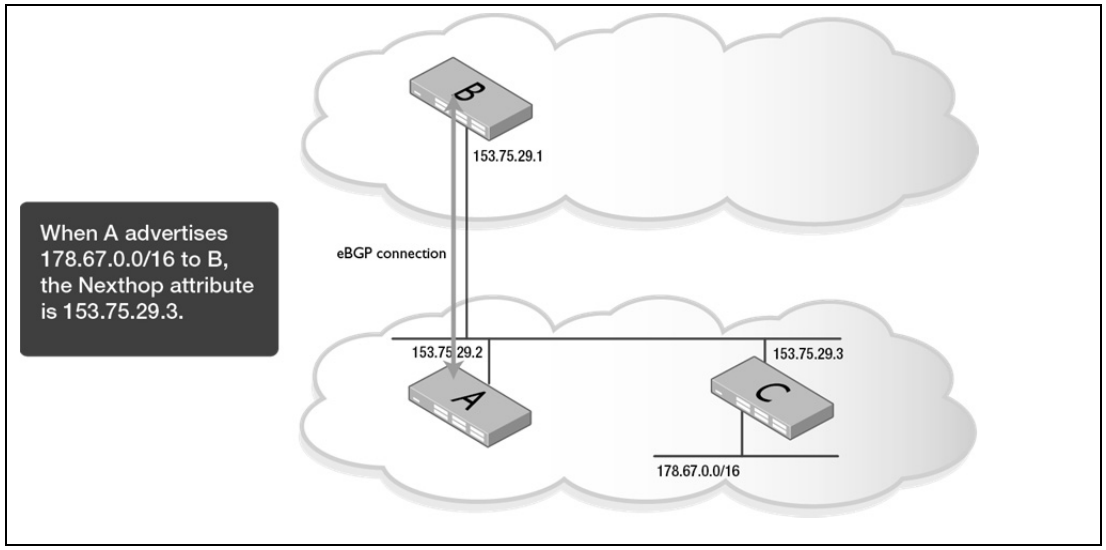
1. For aggregate routes that are created by the BGP router, or connected routes that the router has brought into BGP using the **network** command, the next-hop attribute is set to the source IP of the BGP packets it sends in the given BGP connection. It is not necessarily the IP address on the egress interface via which the BGP session egresses the router; rather it is the address that the device at the other end of the session sees as being the neighbor address of the connection.

   For example, if you are using the loopback address as the source address of the BGP session, then the loopback address will be the next hop address for routes that the router itself is originating.



2. In the case where there is an eBGP connection, and other IGP routers are in the same subnet as that used for the eBGP connection, then the next hop for any routes learnt from those subnet-sharing IGP routers is the IP address of the IGP router.

   For example, in the diagram below, the BGP router A learns the route 178.67.0.0/16 from Router C via an IGP. When Router A then advertises that route to Router B via eBGP the next hop address will not be A's address, but will be C's address (153.75.29.3):



The reason for this choice is that it is more efficient for B to send data directly to C for forwarding to 178.67.0.0/16 hosts. If A advertised its own address as the next hop, then it would have to be forever Layer 3 forwarding packets on to C, for no real benefit.

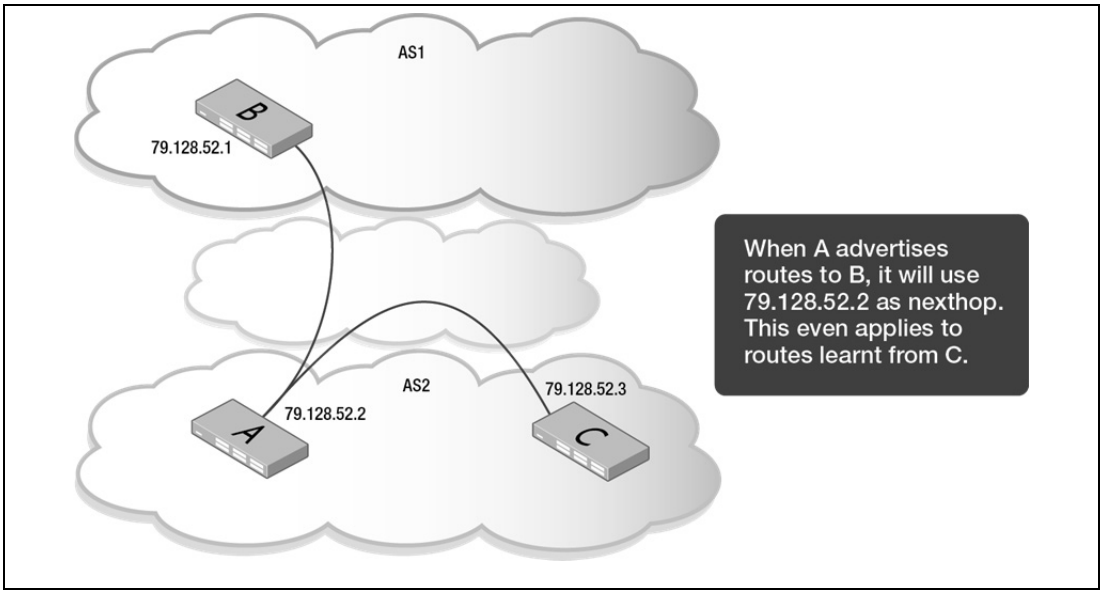
3. In the case that the BGP neighbors, and the originating router of a route, are all connected to a non-fully meshed WAN connection (like a non-fully meshed Frame Relay network), then a different approach is needed.

   For example, in the diagram below, Router B does not have a direct connection to Router C. So, if B is given C's address as the next hop for a route, then that will not be very useful for B.

   In fact, B just needs to be given A's address as the next hop. In this case, A could be configured to put its **own** IP address as the next hop for routes it sends to B.

   The command, in router BGP configuration mode, is:

   ```
   neighbor <neighborid> next-hop-self
   ```



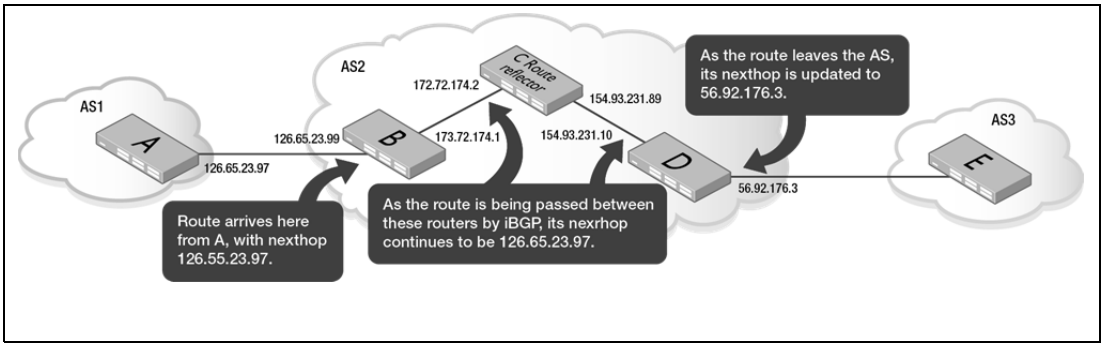
4. In general, the next hop attribute that a route has when it enters an iBGP network is preserved throughout its journey across the iBGP network. The next hop attribute is generally not updated when a route is advertised over an iBGP connection.

   This is because it is assumed that all the routers in an AS have routes to the same set of subnets (which are advertised within the AS by an IGP). When a router learns a route via eBGP, and advertises it into the AS by iBGP, it can assume that because it has a route to the route's next hop, all devices inside the AS will as well.

   As an example, consider the diagram below:

   - A advertises a route to B. The next hop attribute on this route will be 126.65.23.97.
   - When B advertises the route on to C (the Route Reflector), the next hop attribute will remain as 126.65.23.97. B will not change the next hop to 173.72.174.1.
   - Similarly, when C reflects the route on to D, it will not change the attributes.

- When the route arrives at D, the next hop attribute is still 126.65.23.97. So, C and D will certainly need to have learnt routes to 126.65.23.97 via the IGP that is operating within AS2.



5. The internal structure within an AS should not really be the concern of routers outside the AS. When a route emerges from the other side of an AS, and is advertised on to an external AS, the route's next hop attribute is changed to be that router's own IP address (source address of the eBGP session).

   For example, in the diagram above, when D advertises the route on to E, who is in another AS, D will update the next hop attribute to be 56.92.176.3.

6. As an alternative to convention (4), it is not uncommon for the router that first advertises the route into the AS will update the next hop attribute of the route to be its own address, so that routers within the AS are not affected by changes in the route to the external next hop address.

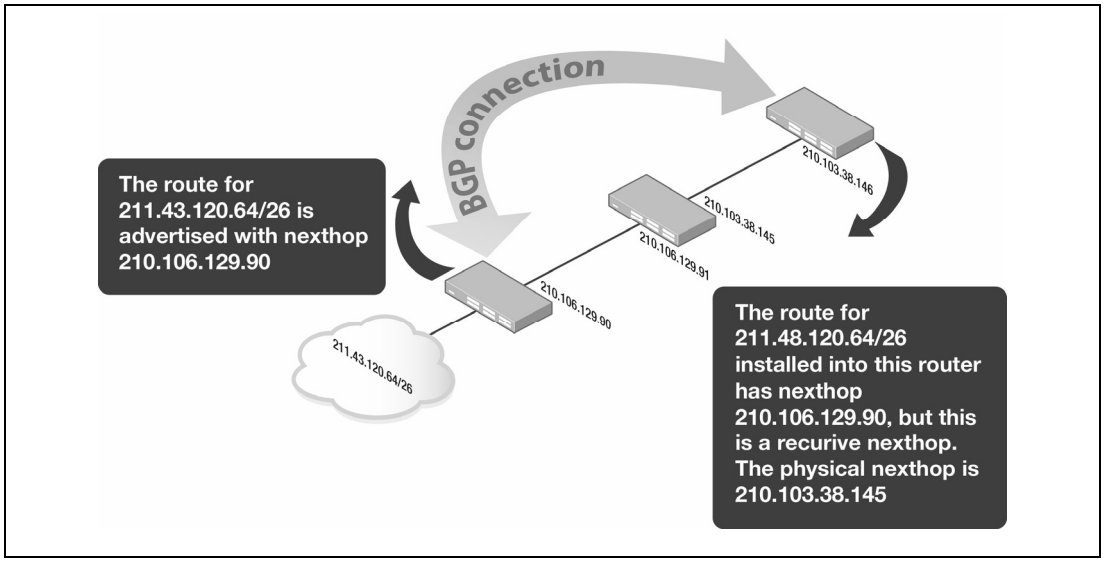
   Again, this is achieved by using the command:

   ```
   neighbor <neighborid> next-hop-self
   ```

   on the router where the route enters the iBGP network.

Recursive routes

Note that when a router learns a route by BGP, and the next hop attribute of the route is not an IP address in the same subnet as the interface that is receiving the BGP session, then the router needs to use recursive next hop resolution. The reality is that when a router forwards a packet, the next router that it forwards the packet to **has** to be in the same subnet as the packet's egress interface. So, even if the router receives a next hop address that is in some subnet three hops away, it still needs to work out the IP address of the **actual** next router to which it forwards packets that are following the route. What it has to do is perform a route lookup to the address of the next hop attribute. The next hop route on that route is the actual next router that we are looking for. This address is the recursively derived next hop for the route.



In the output of **show IP route** a recursive route appears as follows:

```
B 211.43.120.64/26 [20/0] via 210.106.129.90, vlan291 (recursive via 210.103.38.145)
```

Route Flap Dampening

If an interface is going up and down somewhere in a network, and the connected route on that interface is being advertised by BGP, then route flapping will ensue. When the interface is UP, the route will be active, and BGP routers will advertise it to each other as an active route. When the interface goes down, the route becomes unavailable, and BGP routers will tell each other to withdraw the route.

Routers, understandably, become tired of this repeated "*this route is available*", "*no, it isn't*" carry on. It wastes CPU time, and will be annoying to users if their packets occasionally get through to the destination, and occasionally do not. In general, people prefer a hard failure to tantalizing glimpses of connectivity that keep getting snatched away.

The solution

A feature has been developed within BGP that helps reduce the effect of flapping routes. This feature is referred to, quite intuitively, as **Route Flap Dampening**.

The way that Route Flap Dampening works is as follows:

If a router is configured to perform Route Flap Dampening, it keeps track of every route it has learnt from eBGP neighbors. Each route has a **Flapping Penalty value** associated with it. This penalty value is initialized to **zero**.

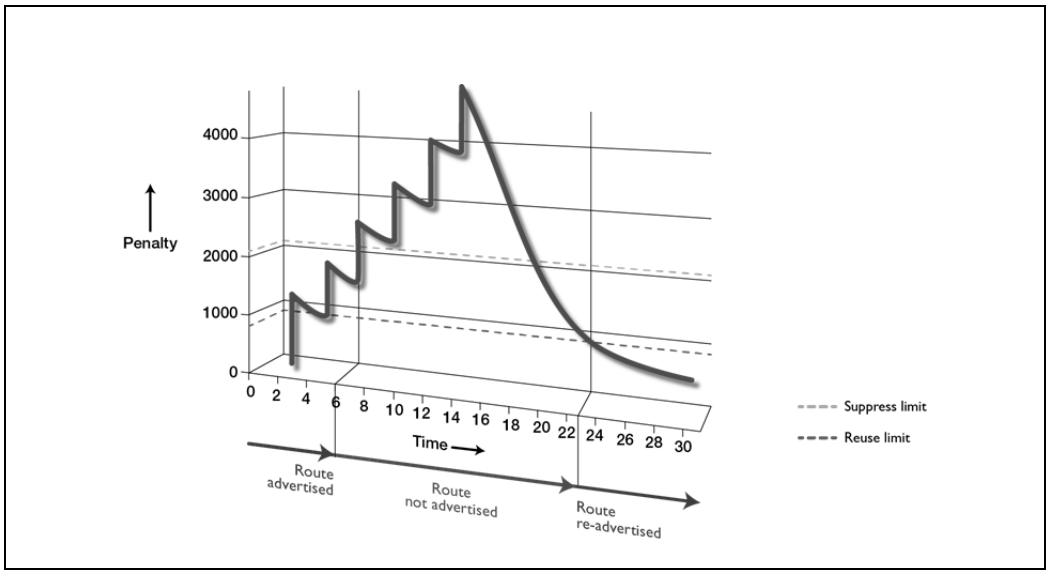
Each time a route is withdrawn, due to receiving a withdraw request from a neighbor, the penalty value is increased by 1000. Although the route is removed from the BGP route table when the withdraw event occurs, BGP still keeps a record of this route, so that its flap penalty value is remembered. There is no addition to the penalty when the route is learnt again; the penalty is only incremented each time the route is withdrawn.

If the penalty value on a route gets above a certain value, called the **Suppress Limit**, then the router stops advertising the route. The **default** value for the Suppress Limit is **2000**.

There also needs to be a method for bringing a route back into service if is behaves itself (stops flapping) for a while. The method for achieving this is the gradual decay of the penalty value. The value of the penalty on each route is exponentially reduced. That is, the rate is not reduced steadily, but is reduced at an ever-changing rate such that the value is halved every 15 minutes. The time period over which the penalty value is reduced to a half is called the **half-life**.

When the value of a route's penalty gets down below a value, called the **reuse** limit, it is allowed to be advertised again. The default value for the reuse limit is 750. Of course, each time the route flaps, 1000 is added to the penalty, and the decay process finds itself operating from a new starting value.

As flapping occurs, the value of the penalty will typically go up in a saw-tooth fashion, as in the diagram below.



When the flapping stops, the penalty is allowed to decay in an unmolested fashion, and the route is eventually brought back into service.

There is also a maximum suppress time. If a route has been in the suppressed state for this length of time, it is automatically brought back into service. By **default**, this time is **4 times the half-life**. Given that the default half-life is 15 minutes, which means that the default maximum suppression time is 60 minutes.

Configuring route dampening

In AlliedWare Plus, BGP route dampening is enabled with the command **bgp dampening** in router BGP mode.

Also, the half-life, reuse limit, suppress limit, and max suppress time can be configured by adding these values as parameters to the **bgp dampening** command.

The syntax is:

```
bgp dampening <reachtime> <reuse> <suppress> <maxsuppress>
```

Where:

- *reachtime* is the half-life which defaults to 15 minutes, and can be configured to be anything from 1 to 45 minutes.
- *reuse* is the reuse limit, which defaults to 750, and can be configured to anything from 1 to 20000.
- *suppress* is the suppress limit, which defaults to 2000, and can be configured to anything from 1 to 20000.
- *maxsuppress* is the maximum suppression time, which defaults to 4 times the half-life, and can be configured to anything from 1 to 255 minutes.

The state of dampening can be viewed by using some **show** commands:

show ip bgp dampening shows the current settings for the parameters listed above.

show ip bgp dampening flap-statistics shows a list of routes that have been known to flap, or are currently being damped. It shows how many times each route has flapped (has been withdrawn) and the period over which it has been flapping.

show ip bgp dampening dampened-path shows a list of routes that are currently being dampened.

BGP and Route Maps

Route maps, while not inherent to the BGP protocol, go very much hand-in-hand with BGP. The great value of BGP is the way that it provides such scope for controlling which routes are advertised where, and which routes are preferred, etc. It is route maps that enable you to make use of BGP's route control capability.

You can use route maps for all sorts of purposes in BGP:

- filtering which routes will be advertised to a neighbor
- filtering which routes will be accepted from a neighbor
- altering the values of attributes on routes learnt from a neighbor
- altering the values of attributes on routes as they are advertised to a neighbor
- filtering which routes will be redistributed from another routing protocol
- altering the values of attributes on routes as they are being redistributed from another routing protocol

- specifying which routes dampening should be applied to
- specifying which routes must be present before a default route will be advertised
- applying attributes to a route being brought into BGP by the **network** command

Route maps are a combination of filters and attribute modifiers. The highly configurable selection criteria that define the routes a route map matches on, and the range of attributes they can modify, mean that they are a key to enabling the selective route manipulation that is a very attractive feature of BGP.

Tweaking attribute-related rules

In the section Attributes on page 137, there are descriptions of various rules regarding how attributes should be used in the algorithm for finding the best route to a destination.

For example, there is a rule that the AS_Path is one of the principal attributes used in deciding a best route; another rule is that the MED attribute is used for comparing between routes advertised from one AS, and so on. However, it is possible that in some networks, due to unusual network designs, or historic choices that have somewhat painted the network into a corner, the network administrator needs to break these rules. There are a set of BGP commands that configure a BGP router to break the normal rules for deciding on the best route to a given destination.

These are commands like:

- `bgp always-compare-med`
- `bgp bestpath as-path ignore`
- `bgp bestpath compare-confed-aspath`
- `bgp bestpath med {[confed] [missing-as-worst]}`
- `bgp deterministic-med`

Capabilities

Over time, the BGP protocol has evolved, and new functionality extensions have been added to the protocol. A number of these extensions require some form of cooperation between BGP neighbors.

If a BGP router is running a BGP implementation that supports one or more such extensions, it is useful for that router to know which, if any, extensions its neighbors support. It would be pointless to try to use an extended feature within the connection to a given neighbor unless you know that this neighbor also supports this extended feature.

Some of the extended features are:

Router Refresh – a BGP router can send a ROUTE_RFRESH packet to a neighbor, to request that the neighbor re-advertise its routes.

Outbound Route Filter – a BGP router can send a neighbor a set of route filters. The router that receives these route filters then must apply these filters to the routes that it sends to the neighbor that sent the filters. In other words, the neighbor that sends the filters is saying, "*Before you send any routes to me please run them through these filters*".

Graceful restart – a BGP router can say to its neighbors:

"*I am going to restart my BGP process, which will mean that my connections to you neighbors will go down. But, please act as though I have not gone away – i.e. do not remove the routes that you learnt from me – because my IP routing will continue to operate, and my BGP process will be up and running again very soon.*"

4-byte AS numbers – as the number of autonomous systems in the Internet has grown; a 2-byte number has not been enough to enumerate them all. In response, a 4-byte AS numbers system has been introduced. But this has required changes to BGP implementations, so that they can handle 4-byte AS numbers in:

- The My AS field of OPEN messages
- AS_Path attributes
- The Aggregator attribute
- Within Community identifiers

Multiprotocol BGP – the ability for BGP to carry route information for protocols other than IPv4.

When two BGP routers establish a connection with each other, their OPEN messages can contain a list of these extended capabilities that they support.

If the routers find that they do not support the same set of capabilities, they can continue to establish their peering connection, or they can terminate the session right there. Under AlliedWare Plus, the command that determines whether the router will accept capabilities mismatches or not is:
neighbor <*neighborid*> strict-capability-match

By default, an AlliedWare Plus router will accept a capability mismatch. But, if it is configured with the command above, then when it detects that is has a capability mismatch with a neighbor, it will simply send that neighbor a Notification message, close the connection, and not try to re-establish the connection.

To deal with the case that a neighbor router will not accept a capability mismatch, it is possible to configure an AlliedWare Plus router to not advertise one or more of its extended capabilities.

The following commands configure an AlliedWare Plus router to not advertise the specified capability to the specified neighbor:

- `no neighbor <neighborid> capability route-refresh`
- `no neighbor <neighborid> capability orf prefix-list`
- `no neighbor <neighborid> capability graceful-restart`
- `no neighbor <neighborid> capability dynamic`

There are some other commands that control the advertising and accepting of capabilities.

dont-capability-negotiate

If one or more of your neighbors were running very early implementations of BGP that don't support the concept of capabilities being advertised in the OPEN packet, then, if you advertise capabilities to those neighbors, the neighbors will invariably send back a Notification message, and close the session. At that point, your router can take note of the fact that the neighbor's notification message included a code to say that the thing it was objecting to is the presence of an unknown option in the OPEN packet. In reaction, your router can then retry the session, but this time without sending any capability options in the Hello message.

OR, if you know in advance that one or more neighbors will not accept capability options in OPEN packets, you can simply configure your router to not send them to those neighbors.

The command to configure this is:

```
neighbor <neighborid> dont-capability-negotiate
strict-capability-match
```

If you desire that you will not tolerate a capability mismatch with a given neighbor, then you can configure your router to terminate the connection with that router if its advertised capability list does not exactly match yours. The command to do this is:

```
neighbor <neighborid> strict-capability-match
```

override-capability

Even though a given neighbor advertises it has a given capability, you may decide that you do not want to make use of that capability on that neighbor (maybe you know that the neighbor's implementation of the capability is flawed).

In fact, the following commands can have another significance:

- `no neighbor <neighborid> capability route-refresh`
- `no neighbor <neighborid> capability orf prefix-list`
- `no neighbor <neighborid> capability graceful-restart`
- `no neighbor <neighborid> capability dynamic`

As well as preventing our router from advertising these capabilities to a neighbor, they can also indicate that we do not wish to make use of a neighbor's support for these capabilities.

The command that brings this second meaning into play is:

```
neighbor <neighborid> override-capability
```

If this command is configured, then even if the specified neighbor does advertise the specified capability, our router will not make use of that capability with that neighbor.

Peer Groups - Making Configuration Tidier

Frequently, a number of neighbors configured on your router will be configured with a very similar set of parameters. For example, they may all be configured with the same filter-list, the same non-default advertisement-interval, the same number of eBGP multihops, etc. To save a lot of retyping of commands to set these same parameter values on each neighbor definition individually, it is more efficient to group the neighbors into a set, and apply the parameters just once, to the whole set.

BGP offers this ability, in the form a **Peer Group**.

To create a Peer Group, use the command:

```
neighbor <group-name> peer-group
```

Then, to add neighbors to the Peer Group, use the command:

```
neighbor <ip-address> peer-group < group-name >
```

Within most of the commands that apply to neighbors, it is possible to specify a Peer Group **name** instead of a neighbor IP address.

For example, in the command:

```
neighbor <neighborid> prefix-list <listname> in
```

the item *<neighborid>* could be either the IP address of a single neighbor, or the name of a Peer Group.

In fact, ease of configuration is not the only advantage that Peer Groups provide. Additionally, they provide some saving of the amount of CPU that BGP consumes. This is achieved by creating just one set of updates for all the neighbors in the Peer Group. The fact that this is possible relies on the requirement that all the members of the peer group must have the same set of outbound filters, distribute lists etc.

The BGP route table can be just taken once through the set filtering and attribute updating actions that are configured on the Peer Group, to generate the set of route updates that can be sent to all members of the Peer Group. This is more efficient than carrying out those filtering and attribute updating actions separately for each member of the Peer Group. But, do keep in mind that this is possible because of the requirement that all the members of the Peer Group must have the same set of outbound filters, distribute lists etc.

There are some other limitations on the sets of neighbors that can be combined in Peer Groups.

- Do not have iBGP and eBGP peers combined in the same Peer Group.
- If the clients of a Route Reflector are combined into a peer group, then the clients need to be fully meshed.

For configuration other than the outbound filtering/attribute updating, it is possible for different members of the Peer Group to have different configuration. An item configured specifically on a Peer Group member will override something that is configured on the Peer Group as a whole.

4-byte AS Numbers

As touched on in the section Capabilities on page 176 the number of autonomous systems in the world is growing to a point where they cannot all be enumerated by a 2-byte number.

Hence, just as address space is being extended by the transition from IPv4 to IPv6, so the autonomous system enumeration space has been expanded by a move from 2-byte to 4-byte autonomous system numbers.

When a parameter changes size like this, it is a bit of a problem for the protocols whose packets transport the parameter. BGP packets have been defined with a 2-byte location for AS numbers. How are they going to squeeze a 4-byte number into there?

Let's work through some of the changes that had to be made to BGP to accommodate the move to 4-byte AS numbers.

The OPEN packet

BGP routers have to put their AS number into the *My Autonomous System* field of the BGP OPEN packet. This field is only 2 bytes in length. If you are a BGP router that resides in an autonomous system that has been allocated a 4-byte AS number, what do you do?

The answer is:

- The value put into the *My Autonomous System* field of the OPEN packet is a special reserved 2-byte AS number, with the value 23456. This reserved AS Number is referred to as AS_Trans.
- The actual 4-byte AS number is put into a Capability option later in the OPEN packet. The capability type 65 is the 4-byte AS number capability. The value carried in the Type-65 capability is the actual 4-byte AS number.

When the router connects to a peer that does not know about 4-byte AS numbers, it just sees the router's AS number as being 23456. When the router connects to a peer that does know about 4-byte AS numbers, that peer knows to look into the Type-65 capability field to find the real AS number.

Note - if a BGP router resides in an AS that has been allocated a 4-byte AS number that is '2-byte mappable' i.e. a number that is less than 65536, so it will actually fit into 2 bytes, then the router puts that actual AS number into the *My Autonomous System* field of the OPEN packet; it does not put AS_TRANS into that field. It should, though, still carry a 4-byte AS Number capability option in the OPEN packet, to inform peers that it is a 4-byte AS number capable router.

The AS_Path

The AS_Path is, of course, one of the most important attributes that a route carries, as it is used for detecting routing loops, and is also a significant parameter in the choice of the best route to a destination. It is important that the managing of AS_Path attributes is well modified to handle the transition to 4-byte AS numbers. In fact, a compromise has been designed, that achieves a good, but not complete, handling of the transition to 4-byte AS numbers.

The main feature that has been added to BGP to enable AS_Paths to work in a mixed 2-byte/4-byte AS number world is a new attribute called the AS4_Path. As the name implies, this is an AS_ Path attribute that carries 4-byte AS numbers.

With the introduction of this attribute, the actions that a 4-byte capable router needs to carry out are as laid out below.

When advertising a route, the router needs to:

- Append AS_Trans to the AS_Path attributes on the route. (Or, if the router's AS number does actually fit into 2 bytes, then it just appends its AS Number to the AS_ Path).
- Append its full 4-byte AS Number to the AS4_ Path attributes on the route. The exception to this is when ALL the autonomous systems that the route has passed through (including the current one) have 2-byte AS numbers. In this case, there will not be an AS4_ Path associated with the route, and the current router must not go and create one. In other words, if a route does not absolutely need an AS4_ Path attribute, then don't give it one.

When receiving a route, the router needs to:

- Read in the AS_ Path and (if it exists) the AS4_ Path attributes associated with the route.
- Do its best to reconstruct the full set of ASs that the route has passed through. This can sometimes be simple, and sometimes impossible.

- If all the that the route has passed through so far have 2-byte AS numbers, then the route will not even have an AS4_ Path attribute, so its AS_Path is completely represented by the AS_ Path attribute. In this case, there is no reconstruction task required.

- If all the routers that the route has passed through are 4-Byte AS number capable, then the AS4_ Path will exactly represent the journey that the route has taken. In this case, there will be a one-to-one correspondence between the items in the AS_ Path, and those in the AS4_ Path. Each entry in the AS4_ Path that is a number greater than 65535 will be matched by an entry in the AS_ Path that has value 23456. Each entry in the AS4_ Path that is a number less than 65535 will be matched by an entry in the AS_ Path that has the same value. In this case, reconstructing the actual AS_Path that the route has passed through is trivially simple.

- If the route had been through one or more autonomous systems with 4-byte AS numbers, then through a series of routers that are not 4-byte AS number capable, then things get a bit more interesting.

  In this case, the AS4_ Path will match the AS_ Path up to a point, but after that, the remaining routers will have simply passed the AS4_ Path attribute along unchanged (it is a transitive attributes, so routers that do not understand it will still pass it along), while the AS_ Path will have been getting updated.

  In this case, the router needs to take the latter part of the AS_ Path attribute, and retrospectively append that onto the AS4_ Path attribute, to get the full AS history of the route. What is more, when the route is readvertised by this router, the AS4_ Path attribute associated with the route must be this reconstructed AS4_ Path (with the current routers AS number appended to it, of course).

  The really difficult situation arises when a router that is not 4-byte AS number capable has aggregated two or more routes that carried AS4_ Path attributes. In this case, the router doing the aggregating will not understand the content of the AS4_ Path attributes, and will not perform and action to aggregate those attributes. It will not be unlikely for some loss of information about the set of autonomous systems with 4-byte AS numbers that the route has passed through. The RFC that defines how BGP should deal with 4-byte AS numbers (RFC 4893) details how a router should do its best to handle this situation; we won't go into those details here.

The Aggregator attribute

The Aggregator is another attribute that carries an AS number. The advent of 4-byte AS numbers has required some action to be taken with regard to this attribute.

Again, a 4-byte version of the attribute has been defined – the AS4_ Aggregator attribute.

If a router in an AS with a 4-byte AS number aggregates some routes, then the router will:

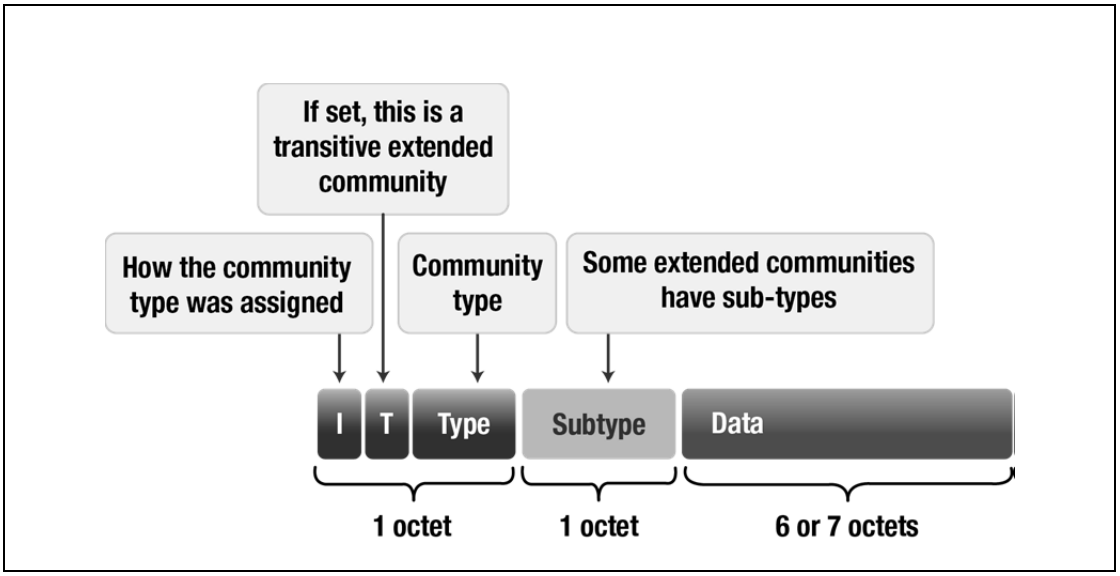
- Put the value AS_Trans into the aggregator attribute on the aggregate route. (Or, if the value of the 4-byte AS number is less than 65535, it will just put the AS number into the attribute).
- Attach an AS4_Aggregator attribute to the aggregated route. The AS number in this attribute is the 4-byte value.

The AS4_ Aggregator attribute is transitive, so routers that are not 4-byte AS Number capable will pass the AS4_ Aggregator attribute along.

Community attributes

The first two bytes of a Community attribute are used to hold the AS number of the autonomous system that defined the attribute.

However, if you have a 4-byte AS number, then these first 2 bytes of the Community attribute are not going to be enough to hold the AS number. Hence a specific type of Extended Community Attribute has been defined, to carry Community attributes that are defined within autonomous systems with 4-byte AS numbers.



Address Families

Whilst BGP was originally developed for the advertising of IPv4 routes, it was soon recognized as a perfectly good mechanism for advertising any type of route. Effectively, the Update packet says "*Here is some information about the reachability of some addresses, and some attributes associated with this reachability information*". There is nothing inherently IPv4-oriented about this, so it is not a difficult job to extend the BGP protocol to carry routing information about IPX or AppleTalk, IPv6, etc.

The extension to enable BGP to become Multiprotocol BGP (**M-BGP**) is essentially just the addition of two new attributes – one that carries information about reachable generic routes, and one that carries information about unreachable generic routes (i.e. an announcement that a route is being withdrawn).

These attributes carry information like:

- which protocol the route belongs to
- the route purpose (unicast forwarding, multicast forwarding, tunneling, etc.)
- the address of the next hop router for this route
- the network address of the route itself

Now, the numbers that identify which protocol a route belongs to are referred to as **Address Family** identifiers. As a result, the term 'Address family' has been used in the commands that are used for configuring BGP to carry different protocols. This, in turn, has resulted in Address Family becoming the term people use to refer to advertising of routes for a specific protocol, or even with the advertising of different segregated sets of routes belonging to one protocol.

Address Families is a bit of an odd term for these activities, but that is the term that has evolved. Let's look at the elements that make up a BGP Address Family.

Configuration

The first, most obvious aspect of an Address Family is the configuration. A command like **address-family ipv4 vrf** <*vrf-name*> starts a section of BGP configuration that is specific to this particular address family. All `network`, `peer`, `redistribute`, `aggregate`, etc. commands between the `address-family` command, and a subsequent `exit-address-family` command, apply only to this Address Family.

Separate BGP route tables

The act of configuring an Address Family will cause BGP to create a separate BGP route table for that Address Family. The routes that are brought into BGP via **network** or **redistribution** commands within the configuration for that Address Family will be held within the route table that BGP has created for this Address Family. Routes that are brought into BGP, by **network** commands or **redistribute** commands configured within that Address Family will only be installed into that Address Family's BGP route table. The routes that are advertised to the peers that are configured within that Address Family will only be routes from that Address Family's BGP route table. Routes that are learnt from peers in that Address Family will only be stored into that Address Family's BGP route table.

Identifiers in the Update packets

As described above, M-BGP defined a new attribute to carry routing information. One of the fields in this attribute is the "*Which protocol the route belongs to*" field. The content of this field is the address-family identifier for the protocol that the route belongs to. These identifiers are listed at:

http://www.iana.org/assignments/address-family-numbers/address-family-numbers.xml

A further field in the attribute is the "*The purpose the route is used for (unicast forwarding, multicast forwarding, tunneling, etc.*)" field. The content of this field is referred to as a "*Subsequent Address Family Identifier*" (SAFI). The SAFI numbers are listed at:

http://www.iana.org/assignments/safi-namespace/safi-namespace.xml

For example, IPv4 routes to be used for multicast purposes have an Address Family ID of 1 and a SAFI of 2.

Usage of Address Families

A common use of Address Families is for the separation of routes belonging to different Virtual Routing and Forwarding (VRF) instances. Because BGP keeps the routes within different Address Families in separate BGP route tables, this works in well with the need to separate the routes belonging to different VRF instances.

If the VRF-specific Address Family is used for IPv4 routes, then the routes are not transported in special M-BGP route attributes; they are just transported as normal routes in an Update packet. No special Address Family identifier is required with these routes, as they are just normal IPv4 routes. The peers configured within the Address Family configured for a given VRF instance must be attached to interfaces that are also within that VRF instance.

Another popular (although not yet supported in AlliedWare Plus) use of Address Families is for the advertising of routes to be used for multicast purposes. These are not routes along which to forward multicast packets, but rather they are the routes that multicast routers use to decide the RPF interface back towards the source of a multicast stream.

A network administrator may wish that multicast packets traverse different links to those used for unicast. If the multicast-capable routers in a network have multiple routes back to the sources of multicast streams, but are explicitly told which route to use when determining the RFP interface, then it becomes possible to force the multicast distribution trees in a network to be confined to certain links.

BGP supports the concept of an IPv4 multicast address family. The routes to be used for this Address Family are kept in the 'IPv4 multicast' BGP route table, and are transported in M-BGP attributes, with Address Family ID of 1 and a SAFI of 2. Then, when peers receive these routes, they will put these routes in their own 'IPv4 multicast' BGP route table; and will tell multicast routing protocols to use these routes for determining RPF interfaces towards multicasts sources.

Packet Formats

The following section describes the BGP packet formats.

BGP header

Marker

The marker is used for authentication. If there is no authentication being used in the session, the marker is just 16 bytes of FF. But, otherwise it is a value that is calculated using authentication information. The algorithm used is such that a router can predict what the marker value should be in the next BGP packet it receives from peer. So, it can know that the packet is authentic, and can detect if packets have been lost.

Length

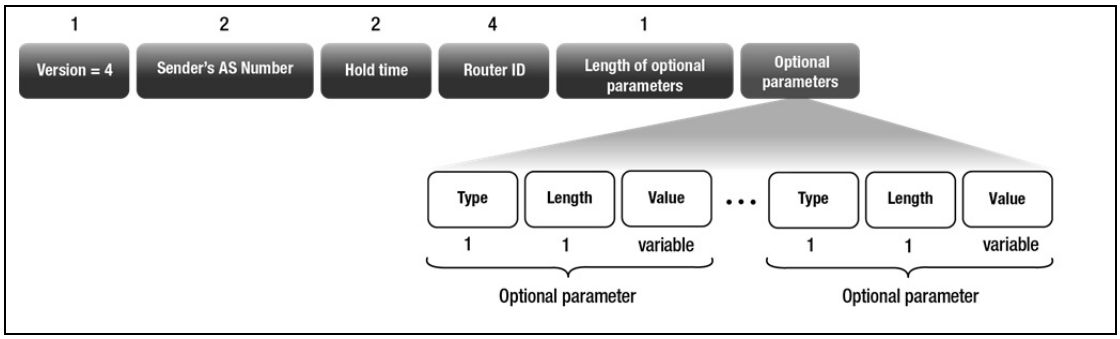
The length is the full length of the current BGP message (which might be split over multiple packets) including the BGP header.

Type

The type of BGP message can be one of the following:

Packet type	Type value
Open	1
Update	2
Notification	3
KeepAlive	4
Route-Refresh	5

Open packet



Version: The version number is currently always 4.

Hold time: This is the length of silence on the part of its peer that a BGP router will tolerate. If the router does not receive a Keepalive or Update from the peer within the hold time, then the router closes the session. The value of the hold time is negotiated between the peers at the start of the session. They each propose a hold time. The lower value wins, and both routers must use this value for the rest of the session.

Router ID: An IP address that the router uses as its ID for BGP.

Optional Parameters length: The sum total of the length of all the optional parameters

Optional parameters: The only optional parameter type currently in use is the capabilities parameter.

The format of the Capabilities parameter is:

Type | Length | Set of Capabilities ..... |

Where the Set of Capabilities is a set of 1 or more TLVs of the form:

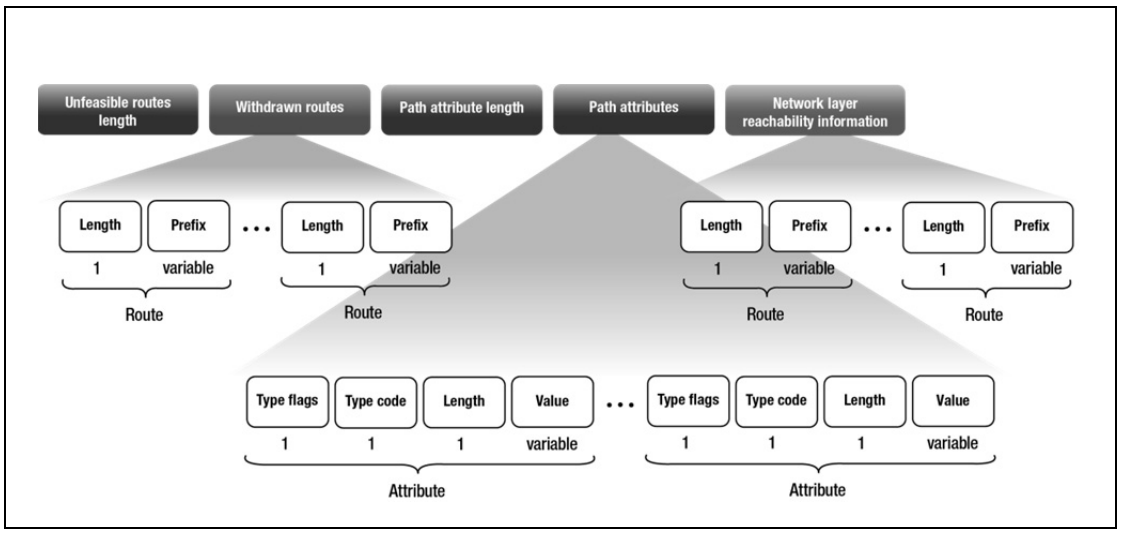
Capability Code | Capability Length | Capability Value

There are a number of capabilities that can be advertised. A list of capability codes can be found at:

http://www.iana.org/assignments/capability-codes/capability-codes.xml

Update packet

The information that the Update message delivers is generically referred to as Network Layer Reachability Information (NLRI). The acronym NLRI is often used in place of the word "*route*" when referring to the routes advertised by BGP.



WithDrawn Routes length: The total length of all the entries in the list of routes being withdrawn. If the length is 0, then this Update message is not withdrawing any routes.

WithDrawn Routes: The list of routes that are being withdrawn. Each entry in the list is effectively an IP route. The entries consist of a prefix length, and then the prefix.

Whilst IP routes written in prefix/length format are typically written in the form a.b.c.d/xx, i.e. 4 address bytes and a prefix-length byte, the trailing bytes in the address are very often zeros, e.g. 172.57.0.0/16. In this case, the zeros are basically superfluous. The first two bytes of the address, along with the prefix length, tell you all the information you need; you can interpolate the two trailing zeros of the address.

In the interests of conservation of packet space, the representation of routes in BGP packets contains only those bytes that contain non-zero bits. If the prefix length is less than or equal to 8, the prefix will be just one byte; if the prefix length is between 16 and 24, then the prefix will be 3 bytes, etc.

Total Path Attributes Length: The total number of bytes that the list of Path Attributes takes up. If the value is zero, it means there are no attributes, AND no advertised (as against withdrawn) routes in this packet.

Path attributes: The attributes in the list apply to ALL the advertised routes in the update. So, BGP has to make sure that routes with different attributes are advertised in different Update packets.

The attributes are carried in TLV fields.

The Type value of the TLV is a 2-byte quantity.

The first byte of the Type value is a Flags field.

Bit	Meaning
0	**Optional/Mandatory. 0=mandatory, 1=optional**
1	**Transitive/non-Transitive. 0=non-transitive, 1=transitive**
2	**Partial/Complete 0=Complete, 1=Partial**
3	**Extended Length 0=one-byte, 1=two-byte**

The second byte is the numerical identifier of the attribute. A list of these identifiers is at:

http://www.iana.org/assignments/bgp-parameters/bgp-parameters.xml#bgp-parameters-2

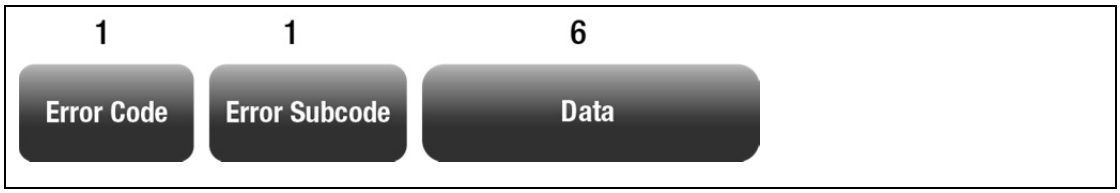
The length field in the TLV can be either a 1-byte or a 2-byte quantity, depending on the value of bit 3 of the flags byte. Then, after the length, is the attribute itself. The content of a number of common attributes is described above in the section Attributes on page 137.

Advertised routes

The advertised routes in the BGP Update packet are frequently referred to as the NLRI (Network Layer Reachability Information). As with the withdrawn routes, each route within the NLRI section of the packet is represented as a prefix length followed by all the bytes of the prefix that have non-zero bits.

Notification packet

The notification contains three fields: Error Code, Error subcode, and Data.



Error Code is a 1-byte field containing the code for the general category that the error falls into, like Header Error, Update Message Error, etc. There is a list of the error codes at:

http://www.iana.org/assignments/bgp-parameters/bgp-parameters.xml#bgp-parameters-3

Error subcode is a 1-byte field that gives a more specific indication of the nature of the error, like "*Invalid value in Message Type field*" or "*Invalid value for Origin attribute*" or "*Malformed AS_Path*" etc.

A full list of the Error subcodes is held at:

http://www.iana.org/assignments/bgp-parameters/bgp-parameters.xml#bgp-parameters-4

Data contains more information about the error. For example, if the error is an unrecognized attribute, then the Data field will contain the Attribute Code of the unrecognized attribute.

Keepalive packet

A BGP Keepalive is just a BGP header with no body.

Route Refresh packet

The purpose of the Route Refresh packet is to enable efficient updating of routing tables when ingress filters are changed.

If a BGP speaker is applying filters and route-maps to the Updates that it receives from its peers, then the content of its BGP route table will be the net result of the actions of those filters and route-maps on the incoming Updates.

But, if the filters and/or route-maps being applied to the incoming Updates are altered, then that will very likely result in alteration to the content of the BGP route table.

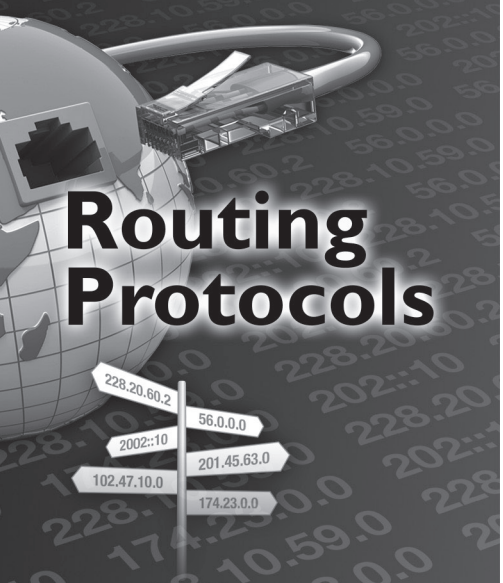
The router needs to re-receive its neighbors' Updates and run them through the new set of filters and/or route-maps, to recalculate the content of the BGP route table.

One way to accomplished this is to keep an unfiltered copy of all the prefixes that have been received from neighbors and internally re-run these prefixes through the filters and/or route-maps, after the alteration has been performed. This is a method known as 'soft-reconfiguration'. It works OK, but keeping a copy of all those received prefixes can use up a lot of memory. A more efficient method might be to just get the neighbors to resend all their Updates. This is exactly what the Route Refresh packet is designed to do. After the ingress filters and/or route-maps have been altered, the route sends Route Refresh packets to its neighbors, to say "*Please resend all your Updates to me, so I can run them through my new filters and/or route-maps*".

The Route Refresh request can specify the specific Address Family for which it wishes to receive the neighbors' Updates.

- AFI - Address Family Identifier
- Res. - Reserved field.
- SAFI - Subsequent Address Family Identifier

For more information on AFI and SAFI fields, see Address Families on page 184.



Routing Protocols

This chapter covers the following topics:

- Filtering and Altering IP Routes
- Overview of the Filter Types Available to BGP
- Configuring Distribute Filters
- Configuring AS_Path filters
- Configuring prefix filters
- Configuring Route Maps
- BGP: Applying distribute, path, prefix, and route map filters to a peer
- Applying Route Maps to Imported Routes
- Other Uses of Route Maps
- OSPF: Configuring Route Maps for Filtering and Modifying OSPF Routes
- OSFP: Applying Route Maps

CHAPTER 4

Route Filtering

Filtering and Altering IP Routes

ISPs transport large volumes of data. They often have to pay large amounts of money to transport their data through hired links, or through other providers' networks. Similarly, they can also charge money for transporting other ISPs' data through their network.

Where significant amounts of money are involved, there are typically complex negotiations involved, and agreements made that are bound by all sorts of rules and restrictions and guarantees. Hence, ISPs need to be able to very precisely control which data gets sent and received on which links. This is achieved by having very precise control over the way the routing tables in their routers are built.

To that end, the BGP implementation in AlliedWare Plus includes a set of facilities for filtering routes, and for altering the attributes that are associated with certain routes in BGP Update messages. This chapter provides an overview of these features, and touches on how to configure them. One of the central route manipulation facilities is the route map.

Route maps can also be used for manipulating OSPF routes, so this document concludes by describing the use of route maps for OSPF.

Overview of the Filter Types Available to BGP

The following sections describe the various types of filters that can be applied to BGP Updates and the hierarchy of the filters.

Filter types

There are four filter types that can be applied to the BGP Updates being exchanged between BGP peers:

Distribute filters

These use ACLs and look at the individual prefixes within an Update message. If a prefix within the Update message matches the filter criteria then that individual prefix is filtered out or accepted depending on what action the filter entry has been configured to carry out. Note that you cannot combine distribute filters and prefix filters.

Path filters

These look at the AS_Path attribute in Update messages. If the AS_Path attribute in the Update matches the filter criteria then the whole Update message is filtered out or accepted, depending on what action the filter entry has been configured to carry out.

Prefix filters

These use prefix lists and look at the individual prefixes within an Update message. If a prefix within the Update message matches the filter criteria then that individual prefix is filtered out or accepted depending on what action the filter entry has been configured to carry out. Note that you cannot combine distribute filters and prefix filters.

Route maps

These are a structured combination of match criteria and actions. They can be used to filter out routes and also to alter the attributes in Update messages.

Note - All of these filter types can be used in incoming or outgoing directions. All the filters can all be used to filter the Update packets that are received from a peer, or the Update packets which the router itself is sending to a peer.

Hierarchy of the different filters

For distribute filters (ACLs), path filters, and prefix filters, the order of application is not important. If an Update is denied by any given filter, it is discarded immediately, and is not run through any of the other filters. If an Update is permitted by one filter, it is passed through to the next filter to be considered. At the end, you end up with the set of Updates that all the filters agree should not be discarded.

However, route maps are applied last, after the other types of filter. This is because route maps can modify Updates, not just accept or discard them.

Configuring Distribute Filters

Distribute filters use ACLs (Access Control Lists) to filter particular routes on the basis of their prefixes.

Distribute filters and prefix filters both filter individual routes out of BGP Update packets. They are mutually exclusive.

About ACLs

From the point of view of route filtering, an ACL is one or more, simple unnumbered filter entries, each with a prefix and an action of deny or permit. You can use any of the following syntax options to create the ACL entries. The main difference is in how you label the ACL—whether you use a name or a number.

```
access-list standard <name> {deny|permit} <ipadd/prefixlength> exact-match
access-list <1-99> {deny|permit} <ipadd> <reverse-mask>
access-list <1300-1999> {deny|permit} <ipadd> <reverse-mask>
```

Entries are unnumbered, so each new entry gets added to the end of the ACL.

Named ACLs

Using a standard named ACL lets you specify whether the prefix needs to be an exact match or not. If you specify **exact-match**, then routes only match the ACL if they have the specified prefix length. Otherwise, routes match the ACL if they have a prefix length equal to or longer than the specified prefix length. For example, if you specify 10.0.0.0/8, then:

- **without** exact-match, the ACL matches all of 10.0.0.0/8–10.0.0.0/32
- **with** exact-match, the ACL only matches 10.0.0.0/8

Numbered ACLs

For numbered ACLs, the mask is a reverse (or wildcard) mask. This is the opposite of a standard mask in dotted decimal notation. However—in line with industry standards—the mask value has no effect. The ACL always applies to all prefix lengths.

Extended ACLs

You can also use an extended ACL (number range 100-199, or 2000-2699, or by using the **extended <*name*>** parameter) but there is no advantage to doing so. Extended ACLs include two prefixes (source and destination), and using two prefixes is meaningless when filtering routes.

Using ACLs as filters

When you have created an ACL, you can use it to filter incoming or outgoing Update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming Updates (received from a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> distribute-list <acl-id> in
```

Filter outgoing Updates (destined for a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> distribute-list <acl-id> out
```

The switch will then compare the prefixes in Update packets with each entry in the ACL, looking for matches.

If a matching entry has the parameter **permit**, then there will be effectively no action. If a matching entry has the parameter **deny**, then the specified prefix will be removed from the Update packet. Once the Update packet has been compared against every entry in the ACL, it will be sent to the neighbor (out filters) or processed (in filters), minus any prefixes that have been removed by the filter.

Example: Distribute filters

Filter out one particular route from a neighbor

1. Create a named ACL to deny the route 52.0.0.0/8 and accept all others. You need to include a **permit any** entry because ACLs end in an implicit **deny any** entry.

```
awplus(config)# access-list standard list1 deny 52.0.0.0/8 exact
awplus(config)# access-list standard list1 permit any
```

2. Set that ACL as the filter for the BGP neighbor 45.45.45.46:

```
awplus(config)# router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 distribute-list list1 in
```

Filter out a range of prefix lengths

This example demonstrates the effect of the **exact** parameter in the ACL by discarding all routes to 52.0.0.0 with prefix lengths of 4 or greater.

1. Create the following ACL. When **exact** is not specified, the ACL entry matches all masks greater than or equal to the specified mask, so the first entry below would block routes like 52.0.0.0/8.

```
awplus(config)# access-list standard list2 deny 52.0.0.0/4
awplus(config)# access-list standard list2 permit any
```

2. Set that ACL as the filter for the BGP neighbor 45.45.45.46:

```
awplus(config)# router bgp 34567
awplus(config)# neighbor 45.45.45.46 distribute-list list2 in
```

Use a numbered ACL instead of a named ACL

This example demonstrates a numbered ACL by discarding all routes to 52.0.0.0.

1. Create a numbered ACL:

```
awplus(config)# access-list 1301 deny 52.0.0.0 0.0.0.255
awplus(config)# access-list 1301 permit any
```

   In line with industry standards, the wildcard mask is required but its value has no effect.

   The ACL always applies to all prefix lengths.

2. Set that ACL as the filter for the BGP neighbor 45.45.45.46:

```
awplus(config)# router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 distribute-list 1301 in
```

Configuring AS_Path Filters

To configure path filters we need to first understand something about AS_Path **lists** and how to use them.

AS_Path lists

Path filters use a construct known as an **AS_Path list**. An AS_Path list has a name and consists of one or more (unnumbered) entries. Each entry specifies:

- which AS_Paths to consider.
- whether the AS_Paths in question should be included or excluded from the list.

The set of paths to consider is specified using regular expressions. The AlliedWare Plus OS supports the full set of syntax elements for simple regular expressions:

Symbol	Character	Used to…
^	Caret	Match the beginning of the input string. When used at the beginning of a string of characters, it negates a pattern match.
$	Dollar sign	Match the end of the input string.
.	Period	Match a single character (white spaces included).
*****	Asterisk	Match zero or more sequences of pattern.
+	Plus sign	Match one or more sequences of pattern.
?	Question mark	Match zero or one occurrences of a pattern.
_	Underscore	Match spaces, commas, braces, parentheses, or the beginning and end of an input string.
[]	Brackets	Specify a range of single-characters.
-	Hyphen	Separate the end points of a range.

For example, you can specify things like:

- any path that contains a certain set of AS numbers: e.g. 23334 45634 8988
- any path that starts with a particular AS number: e.g. ^23334
- any path that ends with a particular AS number: e.g. 8988$
- a specific path: e.g. ^23334 45634 8988$
- an empty path: ^$

Syntax

To create an entry in an AS_Path list, use the command:

```
awplus(config)# ip as-path access-list <list-name> {deny|permit} <regular-expression>
```

There is an implicit "*deny all*" entry at the end of each path list. Any AS_Path that does not match any earlier entry is excluded from the list.

Using AS_Path lists as path filters

When an AS_Path list has been created, it can be applied to filter incoming or outgoing Update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming Updates (received from a particular neighbor):
awplus(config-router)# neighbor <neighbor> filter-list <list-name> in

Filter outgoing updates (destined for a particular neighbor):
awplus(config-router)# neighbor <neighbor> filter-list <list-name> out

The router will then compare the AS_Path attribute in BGP Update packets with each entry in the AS_Path list until a match is found. If the AS_Path list entry that matches has the parameter **permit**, then the Update packet will be allowed through by the filter. If the matching entry has the parameter **deny**, then the Update packet will be blocked by the filter.

Note - All Update packets whose AS_Paths do not explicitly match an entry in the AS_Path list will be dropped, because the list ends in an implicit "*deny all*" entry.

Example: AS_Path filters - discard or allow routes from a neighbor

Explicitly exclude routes that have passed through a particular AS

1. Create an AS_Path access list that denies AS 23456:

   ```
   awplus(config)# ip as-path access-list list1 deny 23456
   ```

2. Set that access list as the in-filter for the BGP neighbor 45.45.45.46:

   ```
   awplus(config)# router bgp 34567
   awplus(config-router)# neighbor 45.45.45.46 filter-list list1 in
   ```

Explicitly include routes that have passed through another AS

1. Add an entry to the AS_Path access list that explicitly includes AS 34568:

   ```
   awplus(config)# ip as-path access-list list1 permit 34568
   ```

2. Check that the AS_Path list shows the two filter entries:

   ```
   awplus(config-router)# do show ip as-path-access-list
   AS path access list list1
   deny 23456
   permit 34568
   ```

Another example: An outgoing filter that uses an AS-Path list

1. Create an AS-Path list that denies empty AS_Paths, but allows AS_Paths that contain the AS number 34567:

```
ip as-path access-list example deny ^$
ip as-path access-list example permit 34567
```

2. Apply this as the out route map for neighbor 45.45.45.46:

```
neighbor 45.45.45.46 filter-list example out
```

Configuring Prefix Filters

Prefix filters use prefix lists to filter particular routes on the basis of their prefixes.

Prefix filters and distribute filters both filter individual routes out of BGP Update packets. They are mutually exclusive.

About prefix lists

A prefix list is a list of prefix entries. Each entry specifies a particular prefix, a mask length or range of mask lengths, and whether or not those prefixes are deemed to explicitly match or explicitly **not** match the prefix list. A prefix list entry is created with the command:

```
awplus(config)# ip prefix-list <list-name> [seq <number>]
                {deny|permit} {any|<ipadd>/<prefix-length>}
                [ge <minimum-length>] [le <maximum-length>]
```

You can choose to give an entry a sequence number by using the optional **seq** parameter. If you do not, the switch assigns sequence numbers in steps of 5 (number 5, 10, 15 etc.) and puts the new entry at the end of the list of entries.

To see entries and their numbers, use the command:

```
awplus# show ip prefix-list
```

Mask length

You can specify a single prefix mask length, or you can use the **ge** and **le** parameters to specify a range of mask lengths for the entry to match.

If you set the mask length to:

- A **single** mask length (by specifying neither the **ge** nor the **le** parameter), then a route matches against this entry if its prefix mask length is exactly that length.
- A **range** of mask lengths, then a route matches against this entry if its prefix mask length is greater than or equal to **ge** and less than or equal to **le**.

For example, to deny the IP addresses between 10.0.0.0/14 (mask of 255.252.0.0) and 10.0.0.0/22 (mask of 255.255.252.0) within the 10.0.0.0/8 (mask of 255.0.0.0) addressing range, use the command:

```
awplus(config)# ip prefix-list mylist deny 10.0.0.0/8 le 22 ge 14
```

The mask length (8 in this example) must be less than the value specified for the **ge** parameter. This mask defines the range of subnets that the matching subnets must fall within.

Using prefix lists as prefix filters

When you have created a prefix list, you can use it to filter incoming or outgoing Update messages for a particular BGP peer, by using the following commands in BGP router mode for the AS.

Filter incoming Updates (received from a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> prefix-list <list-name> in
```

Filter outgoing Updates (destined for a particular neighbor):

```
awplus(config-router)# neighbor <neighbor> prefix-list <list-name> out
```

The router will then compare the prefixes in Update packets with each entry in the prefix list, looking for matches. If a matching entry has the parameter **permit**, then there will be effectively no action. If a matching entry has the parameter **deny**, then the specified prefix will be removed from the Update packet.

Once the Update packet has been compared against every entry in the prefix list, it will be sent to the neighbor (out filters) or processed (in filters), minus any prefixes that have been removed by the filter.

Example: Prefix filters

Filter out one particular route from a neighbor

1. Create an IP prefix list to include the route 45.0.0.0/8. Prefix lists end in an implicit exclude clause, so this will exclude all other routes:

   ```
   awplus(config)# ip prefix-list list1 permit 45.0.0.0/8
   ```

2. Set that prefix list as the prefix list filter for the BGP neighbor 45.45.45.46.:

   ```
   awplus(config)# router bgp 34567
   awplus(config-router)# neighbor 45.45.45.46 prefix-list list1 in
   ```

Filter out a range of different prefix lengths

This example filters out routes within the 45.0.0.0/4 range that have prefix lengths in the range 5-8, accepts routes to 52.0.0.0/8, and implicitly filters out all other BGP routes.

1. Add the following prefix list entries:

```
awplus(config)# ip prefix-list list2 deny 45.0.0.0/4 ge 5 le 8
awplus(config)# ip prefix-list list2 permit 52.0.0.0/8
```

2. Set that prefix list as the prefix list filter for neighbor 45.45.45.46:

```
awplus(config)# router bgp 34567
awplus(config)# neighbor 45.45.45.46 prefix-list list2 in
```

Configuring Route Maps

Route maps are very powerful and flexible entities. Therefore, the configuring of route maps m ust, by necessity, be relatively complex. The purpose of this section is to understand route maps piece by piece and thereby build up a full understanding of how all the parts fit together.

Structure of a route map

There are various levels of structure within a route map:

- A route map is an entity with a name
- Each route map consists of multiple entries, identified by sequence numbers
- Each entry can consist of multiple clauses

In effect, an entry defines an individual filter. It can have a **match** clause that defines what it will match on, and it can have multiple **set** clauses that can specify actions to be taken.

An Update packet is matched against each entry in turn. Once an entry is found that matches the packet, the action(s) associated with that entry is (are) performed, and no further entries are considered.

For example, if you create an entry that will permit an Update packet, followed by an entry that would deny that packet, the packet is permitted. As another example, if you create two conflicting **set** clauses, in different entries, the first change is applied, not the second.

- A route map consists of an ordered set of entries
- Each entry is configured by commands of the form

```
route-map <map-name>  permit|deny <sequence number>
<match clause>
<set clause(s)>
```

There is no command that creates a route map prior to entries being created in the route map. Simply, the first time an entry is created for route map <name> then that route map comes into existence.

The sequence numbers determine the order in which the entries are applied to Update packets.

A route map entry is not required to include a match clause. An entry with no match clause will match every Update packet.

There is an implicit "*match all*" filter at the end of the route map. The action on that implicit entry is **deny**. By default, any Update packet that does not explicitly match any particular entry in the route map will be dropped. You can change this by ending the route map with a "*permit all*" clause, such as the following:

```
awplus(config)# route-map <map-name> permit 65535
```

Clauses

As described above, there are two types of clauses that can be present in a route map entry:

- **match** clauses, which specify attributes or prefixes to match on
- **set** clauses, which specify the changes to be made to attribute values

A given route map entry can never have more than one match clause, but it can have multiple set clauses.

Configuring a match clause

When you configure a match clause, you can match on one of the attributes listed in the following sections.

An AS_Path list

For information about creating an AS_Path list, see AS path lists on page 197.

Once you have made the path list, you can apply it to the match clause of a route map entry by using the commands:

```
awplus(config)# route-map <map-name> {deny|permit} <seq>
awplus(config-route-map)# match as-path <list-name>
```

This clause will cause the router to go through the entries in the AS_Path list. If one of the entries in the AS_Path list matches the AS_Path in the Update packet, and the action on that AS_Path list entry is permit, then the packet is deemed to have matched this route map match clause.

The fact that there are two levels of matching going on, and effectively two different uses of the permit/deny parameters in the one clause, can be a bit confusing.

When working out the correct way to configure a clause that matches on AS_Path list, it is important to first think about how you are configuring the match criteria within the AS_Path list, and then separately think about the actions of the route map entry.

In particular, it is important to note that in this context, the parameters in the AS_Path list do not indicate whether the matching Update packet is being allowed or dropped; "*permit*" and "*deny*" simply indicate whether the Update packet is deemed to match or not match the AS_Path list. This is different to the case where the AS_Path list itself is being used as a filter.

Just as an example, consider these configurations:

Case 1:

```
awplus(config)# ip as-path access-list example deny ^$
awplus(config)# ip as-path access-list example permit 15557
awplus(config)# router bgp 100
awplus(config-router)# neighbor 192.168.200.201 filter-list example out
```

Case 2:

```
awplus(config)# ip as-path access-list example permit ^$
awplus(config)# ip as-path access-list example deny 15557
awplus(config)# route-map rmapexample deny 1
awplus(config-route-map)# match as-path example
awplus(config-route-map)# route-map rmapexample permit 65535
awplus(config-route-map)# router bgp 100
awplus(config-router)# neighbor 192.168.200.201 route-map
rmapexample out
```

Both of these configurations would cause outgoing Update packets with empty AS_Paths to be dropped, and Update packets with an AS_Path containing 15557 to be allowed.

But, to achieve that in the second case, the AS_Path list has to be configured to permit empty paths. This way, the empty path will match the AS_Path list, and be included into the route map's action of dropping packets that match the AS_Path list.

Also, in the second case, the AS_Path list specifically excludes packets whose AS_Path contains 15557. Therefore, these packets are not selected by the AS_Path list, and so are not dropped by the first route map entry. Therefore, they are permitted by the last (permit all) route map entry.

A community list

Creating

A community list is a set of entries that specify which community attribute values are included in or excluded from the list.

There are two types of community list: standard and expanded.

- Standard lists are a just a list of one or more communities. They can be identified by a name, a number, or the word **standard**, and are created by using any of the following commands:

```
awplus(config)# ip community-list <1-99> {deny|permit} {aa:xx|internet|
local-as|no-advertise|no-export}
awplus(config)# ip community-list <list-name> {deny|permit} {aa:xx|
internet|local-as|no-advertise|no-export}
awplus(config)# ip community-list standard <list-name> {deny|permit}
{aa:xx|internet|local-as| no-advertise|no-export}
```

If you have more than one community in an entry, separate them with spaces.

- Expanded lists use regular expressions to specify the communities. They can be identified by a number, or by the word **expanded**, and are created by using any of the following commands:

```
awplus(config)# ip community-list <100-199> {deny|permit} <reg-exp>
awplus(config)# ip community-list expanded <list-name> {deny|permit}
<reg-exp>
```

You can have multiple entries in a community list. Entries are unnumbered, so each new entry gets added at the end of the list. There is an implicit "*exclude all else*" entry at the end of the community list.

Applying

Once you have created a community list, use it in a route map entry by using the command:

```
awplus(config-route-map)# match community <list> [exact-match]
```

The optional **exact-match** parameter specifies that the communities contained in the attribute section of the Update message must exactly match the specified community list. If you do not specify **exact-match**, then the set of communities in the attribute list of the Update message must include all the communities in the specified community list, but can also include other communities.

One or more prefixes, by using a prefix list

For information about creating a prefix list, see About prefix lists on page 199.

Once you have made the prefix list, apply it to the match clause of a route map entry by using the command:

```
awplus(config-route-map)# match ip address prefix-list <list-name>
```

A prefix list can match a subset of prefixes in an Update message. You can use this to change the attributes of some of the prefixes in an outgoing Update, without having to change the attributes of all the prefixes.

But, how can this work? Doesn't an Update message contain just one set of attributes, which must apply to all the prefixes in the Update? So how is it possible for some prefixes in the Update packet to have their attributes changed, and some not?

The solution is simple – split the Update message into two (or more) Update messages. If a route map entry matches on some of the prefixes in a particular outgoing Update message, and has an associated set clause that specifies a change to the attributes for those prefixes, then the switch splits the Update into two Update packets:

- one that contains the original attribute values and the prefixes that were not included by the route map entry

  and

- one that contains the new attribute values and the prefixes that were included by the route map entry.

One or more prefixes, by using an ACL

An ACL is an alternative to a prefix list for matching a prefix in an Update message. For information about creating an ACL, see About ACLs on page 194.

Once you have made the ACL, apply it to the match clause of a route map entry by using the command:

```
awplus(config-route-map)# match ip address <acl-number-or-name>
```

A next hop address

You can use either a prefix list or an ACL to specify a next hop address. Once you have made the prefix list or ACL, apply it to the match clause of a route map entry by using one of the commands:

```
awplus(config-route-map)# match ip next-hop prefix-list <list-name>
awplus(config-route-map)# match ip next-hop <acl-number-or-name>
```

An Update message matches this route map entry if the next-hop attribute in the update equals an IP address permitted in the prefix list or ACL.

An origin

To match an origin, use the command:

```
awplus(config-route-map)# match origin {egp|igp|incomplete}
```

An Update message matches this route map entry if the origin attribute in the Update equals the origin specified in the entry.

A metric (the MED attribute)

To match the MED value, use the command:

```
awplus(config-route-map)# match metric <med-value>
```

An Update message matches this route map entry if the MED attribute in the Update equals the value specified in the entry.

Configuring a set clause

If a packet matches the match clause, then the action of the route map entry will be applied to that packet. The action might simply be to permit or deny the packet, or it might be to update its attributes by applying one or more set clauses.

Note - When configuring a set clause, make sure you are in route map mode for the same route map name sequence number as you used for the match clause. The prompt should look like:

```
awplus(config-route-map)#
```

A set clause can change any of the following attributes on an Update message:

AS_Path

Multiple autonomous system numbers (ASNs) can be added to the AS_Path attribute in the Update packet.

Use the command:

```
set as-path prepend <asn-list>
```

If adding multiple ASNs, separate them with spaces.

Community

Community set clauses give you a lot of control over the Community values in Updates.

You can:

Replace the set of Community values in an Update, by using the command:

```
set community <community-values>
```

Add more Communities to the set of Community values in an Update, by using the command:

```
set community <community-values> additive
```

Remove the Community attribute from an Update, by using the command:

```
set community none
```

Remove Communities from the set of Community values in an Update, by creating a Community list (see A community list on page 204) and then applying it in a set clause.

Use the command:

```
set comm-list <list-id>
```

Localpref

This specifies a value to set as the **Localpref** attribute in the Update message.

Use the command:

```
set local-preference <0-4294967295>
```

Metric (MED)

This specifies a value to set as the MED attribute in the Update message. You can:

- Set the MED in an Update, by using the command:

  ```
  set metric <0-4294967295>
  ```

- Increase or decrease the MED in an Update by a specified amount. Use one of the commands:

  ```
  set metric +<amount>
  set metric -<amount>
  For example, to increase the MED by 2, use the command:
  set metric +2
  ```

Origin

This specifies a value to set as the Origin attribute in the Update message.

Use the command:

```
set origin {igp|egp|incomplete}
```

Aggregator

This specifies the Update's Aggregator attribute. The Aggregator attribute specifies the AS and IP address of the device that performed the aggregation.

Use the command:

```
set aggregator <asn> <ipadd>
```

Atomic aggregate

This adds the Atomic Aggregate attribute to the Update.

Use the command:

```
Set atomic-aggregate
```

Extended community

This specifies a value to set as the Extended Community attribute in the Update message.

Use the command:

```
set extcommunity {rt|soo} <ext-comm-number>
```

The **rt** parameter configures a Route Target Extended Community. This consists of routers that will receive matching routes.

The **soo** parameter configures a site-of-origin Extended Community. This consists of routers that will inject matching routes into BGP.

Next-Hop

This specifies the next hop for matching routes.

Use the command:

```
set ip next-hop <ipadd>
```

Originator ID

This specifies the Originator ID attribute in the Update message. The Originator ID is the Router ID of the iBGP peer that first learned this route, either via an eBGP peer or by some other means such as importing it.

Use the command:

```
set originator-id <ipadd>
```

Weight

Specifies a Weight value for matching routes. The Weight value assists in best path selection of BGP routes. It is stored with the route in the BGP routing table, but is not advertised to peers. When there are multiple routes with a common destination, the device uses the route with the highest Weight value.

Use the command:

```
set weight <0-4294967295>
```

Dampening

This is used to turn on BGP dampening and optionally set dampening parameters for the routes that match the route map entry.

Use one of the commands:

```
set dampening
set dampening <reachtime>
set dampening <reachtime> <reuse> <suppress> <maxsuppress>
```

The effect of different combinations of clauses

A map entry could consist of any one of the following:

- one match clause with an action
- no match clause and one or more set clauses
- one match clause and one or more set clauses

Let us consider each entry type in turn.

One match clause with an action

The effect of an entry that contains a match, but no sets, will be to apply the specified action to all Update messages that match the entry. The action can be either `permit` or `deny`. This is simply a filter—it simply lets Updates through or blocks them; it does not alter Updates in any way.

If the match criterion in the match clause specifies AS_Path .Community, Nexthop or a Metric and the action is deny, then whole Update messages that match the criterion will be discarded.

If the match criterion in the match clause specifies a prefix list or ACL, then only the prefixes which match the prefix-list or ACL will be dropped. Those prefixes will be removed from the Update message, but other prefixes in the Update message will not be removed.

No match clause and one or more set clauses

If an entry has no match clause, then it is assumed to match **all** Update packets.

It is possible to add one set clause for each type of attribute that can be set. Of course, it is not compulsory to add a clause for each of these attributes. If there is no set clause for an attribute, the switch will simply leave it unaltered in Update packets.

It is possible to specify a route map action of deny for a route map that includes a set clause, but then there is no point in having defined any attributes to be set, as the routes are going to be discarded anyway. However, this is a neat way of turning a clause on or off without destroying the configuration entirely.

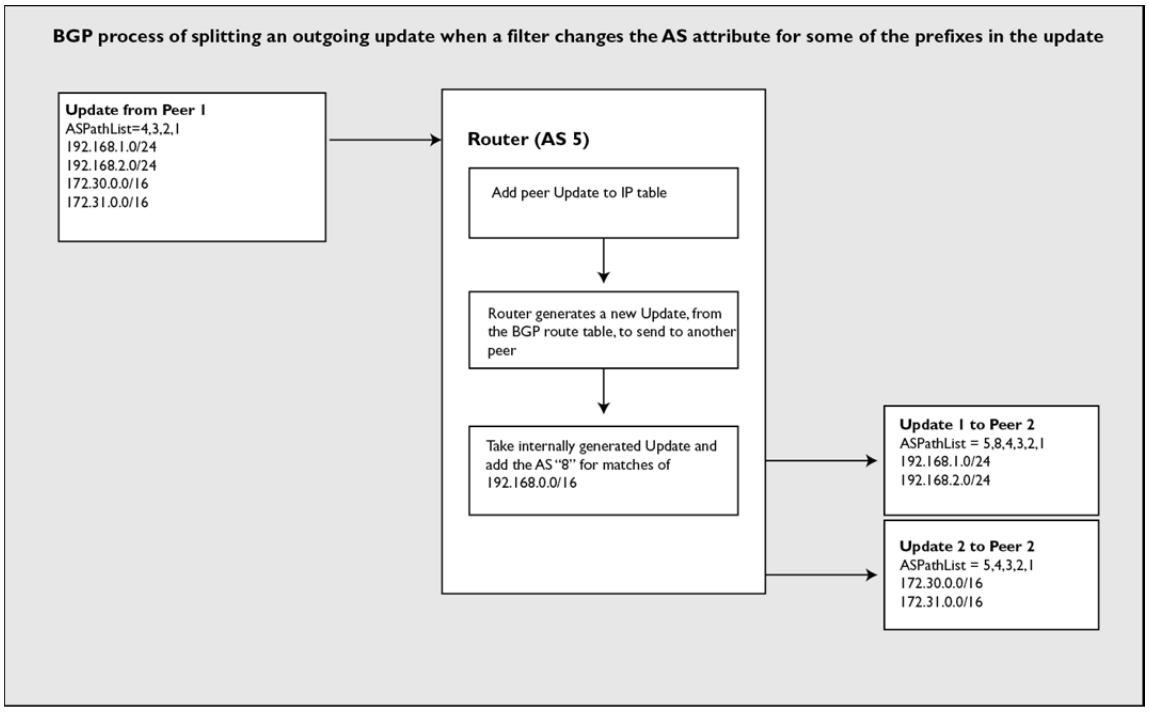
Note - Once you have created an entry that has no match clause, there is no point in adding any more entries below that one, as this entry will match **all** packets, and the entries below it will never be considered.

A match clause and one or more set clauses

The most general case is for an entry to have one match clause (it can never have more than one match clause) and a number of set clauses. Obviously enough, such an entry will apply the activities defined by the set clauses to only those Update packets or prefixes that are picked out by the match clause.

Particular mention, though, has to be made of the case where the match clause specifies prefix list or ACL as the match criterion, and the route map is being applied to outgoing route Updates. The intention of such an entry would be that the attribute values specified in the set clauses be applied to only those routes that are contained in the prefix list or ACL specified in the match clause.

What happens when there is an outgoing Update message that contains several prefixes, some of which are selected by the match clause and some of which are not? Given that an Update packet contains just one set of attributes, it is not possible to say "*Change some of the attributes in the packet, but make them apply to only some of the prefixes contained in the packet*". The attributes in an Update packet must apply to all the prefixes in the packet. The only solution is to break up the Update packet. The switch creates one packet that contains the original attribute values and the prefixes that were not specified in the match clause of the route map entry. Then the switch creates another packet that contains those prefixes of the original packet which did match the prefix list or ACL in the match clause, but with modified attribute values.



BGP: Applying Distribute, Path, Prefix, and Route Map Filters to a Peer

Distribute filters, path filters, prefix filters, and route maps can all be applied to a BGP peer configuration for both incoming and outgoing Updates. However, you cannot combine distribute filters (ACLs) and prefix filters.

First, enter BGP router mode for the AS. The prompt should look like:

```
awplus(config-router)#
```

Then use the following commands.

To apply an ACL to filter incoming route Updates:

```
neighbor <ipadd> distribute-list <acl> in
```

To apply a path list to filter incoming route Updates:

```
neighbor <ipadd> filter-list <list-name> in
```

To apply a prefix list to filter incoming route Updates:

```
neighbor <ipadd> prefix-list <list-name> in
```

To apply a route map to filter incoming route Updates or alter their attributes:

```
neighbor <ipadd> route-map <map-name> in
```

To apply an ACL to filter outgoing route Updates:

```
neighbor <ipadd> distribute-list <acl> out
```

To apply a path list to filter outgoing route Updates:

```
neighbor <ipadd> filter-list <list-name> out
```

To apply a prefix list to filter outgoing route Updates:

```
neighbor <ipadd> prefix-list <list-name> out
```

To apply a route map to filter outgoing route Updates or alter their attributes:

```
neighbor <ipadd> route-map <map-name> out
```

You can also use an AS_Path list, prefix list or ACL in a route map, instead of applying it directly as a filter.

As mentioned in Hierarchy of the different filters on page 194, if you configure multiple types of filter, the switch applies the route maps last. This is true in both the incoming and outgoing directions.

Example A: Match on a prefix-list and set the route metrics

1. Create a prefix list that matches just 52.0.0.0/8

```
awplus (config)# ip prefix-list test1 permit 52.0.0.0/8
```

2. Then, create a route map to match on this prefix-list, and set the metric of matching routes to 665:

```
awplus(config)# route-map test1 permit 1
awplus(config-route-map)# match ip address prefix-list test1
awplus(config-route-map)# set metric 665
```

3. Configure the route-map to operate on incoming Updates from neighbor 45.45.45.46

```
awplus(config-router)# neighbor 45.45.45.46 route-map test1 in
```

Example B: Match on a prefix-list that denies an entry

1. Create a prefix list that excludes 52.0.0.0/8, then includes all other routes:

```
awplus(config)# ip prefix-list test2 deny 52.0.0.0/8
awplus(config)# ip prefix-list test2 permit any
```

2. Create a route map to match on this prefix-list:

```
awplus(config)# route-map test1 permit 1
awplus(config-route-map)# no match ip address prefix-list test1
awplus(config-route-map)# match ip address prefix-list test2
awplus(config-route-map)# set metrics 665
```

Example C: Match on a community list, and apply to a deny entry of a route map

1. Create a community list that has no community types specified:

```
ip-community-list 1 permit
```

2. Create an IP access-list that matches ALL routes:

```
Access-list 1 permit any
```

3. Create a route map that has two clauses: one, which denies all Updates matching the Community list, and one which permits all else.

```
route-map com-deny-test deny 1
match community 1
route-map com-deny-test permit 2
match ip address 1
```

4. Apply this route map as the route map on the neighbor 45.45.45.46

```
awplus(config-router)# neighbour 45.45.45.46 route-map com-deny-test in
```

Example D: Matching on a next-hop prefix-list

1. Create a prefix-list that matches on the neighbor's IP address:

```
awplus(config)# ip prefix-list nh-test permit 45.45.45.46/32
```

2. Create a route map with a permit entry that matches on this prefix-list as a next-hop:

```
awplus(config)# route-map nh-test permit 1
awplus(config-route-map)# match ip next-hop prefix-list nh-test
```

3. Apply this route map as the in route map on the neighbor:

```
awplus(config)# router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 route-map nh-test in
```

Example E: Prepending AS numbers

1. Create a route map that prepends a list of AS numbers to the attribute field of outgoing Updates:

```
awplus(config)# route-map as-list-test permit 1
awplus(config-route-map)# match ip address 1
awplus(config-route-map)# set as-path prepend 11 22 33
```

2. Set this as the out route map for the neighbor 45.45.45.46

```
awplus(config-route-map)# router bgp 34567
awplus(config-router)# neighbor 45.45.45.46 route-map as-list-test out
```

Example F: A community-list based filter dropping incoming Updates based on community number

1. Create a Community-list that matches on some Community values.

```
ip community-list 78 permit 55:66
ip community-list 78 permit 89:89
ip community-list 78 permit 9999:89
```

2. Create a prefix-list that matches the route 52.0.0.0/8

```
ip prefix-list test1 permit 52.0.0.0/8
```

3. Set the Community-list to be the match criterion on a deny entry in a route map

```
route-map com deny 1
match community 78
```

4. Set the prefix-list to be the match criterion on a permit entry in the route map:

```
route-map com permit 2
match ip address prefix-list test1
```

5. Apply this route map as the incoming filter for the neighbor 45.45.45.46:

```
neighbor 45.45.45.46 route-map com in
```

Example G: Using an ACL-match in a route map to update just a single route out of an update

1. Create an ACL that matches just one route:

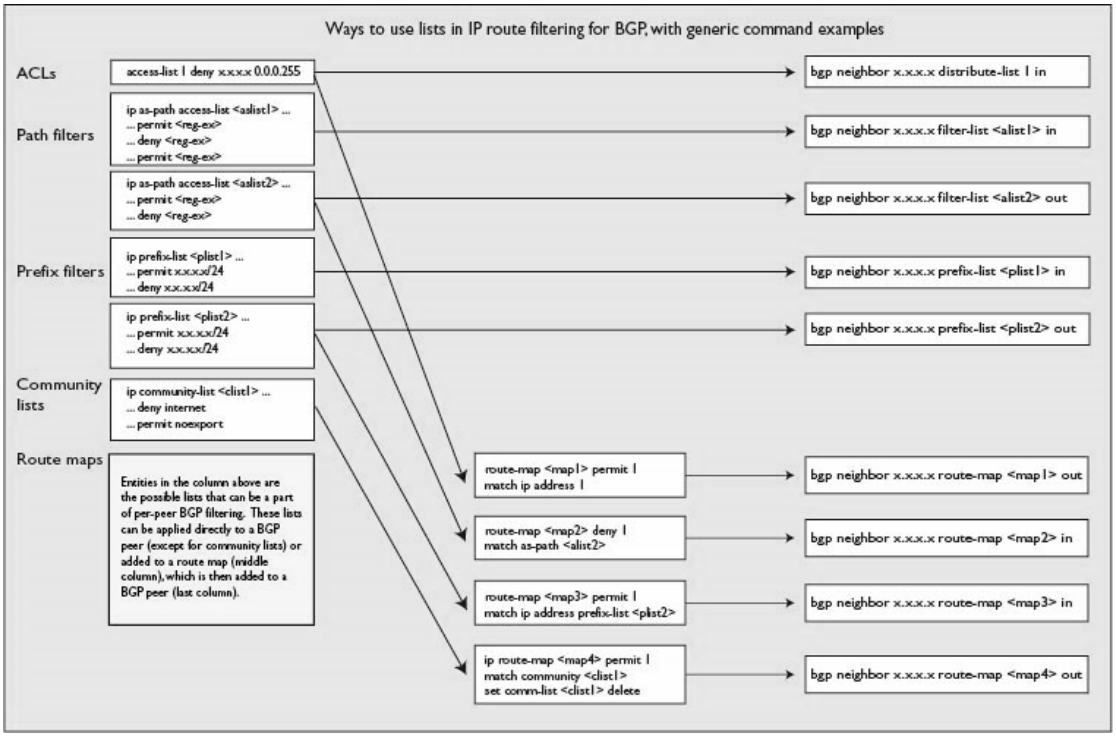
```
access-list standard marker permit 192.34.23.32/29
```

2. Create a route map with one clause that matches on this ACL and prepends an AS_Path (and then has another clause the permits everything else).

```
route-map marker permit 1
match ip address marker
set as-path prepend 7822
route-map marker permit 2
```

3. Apply this route map as the out route map for the peer.

```
router bgp 34567
redistribute connected
redistribute static
neighbour 45.45.45.46 remote-as 34568
neighbour 45.45.45.46 route-map marker out
```



Applying Route Maps to Imported Routes

The switch is able to redistribute routes into BGP that it learnt by non-BGP means. In other words, static routes, or routes learnt by OSPF or RIP can be redistributed into BGP. You can apply a route map to this importation process so that the imported routes are given certain attributes, or so that certain routes are blocked from being imported.

The switch uses the route map to:

- Filter routes or set attributes when it imports the routes into BGP
- Set attributes on any Update message in which it advertises the routes

Note that the entries in route maps applied to BGP importing cannot have match clauses that match on BGP attributes such as AS_Path or Community. This is because BGP attributes are not relevant to non-BGP routes. The route map entries can match on prefix list, origin, or next hop, or the route type if importing OSPF routes.

Syntax

You can apply the route map by using the following commands in BGP router mode for the AS.

```
redistribute {connected|ospf|rip|static} route-map <map-name>
```

Other Uses of Route Maps

You can use route maps in contexts other than filtering routes. Let us look briefly at some of the other contexts in which they are used.

neighbor default-originate

The command **neighbor default-originate** instructs BGP to send a default route to a neighbor. This command includes a parameter for specifying a route map.

The **route map** parameter specifies criteria that must be fulfilled before the switch will advertise the default route to the neighbor. This lets you configure the switch so that it only acts as the default gateway for a neighbor if it has learned a route that makes it a suitable default gateway for that neighbor.

Specifically, the switch must have learnt a route that matches the permit criteria in the route map. If the switch does not know of any routes that match the permit criteria in the route map, then it will not advertise the default route.

neighbor unsuppress-map

Using the **aggregate-address** command with the **summary-only** option, suppresses the more specific routes of the aggregate to all neighbors. If you want to selectively leak some more-specific routes to a particular neighbor, then this is achieved by using the command **neighbor unsuppress-map**.

A required parameter on this command is the route map that specifies the set of more-specific routes to leak. Any route that matches permit criteria in the route map will be leaked.

network

To redistribute particular BGP routes selectively, use the **network** command. Instead of saying "*redistribute all static routes*" or "*redistribute all connected routes*", the **network** command says "*redistribute this particular route*". Of course, the route has to be in the routing table before it can be redistributed, so the **network** command does not say "*always import this route in BGP*", it says "*if this route is in the routing table, then import it into BGP*".

As with other acts of redistribution, the **network** command has an optional **route map** parameter. This lets you set attributes on the route advertised by the **network** command.

You could use the route map for filtering the network, but it would be a little pointless to configure the redistribution of this network if the associated route map simply dropped the route.

show ip bgp

The **show ip bgp** command takes an optional **route map** parameter. The effect of specifying the route map on the command is that the output shows information only on routes that match permit criteria on the route map.

OSPF: Configuring Route Maps for Filtering and Modifying OSPF Routes

Route maps can be applied to OSPF routes as well as BGP routes. Of course, the route maps that can be used with OSPF are rather limited in comparison with those that are used with BGP, as OSPF route Updates do not carry attributes in the way that BGP route Updates do.

Also, OSPF route maps can only be applied to importing:

- OSPF-learnt routes into the main IP route table,

  or

- static, BGP or RIP routes into OSPF

Filtering **cannot**:

- remove an entry from the LSA database once the entry has been added
- prevent the router from advertising an LSA database entry to interfaces in the same area that the entry is relevant to
- prevent updates that OSPF learns from being put into the LSA database
- change the properties of an entry in the LSA database

This is because OSPF shares LSAs between all the routers in an area. The protocol assumes that all the routers in the area have shared all the advertisements among each other, and that all agree on the state of the complete link state database for the area. If some routers in the area are learning certain LSAs, but not advertising them, that breaks the OSPF model.

Configuring a match clause

A match clause can match on the following criteria.

Note - When configuring a match clause, make sure you are in route map mode. The prompt should look like:

```
awplus(config-route-map)#
```

Metric

The entry will match all routes whose metric is equal to that specified in the clause.

To match a metric value, use the command:

```
match metric <value>
```

Interface

The entry will match all routes learnt via the specified VLAN.

To match a VLAN, use the command:

```
match interface <vlan>
```

External route type

The entry will match all routes of either Type-1 External or Type-2 External.

To match a route type, use the command:

```
match external {type-1|type-2}
```

A prefix, by using a prefix list

The entry will match one or more route prefixes. For information about creating a prefix list, see About prefix lists on page 199.

Once you have made the prefix list, apply it to the match clause of a route map entry by using the command:

```
match ip address prefix-list <list-name>
```

A prefix, by using an ACL

An ACL is an alternative to a prefix list for matching route prefixes.

For information about creating an ACL, see About ACLs on page 194.

Once you have made the ACL, apply it to the match clause of a route map entry by using the command:

```
match ip address <acl-number-or-name>
```

A next-hop address

The entry will match the route's next hop.

You can use either a prefix list or an ACL to specify a next hop address. Once you have made the prefix list or ACL, apply it to the match clause of a route map entry by using one of the commands:

```
match ip next-hop prefix-list <list-name>
match ip next-hop <acl-number-or-name>
```

Configuring a set clause

If a route matches the **match** clause, then the action of the route map entry will be applied to that route. The action might simply be to permit or deny the route, or it might be to update its parameters by applying one or more **set** clauses.

Note - When configuring a set clause, make sure you are in route map mode for the same route map name sequence number as you used for the match clause.

The prompt should look like:

```
awplus(config-route-map)#
```

A set clause can alter the following parameters on a route.

Metric

This changes the route metric. You can:

- Set the metric, by using the command:

```
set metric <0-4294967295>
```

- Increase or decrease the metric by a specified amount, by using one of the commands:

```
set metric +<amount>
set metric -<amount>
```

For example, to increase the metric by 2, use the command:

set metric +2

Next-hop

```
set ip next-hop <ipadd>
```

Type

This sets the route type to either Type-1 External or Type-2 External.

Use the command:

```
set metric-type {type-1|type-2}
```

OSFP: Applying Route Maps

To specify a route map to be applied to static, BGP, RIP or connected routes as they are imported to OSPF, use the commands:

```
router ospf
redistribute {bgp|rip|connected|static} route-map <map-name>
```

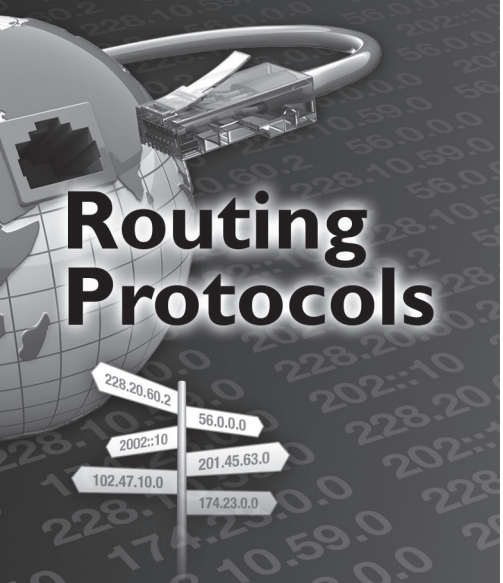
Note that if you want to filter OSPF routes as they are imported into the main IP route table, you need to use a distribute filter instead of a route map.

Use commands like the following:

```
router ospf 88
distribute-list list1 in
```

Section 2

IPv6 Unicast Routing Protocols



Routing Protocols

This chapter covers the following topics:

- IPv6 Addressing
- Types of IPv6 Address
- Setting up an IPv6 Interface using the EUI-64 Algorithm
- RA Guard
- IPv6 Header Structure
- IPv6 Extension Headers
- Encryption and Authentication in IPv6
- Routing
- DHCP Relay
- Management

CHAPTER 5

Introduction to IPv6

Introduction

Internet Protocol version 6, or IPv6, is an improved version of the long-standing Internet Protocol, IPv4.

Address depletion is the primary driver behind the need for IPv6. Commercial opportunities have rapidly increased the need for IP addresses with the demand for wireless devices, peer-to-peer networking and the 'smart home' which, because they access the Internet, require their own IP address. There are more devices connected to the Internet than IP addresses, as NAT has allowed many addresses to 'hide' behind a single public address:

- IPv6 provides an enormous amount of extra address space over IPv4. From IPv4's 32 bits to IPv6's 128 bits, the number of available IP addresses increases from 4 billion to over 340 trillion trillion trillion.
- IPv6 also improves on IPv4 by adding enhancements for security, multimedia traffic management, and simplified network configuration. The transition from IPv4 to IPv6 will be gradual and both IPv4 and IPv6 will coexist for some period of time yet.

In this chapter we describe the IPv6 addressing format, and the differences between IPv4 and IPv6 headers. We look at auto-configuration and neighbor discovery, and network management.

List of terms

Term	Description
Stateless Address Auto Configuration (SLAAC)	A process whereby an enabled IPv6 host can work out its own IPv6 address.
Solicited Node Address	A special multicast address, used in neighbor discovery, that narrows down the number of hosts that need to process the request
Extension headers	Option fields that can be chained onto the end of an IPv6 header
EUI-64	An algorithm for calculating an IPv6 interface ID from a MAC address.
Anycast	A method of IP addressing in which several hosts share the same address, and packets are directed to the nearest of these hosts
NAT	Network Address Translation enables multiple hosts on a private network to access the Internet using a single public IP address.

IPv6 Addressing

In this section we briefly touch on IPv6 addressing and how IPv6 addresses are represented.

IPv6 address formats

IPv6 addresses are **128** bits long whereas IPv4 addresses are only 32 bits long. The new 128 bit IPv6 addresses are written as eight hexadecimal groups. Each hexadecimal group is separated by a colon (**:**) and consists of a 16 bit hexadecimal value.

A complete IPv6 address could look like this:

xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx

A group of xxxx represents a 16 bit hexadecimal value with each individual x representing a 4 bit hexadecimal value. The following is an example of a possible IPv6 address:

`2001:0340:0000:0000:0000:F673:0029:0564`

The IPv6 prefix

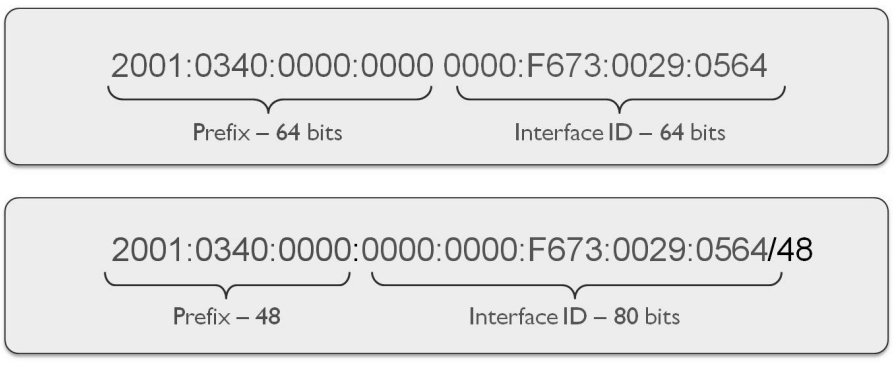
IP addresses combine, in a single address, a network identifier (called the prefix) and a device identifier (the interface ID). IPv4 utilizes a subnet mask to define the network **prefix** and **host** portions of an address. The mask contains a contiguous set of 1's that indicate the prefix portion of the address, following by a contiguous set of 0's that mark the host portion of the address. The length of the prefix portion of the address is given by the number of 1's in the mask. This number is referred to as the 'prefix length'.

IPv4 addresses are typically written in either address/mask format or address/prefix-length format. For example: `192.168.128.1/255.255.255.0` or it can also be written as `192.168.128.1/24`

IPv6 always uses the **address/length** notation. The prefix length is written as /xx at the end of the address:

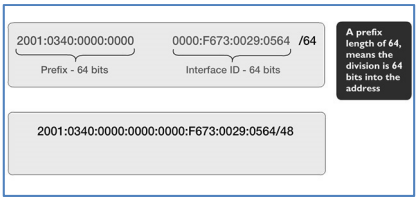
e.g. `2001:0DB8:0000:0000:0000:F673:0029:0564/48`

Let's look at how the prefix length can enable us to split an IPv6 address into it prefix and host portions:



To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/Vx8w9qnglHo

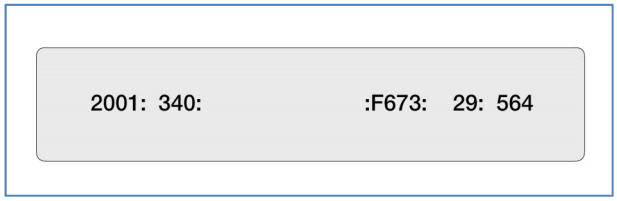


Simplified address representations

To make IPv6 addresses easier to write, the leading zeros in a 4-digit block can be removed. Also, contiguous sets of 4 zeros, and their separating colons, can be completely removed, and replaced by **::**.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/8iH9if5e4vA



To avoid ambiguity, it is only possible to have one place in the address where a continuous set of 0s is replaced by **::**.

Therefore, `2001:0DB8:0000:0000:F673:0000:0000:0564`

can be written as:

`2001:0DB8::F673:0000:0000:564` or `2001:0DB8:0000:0000:F673::0564`

But **not** as: `2001:0DB8::F673::564`

Configuring an IPv6 interface on an AlliedWare Plus switch

Use the following commands to configure an IPv6 address on VLAN1:

```
awplus# conf t
awplus(config)# int vlan1
awplus(config-if)# ipv6 address 2003:78:ab34:9e43::1/64
```

You can configure multiple addresses on one interface, as shown in the example below:

```
awplus(config)# int vlan1
awplus(config-if)# ipv6 address 2003:42c:ab34:9e43::1/64
awplus(config-if)# ipv6 address 2003:af0e:ab34:9e43::1/64
```

Checking the configuration

To check your IPv6 configuration use the **`show ipv6 interface`** command, as shown in the example output below:

```
awplus#show ipv6 interface
Interface IPv6-Address                  Status      Protocol
vlan1     2003:78:ab34:9e43::1/64       admin up    running
          2003:42c:ab34:9e43::1/
          2003:af0e:ab34:9e43::1/64
          fe80::215:77ff:fead:fbcd/64
```

Note the link-local address: `fe80::215:77ff:fead:fbcd/64`. We will discuss link-local addresses in the section : Unicast on page 226.

Types of IPv6 Address

There are three methods of IP addressing that are supported in IPv6:

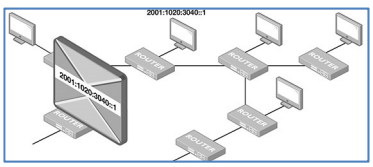
- Unicast
- Multicast
- Anycast

Unicast

A single host possesses the unicast address and irrespective of where a packet destined to that address comes from, it will go to the same host.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/zZ7-ucXaej4



Reserved or special unicast IPv6 addresses

The first field of a reserved or special IPv6 address will always begin with 00xx. Reserved addresses represent 1/256th of the available IPv6 address space. Various reserved addresses exist, including:

`0:0:0:0:0:0:0:0` (or `::`)—an unspecified or unknown address.

This address is the equivalent of the IPv4 0.0.0.0 address, which indicates the absence of a configured or assigned address. In routing tables, the unspecified address is used to identify all or any possible hosts or networks (i.e., the default route, or route of last resort).

`0:0:0:0:0:0:0:1` (or `::1`)—the loopback or local host address. This is the equivalent of the IPv4 `127.0.0.1` address.

Categories of unicast addresses

Unicast addresses can be grouped into three subcategories:

1. Link-local
2. Unique-local
3. Global

Link-local

These addresses start with FE8x: and are used in a single link or subnet. Any packets that are transmitted with a link local source/destination address are never routed out of that subnet. Typically, when IPv6 is enabled on an interface, the interface will be automatically assigned a link-local address, calculated from the interface's MAC address. Then, any intra-subnet IPv6 communication via that interface can use its link-local address.

Unique-local

Originally called site-local, these are the equivalent of IPv4 **private** addresses (RFC1918) that are used within a local organization. Unique-local addresses cannot be routed across the global Internet IPv6 address space. L3 devices will not forward any packets with unique-local source or destination addresses outside of the private enterprise or customer site. IPv6 routing between multiple unique-local subnets within a private enterprise is allowed.

There is a bit of history to which address ranges have become used for local addresses. Originally it was the range FEC0 : : /10 (RFC 1884). But the term 'site-local' was not well defined in the original definition of site-local addresses. The use of FEC0 : : /10 was deprecated in RFC3879. Shortly later, a new range was defined FC00 : : /7 (RFC 4193) for unique-local address ranges.

Global

These addresses start with either a 2xxx: or 3xxx: they are the equivalent of public IPv4 addresses. Global addresses can be routed publicly in the Internet. Any device or site that wishes to transmit packets to another site must be uniquely identified with a global address. Typically, enterprises will be allocated a block of addresses with a 48 bit or 56 bit prefix length. Then, within the enterprise, this block of addresses will be split up into subnets with a 64 bit prefix length.

48-bit network prefix assigned by ISP	**16 bits of address space for subnetting**	**64-bit interface ID (host address) derived from the interface's MAC address**

Some global addresses are allocated to special purposes:

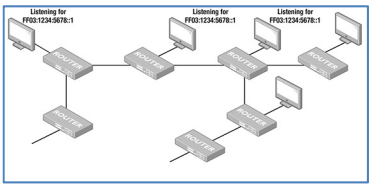
- Documentation: 3FFF:FFFF::/32:2001:ODBB:/32
- 6 to 4 tunneling: 2002::/16
- IPv4 mapped IPv6 addresses: ::ffff:0:0/96

Multicast

Multicast addresses start with **FFxx**: and they operate the same as IPv4 multicast addresses. Interfaces can belong to one or more multicast groups and will accept a multicast packet only if they belong to the group corresponding to the packet's destination address.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/TpgbQ0ctTq4



Multicast is more utilized within IPv6 than it is in IPv4. This is because there are no broadcast packets in IPv6, instead the IPv6 protocol uses IPv6 multicast packets to do the job of an IPv4 broadcast packet. Multicasting provides a much more efficient mechanism than broadcasting, as broadcasting requires that every host on a link accept and process each broadcast packet, whereas hosts can pick out which multicast-destined packets they need to process.

Multicast address scopes

The scope of a multicast address is indicated by the **fourth** hex digit in the address, i.e. The digit 'S' in FF0**S**::

Value	Scope	Meaning
FF01::	node-local	Contained within a single device*
FF02::	link-local	Forwarded only within a subnet on an Ethernet segment
FF04::	admin-local	Forwarded within a small administratively-defined topology
FF05::	site-local	Forwarded only within a single site
FF08::	organizational-local	Forwarding can span multiple sites of a single organization
FF0E::	Global	Can be sent across the Internet

*Node-local means that the scope for the address is within the node itself, e.g. if a PC streams multicast data with node-local scope, then only other applications inside PC can join/see the stream, and the stream never goes out of the PC on any of its interfaces.

Reserved IPv6 multicast addresses

The following is a list of common, well-known IPv6 multicast addresses.

Address	Meaning	IPv4 Equivalent
Node-local scope		
FF01::1	All listeners within the node	
Link-local scope		
FF02::1	All-nodes address	224.0.0.1
FF02::2	All-routers address	224.0.0.2
FF02::5	OSPFv3 (OSPFv6) All SPF Routers	224.0.0.5
FF02::6	OSPFv3 Designated Routers	224.0.0.6
FF02::9	RIPng Routers	224.0.0.9
FF02::13	PIM Routers	224.0.0.13
FF02::16	MLDv2 Reports	224.0.0.22
Site-local scope		
FF05::2	All-routers address	

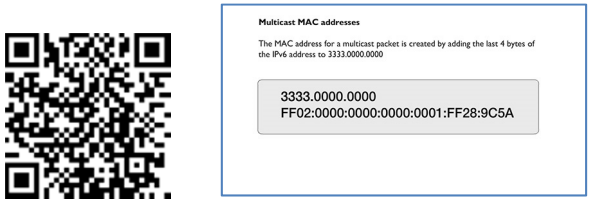
Multicast MAC address

The MAC address for a multicast packet is created by adding the last 4 bytes of the IPv6 address to 3333.0000.0000.

For example if the IPv6 group address is: FF02:0000:0000:0000:0001:FF28:9C5A, then the MAC address for the group is: 3333.FF28.9C5A

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/ujY6Z5CY50U



Anycast

In the anycast addressing mode, multiple hosts are configured with an identical address. Packets sent to an anycast address are sent to the nearest (least number of hops) host that possesses the address. Anycast addresses are indistinguishable from any other IPv6 unicast address. Anycast does not define a class of addresses; it defines how the addresses are used.

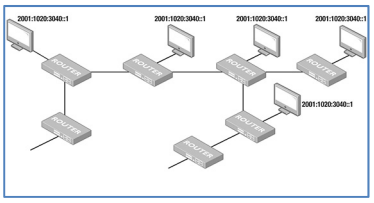
One possible application would be a server farm in which all servers provide an identical service or function, in which case anycast addressing would allow clients to connect to the nearest server.

Like multicast, multiple nodes may be listening on an anycast address. Like unicast, a packet sent to an anycast address will be delivered to one (and only one) of those nodes.

The exact node to which it is delivered is based on the IP routing tables in the network. Anycast is therefore as a cross between unicast and multicast.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/yK47tvyGcv0



Auto-configuration and Neighbor Discovery

Neighbor discovery

Neighbor discovery is an ICMPv6 function that allows a router or host to identify other devices on its links. Neighbor discovery messages are used:

- In address auto-configuration, for duplicate address detection
- To redirect a node to use a more appropriate router if necessary
- To maintain reachability information with its neighbors

There are five IPv6 Neighbor Discovery messages that replace existing IPv4 messages:

IPv6 Discovery Message	of ICMPV6 Type	Replace IPv4 Message
Router Solicitation	133	ICMPv4 Router Discovery
Router Advertisement	134	
Neighbor Solicitation	135	ARP
Neighbor Advertisement	136	
Redirect	137	ICMPv4 Redirect

The three processes under the umbrella of neighbor discovery are:

- **Neighbor Solicitation** – finding out identities of neighbor devices.
- **Router Solicitation** – enabling hosts to discover routers.
- **Redirect** – directing a host to a better gateway for a given destination.

In this section, we will look at the processes that use the neighbor discovery messages.

Stateless address auto-configuration

When IPv6 has been enabled on a network interface, the device can automatically configure itself with an IPv6 address by using ICMPv6 router discovery messages. This is known as Stateless Address Auto-Configuration (SLAAC). SLAAC occurs when a host configures its own address—the address is generated, not allocated.

The major benefit for IPv6 devices is that they can just be plugged in, switched on, and they are globally routable. In summary, the benefits of SLAAC are:

- Devices can be plugged into a network without manual configuration an IP address.
- Has pug and play functionality which makes networks much easier to set up.
- Simplifies the process of renumbering a network.

This process is described below in the section Setting up an IPv6 Interface using the EUI-64 Algorithm on page 232.

There are two halves to the SLAAC process, the client side and the router side. The router provides the prefix, and the client creates the host part of the address. **AlliedWare Plus supports both the client and the router side of stateless address auto-configuration.**

Router side of stateless address auto-configuration – a router can be configured to send out the prefix information in an RA (Router Advertisement) so the client is able find out the prefix part of its address.

For example, on a VLAN interface of an AlliedWare Plus device, the following configuration can be used to send the Prefix:

```
awplus(config)# int vlan10
awplus(config-if)# ipv6 nd ra-interval 10
awplus(config-if)# ipv6 nd prefix 2001:1db9:1:2::/64
awplus(config-if)# no ipv6 nd suppress_ra
```

Client side of stateless address auto-configuration – The client sends out solicitation messages to find out if there is a router that will inform it of the prefix it should use. Having learnt the prefix from the router, the client then generates the host part of the address (i.e. performs SLAAC).
To configure this mode on an interface of an AlliedWare Plus device, use the command:

```
awplus(config-if)# ipv6 address autoconfig
```

Setting up an IPv6 interface using the EUI-64 algorithm

Here is an overview of the steps that occur when a switch performs SLAAC using the EUI-64 algorithm:

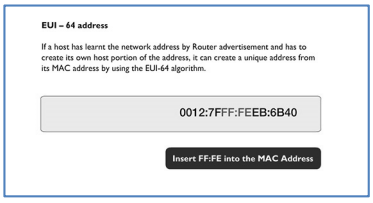
1. Generate a 64 bit interface identifier using the EUI-64 algorithm.

   The host has to create its own host portion of its IPv6 address. It can create a unique address from its MAC address by using the EUI-64 algorithm, here is how it works:

Step	Address
1. Start with the MAC address	0012.7FEB.6B40
2. Split the MAC address in half	0012:7F EB:6B40
3. Insert FF:EE into the MAC address	0012:7FFF:FEEB:6B40
4. Change the 7th bit to '1'	0212:7FFF:FEEB:6B40

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/GO-zlED6qq8



2. **Create a Link-local node address.**

   When IPv6 has been configured on an interface, the switch will automatically assign a link-local address to that interface. Link-local addresses are used as the source address for packets that stay within the subnet, for example:

   - Automatic address configuration
   - Neighbor discovery
   - OSPF exchanges etc.

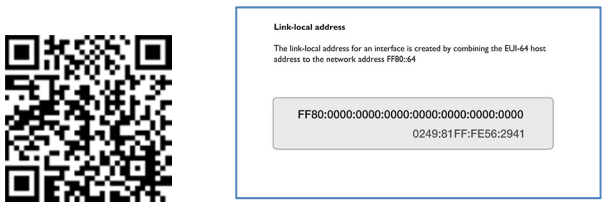
   Any packets that are transmitted with a link-local source/destination address are never routed out of that subnet and are assigned the `fe80::/10` prefix , equivalent to the IPv4 address block 169.x.x.x.

   The link-local address for an interface is created by combining the EUI-64 host address to the network address `FF80::64`

   FF80:0000:0000:0000: + **0212:7FFF:FEEB:6B40 = FF80 : : 0212:7FFF:FEEB:6B40**

   To watch this video, browse to the link or scan the QR code with your smart phone:

   http://youtu.be/_83J0zJmSAM

3. **Send router solicitation messages to all routers on the local link multicast address**. If there is no response, SLAAC ends with only a link-local address generated.

   Note - If a periodic RA containing the appropriate information is received at any time, SLAAC will use it immediately. Also, if an RA is received that reduces the lifetime of a prefix to zero, SLAAC will immediately deprecate the address (the system will then cease using it for new connections, but existing ones will continue).

4. **Prepend the prefix to the EUI-64 interface ID.**
   Once a prefix is learnt by RA, prepend the prefix to the EUI-64 interface ID, to create the full IPv6 address.

   `2001:639A:1234:5678:` + **0212:7FFF:FEEB:6B40**
   =
   `2001:639A:1234:5678:0212:7FFF:FEEB:6B40`

5. Find the default gateway (default routers).
   On receipt of a valid Router Advertisement, a host extracts the source address of the packet and does the following.

   If the address is:

   - **Not** already present in the host's Default Router List, and the advertisement's Router Lifetime is non-zero, it creates a new entry in the list and initializes its invalidation timer value from the advertisement's Router Lifetime field.
   - Already present in the host's Default Router List as a result of a previously received advertisement, it resets its invalidation timer to the Router Lifetime value in the newly received advertisement.
   - Already present in the host's Default Router List and the received Router Lifetime value is zero, it immediately times-out the entry as specified.

To limit the storage needed for the Default Router List, a host may choose not to store the entire set of router addresses discovered via advertisements. However, a host **must** retain at least two router addresses and should retain more. Default router selections are made whenever communication to a destination appears to be failing. Thus, the more routers on the list, the more likely an alternative working router can be found quickly (without having to wait for the next advertisement to arrive).

Neighbor Solicitation

IPv6's replacement for ARP is Neighbor Solicitation, which uses two ICMP messages:

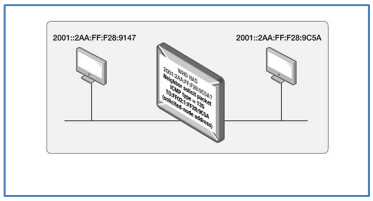
- Neighbor Solicitation (ICMPv6 Type 135)
- Neighbor Advertisement (ICMPv6 Type 136)

Neighbor Solicitation messages perform the following functionality:

- They allow IPv6 nodes (IPv6 hosts and IPv6 routers) to resolve the Link Layer address of a neighboring node (a node on the same physical or logical link).
- When the Link Layer address of a neighboring node has changed, Neighbor Discovery messages allow the other IPv6 nodes to learn that this address has changed.
- They enable IPv6 nodes to determine whether neighboring nodes are still reachable.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/zdzfQY20JIo



Solicited-node multicast address

When requesting the identity of the host that possesses a given IPv6 address, it is more efficient to multicast the request to potential candidates, rather than broadcast to all hosts. This means that hosts that cannot possibly possess the address do not have to process unnecessary broadcast packets. Solicited Node addresses are often flooded by switches and filtered by NIC cards drivers. The multicast address used is called the solicited-node multicast address. It is created by attaching the last 3 bytes of the requested address to FF02::1:FF00:0

For example:

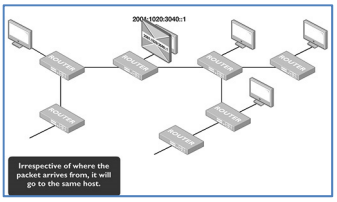
Address being requested: 2001: : 2AA:FF:F**28:9C5A**

1. Begin with FF02:: 1: FF00:0
2. Take the last 3 bytes of the requested address: **28:9C5A**
3. Attach them to the address FF02::1:FF00:0

This is the solicited-node multicast address: FF02::1:FF28:9C5A

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/x_nylx5HYbc



Router discovery

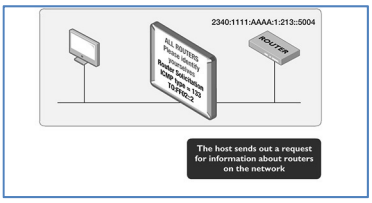
IPv4 hosts need either an administrator to manually configure the default gateway or DHCP to provide this information. When IPv6 is being used, the hosts themselves can automatically locate routers on the LAN. The host achieves this by using two different ICMPv6 messages. They are:

- Router Solicitation (ICMPv6 Type 133)
- Router Advertisement (ICMPv6 Type 134)

When a host is first connected to a LAN, it will send an IPv6 Router Solicitation packet to request information about routers on the network. Each router which is active on the LAN will respond to this packet by sending a Router Advertisement (RA) with its address to all nodes in the group. It informs the host what network address(es) is(are) in use on the subnet. It also informs the host if it is a default gateway.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/Y2eWzIiAYUU



As well as responding to Router Solicitation events, a router will also send out Router Advertisements packets at regular intervals.

Configuring Router Advertisements on AlliedWare Plus

Router Advertisements are configured on AlliedWare Plus on a per-interface basis.

To enable RA advertisements use the command:

```
awplus(config-if)# no ipv6 nd suppress_ra
```

The options available are:

`IPv6 nd prefix<x:x.../N>` which sets the prefix to advertise

`IPv6 nd ra-interval <seconds>` which sets the period of periodic advertisements

`IPv6 nd ra-lifetime <seconds>` which sets the time for which the router will act as a default router, set this to zero to inform hosts that this is not a default router.

The M and O flags

A router advertisement packet includes a set of flags that indicate whether various options are enabled on the router. The most well known of these flags are the M and O flags.

These flags indicate what information hosts can obtain by DHCP.

The M flag represents "Managed address configuration".

If this flag is set in the Router Advertisements sent out from a router, that indicates to the receiving hosts that a DHCP server is available, and can allocate IPv6 addresses to the hosts. Therefore, the hosts have the option of obtaining their addresses by DHCPv6, rather than using auto address configuration.

The O flag represents "Other configuration".

If this flag is set in the Router Advertisements sent out from a router, that indicates to the receiving hosts that a DHCP server is available, and can provide information like DNS server addresses, SNTP server addresses, etc. If the O flag is set, and the M flag is not set, then that indicates that the DHCP server is not handing out IPv6 addresses. In this case, hosts are expected to create themselves an IPv6 address by auto address configuration, and then use DHCP to gather "other" information (DNS server address, etc).

Under AlliedWare Plus, the setting, or not setting, of these flags in the Router Advertisements the device transmits from a given interface is configured by:

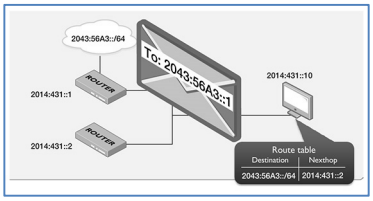
```
awplus(config)# interface <VLAN name>
awplus(config-if)# ipv6 nd managed-config-flag
awplus(config-if)# ipv6 nd other-config-flag
```

Redirect

Redirect uses ICMP type 137 to inform a host of a better router to use as the gateway to a given destination. If a router receives a packet and has to forward that packet to another router in the same subnet, it will also send a redirect back to the sender, telling it to send directly to the other router.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/jBYyXDmDMW8



RA Guard

Router Advertisements (RA) and Router Redirects are key to the Network Discovery Protocol (NDP) used to manage IPv6 networks. RA messages advertise a router's presence and specify network parameters that are used by hosts as part of address auto-configuration and next hop routers for particular destinations.

Subverting this process can severely disrupt the operation of an IPv6 network. RA Guard is a feature that protects the RA process from being subverted.

RA Guard is positioned in between routers and hosts, and acts as an **authorization proxy**. RA Guard drops bad RAs before they reach hosts.

RA Guard operates on all AlliedWare Plus Layer 3 switches, including stacked environments.

Rogue RAs

A rogue RA is an RA that contains invalid information that could cause unwanted changes in the network configuration. These could be generated unintentionally through misconfiguration or maliciously by someone wanting to disrupt or gain access to the network.

A switch can be configured to be selective about the RA and redirect packets it will accept. Ports are configured to trust or not trust the RA and redirect packets they receive.

RA Guard on AlliedWare Plus switches

Ports can be configured to be RA untrusted ports, i.e. RA Guard is disabled by default, but may be **applied** to ports on a per-interface basis and enabled on the following:

- Standalone ports.
- Individual ports in a dynamic (LACP) aggregator, but is not supported on the dynamic aggregator itself.
- A static aggregator, but is not supported on individual ports in a static aggregator.

RA Guard is enabled on an interface as follows:

```
awplus# conf t
awplus(config)# int port1.0.2
awplus(config-if)# ipv6 nd raguard
```

RA Guard classifiers

The actual security enforcement of RA Guard is handled through hardware classifiers, which are dynamically added when a port is marked as trusted or untrusted. RA Guard blocks RAs and router redirects on untrusted ports with filters for ICMPv6 type 134 and 137.

IPv6 Header Structure

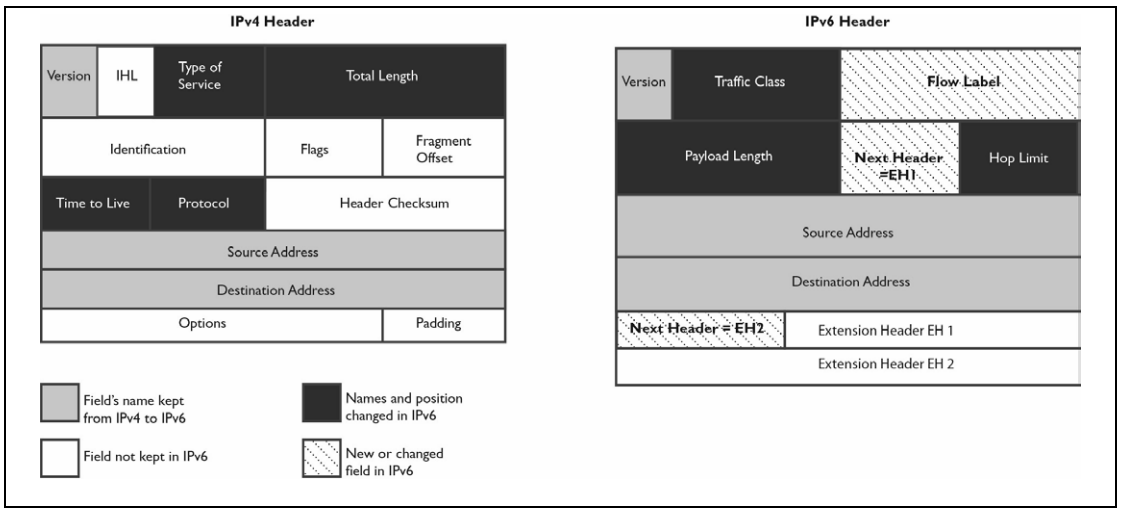
Basic IPv6 header

An IPv6 data packet comprises of two main parts: the header and the payload. The header is the first 40 bytes/octets (40 x 8 = 320 bits) and contains the following fields:

Header Field	Size (Bits)	Description
Source address	128	IPv6 address of the originating node of the packet.
Destination	128	Address of the intended recipient of the IPv6 packet.
Version/IP version	4	Indicates the version of the IPv6 protocol
Packet priority/Traffic class	8	Identifies the data packets that belong to the same traffic class and distinguishes between packets with different priorities.
Flow label/QoS management	20	Identifies a set of packets belonging to the same flow.
Payload length in bytes	16	Length of data field following the IPv6 packet header.
Next header	8	Usually specifies the Transport Layer protocol used by a packet's payload, e.g. TCP (6) and UDP (17).
Time to Live	8	The main function of this field is to identify and to discard packets that are stuck in an indefinite loop due to any routing information errors.

What are the differences between IPv4 and IPv6 headers?

The differences between IPv4 and IPv6 headers are illustrated in the diagram below:



In brief, the changes between the IPv4 and IPv6 headers are as follows:

- The IPv6 header is 40 bytes long (IPv4 header is 20 bytes).
- Each IPv6 address is four times the length of an IPv4 address.
- The IPv6 header no longer contains the header length, identification, flags, fragment offset and header checksum fields. Some of these options have been placed in extension headers.
- The Time To Live field has been replaced with the Hop Limit field.
- The IPv4 Type of Service field is now replaced with a Traffic Class field.
- A Flow Label field has been added.
- The Options field has been replaced by Extension Headers.

For further details on IPv6 address formats, please see **RFC 2460** Internet Protocol, Version 6 (IPv6).

IPv6 Extension Headers

In IPv6, the Options field has been replaced with a set of headers called Extension Headers.

The Options field in IPv4 was too limited, so in IPv6 it has been replaced by the more flexible and extendable concept of Extension Headers. The idea is that Extension Headers each carry information about some aspect of the packet. There is no fixed size to the headers and no limit to how many there are. Customized Extension Headers are added as required.

The basic structure of Extension Headers

To enable routers to efficiently parse the IPv6 packet, each Next Header field gives a pointer to the next Extension Header, further on in the packet, so the processing required to move from one header to the next is simple.

The first Next Header field is the Main IPv6 Header, as illustrated in the diagram above; see What are the differences between IPv4 and IPv6 headers? on page 240 .

In that example, the first Next Header field tells us that the next Extension Header is Extension Header 1 (EH1). The Next Header field in Extension Header 1 tells us that the next Extension header is Extension Header 2 (EH2).

Now let us look at a specific example of packet decode.

We can see the structure of the Extension Headers in the IPv6 section of the packet:

Traffic Class 8 bits 6 for DSCP 2 for ECN	`1110 0000 .... ... = Traffic class: 0x000000e0` .... 1110 00.. ....= Differentiated Services Field: Class Selector 7 ..0. = ECN-Capable Transport(ECT):Not Set..0 = ECN-CE: Not set
Flow Label	`0000 0000 0000 0000 0000 = Flowlabel: 0x00000000`
Payload length	`Payload length: 52`
In the Main IPv6 Header the Next Header is AH – Authentication Header	
Next header	`Next header: AH (0x33)0x33 Hex` (51 Decimal)
Hop Limit	`Hop limit: 1`
Source IPv6 address	`Source: fe80::1 (fe80::1)`
Dest. IPv6 address	`Destination: fe80::2 (fe80::2)`
	`Authentication Header`
And in the AH Header we see that the Next header is OSPF	

AH Extension Header	`Next Header: OSPF IGP 0x59 Hex (`89 Decimal)
	`Length: 24`
	`AH SPI: 0x00000100`
	`AH Sequence: 22`
	`AH ICV: d0883638d39101562e836679`

There is no Next Header field in the OSPF Header, so we know that this is the last Header in this IPv6 packet.	

OSPF Extension Header	`OSPF Header`
	`OSPF Version: 3`
	`Message Type: DB Description (2)`
	`Packet Length: 28`
	`Source OSPF Router: 1.1.1.1 (1.1.1.1)`
	`Area ID: 0.0.0.1`
	`Packet Checksum: 0xe471 [correct]`
	`Instance ID: 0 (IPv6 unicast AF)`
	`Reserved: 0`
	`OSPF DB Description`

Possible values for the Next Header field

Value	Field	Description
0	HBH	Hop-by-Hop option (IPv6)
1	ICMP	Internet Control Message (IPv4)
2	IGMP	Internet Group Management (IPv4)
3	GGP	Gateway-to-Gateway Protocol
4	IP	IP in IP (IPv4 encapsulation)
5	ST	Stream
6	TCP	Transmission Control
8	EGP	Exterior Gateway Protocol
9	IGP	Any private interior gateway
16	CHAOS	Chaos
17	UDP	User Datagram
29	ISO-TP4	ISO Transport Protocol Class 4
36	XTP	XTP
43	RH	Routing header (IPv6)
44	FH	Fragmentation header (IPv6)
45	IDRP	Inter-Domain Routing Protocol
46	RSVP	Reservation Protocol
50	ESP	Encapsulating Security Payload
51	AH	Authentication header (IPv6)
54	NHRP	NBMA Next Hop Resolution Protocol
58	ICMP	Internet Control Message (IPv6)
59	Null	No next header (IPv6)
60	DOH	Destination Options header (IPv6)
80	ISO-IP	ISO Internet Protocol (CLNP)
83	VINES	VINES
88	IGRP	IGRP
89	OSPF	Open Shortest Path First (OSPF)
93	AX.25	AX.25 Frames

RFC 2460 recommends the order in which the Headers should be chained in an IPv6 packet:

Order	Header Type
1	Basic IPv6 Header
2	Hop-by-hop options
3	Destination options (with routing options)
4	Routing header
5	Fragment header
6	Authentication header
7	Encapsulation security payload header
8	Destination options
9	Mobility header
	No next header
Upper Layer	TCP
Upper Layer	UDP
Upper Layer	ICMP v6

However, this is only a recommendation; the only absolute requirement is that the Hop-by-Hop Extension Header has to be the first one. The Hop-by-Hop Extension Header is the only Extension Header that must be fully processed by all network devices. This is why this Extension Header must be the first in a sequence of Extension Headers.

The purpose of the Hop-by-Hop option (also called the CSI option) is to store status information of nodes along the packet's path. New IPv6 ICMP messages **Status Request**, **Status Reply** and **Status Report** use the Hop-by-Hop option to gather network information that can be used to determine the best path through a network, as well as detect network problems. Network devices are not required to process any of the other IPv6 extension headers when simply forwarding the traffic. Each extension header should occur at most once, with the only exception being the **Destination Options** header, which can occur twice in different positions.

The Hop-by-Hop and Destination Headers carry variable numbers of Options (other Extension Headers do not) – these are carried in a TLV format:

- Option Type
- Option Data Length
- Option Data

Encryption and Authentication in IPv6

IPv4 protocols such as OSPFv2 have authentication incorporated into their own protocol header. In IPv6, authentication and encryption are performed by separate IP headers, completely independent to the enclosed protocol.

AH (Authentication Header)

The authentication information for the Authentication Header is calculated using all the fields of the datagram that do not change in transit. The value stored in the header is the result of passing those fields though a hash algorithm – most commonly MD5 or SHA.

This header can be used as part of IPSec to authenticate packets. The authentication value is calculated and inserted by the device at one end of the conversation, and is checked by the device at the other end of the conversation. This can be used to protect protocols like OSPFv3, IPv6 BGP, RADIUS, TACACS+, and RIPng.

ESP (Encapsulated Security Payload)

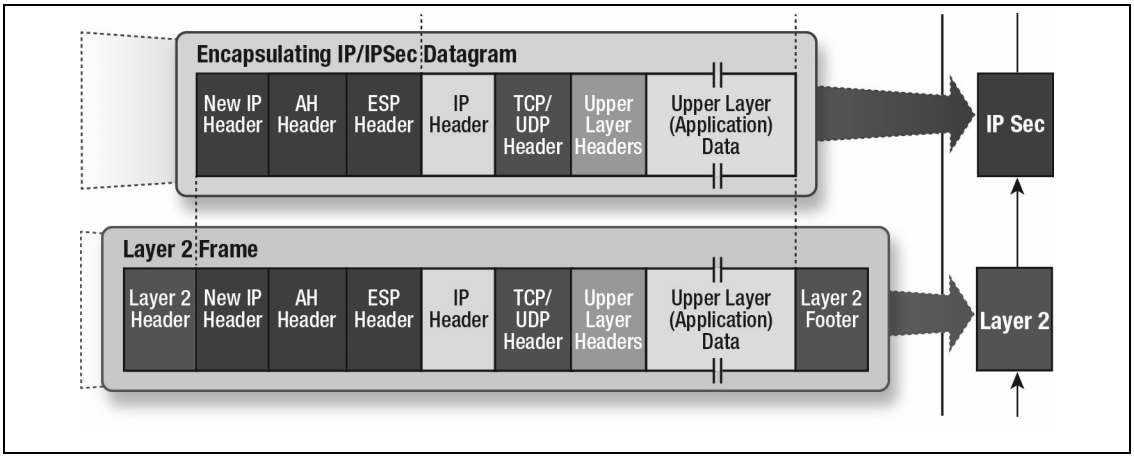
ESP is used to carry encrypted data within an IP datagram. The encrypted data is obtained by applying a specified encryption transform to the data – most commonly 3DES or AES. The recipient of the packet requires a key in order to return the encrypted content to plaintext.

There are two modes used for ESP:

- Tunnel mode
- Transport mode

Tunnel mode, where the entire IP packet is encrypted and/or authenticated. It is then encapsulated into a new IP packet with a new IP header.

ESP Tunnel mode encrypts the whole IP datagram:



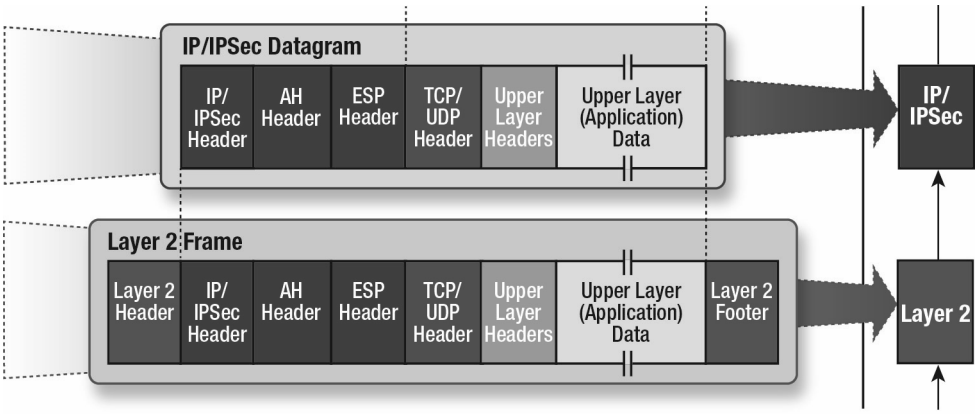
In ESP Tunnel mode, the Authentication Header appears as an Extension Header of the new IP datagram that encapsulates the original one being tunneled.

Transport mode, where **only the payload** of the IP packet is encrypted and/or authenticated – not the original IPv6 Header.

ESP Transport mode encrypts only the payload (the Transport Layer message of the IPv6 datagram). In ESP transport mode, the Authentication Header is placed into the main IP Header before any Destination Options header and before an ESP header.

The Extension Headers are used to secure the IP communication between two hosts, Authentication and Encapsulating Security Payload Headers, are ignored by the intermediary network devices while forwarding traffic. These Extension Headers are relevant only to the source and destination of the IP packet.

All information following the ESP Header is encrypted and cannot be examined by any intermediate device.



The QoS (Quality of Service) Flow Label

The QoS Flow Label is a 20 bit field in the IPv6 packet header which provides an efficient way for packet marking, flow identification, and flow state lookup.

This field can be used by a source to label a set of packets belonging to the same flow.

The switch must process the packets in the same flow in the same manner.

When a Flow Label-aware router receives the first packet of a new flow, it sets up a new flow entry using the information carried by the IPv6 header, Routing header, and Hop-by-Hop Extension Headers, and stores the result.

It then uses the flow entry to route all other packets belonging to the same flow, which will have the same source address and the same Flow Label.

Routing

Static routes

IPv6 forwarding must be enabled before AlliedWare Plus will route IPv6 data packets:

```
awplus# conf t
awplus(config)# ipv6 forwarding
```

This is how you add a **static** route to a switch:

```
awplus# conf t
awplus(config)# ipv6 route 2003:3333::/64 2002: 1111::1
```

This is how you add a **default** route:

```
awplus# conf t
awplus(config)# ipv6 route ::/0 2002:4444::2
```

Dynamic routing protocols for IPv6 are discussed in chapters on OSPFv3, RIPng and BGP4+.

DHCP

While stateless address auto-configuration is the IPv6 way of automatically configuring hosts' network information, it is not fully supported by some vendors, so Dynamic Host Configuration Protocol (DHCP) is still commonly used.

AlliedWare Plus devices can operate as DHCPv6 clients, DHCPv6 relays, and DHCPv6 servers. Note that DHCPv6 client, DHCPv6 server and DHCPv6 relay are mutually exclusive on an interface.

DHCP client

An interface on an AlliedWare Plus device can be set up as a DHCPv6 client interface, i.e. an interface that obtains its IPv6 address by DHCP.

```
awplus# configure terminal
awplus(config)# interface vlan10
awplus(config)# ipv6 enable
awplus(config-if)#  ipv6 address dhcp
```

As well as obtaining an IPv6 address, the client interface will learn a subnet mask, a set of default routes, the addresses of available DNS servers, and a domain-name.

DHCP relay

Where the DHCPv6 server does not reside on the same IP subnet as its clients, a relay agent can act as an intermediate device between the two subnets. The AlliedWare Plus DHCP Relay supports IPv6 addresses, in addition to support for IPv4 addresses.

Example

To enable the DHCP relay agent on your device to relay DHCP packets on interface VLAN 1 to the DHCP server with the IPv6 address `2001:0db8:010d::1` reachable via interface VLAN 2, use the following command sequence:

```
awplus# conf t
awplus(config)# service dhcp-relay
awplus(config)# interface vlan1
awplus(config-if)# ip dhcp-relay server-address 2001:0db8:010d::1 vlan2
```

You can configure up to five servers per interface that DHCP requests can be relayed to.
Note - A DHCP relay configured with an IPv6 server address will relay only IPv6 DHCP packets. To relay IPv4 DHCP packets, a separate relay instance, using an IPv4 server address, is required.

DHCP server

Configuration of the DHCPv6 server in AlliedWare Plus has similarities and differences with the configuration of the DHCP v4 server.

The DHCPv6 server is enabled per interface:

```
interface VLAN10
IPv6 dhcp-server
```

This is different to the approach taken with the IPv4 DHCP server, where the server operates globally. In a similar manner to the DHCPv4 server, pools are defined, from which the server instances can dish out addresses.

```
awplus(config)# ipv6 dhcp pool <name>
```

Within the pool, addresses are defined, which can be allocated from the pool

The addresses can be defined as a subnet, with a prefix and a prefix length:

```
address prefix <ipv6-prefix/prefix-length> [lifetime {<valid-time>|infinite}
{<preferred-time>|infinite}]
```

or as a range of addresses:

```
address range <first-ipv6-address> <last-ipv6-address> [lifetime {<valid-
time>|infinite} {<preferred-time>|infinite}]
```

The pool can be configured with a DNS server address, a domain name, and an SNTP server address to allocate to the clients

```
awplus(config)# ipv6 dhcp pool <name>
awplus(dhcpv6-config)# domain-name <domain-name>
awplus(dhcpv6-config)# dns-server <ipv6-address>
awplus(dhcpv6-config)# sntp-address <ipv6-address>
```

Additionally, user defined options can be created, and configured on a pool

For example:

```
awplus(config)# ipv6 dhcp option 56 name perform-router-discovery flag
awplus(config)# ipv6 dhcp pool <name>
awplus(dhcpv6-config)# option perform-router-discovery true
```

Pools are created in global configuration mode, and then attached to DHCP server definitions in interface configuration mode:

```
interface VLAN10
IPv6 dhcp-server <pool name>
```

Prefix delegation

An aspect of DHCP in IPv6 that does not exist in IPv4 is Prefix Delegation. This is a process whereby a DHCP server allocates a subnet to a downstream router, and that router uses this as the subnet on it downstream interface. If the router has multiple downstream interfaces, it will split up the allocated subnet up into multiple smaller subnets, and spread these smaller subnets across the downstream interfaces.

The subnets that the router allocates to its downstream interfaces then define the prefixes that hosts on those downstream networks will use. The router might simply advertise the prefix in Router Advertisement messages, and the hosts use that prefix as part of their automatic address creation process. Alternatively, the router may, itself, act as a DHCP server, and use the allocated subnets as the pools from which it hands out addresses to DHCP clients on its local networks.

Configuring Prefix Delegation in AlliedWare Plus

This configuration is setup on both the server and the downstream router ('the delegatee').

Server configuration

Configure the server with the following:

1. A subnet to act as a pool from which smaller subnets are handed out to the client routers.

   This is created by using the command:

   ```
   ipv6 local pool <delegation_pool_name> <delegated-prefix-name> <ipv6-
   prefix/prefix-length> <assigned-length> [exclude-local-prefix]
   ```

Where:

- <Prefix-length> the length of the prefix that defines the whole subnet from which the smaller subnets will be allocated to the downstream routers.
- <assigned-length> prefix-length of the smaller subnets allocated to downstream routers
- Of course, <assigned-length> must be a larger number than the <Prefix-length>

2. A server pool definition that is told to use this subnet to satisfy prefix delegation requests:

```
ipv6 dhcp pool <DHCPv6-poolname> prefix-delegation pool <
delegation_pool_name > [lifetime {<valid-time>|infinite} {<preferred-
time>|infinite}]
```

Note - this pool can also contain addresses that are allocated to directly connected hosts, specified with the **address range** and/or **address prefix** commands.

3. A DHCP server definition that uses this pool:

```
interface vlan1
ipv6 dhcp server <DHCPv6-poolname>
```

Putting this all together:

```
ipv6 local pool pd_direct_vlan1 2001:db8:50::/48 56
!
ipv6 dhcp pool pool1
prefix-delegation pool pd_direct_vlan1
!
interface vlan1
description to_PD_Subdelegation
ipv6 address 2001:db8:10::1/64
ipv6 enable
no ipv6 nd suppress-ra
ipv6 dhcp server pool1
```

Client configuration

A router that is operating as a Prefix Delegation client requires:

1. A DHCP pool that will be used to hold the delegated prefix

   ```
   ipv6 dhcp pool <prefix-pool-name>
   ```

2. An upstream interface that is configured as a prefix delegation client interface

   ```
   interface <upstream interface name>
   ipv6 dhcp client pd <prefix-pool-name>
   ```

This configuration causes the router to:

- Send prefix delegation requests out of this interface
- When it receives a delegated prefix, it populates the pool <prefix-pool-name> with the set of addresses in the delegated prefix.

3. One or more downstream interfaces that use subnets from within the delegated prefix

   ```
   interface <downstream interface name>
   ipv6 enable
   no ipv6 nd suppress-ra
   ipv6 address [<prefix-pool-name>] <ipv6-addr/prefix-length> [eui64]
   ```

Where:

- `<prefix-pool-name>` refers to the internal address pool that is defined by the **ipv6 dhcp client pd** command on the upstream interface
- `<prefix length>` is the length of the subnet to be pulled from the pool and used on this downstream interface.
- If `[eui64]` is specified, then the router will use eui64 to create the interface portion of the IPv6 address on this interface. Otherwise, it will use the interface portion defined by `<ipv6-addr>`.

For example:

```
ipv6 dhcp pool pool1
interface vlan6
ipv6 address autoconfig
ipv6 dhcp client pd pool1
!
interface vlan7
ipv6 enable
no ipv6 nd suppress-ra
ipv6 address pool1 ::/64 eui64
```

It is possible to configure an AlliedWare Plus device to use a delegated prefix as the pool for DHCP allocation of addresses on a downstream interface.

```
ipv6 local pool pdpool1 pool1 ::/56 64
!
ipv6 dhcp pool pool1
prefix-delegation pool pdpool1
!
interface vlan1
ipv6 address auto
ipv6 dhcp client pd pool1
!
interface vlan6
ipv6 enable
no ipv6 nd suppress-ra
ipv6 address pool1 ::1/64
ipv6 dhcp server pool1
```

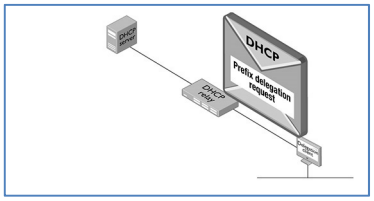
Routing considerations relating to Prefix Delegation

The act of delegating a Prefix to a downstream router requires routing updates in the network, as a subnet is now being used in a certain location in the network, and packets will need to be correctly routed to that subnet. Hence, when a server delegates a prefix to a router that is downstream of a given interface, it must create a route to that subnet, directed out through that interface.

Similarly, when a DHCP relay sees that a server on its upstream side is delegating a prefix to a router on its downstream side, it must make a route to that delegated subnet, directed towards the router that receives the delegated prefix. And, of course, when the router that receives the delegated prefix relinquishes that prefix, the associated routes on the DHCP server and relay need to be deleted.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/bRxK7XyKOm4



Management

Network Time Protocol v6

Network Time Protocol (NTP) is a service that allows networked devices to solicit their system time from a central source. This can either be internal to a company or derived from one of a handful of atomic clocks who provide this time as a service to the Internet community. NTP on AlliedWare Plus can accept time from upstream devices, synchronize with peers, and pass the time on to downstream devices. NTP over IPv6 is identical to NTP over IPv4, except that it uses IPv6 as the network protocol.

Adding an NTP peer

```
awplus# conf t
awplus# ntp peer 2001:0db8:010d::1
```

Once a NTP peer has been configured we can use the **show ntp associations** command to see the current list of NTPv6 associations to the switch.

To see output of more detail on the NTPv6 peer associations, use the following command:

```
show ntp associations detail
```

DNS Client and DNS Relay over IPv6

The Domain Name System (DNS) allows you to access remote systems by entering human-readable device host names rather than IP addresses.

AlliedWare Plus supports Domain Name translation for internal switch applications using remote DNS servers over an IPv6 network via the following commands:

```
ip name server
show ip name-server
```

Additionally **ip dns forwarding** inherently supports the ability for connected hosts to resolve domain names via a remote DNS server over an IPv6 network.

The **ip name-server** command is required in addition to the **ip dns forwarding** command to ensure DNS requests from connected hosts can be relayed to the DNS server.

When your device is using its DHCP client for an interface, it can receive Option 6 from the DHCP server. This option appends the name server list with more DNS servers. The DNS client on your device sends DNS queries to devices on this list when trying to resolve a hostname. Your device cannot resolve a hostname until you have added at least one server to this list. There is no limit on the number of servers you can add to the list.

Example

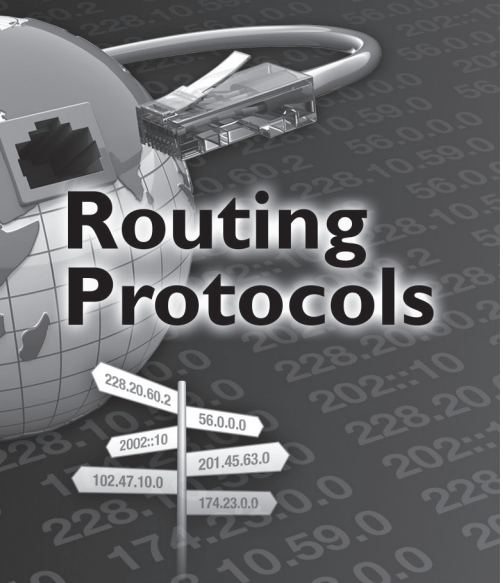
To allow your device to send DNS queries to a DNS server with the IPv6 address `2001:0db8:010d::1`, use the commands:

```
awplus# configure terminal
awplus(config)# ip name-server 2001:0db8:010d::1
```

```
awplus#show ip name-server
Nameservers:
10.10.0.123
10.10.0.124
2001:0db8:010d::1
```

It is important to bear in mind that the type of address returned in response to a DNS query is independent of the IP version that was used to transport the query.

- A DNS query sent by IPv4 (to a server with an IPv4 address) can request an IPv6 address (and, receive an IPv6 address in response).
- A DNS query sent by IPv6 (to a server with an IPv6 address) can request an IPv4 address (and, receive an IPv4 address in response).



Routing Protocols

This chapter covers the following topics:

- Introduction
- Configuring RIPng
- Viewing RIPng information under AlliedWare Plus
- Packet Formats

CHAPTER 6

Routing Information Protocol IPv6 (RIPng)

Introduction

RIPng is the name given to the application of RIP to IPv6.

It is extremely closely based on the IPv4 RIPv2. Really, RIPng is just RIPv2 updated with the minimum set of changes required to enable it to advertise IPv6 routes. Unfortunately, the incorporation of IPv6 into RIP was not just a simple matter of defining a new Address Family ID to be used with RIP, and retaining the existing packet format. Because IPv6 addresses are so much larger than IPv4 addresses, the packet format did require revision.

In the process, a few changes have been made:

1. RIPng uses UDP port 521, rather than RIP's UDP port 520. The destination multicast address used for RIPng packets is FF02::9 – which is the IPv6 equivalent of RIPv2's 224.0.0.9.

2. Next-hop addresses are carried differently in RIPng packets. In RIPv2, there was a next hop field carried with every route entry in the packet. But in RIPng, carrying a next-hop address with each route entry would greatly reduce the number of route entries that could be carried in a single packet, due to the sheer size of IPv6 addresses. Instead, a special Route Entry is used to define a next-hop address. These special next hop entries can appear anywhere in a RIP packet. The next hop address then applies to all the following route entries up until the next next hop entry in the packet.

3. Authentication is not part of the RIPng protocol. Instead, Authentication and/or encryption of RIPng packets is performed by the inherent IPv6 IPSEC, using the relevant next-headers in the IPv6 packet that carries the RIPng data.

4. No Route tag information is carried in RIPng packets.

However, a lot has remained the same:

- Updates are sent at regular intervals (30 seconds, by default).
- The Update packets advertise all the routes in the IPv6 RIP route table, (except those prevented by Split Horizon considerations).
- Split Horizon and Poison Reverse still operate in the same way.
- A metric of 16 is still considered infinite.

- Routes are still set to metric 16 if their invalid timer (180 seconds, by default) expires.
- There is still a garbage collection time (2 minutes, by default).
- Triggered updates are still sent when route changes occur.
- The ability to request route updates remains.

Configuring RIPng

Configuration of RIPng in AlliedWare Plus is one aspect that does differ noticeably from IPv4 RIP.

Rather than having a `Network` command implicitly enable RIP advertisements on the interfaces whose addresses are within the address range specified by the `Network` command, RIPng needs to be explicitly enabled on given interfaces.

The commands to enable RIPng on a VLANx interface are:

```
conf t
interface VLANx
IPv6 router RIP
```

In addition, the RIPng process needs to be enabled globally.

```
conf t
router IPv6 RIP
```

Once RIPng has been configured on an interface, the connected IPv6 route on that interface is immediately imported into the IPv6 RIP route table. Routes can also be redistributed into RIPng

```
redistribute {connected|static|ospf} [metric <0-16>] [route-map <route-map>]
```

Or, a route can simply be defined as an IPv6 RIP route, by using the **route** command:

```
configure terminal
router ipv6 rip
route 2001:db8::1/64
```

RIPng will be sent by multicast from the interfaces on which it is configured. To change to unicasting RIPng to specific neighbors on an interface, define the interface to be passive, and configure the neighbor addresses

```
configure terminal
router ipv6 rip
passive-interface vlan2
neighbor 2001:db8:1::1 vlan2
```

Aggregate routes

RIPng under AlliedWare Plus supports the concept of an aggregate route. An aggregate route can be added by using the commands:

```
conf t
router IPv6 RIP
aggregate-address 2001:dbb:8923::/48
```

The route will not be added to the RIPng database unless the database contains at least one route which is contained within the address range covered by the aggregate route.

As soon there are any such component routes in the RIPng database, then the following occurs:

- The aggregate route is added to the RIPng database
- All the component routes that are within the address range covered by the aggregate route are retained in the RIPng database, but are marked as suppressed routes. The aggregate route will be advertised in RIPng updates, and the component routes will no longer be advertised.

Note that simply having a component route in the IPv6 route database is not a sufficient condition for the aggregate route to be included into the RIPng database. The component route(s) must be in the RIPng database before the aggregate route will be included in the RIPng database. There is no restriction on the method by which the component routes have arrived into the RIPng database - it can be by being connected RIP interfaces, by redistribution or by direct inclusion using the **route** command in **router IPv6 RIP** configuration mode.

Default metric

The following commands set the default value for the metric assigned to routes that are redistributed into RIPng:

```
conf t
router IPv6 RIP
default-metric  x
```

Note - this metric is not applied to routes that are brought into RIPng by using the **route** command in **router IPv6 RIP** configuration mode. This metric is applied to any RIPng aggregate routes that have been brought into the RIPng database due to the presence of a component route that was redistributed into RIPng.

Next Hops

As mentioned above, RIPng does not include a next hop IPv6 address with each Route Table Entry (RTE) in the Update packet. Because IPv6 addresses are long (128 bytes) it is a good idea to look for ways to avoid putting the same IPv6 address into a packet multiple times. Very often, all the RTEs in an Update packet are going to share the same next hop (i.e. the address of the sender of the Update packet). Even if all the RTEs are not sharing the one next hop address it is quite rare that most of the routes in an IPv6 Update packet are going to share more than the same 2 or 3 next hops. So it certainly makes sense to just specify those next hop addresses once, and then associate a set of RTEs with each next hop address.

The method by which a next hop address is associated with a set of RTEs is simple – a next hop entry is created in the packet, and then all the RTEs thereafter are deemed to be associated with that next hop. If there is another next hop entry later in the packet, then the RTEs after that are associated with this new Nexthop, and so on.

As we will see in the section

Packet Formats on page 263, a next hop entry in the packet is actually just a special-format RTE entry.

There are a few of rules about next hop addresses:

1. They must be link-local addresses.
2. The special case of a next hop of 0:0:0:0:0:0:0:0 says that the next hop address is the source IPv6 address of the Update packet. (As we see below, the source IPv6 address of an Update packet must be a link-local address).
3. If the sender has incorrectly put a non-link-local IPv6 address in a next hop entry, then the receiver should act as though the next hop entry contained the address 0:0:0:0:0:0:0:0 (i.e. use the source IPv6 address of the packet as the next hop for the associated RTEs).

Maximum Number of Route Table Entries

In RIPv2, there is a hard limit of a maximum of 25 RTEs in an Update packet. But, RIPng does not put this hard limit on the number of RTEs. Instead, RIPng allows an Update packet to be as large as the link MTU. Because a router never forwards a RIP packet, there is no problem with making the packet as large as the MTU on the sending router's egress interface. There is never going to be a problem with the packet exceeding the MTU of some link further on in its journey, as there are no links further on in its journey. The journey stops at the next router.

The number of RTEs that can be put into a RIPng packet is the number of RTEs that will fit in an MTU-sized packet along with all the headers – IPv6 header, UDP header, and RIP header. This means the RTEs are allowed to fill up all the space that is left once all the headers have been put into the packet.

Miscellaneous Requirements on RIPng Packets

1. The source IPv6 address of a RIPng update must be a link-local address (except in the case of a response to a special 'routed request' messages mentioned in (3) below). If the router has multiple possible link-local addresses on an interface, it needs to keep on using the same link-local address for its Update packets, unless that link-local address becomes invalid for that interface. The reason for maintaining consistency in the source address is that the recipients of the RIP packets are using the source address to identify the neighbor who sent the updates and most likely as the next hop address for the routes learnt from the neighbor.

2. The hop-count in RIPng periodic Update packets must be 255. This provides an extra way for a recipient to verify that the packet arrived from a directly connected neighbor, and has not been inadvertently routed across subnet boundaries.

   Typically, requests and responses will also be within the same subnet, and so will not have the hop-count decremented. But, RIP does provide for the possibility that a server might send routed requests to RIP routers in remote subnets, for monitoring purposes. Such route requests (which will have a source port other then 521) and the responses to those requests will end up with hop counts of less than 255 when they reach their destinations.

3. No route advertised by a RIPng update should ever be a link-local address.

Viewing RIPng information in AlliedWare Plus

Viewing the content of the RIPng database is done by using the command:

`show IPv6 RIP`.

```
awplus#show ipv6 rip
Codes: R - RIP, Rc - RIP connected, Rs - RIP static, Ra - RIP aggregated,
       Rcx - RIP connect suppressed, Rsx - RIP static suppressed,
       C - Connected, S - Static, O - OSPF, B - BGP

   Network                    Next Hop                 If       Met Tag  Time
Rc 2001:dbb:789a::/64         ::                       vlan8      1   0
Ra 2001:dbb:8923::/48                                   --        5   0
Sx 2001:dbb:8923:3456::/64    2001:dbb:789a::2         vlan8      5   0
Rs 2001:dbb:adf4::/64         ::                        --        1   0
S  2001:dbb:c72f::/64         2001:dbb:789a::2         vlan8      5   0
Rc 2001:dbb:d29a::/64         ::                       vlan4      1   0
Ra 2001:dbb:fc31::/48                                   --        1   0
Rsx 2001:dbb:fc31:5ae9::/64   ::                        --        1   0
```

In the example above, you can see various types of routes:

Rc routes are connected routes on interfaces that have been configured with the command: router ipv6 `RIP`.

Ra routes are routes created by the `aggregate route` command in `router IPv6 RIP` configuration mode.

Rs routes are routes that have been brought into RIPng by using the Route command in `router IPv6 RIP` configuration mode.

S routes are static routes that have been redistributed into RIPng

The Rsx and Sx routes are routes that have been suppressed because they are component routes of the aggregate routes that have been brought into the RIPng database.

Note that all Ra and Rs routes have "—"in the `if` (interface) column, as those entries are internally generated and so are not associated with any egress interface.

Some information about the active (and inactive) RIPng interfaces is provided by the command:

`show IPv6 RIP interface`

```
awplus#show ipv6 rip interface
lo is up, line protocol is up
  RIPng is not enabled on this interface
vlan1 is up, line protocol is up
  RIPng is not enabled on this interface
vlan2 is up, line protocol is up
  RIPng is not enabled on this interface
vlan3 is up, line protocol is up
  RIPng is not enabled on this interface
vlan4 is up, line protocol is up
  Routing Protocol: RIPng
    Passive interface: Disabled
    Split horizon: Enabled with Poisoned Reversed
    IPv6 interface address:
      2001:dbb:d29a::1/64
      fe80::eecd:6dff:fe48:e67c/64
vlan5 is up, line protocol is up
  RIPng is not enabled on this interface
vlan6 is up, line protocol is up
  RIPng is not enabled on this interface
vlan7 is up, line protocol is up
  RIPng is not enabled on this interface
vlan8 is up, line protocol is up
  Routing Protocol: RIPng
    Passive interface: Disabled
    Split horizon: Enabled with Poisoned Reversed
    IPv6 interface address:
      2001:dbb:789a::1/64
      fe80::eecd:6dff:fe48:e67c/64
```

And, information about the configuration of RIPng itself is provided by the command:

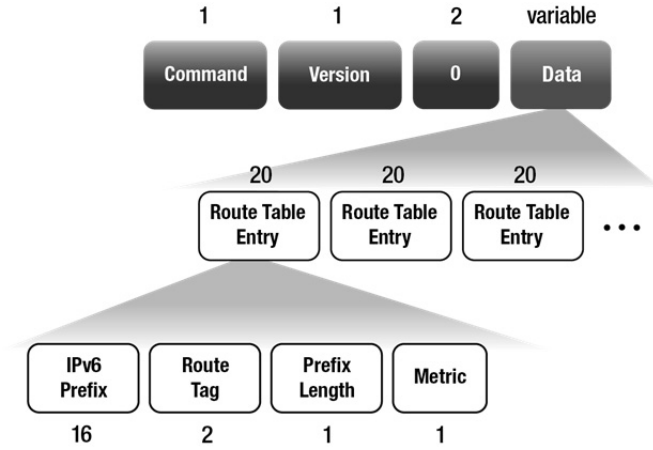
```
show IPv6 protocols rip
```

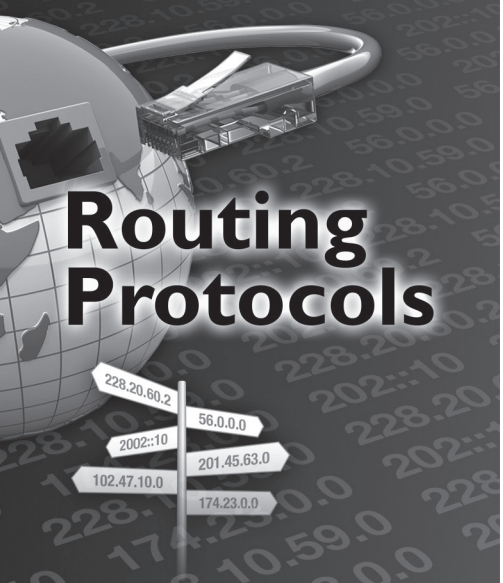
```
awplus#sh ipv6 protocols rip

Routing Protocol is "RIPng"
  Sending updates every 30 seconds with +/-5 seconds, next due in 6 seconds
  Timeout after 180 seconds, garbage collect after 120 seconds
  Outgoing update filter list for all interface is not set
  Incoming update filter list for all interface is not set
  Default redistribute metric is 5
  Redistributing: static
  Interface
    vlan4
    vlan8
  Routing for Networks:
```

Packet Formats

The RIPng packet format is:





Routing Protocols

This chapter covers the following topics:

- Introduction
- Summary of Differences between OSPFv2 and OSPFv3
- What aspects of OSPFv3 are the same as OSPFv2?
- LSAs in OSPFv3
- Comparison of OSPFv3 and OSPFv2 LSAs
- Configuring OSPFv3 in AlliedWare Plus
- Instances and Processes
- Configuring OSPFv3 Authentication
- OSPFv3 Packet Formats

CHAPTER 7

OSPFv3

Introduction

OSPFv3 is OSPF for IPv6. This version of the protocol has no support for IPv4, it is purely for IPv6. So, there is no sense in which OSPFv3 supersedes OSPFv2. Rather, OSPFv2 is still the current OSPF version for IPv4 routing, and OSPFv3 is the current OSPF version for IPv6 routing.

As you might expect, OSPFv3 is closely modeled on OSPFv2. So, OSPFv3 can be looked upon as an adaptation of OSPFv2 to IPv6 routing. Given that OSPFv2 has been very successful as an IPv4 routing protocol, there has not been a burning need to make significant changes when adapting it to IPv6.

However, there are some differences between OPSFv2 and OSPFv3. The prime reasons for the changes are:

- OSPFv3 runs on IPv6, and so can make use of some of the built-in features of IPv6 that were not present in IPv4
- As OSPFv2 was used in the field, various shortcomings became evident. The lessons learnt from the field experience of OSPFv2 were fed into the development of OSPFv3, so that is would not exhibit the same shortcomings.

There is little point in this chapter dwelling on the detail of all those aspects of OSPFv3 that are the same as OSPFv2, as those areas have already been described in detail in the chapter on OSPFv2. So, the main focus of this chapter will be the aspects of OSPFv3 that differ from OSPFv2.

Let's begin with an overview of what these areas of difference are.

Summary of Differences between OSPFv2 and OSPFv3

Separation of topology from addressing

The biggest change that has been introduced in OSPFv3 is the separation of network topology from the location of addresses.

In OSPFv2, the Router LSAs and Network LSAs contain IP address information. The Network LSA's LSID (along with the mask) identifies the IP subnet in use on the network in question. The Link entries in Router LSAs contain the IP addresses in use on the link. So, these OSPFv2 LSAs contain both topological information (who is connected to who) and address information (what IP addresses are in use on these topological connections).

One of the drawbacks of the close connection between IP addressing and topology in OSPFv2 is that secondary IP addresses are not entirely well handled. OSPFv2 will advertise secondary IP addresses as stub networks, even though they could well be equally as distributed across a transit network as the associated primary addresses are.

In the development of OSPFv3, recognition has been given to the fact that multiple address ranges can co-exist on the same physical network segment. So, to accommodate this situation, OSPFv3 works by advertising just topological information in Router and Network LSAs, and advertising address information in quite separate LSAs (new LSA types that did not exist in OSPFv2). In this way, the SPF calculation can work out the optimal topology, and then the IPv6 subnets in use on the different segments of the network can be separately overlaid on that topology.

To enable OSPFv3 to operate in this way, a number of changes have been made:

- Router and Network LSAs do not advertise any IP addresses. This is discussed in more detail in the sections Router LSA and Network LSA on page 275
- A new 'link' LSA has been introduced, that advertises the IPv6 addresses that a router is using on the interface that connects to a given link. This is discussed in more detail in the section Link LSA on page 277.
- A new intra-area-prefix LSA has been introduced. An intra-area-prefix LSA can be a companion to a Router LSA or to a Network LSA (some fields in the intra-area-prefix LSA identify the particular Router or Network LSA to whom it is a companion). The purpose of the intra-area prefix LSA is to carry the address information relating to its companion LSA. This is discussed in more detail in the section Intra-Area Prefix LSA on page 281.

Explicit identification of flooding scope for LSAs

In OSPFv2, there is a concept of flooding scope for LSAs. Most LSAs (Router, Network, Area Summary, ASBR Summary, Type-7) have a flooding scope of the local area – they are flooded throughout the local area, and are not flooded anywhere else. External LSAs have a flooding scope of the whole AS (except for stub areas).

These conventions are simply just known; they are not coded into the LSAs in any way.

In OSPFv3, there is a definite drive to make the protocol more extendable. One barrier to extendibility is conventions that are 'just known'. If a new type of LSA comes along in OSPFv2 (like the various Opaque LSAs), then older implementations of OSPFv2 don't know what to do with these LSAs, as there is nothing in the LSA that encodes its flooding scope.

OSPFv3 introduces a field to the LSA header that explicitly states the flooding scope of the LSA.

Moreover, OSPFv3 introduces a new type of flooding scope – the **link scope**. LSAs with link flooding scope are received by neighbors but go no further.

The explicit encoding of flooding scope is discussed in more detail in the section Explicit identification of flooding scope for LSAs on page 267. The concept of link flooding scope is discussed in more detail in the same section.

Ability to deal more effectively with unknown LSA types

As part of the move to make the protocol more extensible, LSAs in OSPFv3 contain a field that explicitly instructs routers what to do with the LSA if they do not recognize its type. The field tells the router to either treat the unknown LSA as though it had link flooding scope, or to treat it according to the flooding scope encoded in the LSA's flooding scope field. If the LSA is to be treated as though it had link flooding scope, then it is simply stored away, but not sent on to any neighbors. If it is to be treated according to the flooding scope in its flooding scope field, then it is stored in the LSA database, and forwarded the appropriate neighbors in the same way that a known LSA type with the same flooding scope would be.

Simply dropping unknown LSAs is out of the question.

The fact that routers are obliged to forward unknown LSAs into other areas if the LSA has AS flooding scope, there is a danger that large numbers of unknown LSAs could be flooded into stub areas, which rather defeats the goal of keeping the LSA databases in stub areas small. Special rules have been added with regard to how the border routers on the border of stub areas should treat unknown LSAs.

The whole business of dealing with unknown LSAs is discussed in more detail in the section Flag for how to handle unknown LSAs on page 272.

LSA and packet format changes

A number of references are made just above and just below, to fields that have been added to, or removed from, the OSPF packet and LSA formats. There are also a certain number of other changes that are not referred to just here. Each change has a good reason for it, and it is worth understanding each change, and why it has been made.

The changes are not all discussed together in one place, but are discussed at various appropriate places.

Splitting data across multiple LSAs

As described in Router LSA on page 275, a router's link information can be spread across multiple router LSAs.

Authentication

In OSPFv2, there is an authentication capability built into the OSPF protocol. But, OSPFv3 is an IPv6 protocol, and one of the inherent improvements in IPv6 is the provision of a generic authentication capability, in the form of extension headers. Therefore, OSPFv3 does not need an authentication capability in the protocol. Instead, if users decide to employ authentication of their OSPFv3 exchanges, the IPv6 stack will insert the appropriate extension header into the packets.

Using link-local IPv6 addresses

Another little improvement that came along with IPv6 is the concept of the link-local address. For situations where routers simply want to talk directly to each other on a given link, rather than route data to some further destination, they can use link-local addresses on the packets. OSPF, of course, is a classic example of routers talking directly to each other.

The exchange of OSPF packets is exactly the sort of conversation that the link-local address was designed for.

Not surprisingly, then, OSPFv3 packets have link-local source and destination addresses. The exception to this, of course, is virtual links. Packets sent over a virtual link must use the routable IP addresses of the source and destination interfaces, as these packets are forwarded through routers.

Note that the use of link-local addresses for OSPF packets effectively removes one of the criteria that routers must check when deciding if they can become neighbors. The criterion in question, of course, is whether the source IP address of the incoming Hello packets is in the same subnet as the IP address on the receiving interface. This becomes a non-issue in OSPFv3 because link-local addresses, by their very nature are in the same subnet.

This is another aspect of the way that OSPFv3 separates topology from addressing. The addresses on the participating interfaces are irrelevant to the formation of neighbor relationships. Routers simply become neighbors because they share a link – i.e. because of their topological relationship. IP addresses are not relevant to this act of forming the neighbor relationships that build up the OSPFv3 topological map.

Running multiple OSPF instances on a shared link

Sometimes, if different service providers, with separate routing domains, are sharing a common link, then it is useful for them to run two separate instances of OSPF over that link.

In OSPFv2, routers can run multiple OSPF instances, but support for this is implementation-specific, rather than inherent in the protocol. When different OSPF instances share the same link, things can get awkward, because nothing in the packets identifies which instance they belong to. Typically, people have resorted to using authentication to enable the different OSPF instances to process only those packets that are intended for them.

In OSPFv3, the **Instance ID** is an actual field in the OSPF packet header, so the separation of different instances on the same link is inherently supported by the protocol. We will not look at multi-instance OSPFv3 in any detail, but will touch on the Instance ID when discussing the format of OSPFv3 packet headers. See OSPFv3 Packet Formats on page 291.

What Aspects of OSPFv3 are the Same as OSPFv2?

The simple answer is "*just about everything else*".

- The formation of neighbor relationships is the same. Routers still discover each other using Hello packets, and still check the same criteria before forming a neighbor relationship (although, as noted above, the checking that their IP addresses are in the same subnet has become effectively meaningless in OSPFv3). The neighbor relationship establishment still goes through the same states (Two-Way, Exstart, Exchange, etc). The timers (Hello, Dead timer, etc) are still the same.
- The various modes of operation on different network types (Broadcast, NBMA, Point-to-Point, Point-to-Multipoint) are still the same.
- Electing DR and BDR on broadcast and NBMA networks is still the same.
- The packet types (Hello, LSupdate, Database Description, etc) are the same.
- The rules about areas are still the same – the AS must have a backbone area, all other areas must touch the backbone (even if only via a virtual link). Stub areas and Totally Stubby areas and Not So Stubby areas are all still there.
- Routers still have link-state databases, and have to reoriginate their LSAs every 30 minutes, and age out LSAs that have an age over 60 minutes. All the LS databases in a given area still have to be identical.
- The reliable transport process for LSAs (acknowledgements, retransmits) still works in the same way. Sequence numbers are still used to determine the latest version of an LSA.
- External LSAs still come in Type-1 and Type-2 flavors.
- The split horizon rules are still the same.
- Redistribution, Summarization, Filtering, are all essentially the same.
- Graceful Restart still operates the same way as in OSPFv2

LSAs in OSPFv3

The changes made to LSAs between OSPFv2 and OSPFv3 fall into the following categories:

- changes to the format of the LSA header
- separation of topology from addressing
- introduction of new LSA types
- changes to inter-area and AS External LSAs

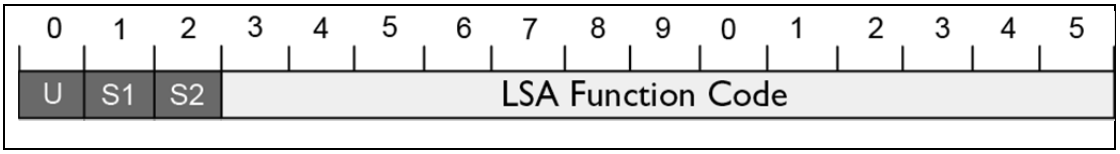
Let us work through each of these categories.

Changes to the format of the LSA header

The new format for the LSType field

In OSPFv2, the LSType is simply a 2-byte field that holds a number indicating what type of LSA this is. The LSType for a Router LSA is 1, for a Network LSA is 2, and so on.

In OSPFv3, the LSType field has been expanded to 16 bits, so that it can include some new flags. The format of the LSType field is now:



Where:

U is the flag that indicates how to handle this LSA if the router does not recognize its type.

The details of this are described below in the section Flag for how to handle unknown LSAs on page 273.

S1, S2 are the two bits that make up the indicator of the LSA's flooding scope. The details are described below in the section Flooding scope indicator on page 273.

LSA function code is the number that actually identifies the type of LSA.

The following table contains the standard LSA types, as stated in RFC 2740:

LSA function code	LS Type	Description
1	**0x2001**	**Router-LSA**
2	**0x2002**	**Network-LSA**
3	**0x2003**	**Inter-Area-Prefix-LSA**
4	**0x2004**	**Inter-Area-Router-LSA**
5	**0x2005**	**AS-External-LSA**
6	**0x2006**	**Group-membership-LSA**
7	**0x2007**	**Type-7-LSA**
8	**0x2008**	**Link-LSA**
9	**0x2009**	**Intra-Area-Prefix-LSA**

The LSA function code values are the familiar values from OSPFv2 - the value for the Router LSA is 1, the value for the Network LSA is 2, and so on. Then, the actual LSType values are these crazily large values like 0x2001. The values are so large because the U an S flags are the highest-order bits of the LSType field. A Router LSA, for example, has a zero for its U flag, and a value of 1 for its flooding scope (1 = flood through the local area). Those flag bits, situated right up at the expensive end of the 4-byte number, add 0x2000 to the Router LSA's 'LSA function code', to give an overall value of 0x2001 as the LSType for a Router LSA.

Flag for how to handle unknown LSAs

As described above in the section Changes to the format of the LSA header on page 267, the LSA header now contains a field that tells a router how to deal with the LSA if the router does not recognize the LSA's type. In fact, this field is a bit that is included in the LSType field. Being a one-bit field, this flag can have two possible values: 0 or 1.

0 - indicates that the LSA should be treated as though it has link-local flooding scope. What this treatment amounts to is that the LSA is stored in the LSA database, in a section of the database that is specific to the interface on which the LSA was received. The contents of the LSA are not examined in any meaningful way, and no routing calculations are performed due to the arrival of the LSA.

The LSA is not flooded anywhere, as the only place to flood it is back out the interface it arrived on, which is pointless. But despite that, there is some point in storing the LSA, as there are occasions when a router will have cause to transmit a link-local LSA that it received from someone else. For example, when a neighbor comes out of Graceful Restart, a router is expected to send back to it any LSAs that it had received from that neighbor.

1- indicates that the LSA should be stored, and flooded in accordance with its indicated flooding scope (indicated by the next 2 bits in the LSType field). This means that as far as storing and flooding is concerned, this LSA should be treated as though the router recognizes its type. Of course, this does not mean that the router that does not recognize the LSA's type will not be able to make use of the content of the LSA in any meaningful way. But, it does mean that they LSA will be forwarded on to other routers that may recognize its type, and know what to do with its contents.

Flooding scope indicator

The second and third bits in the LSType field are called S1 and S2. Together, they form a two-bit number that indicates the flooding scope of the LSA.

Being a two-bit number, this indicator can take on four possible values:

S2	S1	Value	Flooding Scope
0	0	0	Link-Local Scoping
0	1	1	Area Scoping
1	0	2	AS Scoping
1	1	3	Reserved

Most of the standard LSAs (Router LSA, Network LSA, etc.) has an area flooding scope – they are sent to all routers in the same area, but not further.

Of the standard LSAs, the only one with a AS flooding scope is the External LSA.

There is one standard LSA, the new Link LSA that has link-local flooding scope – it is not forwarded on to anywhere.

Less significance in the LS ID

In OSPFv2, for a lot of types of LSA, the LS ID is strongly associated with the content that the LSA is carrying. For example, for each of Network LSAs, Area Summary LSAs and External LSAs, the LS ID is the network address being advertised by the LSA.

In keeping with OSPFv3's drive to achieve expandability, and keep IDs more generic, for most OSPFv3 LSAs, there is no connection between the LS ID and the content of the LSA. For most LSAs, the LS ID is just a label and nothing more.

The exceptions to the 'the LS ID is just an arbitrary label' rule are the Network LSA and the Link LSA. In the case of the Network LSA, the LS ID is the interface ID of the interface via which the DR connects to the network in question. In the case of the Link LSA, the LS ID is the interface ID of the interface via which the originating router connects to the link in question. The values used for Interface IDs will be described below in the section Interface ID on page 274.

The Options field

OSPFv2 defines an Options field that appears in all LSA headers, and in Hello and DD packets. The Options field contains indicators of the properties of the originating router – for example, whether its sending interface is in a stub area, whether it supports OSPF-on-demand, how it handles Type-7 LSAs, etc.

OSPFv3 also defines an Options field. It has a slightly different set of options to OSPFv2, and has been expanded out to 3 bytes, to make generous provision for adding new options in the future. It continues to appear in Hello and DD packets, but no longer appears in the LSA header. The reason that it no longer appears in the LSA header is that for most types of LSA, the information in the Options field was not particularly relevant; it was just extra data being carried around for no benefit.

However, for those LSAs for which the Options field is relevant – Router, Network, Inter-Area Router, and Link LSAs – the field is still present. It is carried in the body of the LSA, rather than the header.

The options defined in OSPFv3 are described in more detail in the section Options flags on page 284.

Separation of topology from addressing

The two elements that carry around topological information in OSPF are the Router LSA and the Network LSA. In OSPFv2, these LSAs carry around addressing information as well as topological information. However, in OSPFv3, a separation between topology and addressing has been enacted. To understand how that has actually been achieved, let us look at the structure of the Network and Router LSAs in OSPFv3.

But, first, it is useful to understand one small aspect of OSPFv3 that is a key to making the whole separation of topology and addressing possible. This aspect is the Interface ID.

Interface ID

In OSPFv2, interfaces are typically identified by their IP address. For example, the links in a Router LSA are mostly identified by the IP address of the interface where the link attaches to the router.

In OSPFv3, the idea is to identify interfaces by something other than their IPv6 address, so that addressing is separated from topology. Hence, IPv6 uses an "*Interface ID*" to identify interfaces.

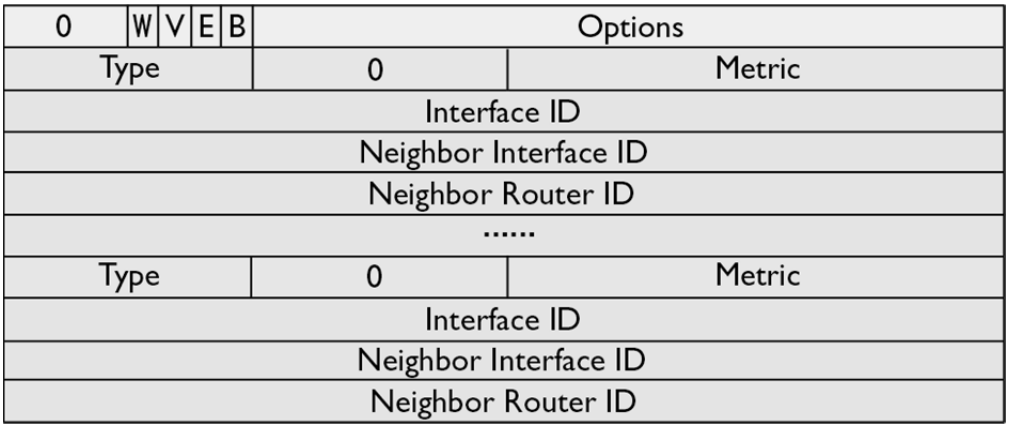
There is no rule about what numbers or strings that routers use as Interface IDs. Routers can choose to use whatever scheme they like for the deciding on the IDs to apply to their interfaces. The only requirement is that each interface on a router must have a different ID. A popular convention is to use the **IfIndex** that is allocated to the interface by the SNMP Interface MIB.

Router LSA

The Router LSA in OSPFv3 has the same structure as that in OSPFv2, except that:

- Interface IDs are used instead of IP addresses to identify the interfaces where the LSA's links connect to the router.
- Interface IDs are used to identify the interfaces where the LSA's links meet the neighbors on these links. For instance, for a link into a broadcast or NBMA subnet, the 'neighbor ID' is the Interface ID that the DR advertises in the Hello packets that it sends into this subnet.
- There is an Options field within the LSA. The details of this field are described just below in the section Router LSA Format.
- The fields that advertise different metrics for different TOS values has been removed. The business of having different metrics for different TOS values seemed like a good idea when OSPFv2 was first defined. However it was not useful in practice. So, the idea has not been carried through into OSPFv3.
- The advertised information may be spread across multiple router LSAs. In OSPFv2, a router would originate a Router LSA from each of its interfaces. This Router LSA would contain all the Router-LSA information that the router needed to advertise into the area attached to the originating interface. But in OSPFv3, it is possible to send a set of Router LSAs out a given interface. The information that the router needs to advertise into the attached area is spread across the LSAs – some information in the first one, some more in the second one, yet more in the third one, and so on. Each of the Router LSAs in the set has a different LS ID. Receiving routers need to concatenate together all the information they receive from the set of Router LSAs.

Router LSA Format



In addition to the Options field, there are some specific flags in the router LSA:

- The V, E and B flags are equivalent to those in OSPFv2.
- The W flag is specific to Multicast OSPF. It indicates that this router wishes to be on the forwarding path for ALL multicast streams being transmitted into the network.

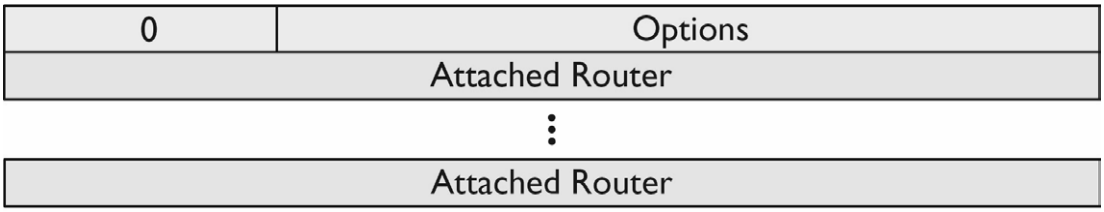
This sort of Wildcard router (i.e., one that receives all streams (*,*)) is typically the router that feeds multicast out of the current multicast domain into a neighboring domain. Of course, MOSPF has not had much usage in the industry, so the W flag is not of much importance.

Network LSA

The main changes between the Network LSA in OSPFv2 and OPSFv3 are:

- The LSID of the OSPFv3 Network LSA is the Interface ID of the interface via which the Designated Router connects to this network. Whereas in OSPFv2, the LSID of a network LSA is the IP address of the interface via which the Designated Router attaches to the subnet in question.
- The Options field is included within the LSA, rather than in the LSA header.
- There is no Network Mask field in the LSA. This is, yet again, because of the separation of topological information and address information. The network mask is part of the addressing information carried in the OSPFv2 Network LSA. But the OSPFv3 Network LSA is an addressing-information-free zone, so it has no reason to carry a Network Mask field.

Network LSA format



Each of the attached routers, i.e. the routers in the subnet that are fully adjacent to the Designated Router, is represented by its Router ID. In fact, the Designated Router is also included in the list of Attached Routers.

The way that the Options field is filled in is interesting. The options in that field really are quantities that describe properties of individual routers – like whether it supports OSPF-on-demand, whether it is an active router, etc (the full list is described below in Options Flags). So, it is a little odd for a Network to have these attributes, and initially hard to know how a network would determine its values for these attributes. In fact, as we will see in more detail below in the section Link LSA on page 277, the value of the items in the Options field in the Network LSA is a logical OR of all the attached router's values for those options. The attached routers advertise their option values to the Designated Router in Link LSAs, and then the Designated Router combines them to create the Options field for the Network LSA.

Introduction of new LSA types

OSPFv3 introduces two new LSA types, both of which assist with the separation of topology and addressing. The two new LSA types are:

- Link LSA
- Intra-Area Prefix LSA

Link LSA

Despite its name, the Link LSA does not end up being a topological entity in the SPF calculation. Instead, it is an entity that enables a router to send its interface address(es), and other information, to its neighbor routers on a given link.

The Link LSA is just a way of filling in other routers with the information they will not have got from the router's Router LSA.

Each interface of a router is attached to a different link. A router will advertise a different Link LSA out each of its interfaces. The main difference between the Link LSA that a router sends of one link and that which sends on another link are the interface addresses that it contains. Each different Link LSA originated by a router will contain the addresses of the interface out of which the Link LSA is advertised.

In fact, the Link LSA has three information-sharing purposes:

1. To let other routers on a link know the link-local address of the originating router. Then, when those routers need to put a next hop address onto the routes that they calculate to be via this router, they know what link-local address to use as that next hop.

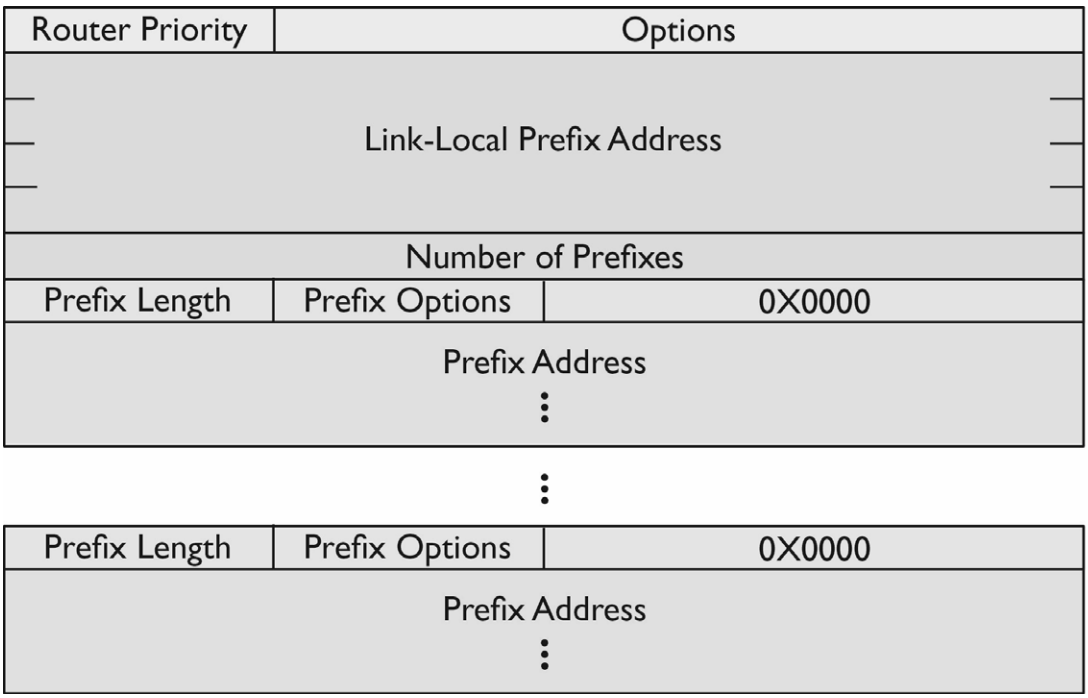
   You might be thinking "*Well, won't the other routers on the link learnt this router's link-local LSA from the source address of its Hello packets?*". In fact, most of the time they will. But in NMBA networks, it is possible that not all routers on the same link (well, in the same subnet) will receive each others' Hello packets. But, the Designated Router is responsible for ensuring that they all see each other's Link LSAs.

2. To advertise the IPv6 prefix(es) that the router has been configured with on the interface which is transmitting the Link LSA. In particular, this informs the Designated Router which Prefix(es) the originating router has. The Designated Router needs to keep a list of all the Prefixes in use on its link, so it can advertise this in an Intra-Area Prefix LSA.

3. To send the router's option values to the Designated Router. Each router on a link effectively tells the Designated Router "*these are my option values; please take note of them when you work out what to send in the Network LSA*". The Designated Router accumulates all the option values it receives from its neighbors on the link. Then, when it needs to fill in the Options field in its Network LSA, it performs a logical OR on all the Options fields it has received in the Link LSAs from the neighbors on the link that the Network LSA

represents. Effectively, if any router on a given link has a "1" for any given option, then the Network LSA for that link will have a "1" for that option.

Link LSA format

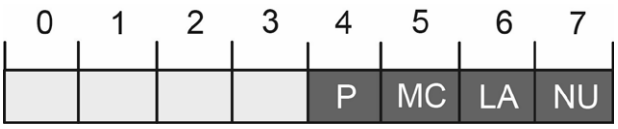


The LS ID of a Link LSA is the Interface ID of the originating interface.

The Router Priority is the value used in electing the DR. The router with the highest Router Priority on a given link will win the DR election (the tie-breaker is the Router ID).

Each prefix that is configured on the originating Interface is included in the Link LSA. The information advertised for each prefix is: Prefix Address, Prefix Length and Prefix Options.

The Prefix Options is an 8-bit field. So far, meanings have been defined for only 4 of those bits.



The options are used when routers are doing route calculations.

NU is the "*no unicast*" bit. If this bit is set in the Options field for a prefix, then the prefix should be omitted from any calculations of IPv6 unicast routes.

LA is the "*Local Address*" bit. This indicates that the prefix is an IPv6 address that is in use on the originating interface.

MC is the "*multicast*" bit. This indicates that the prefix should be used in calculations for the Multicast route table. So, it represents a prefix that is used for the RPF path in multicast path calculations.

P is the "*propagate*" bit. This is a prefix that originates in an NSSA area, and is allowed to be seen by routers outside of the originating area. So, the ABR at the border of the originating NSSA is allowed to propagate this prefix out to other areas in an Inter-Area Prefix LSA.

Intra-Area Prefix LSA

Because the Router and Network LSAs do not carry around any address information, then some other LSA has to do that job for them. The LSA that picks up the slack is the Intra-Area Prefix LSA.

Every router originates zero or more Intra-Area Prefix LSAs to advertise the addresses on some of the links that are advertised in its Router LSA(s). The rules about which links' addresses are included are discussed below.

Every router that is a Designated Router on a Broadcast or NBMA segment originates one or more Intra-Area Prefix LSAs to advertise the addresses that is has learnt are in use on that segment.

Advertising addresses on behalf of router LSAs

A router runs through all the IPv6 prefixes that is has configured on its links:

- Stub
- Point-to-Point
- Point-to-Multipoint

It creates Intra-Area Prefix LSAs to advertise these prefixes. The prefixes can be distributed across multiple Intra-Area Prefix LSAs. If the router is advertising multiple Router LSAs, there is no requirement that there be a one-to-one correspondence between the Intra-Area Prefix LSAs and the Router LSAs. I.e., if there is a Router LSA that advertises a certain set of links, then there is no need for one of the Intra-Area prefix LSAs to advertise the addresses for that exact set of links. So, the router can distribute the prefixes across the Intra-Area LSAs in whatever way it sees fit.

Note - the router does not create Intra-Area Prefix LSAs for transit links (links into Broadcast or NBMA networks on which there are OSPF neighbors attached). This is because the Designated Router on those networks will advertise those prefixes in the Intra-Area Prefix LSA(s) that correspond to its Network LSA.

If all the links attached to a given Router are links into Transit networks, then the router will not have to originate any Intra-Area Prefix LSAs on behalf of its Router LSA(s).

Advertising addresses on behalf of Network LSAs

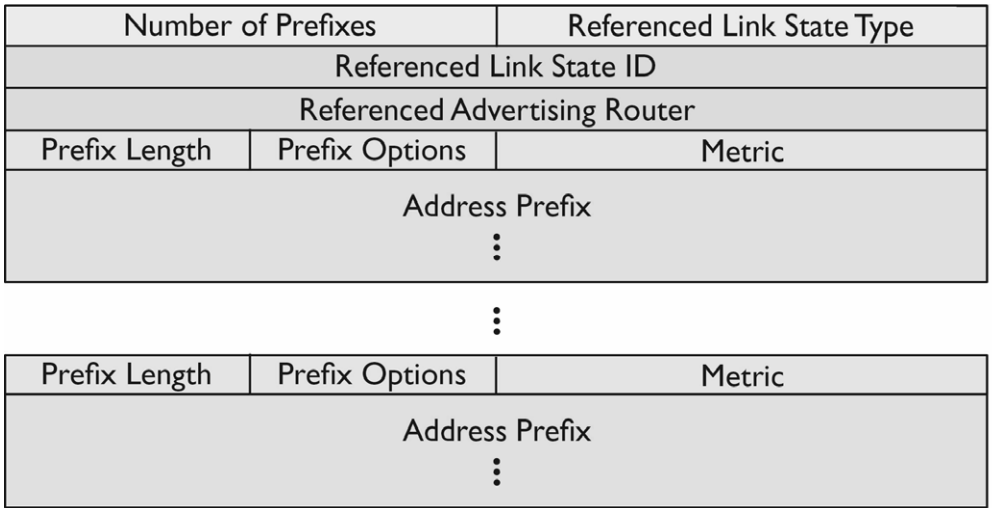
As described above in the section Link LSA on page 277, the Designated Router on a Broadcast or NBMA network will receive a list of the prefixes being used on the network from its neighbors' Link LSAs.

The Designated Router works through this list of Prefixes, and sends them out in one or more Intra-Area Prefix LSAs. Prefixes that have been advertised with the NU or LA bit set in their corresponding Prefix Options are not included in any Intra-Area Prefix LSA.

It is highly likely that if a Designated Router has multiple neighbors then it will receive multiple copies of some, if not all, the prefixes it receives. This is because it is extremely likely that all the routers attached to a given network are going to be using at least some prefixes in common. If the Designated Router does have multiple copies of some prefixes, it treats them as duplicates, so that the Intra-Area Prefix LSAs have just one entry for any given prefix.

The Prefix Options associated with a Prefix is a logical OR of the Prefix Options with which the different copies of the Prefixes were advertised to the Designated Router.

Format of the Intra-Area Prefix LSA



The Referenced Link State field will have one of two values:

- 0x2001 when the Intra-Area Prefix LSA is advertising prefixes on behalf of a Router LSA
- 0x2002 when the Intra-Area Prefix LSA is advertising prefixes on behalf of a Network LSA

The Referenced Link State ID is filled in as:

- 0, when the Intra-Area Prefix LSA is advertising prefixes on behalf of a Router LSA.
- The Network LSA's LS ID (i.e. the Interface ID of the Interface by which the Designated Router connects to the network in question), when the Intra-Area Prefix LSA is advertising prefixes on behalf of a Network LSA.

The Referenced Advertising Router is the originating router, in both the cases.

Changes to Inter-Area and as External LSAs

The Type-3, Type-4, Type-5 and Type-7 LSAs are quite similar in OSPFv3 to OSPFv2. They carry essentially the same information in OSPFv3 as they do in OSPFv2; however the information is structured a bit differently.

There are some cosmetic changes:

- The Type-3 LSA has a different name in OSPFv3, where it is called the Inter-Area Prefix LSA.
- The Type-4 LSA has a different name in OSPFv3, where it is called the Inter-Area Router LSA.
- The LSA header on these LSAs has been changed, as it has been for all LSAs.

And, there are changes that more significantly change the structure of the LSAs.

Let's look at each of the three LSA types, and understand how their structures have changed in OSPFv3.

Inter-Area Prefix LSA

The Link State ID of an Inter-Area Summary LSA in OSPFv2 is the IP network address of the route being advertised by the LSA. In OSPFv3, the Link State ID of an Inter-Area Prefix LSA has no addressing significance at all, it is simply an identifier.

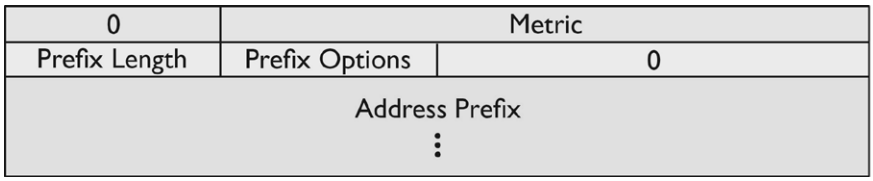
In OSPFv2 the body of the Inter-area Summary LSA contains the route's network mask, its metric, and then an optional set of per-ToS metrics.

The content of the OSPFv3 Inter-Area Prefix LSA is actually quite different.

It contains:

- a metric
- the prefix length (as against a network mask)
- a set of Prefix Options (as is typical of OSPFv3 LSAs)
- the IPv6 prefix address.

Format of the Inter-Area Prefix LSA



Inter-Area Router LSA

The format of a Type-4 Summary LSA in OSPFv2 is exactly the same as that of a Type-3 Summary LSA.

However in OSPFv3, the Inter-area Router LSA has a different format to the Inter-Area Prefix LSA. The Inter-area Router LSA has no need for a prefix length, but it does contain an Options field. The content of this Options field is the same as the content of the Options field in the target ASBR's Router LSA. So, the Inter-Area Router LSA informs recipients of the capabilities of the target ASBR.

The final field in the Inter-Area Router LSA is the Router ID of the target ASBR.

As with the Inter-Area Prefix LSA, the LS ID of the Inter-Area Router LSA has no significance – it is simply an identifier.

Format of the Inter-Area Router LSA

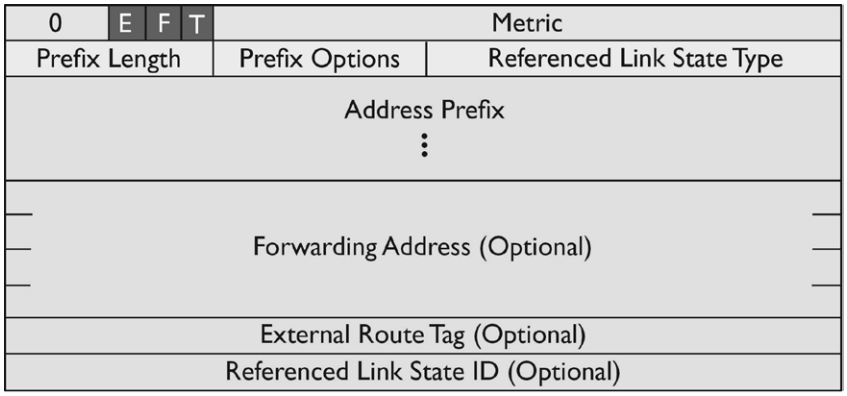
0	Options
0	Metric
Destination router ID	

ASExternal LSA

The ASExternal LSA is another in which the LS ID has no significance, and is simply an identifier. The OSPFv3 ASExternal LSA has some fields that are not present in the OSPFv2 ASExternal LSA.

- There are some extra flags.
- Prefix Length replaces Network Mask.
- Prefix Options are added.
- The concept of a reference LSA is added. This is an LSA that carries extra (usually non-OSPF-related) data across an OSPF autonomous system.

Structure of an ASExternal LSA



Here is a description of the fields:

Field	Description
E flag	As with OSPFv2, indicates whether this is a Type-1 or Type-2 External LSA. If the flag is set, then it is a Type-2 External LSA
F flag	This indicates whether there is a forwarding address in the LSA. The forwarding address field is something that also exists in the OSFPv2 ASExternal LSA, so the idea of a forwarding address is nothing new. The flag has been added as an extra convenience for routers processing the LSAs, so they can quickly know whether or not they need to process a forwarding address in the LSA.
T flag	This indicates whether there is an External Route Tag in the LSA. The External Route Tag is something that also exists in the OSFPv2 ASExternal LSA, so the idea of a External Route Tag is nothing new. The flag has been added as an extra convenience for routers processing the LSAs, so they can quickly know whether or not they need to process a External Route Tag in the LSA.
Metric	The external metric that the route was allocated when it was imported into OSPF.
PrefixLength, Prefix Options, Address Prefix.	The standard fields for representing a prefix in an OSPFv3 LSA. The NU bit in the prefix options for an ASExternal LSA should never be set. It is assumed that the ASExternal LSA is always advertising a route that is used for unicast forwarding.
Referenced LS Type, Referenced LS ID	The External Route Tag can tunnel a certain amount of data across an OSPF AS, to provide external routing protocol information about the route being carried in the ASExternal LSA. However, it may be necessary to carry more data than can be fitted into the External Route Tag. In this case, new LSA types may be defined in the future, that will carry larger amounts of data. When such LSAs are being used, the ASExternal LSA will use the Referenced LS Type and Referenced LS ID fields to point to the associated LSA that is carrying information about this route.
Forwarding address	This has the same meaning as in OSPFv2, i.e. it indicates the gateway address to the route in the ASExternal LSA, in the case that the route is via a non-OSPF router that is attached to the same link as the advertising ASBR. If there is an address in the Forwarding Address field, then the F flag on the LSA must be set.
External Route Tag	Used to carry some data along with the route. The data will not be significant to OSPF, but will be significant to an external routing protocol that the route will be redistributed to when it arrives at another ASBR. If the T bit is not set on the LSA, then the External Router Tag field will not be present.

Comparison of OSPFv3 and OSPFv2 LSAs

OSPFv3 LSAs		OSPFv2 LSAs	
LS Type	Name	Type	Name
0x2001	Router LSA	1	Router LSA
0x2002	Network LSA	2	Network LSA
0x2003	Inter-area Prefix LSA	3	Network Summary LSA
0x2004	Inter-area Router LSA	4	ASBR Summary LSA
0x2005	AS-External LSA	5	AS-External LSA
0x2006	Group Membership LSA	6	Group Membership LSA
0x2007	Type-7 LSA	7	NSSA External LSA
0x2008	Link LSA		No corresponding LSA
0x2009	Intra-Area Prefix LSA		No corresponding LSA

Options flags

The Options field in OSPFv3 LSAs is reasonably different to that in OSPFv2. Let's look at the details of the Options field in OSPFv3.

The Options field appears in the body of the OSPFv3:

- Hello packet
- Database Description packet
- Router LSA
- Network LSA
- Inter-Area Router LSA
- Link LSA

The purpose of the Options field is to enable routers to inform each other about which optional capabilities they support. That way, routers can be aware of the capabilities of their peers, and make decisions based on those peers' capabilities.

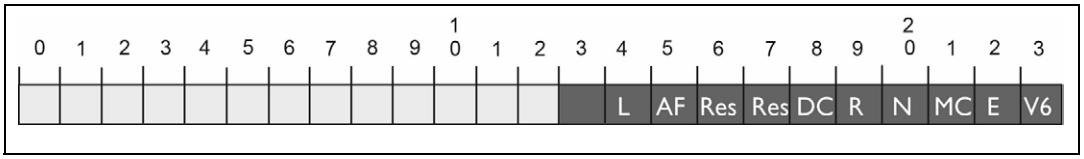
The sort of decision that routers make when they realize that another router's capabilities differ from its own are things like not forming a neighbor relationship with the other router, or not accepting certain LSAs.

There were originally 6 options defined for OSPFv3, but over time there have been others added. So, it is possible that one router will be advertising option bits that another router does not even recognize. If a router does receive packets in which unrecognized option bits are set, it should just ignore those options, and just continue on the basis of the option bits that it does recognize.

The option bits defined in OSPFv3 are:

Value	Description	Reference
0x000001	v6-bit	[RFC5340]
0x000002	E-bit	[RFC2328]
0x000004	Deprecated	[RFC5340]
0x000008	N-bit	[RFC3101]
0x000010	R-bit	[RFC5340]
0x000020	DC-bit	[RFC1793]
0x000040	Reserved for OSPFv2 migrated options	[RFC5340]
0x000080	Reserved for OSPFv2 migrated options	[RFC5340]
0x000100	AF-bit	[RFC5838]
0x000200	L-bit	[RFC5613]

The options field contains 24 bits, of which 10 are defined.



Let's go through each option in turn, and look at its meaning:

Option	Description
R	The R bit indicates that the originator of the packet is a router. So, routes that go via this device can be put into the route table. The most likely occasion when the R bit would be set to 0 would be when the originator is a host that has multiple IP interfaces, wants to learn which routes are accessible via each interface, but does not actually wish to route traffic between its interfaces. So, it participates in OSPF to learn routes, but is not actually a router itself.
v6	Indicates the originator of the packet is an IPv6 router. The v6 bit is a refinement of the R bit. There could be occasions where a device will route protocols other than IPv6, but will not route IPv6. So, such a device would have the R bit set to 1, but the v6 bit set to 0. In this case, any IPv6 routes that go via this device cannot be put into the IPv6 route table.
E	The E bit has the same meaning in OSPFv3 as it does in OSPFv2, i.e. that the originator is capable of accepting ASExternal LSAs. In other words, the interface via which the originator is sending this packet is not in a stub area.
MC	Another bit that is the same as OSPFv2. It used to indicate that the route in the LSA can be used as the Reverse Path to a multicast source. But this has now been deprecated, due to the abandoning of MOSPF
N	As with OSPFv2, this bit indicates that the interface originating the packet is in an NSSA area. This bit is only used in Hello packets. For two switches to become neighbors they must have the same value for N in their Hello packets – i.e. both have N=0 or both have N=1.
DC	Switches that have this bit set are Demand-Circuit capable. This has the same meaning as in OSPFv2.
AF	This bit indicates that the originator supports Address Families.
L	This bit is set in Hello and DD packets. It indicates that the packet contains a field called the Link-Local Signaling (LLS) Data Block. This is a special data block that carries extra information between neighbors.

Configuring OSPFv3 in AlliedWare Plus

The configuration of OSFPv3 in AlliedWare Plus is somewhat different to the configuration of OSPFv2.

The main change is the removal of the **network** command that exists in router OSPF mode for OSPFv2.

The removal of this command has more effect than just one less command; it actually changes the paradigm for determining which interfaces are involved in OSPF.

In OSPFv2, interfaces are not explicitly configured as OSPF interfaces. Rather, they are implicitly configured if they happen to have an IP address that falls within the range that is covered by the network added in an OSPF **network** command.

But, in OSPFv3, a more intuitive approach is taken. Rather than configuring a network in **Router IPv6 OSPF** mode, and having that pull some interfaces into OSPF, instead the interfaces are configured as OSPFv3 interfaces, and that pulls their connected networks into OSPFv3.

To configure OSPFv3 on an interface, first enter interface configuration mode for that interface, then enter the command:

```
ipv6 router ospf area <area-id> [tag <process-id>][instance <inst-id>]
```

The area referred to by <*area-id*> does not need to have been pre-defined. Simply specifying an <**area-id**> in an `ipv6 router OSPF` command makes the router aware of that area. By default, the router assumes an area to be a transit (non-stub) area. If you want a particular area to be a stub area and/or to configure summary routes for the area and/or you want the cost for summary routes to be something other than 1, then you can configure those properties of the area in `router IPv6 OSPF` mode.

The <*process-id*> in this command is used in the case that there are multiple OSPF processes running on the router. The <*process-id*> needs to be specified on an `ipv6 router OSPF` command.

The <*instance-id*> on the command specifies an OSPFv3 instance. The value of the <**instance-id**> will be put into the Instance ID field of the headers of the OSPFv3 packets sent from that interface.

Apart from the difference in the way that an interface is brought into OSPF, the configuration of OSPFv3 is very similar to that of OSPFv2.

There are a number of commands that define the properties of an OSPF interface, and are all configured in Interface Mode for the interface in question:

- IPv6 OSPF Cost
- IPv6 OSPF dead-interval
- IPv6 OSPF hello-interval
- IPv6 OSPF priority
- IPv6 OSPF retransmit-interval
- IPv6 OSPF transmit-delay

These commands all have the same effect as their OSPFv2 counterparts.

And, there are a set of commands that are used in `router IPv6 OSPF <process-ID>` mode:

- Router-ID
- Redistribute
- Passive-interface
- Default-metric
- Auto-cost Reference-Bandwidth
- Area virtual-link
- Area stub
- Area default-cost
- Area range

Again, these commands have the same effect as their OSPFv2 counterparts.

Instances and Processes

OSPFv3 in AlliedWare Plus has two different ways of keeping sets of OSPFv3 operations separate from each other.

To quite an extent, the two methods are aimed at the same goal – namely to have multiple OSPF networks in operation on the same physical network, but not communicating with each other. The fact that there are two methods is mostly historical.

OSPFv2 on AlliedWare Plus has the concept of OSPF **Processes**. But, this is purely a vendor-specific feature; it is not something that is defined in the OSPFv2 RFCs.

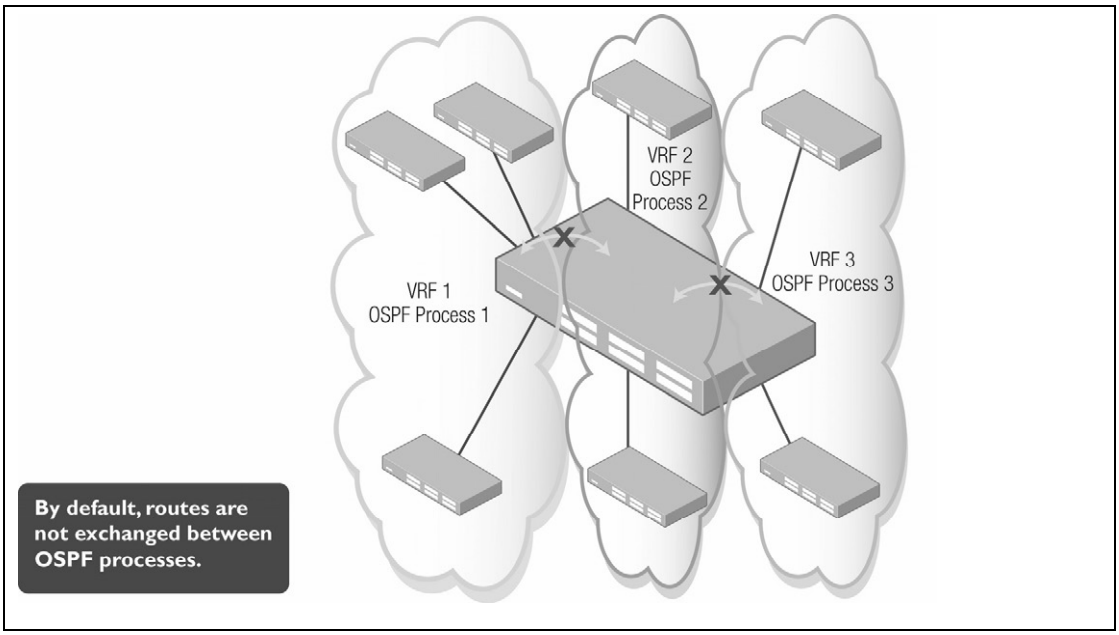
Then when OSPFv3 came along, the OSPFv3 RFC defined the idea of **Instances**.

AlliedWare Plus has brought the processes concept through from OSPFv2 to OSPFv3; and has added the instances concept, as it is part of the RFC.

So, what are the differences between Processes and Instances?

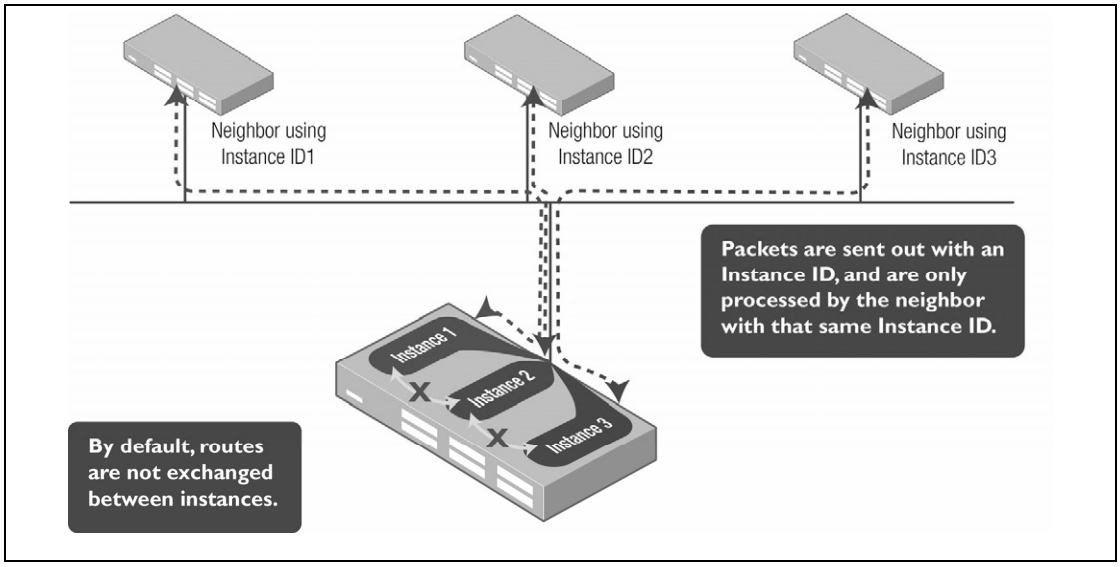
Processes are separate instantiations of OSPF within the same switch. That is, they are whole complete OSPF environments running independently in the same router. They can have different Router IDs, different default reference bandwidths. They can have different sets of routes being redistributed into them. They manage separate OSPF route tables. A given OSPF interface cannot belong to more than one process. The process to which an interface belongs is one of the parameters that is specified when it is configured as an OSPF interface.

Typically, OSPF processes are used for VRF – different OSPF processes are used within different VRF instances. There is no external visibility of OSPF processes. No identifier is put into packets to indicate which OSPF process a packet belongs to. However, in the typical use-case, OSPF processes will be used to create separate OSPF networks running in the same physical network. In fact, the typical use-case will be an OSPF process per VRF instance. The interfaces that belong to OSPF process X will be the interfaces that belong to VRF instance X.



Instances are, again, separate OSPF networks running in the same physical network. However, there is no specific separation of instances within a router. While a given interface can belong to only one instance, if multiple interfaces belong to the same process there is nothing to stop them all belonging to different instances.

There is external visibility of instances in OSPFv3. The instance to which an OSPF interface belongs is written into the Instance ID field of the packets the interface originates. So, if multiple devices have interfaces all attached to a common link, then the interfaces that belong to common instances will find each other, and they will not talk to interfaces that belong to other instances.



Configuring OSPFv3 Authentication

Once you configure OSPFv3 and decide on using authentication or encryption, you need to define a security policy on each OSPFv3 interface of each device in the network.

A security policy consists of:

- an SPI (Security Parameter Index), which is simply an ID number for the policy
- an authentication algorithm (if the policy is to perform authentication)
- an authentication key (if the policy is to perform authentication)
- an encryption algorithm (if the policy is to perform encryption)
- an encryption key (if the policy is to perform encryption)

The available authentication algorithms are MD5 and SHA.

The available encryption algorithms are AES and 3DES.

The command to configure OSPFv3 authentication on an interface is

```
ipv6 ospf authentication ipsec spi <256-4294967295> {md5 <MD5-key>|sha1
<SHA1-key>}
```

The command to configure OSPFv3 encryption on an interface is:

```
ipv6 ospf encryption ipsec spi <256-4294967295> esp {aes-cbc <AES-CBC-
key>|3des <3DES-key>|null}
```

Both OSPFv3 encryption and authentication can be configured together on an interface with the command:

```
ipv6 ospf encryption ipsec spi <256-4294967295> esp {aes-cbc <AES-CBC
key>|3des <3DES-key>|null}> {md5 <MD5-key>|sha1 <SHA1-key>}
```

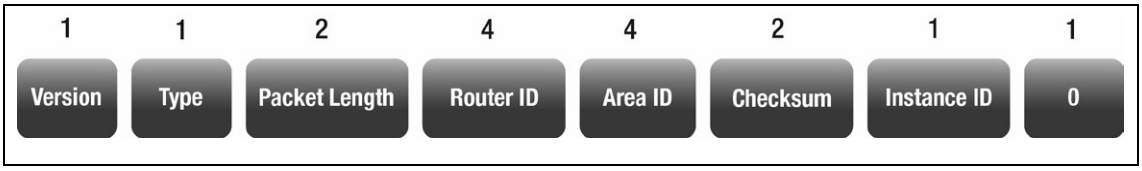
If routers connect to the same link, and they are performing authentication and/or encryption then they must have the same security policy (same SPI, same algorithms, and same keys) in order to become neighbors.

It is actually possible to configure authentication and/or encryption on an area instead of just a single interface within the area. If authentication and/or encryption is configured on an area AND is also configured on an interface within the area, then the configuration on the interface takes precedence, for that interface.

OSPFv3 Packet Formats

OSPFv3 Header

An OSPFv3 Header consists of the following fields:



Version Number – For OSPFv3, the version is, of course, 3.

Type – The possible values for the packet Type are:

Type Value	Name
1	**Hello**
2	**Database Description**
3	**Link State Request**
4	**Link State Update**
5	**Link State Acknowledgement**

Packet Length - this is the total number of bytes in the OSPF portion of the packet, including the OSPF header, but excluding IPv6 header(s) and Layer 2 protocol header(s).

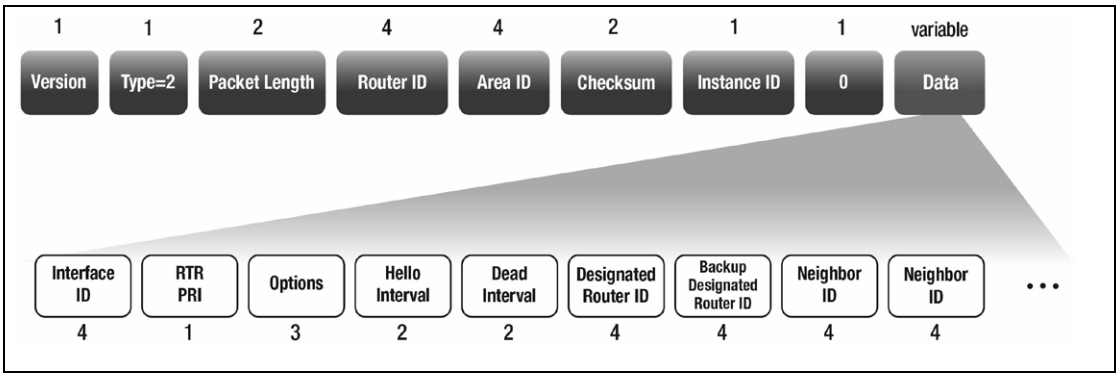
Router ID - the Router ID of the packet's originator.

Area ID – the ID of the area that the packet is being transmitted into. If a packet is being sent into a virtual link, it is deemed to belong to Area 0.

Checksum – the algorithm for calculating the checksum is described in section A.3.1 of RFC2740.

Instance ID – the concept of OSPFv3 instances is described above in the section Instances and Processes on page 288. The value in the Instance ID field identifies which Instance a packet belongs to.

Hello packet



Interface ID – identifies the egress interface via which the originating router transmitted the packet. The ID is a router-specific value. It is customary to use the ifIndex as defined in the Interface MIB. But, each implementer is free to choose whatever Interface ID scheme they like. Interface IDs are described above in the section section Interface ID on page 274.

Rtr Priority – this is the router's priority for election as the DR (or backup DR) for the subnet this packet is being transmitted into.

Options – a set of flags that indicate the originating router's capabilities. These are described in detail in the section Options flags on page 284.

HelloInterval – the time (in seconds) between the sending of Hello packets.

RouterDeadInterval – the number of seconds that the router will wait to receive a Hello packet from a neighbor before deciding that neighbor is dead.

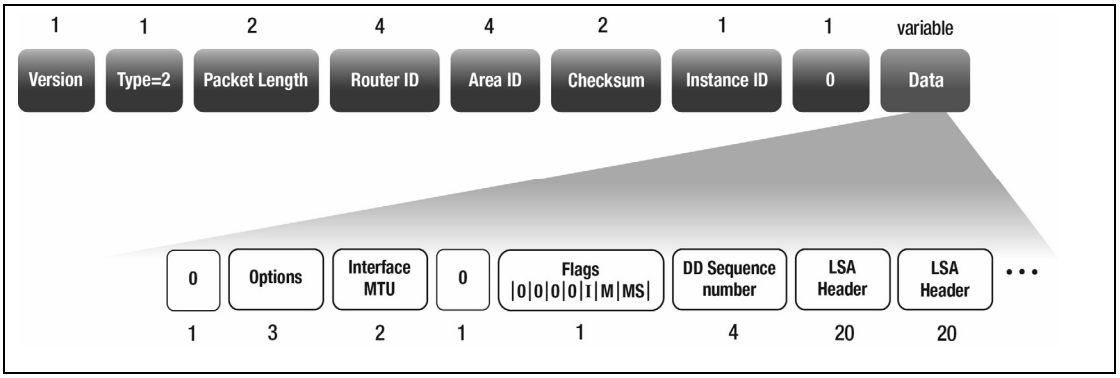
Designated Router ID – the Router ID of the router that the originator of this packet believes to be the DR on the current subnet.

Backup Designated Router ID – the Router ID of the router that the originator of this packet believes to be the BDR on the current subnet.

Neighbor ID – a list of the Router IDs of neighbors that the originating router has received compatible Hello packets from within the last RouterDeadInterval seconds.

The Database Description packet

Database Description packets are OSPF packet type 2. These packets are exchanged when an adjacency is being initialized. They describe the contents of the link-state database. Multiple packets may be used to describe the database. For this purpose a poll-response procedure is used. One of the routers is designated to be the master, the other the slave. The master sends Database Description packets (polls) which are acknowledged by Database Description packets sent by the slave (responses). The responses are linked to the polls via the packets' DD sequence numbers.



The format of the Database Description packet is very similar to both the Link State Request and Link State Acknowledgment packets. The main part of all three is a list of items, each item describing a piece of the link-state database.

Options

The optional capabilities supported by the router, as described in Options flags on page 284.

Interface MTU

The size in bytes of the largest IPv6 datagram that can be sent out the associated interface, without fragmentation. Interface MTU should be set to 0 in Database Description packets sent over virtual links.

I-bit

The Init bit. When set to 1, this packet is the first in the sequence of Database Description Packets.

M-bit

The More bit. When set to 1, it indicates that more Database Description Packets are to follow.

MS-bit

The Master/Slave bit. When set to 1, it indicates that the router is the master during the Database Exchange process. Otherwise, the router is the slave.

DD sequence number

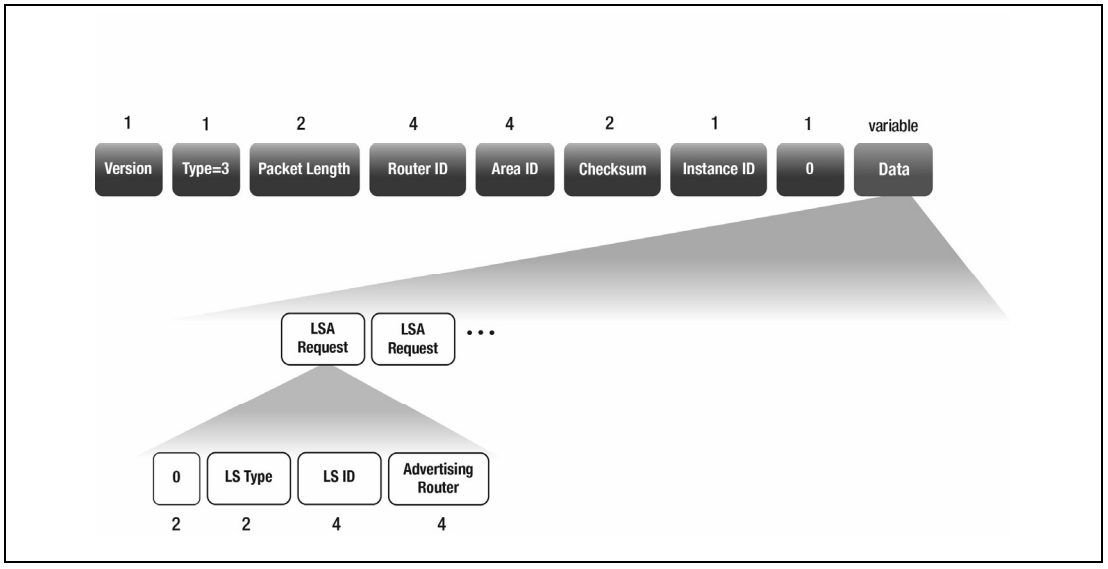
This is used to sequence the collection of Database Description Packets. The initial value (indicated by the Init bit being set) should be unique. The DD sequence number then increments until the complete database description has been sent.

The rest of the packet consists of a (possibly partial) list of the link-state database's pieces. Each LSA in the database is described by its LSA header.

The Link State Request packet

Link State Request packets are OSPF packet type 3. After exchanging Database Description packets with a neighboring router, a router may find that parts of its link-state database are out-of-date. The Link State Request packet is used to request the pieces of the neighbor's database that are more up-to-date. Multiple Link State Request packets may need to be used.

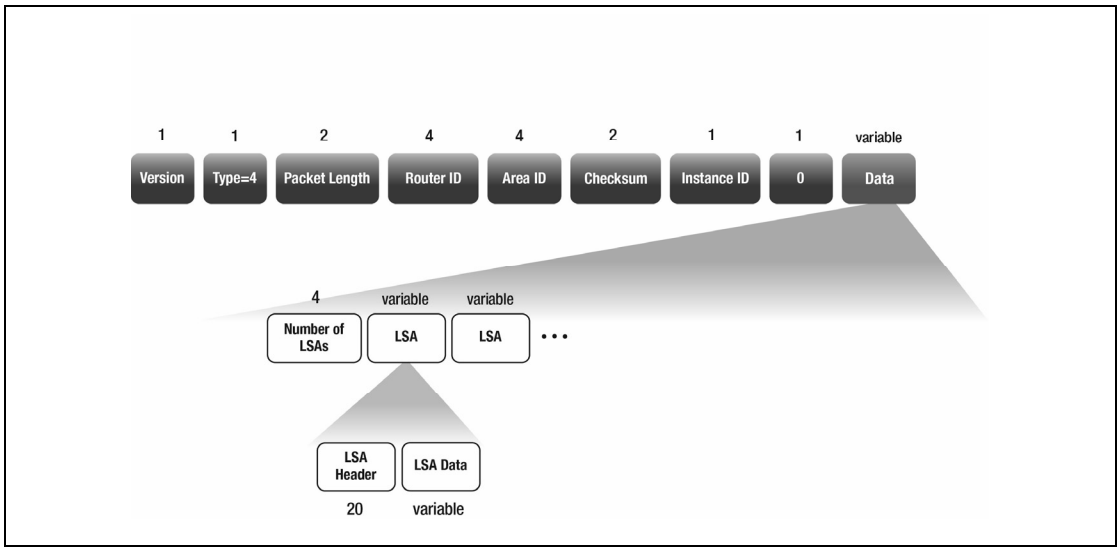
A router that sends a Link State Request packet has in mind the precise instance of the database pieces it is requesting.



Each LSA requested is specified by its LS type, Link State ID, and Advertising Router. This uniquely identifies the LSA, but not its instance. Link State Request packets are understood to be a request for the most recent instance (whatever that might be).

The Link State Update packet

Link State Update packets are OSPF packet type 4. These packets implement the flooding of LSAs. Each Link State Update packet carries a collection of LSAs one hop further from their origin. Several LSAs may be included in a single packet. Link State Update packets are multicast on those physical networks that support multicast/broadcast. In order to make the flooding procedure reliable, flooded LSAs are acknowledged in Link State Acknowledgment packets. If retransmission of certain LSAs is necessary, the retransmitted LSAs are always carried by unicast Link State Update packets.

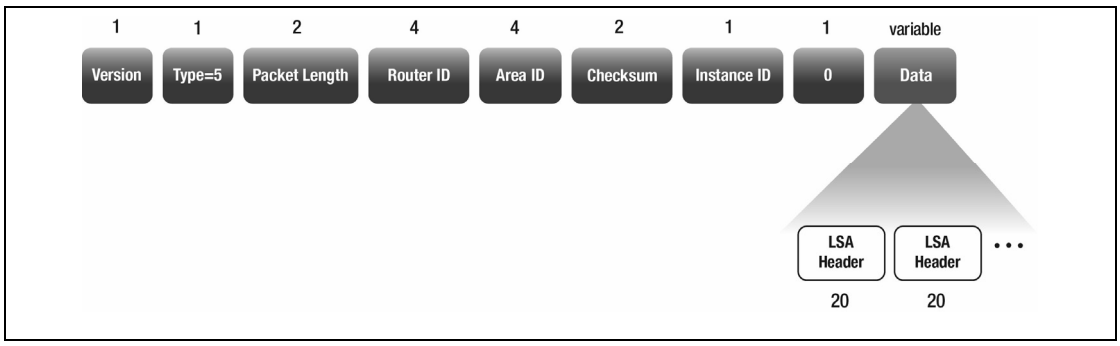


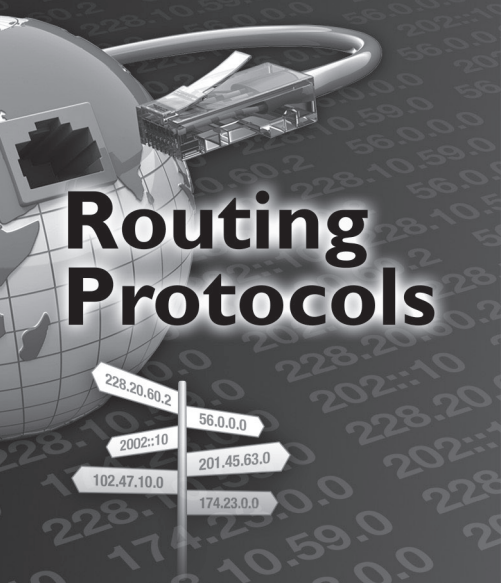
The body of the Link State Update packet consists of a list of LSAs. Each LSA begins with a common 20 byte header, described in Section A.4.2. Detailed formats of the different types of LSAs are described in Section A.4. http://www.freesoft.org/CIE/RFC/1583/109.htm

The Link State Acknowledgment packet

Link State Acknowledgment Packets are OSPF packet type 5.

To make the flooding of LSAs reliable, flooded LSAs are explicitly acknowledged. This acknowledgment is accomplished through the sending and receiving of Link State Acknowledgment packets. The format of this packet is similar to that of the Database Description packet. The body of both packets is simply a list of LSA headers.





Routing Protocols

This chapter covers the following topics:

- Introduction
- BGP4+ Routing Attributes
- New NLRI attributes
- The Multiprotocol Extension capability
- Address Families

CHAPTER 8

Differences between BGP4 and BGP4+

Introduction

This chapter describes the differences between BGP version 4 (BGP4) and BGP version 4+ (BGP4+).

BGP4 was originally developed for IPv4 only, when IPv6 was still in its infancy.

BGP4+ is a development of BGP4 that has added support for IPv6 and other traffic types, like IPv4 multicast and MPLS VPN labels. These extensions are known as the Multiprotocol Extensions for BGP4, defined in RFC 2858. The original messaging and routing mechanism of BGP4 has not been changed.

Because of the way that BGP4+ has been designed, using extensions to the existing BGP4 protocol, a BGP4 Speaker can interoperate with a BGP4+ Speaker.

BGP4+ Routing Attributes

There are three pieces of information carried by BGP4 that are IPv4 specific:

1. The NEXT_HOP attribute
2. The AGGREGATOR attribute
3. NLRI (Network Layer Reachable Information)

To support IPv6, the BGP4**+** protocol has to carry the following IPv6 information:

- NLRI
- Next Hop

How can this be achieved while still maintaining backward compatibility with BGP4?

The problem is that the BGP4 Update packet has been defined to have just one set of reachable routes, and one set of unreachable routes; and the routes in each of these sets are IPv4 routes. A BGP4 Speaker will always interpret the information in the *Withdrawn Routes* and *NLRI* sections of the Update packet as sets of IPv4 routes.

There is no provision in the protocol for having other sections in the packet that represent the *IPv6 Withdrawn Routes* or the *IPv6 NLRI*.

If, for example, an *IPv6 Withdrawn Routes* section was defined, and placed into the packet directly after the original *Withdrawn Routes* section, then a BGP4 Speaker would attempt to interpret this data as a set of path attributes, as that is exactly what a BGP4 speaker expects to find directly after the *Withdrawn Routes* section of the Update packet.

It was decided that the best way to introduce IPv6 routing information into BGP packets was to introduce special **attributes** that carry the IPv6 routes. This may seem strange at first – how can routes be attributes? Surely attributes are properties of routes, not the routes themselves?

Yes, it is a little odd, but it is actually the best solution.

- BGP4 Speakers are already well used to dealing with attributes that they do not recognize, so if new attributes appear that are carrying IPv6 routes, this will not cause a BGP4 speaker to terminate the BGP session.
- There is no technical reason why route data cannot be contained in an attribute – it is as good a type of data structure as any within which to hold route data.
- It does not matter that the route information is all contained within the attributes, and that there will not be an NLRI section appearing after the attributes; it is OK for a BGP Update packet to have a zero-length NLRI section.

New NLRI attributes

BGP4+ has introduced two NLRI attributes:

1. **MP_REACH_NLRI** (Multiprotocol Reachable NLRI) which is used to carry the set of reachable destinations, together with the next hop information to be used for forwarding to these destinations.
2. **MP_UNREACH_NLRI** (Multiprotocol Unreachable NLRI) which is used to carry the set of unreachable destinations, i.e. routes that are being withdrawn.

Note - In BGP4+, the next-hop attribute is expressed by an IPv6 address, which can be an IPv6 global unicast address or a link-local address.

You may be wondering why we do not also need an IPv6 version of the Aggregator attribute. This is because the IPv4 address used in this attribute is the BGP Identifier (Router ID), a 4 octet integer which is still used in BGP4+.

Both of these new BGP4+ attributes are optional and non-transitive, thereby allowing backward compatibility. This way a BGP4 speaker (which doesn't support the multiprotocol capabilities) will just ignore the information carried in these attributes and will not pass it to other BGP speakers, allowing it to still interoperate with a BGP4+ speaker.

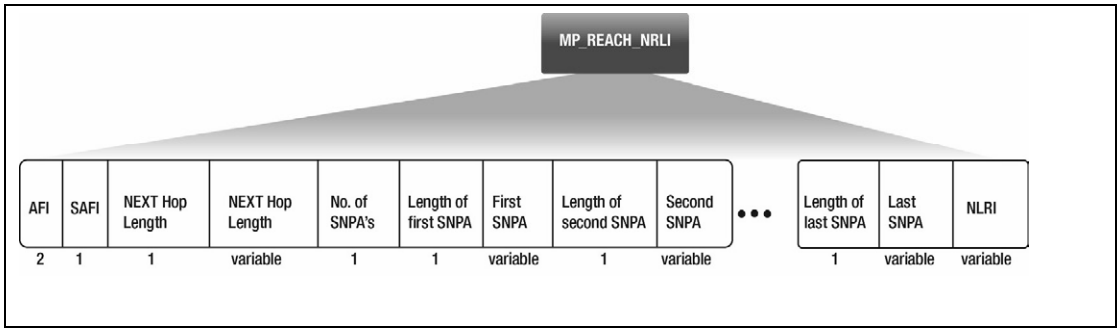
The new BGP4+ NLRI attributes in more detail

MP_REACH_NLRI (Type Code 14):

The three main pieces of information that this attribute carries are the:

- Actual reachable routes that the router is advertising.
- Network Layer address of the device that is the next hop on the path to the advertised routes.
- Subnetwork Points of Attachment (SNPAs) associated with the next hop (i.e. the Layer 2 address(es) of the next hop).

Here is a diagram of the MP_REACH_NRLI attribute, with the number of octets in each field shown:



AFI - Address Family Identifier

This identifies the protocol or service that the information in this attribute pertains to. For example: IPv4, IPv6, AppleTalk, Banyan Vines, EIGRP, etc.

For the full list is available at:
http://www.iana.org/assignments/address-family-numbers/address-family-numbers.xhtml

SAFI - Subsequent Address Family Identifier.

This identifies a subcategory of the protocol or service identified by the AFI. For example, if the attribute is carrying information for IPv6 mulitcast then the AFI would identify IPv6, and the SAFI would identify multicast.

For the full list is available at:
http://www.iana.org/assignments/safi-namespace/safi-namespace.xhtml

Example values are: IPv6 unicast routing, multicast, both unicast and multicast.

SNP - Subnetwork Points of Attachment. The Layer 2 address of a next hop device.

NLRI - Network Layer Reachability Information. The routes being advertised.

Border Gateway Protocol Update Message with the MP_REACH_NLRI attribute:

Here we have part of a BGP4+ Update message, showing the Path Attributes, which include an MP_REACH_NRLI attribute:

```
Path attributes
        ORIGIN: INCOMPLETE (4 bytes)
            Flags: 0x40 (Well-known, Transitive, Complete)
                0... .... = Well-known
                .1.. .... = Transitive
                ..0. .... = Complete
                ...0 .... = Regular length
            Type code: ORIGIN (1)
            Length: 1 byte
            Origin: INCOMPLETE (2)
        AS_PATH: 65001 (9 bytes)
            Flags: 0x40 (Well-known, Transitive, Complete)
                0... .... = Well-known
                .1.. .... = Transitive
                ..0. .... = Complete
                ...0 .... = Regular length
            Type code: AS_PATH (2)
            Length: 6 bytes
            AS path: 65001
                AS path segment: 65001
                    Path segment type: AS_SEQUENCE (2)
                    Path segment length: 1 AS
                    Path segment value: 65001

                MP_REACH_NLRI (59 bytes)
            Flags: 0x90 (Optional, Non-transitive, Complete, Extended Length)
                1... .... = Optional
                .1.. .... = Non-transitive
                ..0. .... = Complete
                ...1 .... = Extended length
            Type code: MP_REACH_NLRI (14)
            Length: 55 bytes
            Address family: IPv6 (2)
            Subsequent address family identifier: Unicast (1)
            Next hop network address (32 bytes)
                Next hop: 2005:5555::1 (16)
                Next hop: fe80::200:cdff:fe29:90c7 (16)
            Subnetwork points of attachment: 0
            Network layer reachability information (18 bytes)
                2002:2222::/64
                    MP Reach NLRI prefix length: 64
                    MP Reach NLRI prefix: 2002:2222::
                2005:5555::/64
                    MP Reach NLRI prefix length: 64
                    MP Reach NLRI prefix: 2005:5555::
```

In the output above, firstly we have the 'normal' Path attributes (Origin and AS_Path).

In the case that the Path Attributes section of an Update packet contains both 'normal' attributes and an MP_REACH_NRLI attribute, then the 'normal' attributes are applied to the routes in the MP_REACH_NRLI attribute. In this case we have the Origin and AS_Path attributes preceding the MP_REACH_NRLI attribute.

- The Flags show us that the MP_REACH_NRLI attribute is Optional and Non-transitive.
- The Address Family is IPv6 (AFI=2) and the Subsequent Address Family Identifier shows us that it is carrying routes that are to be used for unicast data forwarding (SAFI=1).
- There are two next hop addresses – these are the unicast IPv6 address of the next hop, as well as its link-local address. In this case the originator of the Update packet is advertising itself as the next hop.
- The Sub Network Points of Attachment field is set to 0 for IPv6, which means the SNPA field is omitted.
- Lastly, we can see that the switch is advertising routes 2002:2222: :64 and 2005:5555: :64.

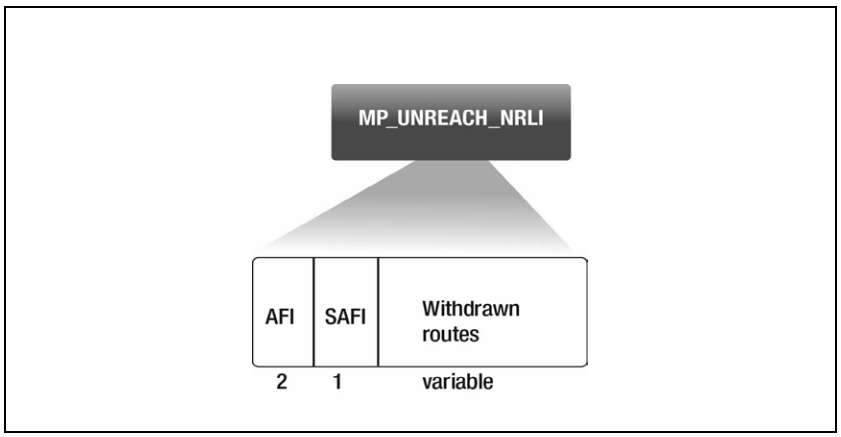
Multiple IPv6 routes are contained in a single MP_REACH_NRLI attribute in the Network Layer Reachability Information field.

Multiprotocol Unreachable NRLI - MP_UNREACH_NLRI (Type Code 15) attribute

Just as the MP_REACH_NRLI transports reachable routes, there is a companion attribute, the MP_UNREACH_NLRI, that carries the list of routes that are being withdrawn.

Again, this is an optional, non-transitive attribute.

Here is a diagram of the MP_UNREAC_NRLI attribute with the number of octets in each field shown.



Border Gateway Protocol UPDATE Message with the MP_UNREACH_NLRI attribute

Here we have part of a BGP4+ Update message, showing the MP_UNREACH_NRLI section and its contents:

```
MP_UNREACH_NLRI (16 bytes)
Flags: 0x90 (Optional, Non-transitive, Complete, Extended Length)
1... .... = Optional
.1.. .... = Non-transitive
..0. .... = Complete
...1 .... = Extended length
Type code: MP_UNREACH_NLRI (15)
Length: 12 bytes
Address family: IPv6 (2)
Subsequent address family identifier: Unicast (1)
Withdrawn routes (9 bytes)
2002:2222::/64
MP Unreach NLRI prefix length: 64
MP Unreach NLRI prefix: 2002:2222::
```

The route 2002:2222::/64 has been withdrawn. In this case, there is only one route in the MP_UNREACH_NLRI attribute, but the attribute can contain multiple routes.

The Multiprotocol Extensions capability

When the BGP4+ session is started with its peer, the OPEN message must indicate that it supports multiprotocol extensions by listing the capabilities of the BGP speaker.

Here we can see that this BGP device is advertising its capabilities for both IPv4 and IPv6:

```
Border Gateway Protocol - OPEN Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 61
    Type: OPEN Message (1)
    Version: 4
    My AS: 65002
    Hold Time: 90
    BGP Identifier: 1.1.1.1 (1.1.1.1)
    Optional Parameters Length: 32
    Optional Parameters
        Optional Parameter: Capability
            Parameter Type: Capability (2)
            Parameter Length: 6
            Capability: Multiprotocol extensions capability
                Type: Multiprotocol extensions capability (1)
                Length: 4
                AFI: IPv4 (1)
                Reserved: 00
                SAFI: Unicast (1)
        Optional Parameter: Capability
            Parameter Type: Capability (2)
            Parameter Length: 6
```

```
Capability: Multiprotocol extensions capability
    Type: Multiprotocol extensions capability (1)
    Length: 4
    AFI: IPv6 (2)
    Reserved: 00
    SAFI: Unicast (1)
```

In the packet decode above, we can see that:

The Optional Parameter **Capability** is set, which was introduced in RFC 5492 for BGP4.

This is used to allow BGP4 peers to negotiate a set of mutually supported capabilities, rather than just terminating the connection if both peers had different Optional Parameters (as in the base BGP4 specification). Now, in BGP4+, we have a new Capability – **Multiprotocol Extensions**.

We have two **Optional Parameter** Capabilities in the output above - one for Address Family IPv4 and another for Address Family IPv6. Although IPv4 can be configured in the base BGP4+ configuration section, rather than in a separate IPv4 Address-Family, it is implemented in BGP4+ as a separate Address Family and will have its own separate section in a BGP4+ Open message.

The **Capability Type** code is set to 1 - multiprotocol extensions

The **Capability AFI** (Address Family Identifier) codes are set to 1 (for IPv4) and 2 (IPv6)

Address Families

As we know, when BGP4 was originally developed, only IPv4 was considered, as IPv6 was not yet well established. Therefore, all configurations were performed in the main BGP configuration section.

As BGP4+ can support multiple protocols, the concept of address families was implemented to allow separate configuration of each protocol under a separate BGP configuration subsection.

Which address families are supported will vary from vendor to vendor, but can include IPv4, IPv6, Multicast and VPN. The address families configured on a device are identified in the AFI (Address Family Identifier) codes within the Multiprotocol Extension Capability section of the BGP4+ Open message.

Parameters that are not specific to a particular address family are configured directly under the global BGP configuration section. These include:

- Defining a peer
- Defining a peer group
- Setting **ebgp multihop** to define the number of hops to reach the external peer
- Timers

Commands which enable BGP functions (for the whole BGP protocol) are also configured under the global BGP section:

- BGP Router ID
- BGP bestpath (assortment of best path algorithm changing commands)
- BGP graceful restart (functionality - not capability advertisement)

Commands which setup or advertise, like capabilities, policies, redistribution, networks, attributes, are configured under the address family:

- Next-hop-self
- Applying route maps, prefix lists, filter lists, distribution lists
- Route servers
- Route reflectors
- Maximum prefixes
- Capability advertisement (graceful restart)
- Assigning a peer to a peer group.
- Redistribution of static and/or connected routes
- Networks
- Aggregate-address

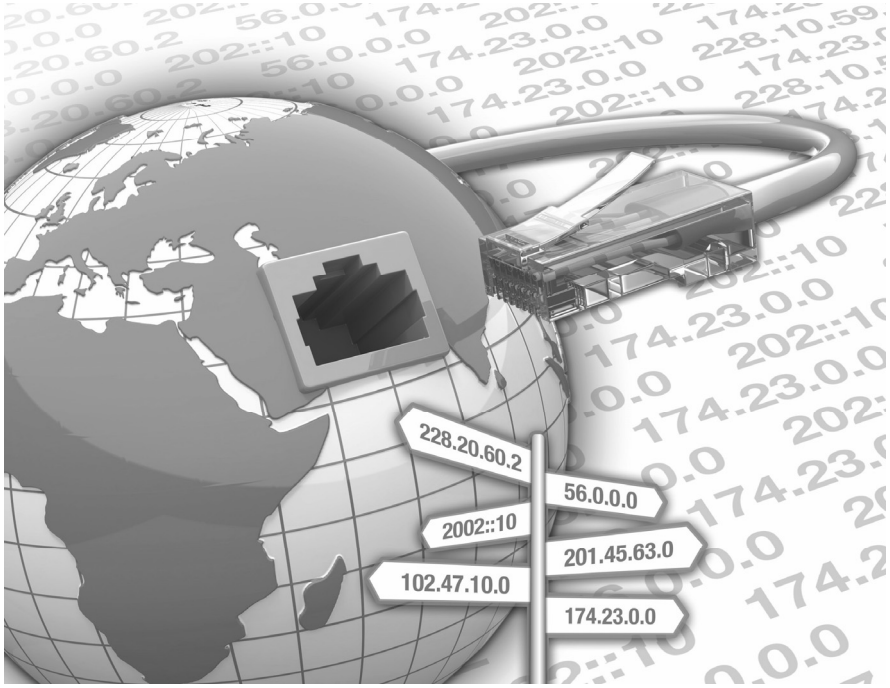
Here is an example (and description) of a BGP configuration with an IPv6 address-family:

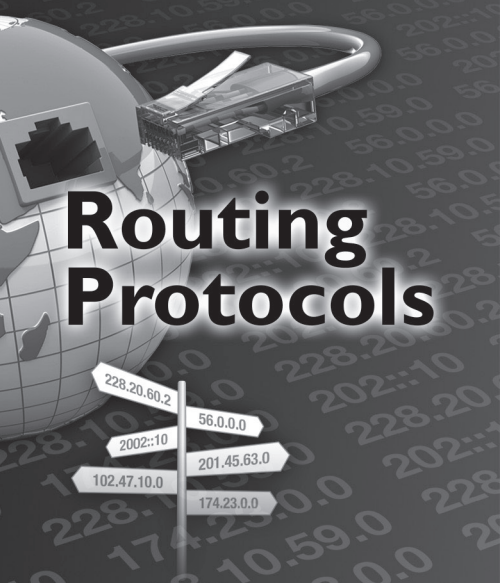
`router bgp 65001`	The Local autonomous system Number.
`bgp router-id 1.1.1.1`	The BGP Identifier.
`network 192.168.1.0/24`	An IPv4 network/mask which will be advertised.
`neighbor 192.168.1.2 remote-as 65002`	The IPv4 BGP peer and its autonomous system Number.
`neighbor 2222::2 remote-as 65002`	The IPv6 BGP peer and its autonomous system Number.
`!`	
`address-family ipv6`	The IPv6 address-family where we configure IPv6 specific options.
`neighbor 2222::2 activate`	Enables the exchange of information with a BGP peer.
`network 2222::/64`	The network/prefix that will be advertised.
`redistribute connected`	Connected routes will be redistributed into BGP.
`redistribute static`	Static routes will be redistributed into BGP.
`exit-address-family`	

There are actually no new commands in BGP4+ - the same commands are used as in BGP4, but with multiprotocol options rather than just IPv4. The behaviour of BGP4+ is also the same as with BGP4.

Section 3

Multicast Routing Protocols





Routing Protocols

This chapter covers the following topics:

- Introduction
- What exactly do the routers and switches in the network need to do?
- Operation Differences
- Terminology

CHAPTER 9

Layer 2 and Layer 3 Multicasting

Introduction

Some forms of communication essentially operate on a one-to-one basis, and some operate on a one-to-many basis. The characteristics of these forms of communication are quite different.

For example, a telephone call is a one-to-one form of communication. The caller dials the specific phone number of the recipient. The recipient answers the call, and the two parties communicate directly with each other.

On the other hand, a radio station operates on a one-to-many basis. The radio station simply sends out a signal on a particular frequency, and people can tune their radios into that frequency to listen to that station. The radio station's signal does not specify any particular individual radios as the recipients of the signal; it just has a published frequency that listeners can tune into if they wish to. What is more, the communication is one-way; the listeners' radios do not send a signal back to the radio station, they just passively receive the station's signal. What is more, the radio station does not have to alter the strength of its signal, or send out multiple copies of the signal, as more radios tune in. The signal sent out by the radio station is exactly the same whether zero radios are tuned into it, or 100,000 radios are tuned into it.

IP data networking has modes of operation that are analogous to the telephone call and to the Radio Station.

The telephone-like one-to-one form of communication is referred to as **unicast** communication. The Radio-like one-to-many form of communication is referred to as **multicast** communication.

In unicast IP communication, one device sends packets to the specific IP address of another device, very much like dialing someone's phone number, and the other device (usually) sends packets back in reply.

In multicast IP communication, a device sends packets to an IP address that resides within a range that is reserved for multicasting (224.0.0.0/4), and other devices can decide to receive these packets or not.

This chapter presents a general overview of how IP multicasting works, in both Layer 2 and Layer 3 networks.

Structure of a multicast network

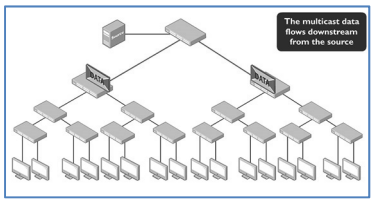
In essence, a multicast network consists of:

- One or more sources of multicast data. These sources just send their multicast data, quite oblivious as to whether or not anybody is listening for it.
- One or more receivers of the multicast data.
- Routers and switches in between, that forward multicast data to the receivers.

Because the flow of the multicast is unidirectional, it is possible to use the analogy of the flow of a river. The individual channels of multicast data are frequently referred to as 'streams'. The direction of flow is 'downstream', so the multicast sources are said to be 'upstream', and the multicast receivers are 'downstream'.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/z76R4pY0nfY



What exactly do the Routers and Switches in the Network Need to do?

To understand the task required of the routers and switches, you first need to understand that a **key** reason for using multicasting at all (as against always just using unicast) is to achieve **efficient use of bandwidth**. The great strength of multicasting is that a multicast source needs send out only one copy of a given multicast stream, but that stream can be delivered to multiple receivers. The duplication of the stream, for delivery to multiple receivers, is done as late in the packets' journey as possible.

Multicast aims to deliver data to multiple receivers with as little packet duplication as possible. With unicast, there needs to be a separate stream for each receiver, all the way from source to receiver, which uses considerably more bandwidth, and puts greater load on the source device.

Hence, the main role of the routers and switches in a multicast network is to decide exactly where the downstream receivers are for any given multicast stream, and forward the stream towards those receivers, without unnecessarily forwarding the stream anywhere else.

Overall, this is not a highly complicated process, but there is always plenty of devil in the detail. The sort of factors that add complication to multicast routing are:

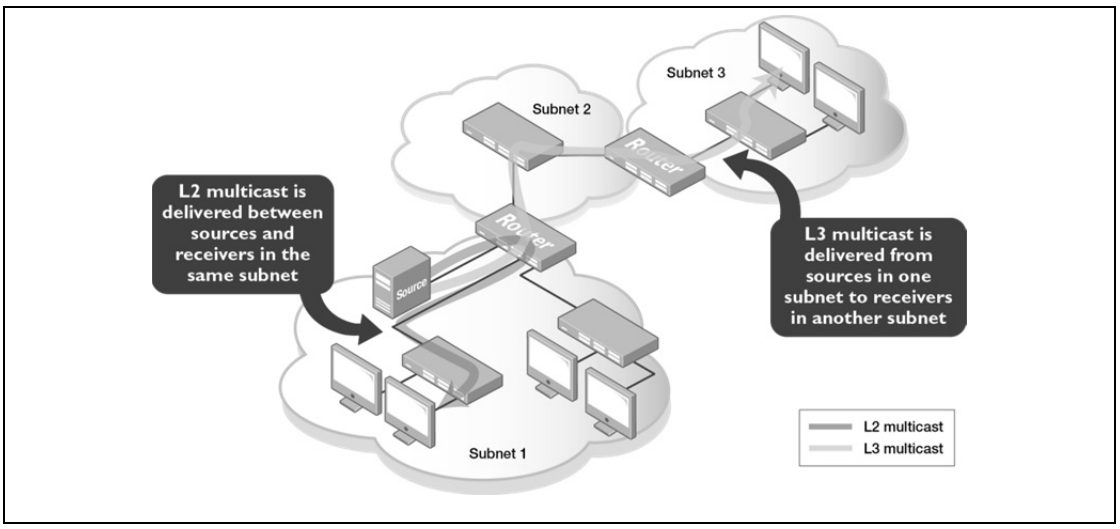
- Handling looped topologies—if there are multiple paths from a given source to a given receiver, which path should be chosen?
- Reacting correctly to changes in the network topology—if a stream that had been arriving on interface X now suddenly starts arriving on interface Y, should the switch still continue forwarding the stream? Or is there now a better path on which the stream is now being forwarded?
- Reacting quickly to the arrival and departure of receivers—when a receiver requests a given stream, the switches on the path back to the source need to react quickly, and start forwarding the stream to this receiver. Similarly, when a receiver signals that it no longer wants to receive a stream, the switches need to react quickly to stop unnecessarily forwarding the stream (bearing in mind that a goal of multicasting is to avoid unnecessary usage of bandwidth).
- Deciding if a receiver has quietly departed—if a receiver has not explicitly signaled that it no longer wishes to receive a stream, but also has not recently signaled that it wishes to continue receiving the stream, at what point does a switch decide to no longer forward that stream to that receiver?
- Keeping track of the fact that a receiver has requested a stream that is not currently arriving from any source, so that the stream will be forwarded to that receiver if it does start being transmitted by some source.

In this chapter and the following chapters, we will look at how to solve all these problems, and more.

Operation Differences

The operation of multicast within a Layer 2 domain is controlled by the IGMP protocol (or, the MLD protocol, in the case of IPv6). The Layer 3 forwarding of multicast is controlled by a Layer 3 multicast protocol like PIM. So, there are different protocols used for Layer 2 and for Layer 3 IP multicasting. This begs the question – *"What is the difference between Layer 2 and Layer 3 multicast forwarding?"*

In essence, the difference is that Layer 2 multicast forwarding covers the scenario where the multicast sources and receivers are in the same IP subnet or VLAN. Layer 3 multicasting forwarding refers to the situation where at least some sources are in **different** IP subnets (VLANs) from at least some of the receivers, so that multicast streams have to cross subnet boundaries.



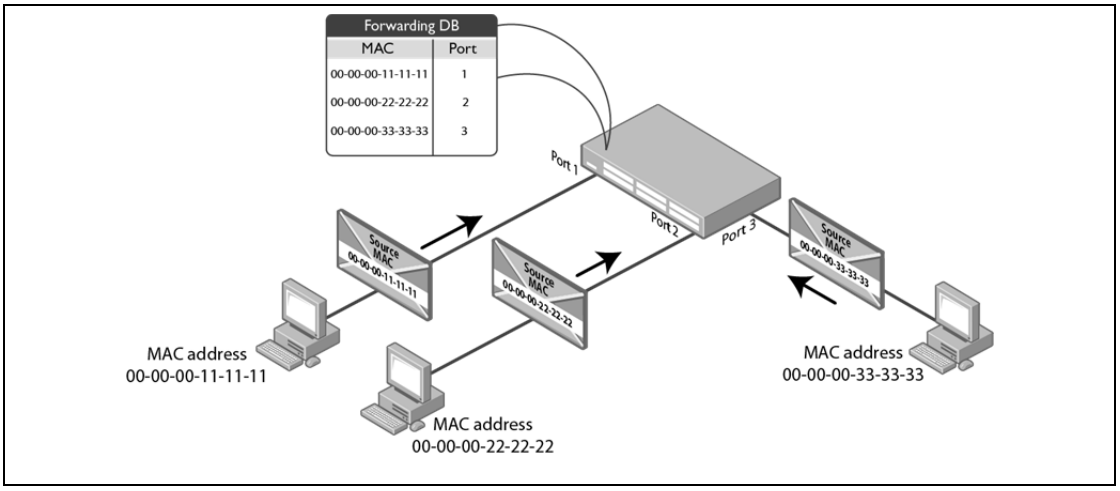
To understand this in a bit more detail, let's get a brief understanding of the processes involved in Layer 2 and Layer 3 multicasting, and also compare and contrast these processes with the processes of Layer 2 and Layer 3 unicast forwarding.

Layer 2 unicast and multicast forwarding

In an Ethernet LAN, the process of Layer 2 multicast forwarding is not radically different from Layer 2 unicast forwarding. Unicast forwarding involves learning the ports that lead to given MAC addresses, and forwarding packets out the port that corresponds to their destination MAC address. Multicast forwarding involves learning the one or more ports that lead to receivers of given multicast streams, and forwarding multicast packets to the appropriate ports, based on their destination multicast MAC address.

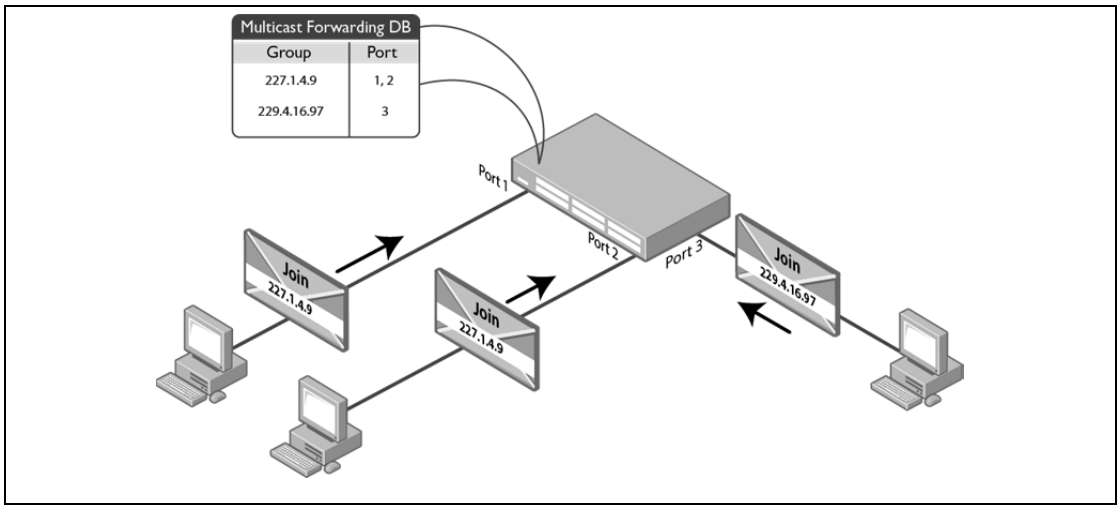
The major difference (apart, of course, from the fact that multicast packets can be sent to multiple egress ports) is the fact that in multicasting, the learning of the egress ports uses IGMP signaling (or, for IPv6, MLD signaling), whereas in unicasting, it is simply a matter of learning the source MAC addresses of the packets that arrive on given ports.

In unicast communication, a host does not need to actively signal to the switch where it resides; the switch simply learns that by seeing what port the packets with that host's source MAC address arrive on.



In multicast communication, a host that wishes to receive a certain multicast group actively signals this fact by transmitting IGMP reports that request this group. (And, conversely, transmitting IGMP leaves when it no longer wishes to receive that group).

Note – for simplicity, throughout the rest of this chapter, we will use IGMP as the example of a Layer 2 multicast protocol. The operation of MLD is almost identical to that of IGMP, so the statements made in this chapter about IGMP apply equally to MLD.

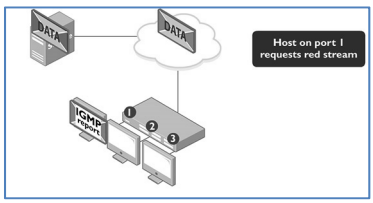


In general, within a multicast network, multiple multicast streams will be flowing. Different receivers in the network will want to receive zero, one, or more of the streams. The receivers use IGMP reports to tell the switches which, if any, streams they want to receive. If a switch has not received any IGMP reports on any of its ports, it will just drop any multicast streams it is receiving. As the switch receives IGMP reports, it creates forwarding entries for the groups requested in the IGMP reports.

If it is receiving the streams for those groups, the streams will be forwarded by these forwarding entries. Different streams will be forwarded to different sets of ports. Some ports may transmit more than one stream.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/bkxLHxCwzn8



Of course, in both unicast and multicast forwarding, it is not just the switch that is directly connected to the receiving host(s) that needs to know the direction toward the host(s). All the switches back up the chain to the source of the data also need to know where to forward the packets in the data stream.

In the case of **unicast** forwarding, this process is still a simple matter of learning source MAC addresses. As packets are forwarded through the network, the port that should transmit packets to a given MAC address is determined by learning the port on which packets with that source MAC address arrive.

In the case of **multicast** forwarding, a little more structure is required. A multicast forwarding tree needs to be created within the network.

Creating a Layer 2 multicast forwarding tree

When a switch receives an IGMP report from a directly connected host, it needs to forward that IGMP report somewhere, so that other switches can work out the ports they need to forward the relevant multicast stream from in order to reach this receiver.

They could just simply forward this IGMP report to all ports other than that which the report arrived on (in the way that a unicast packet is flooded when the destination MAC address has not yet been learnt) however, that would result in all switches believing that they need to forward streams on ports that are not necessarily in the path from the source to the receivers of those streams. This, of course, defeats the chief purpose of multicasting, which is to forward the minimum possible number of copies of any given multicast stream.

To efficiently use bandwidth, multicasting needs to impose a tree structure onto a Layer 2 network, so there is a single unique path that runs from any given source to any given receiver. There needs to be a single switch that acts as the root of this tree.

This root switch is the **Querier**—a switch that is specifically designated the task of sending out IGMP queries (packets which periodically ask receivers "*do you still wish to receive the stream(s) you have requested?*"). Query packets emanate out from the Querier in all directions, and all switches forward the queries on all ports other than that which receives them.

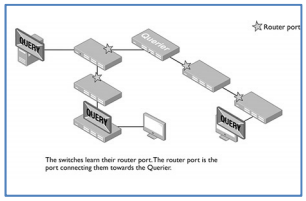
Because of this, all switches will learn which port connects them back to the Querier. This port is referred to as the **Router Port** or **All Groups** port.

The rules are:

- IGMP *reports* and *leaves* received from downstream are retransmitted out through the Router Port.
- The Router Port is added to the list of egress ports for ALL the groups that the switch is forwarding, so that all streams that arrive from anywhere (except those arriving on the Router Port) are forwarded toward the Querier.

To watch this video, browse to the link or scan the QR code with your smart phone:

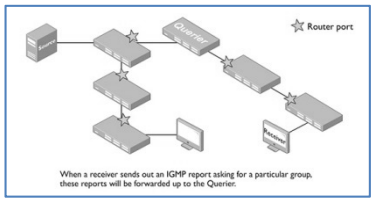
http://youtu.be/CrQGFUGFfgc



This way, the Querier becomes the root of the tree down which all multicast streams are distributed. When a receiver sends out an IGMP report asking for a particular group, these reports will be forwarded up to the Querier and all switches on the path from the receiver to the Querier learn where they need to forward this group if they start receiving it.

To watch this video, browse to the link or scan the QR code with your smart phone:

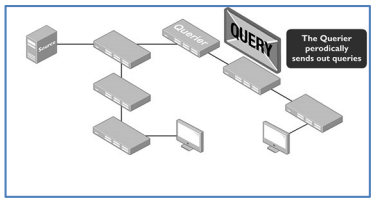
http://youtu.be/01oyjislCbE



When a source starts transmitting a stream, this stream is forwarded up to the Querier, so that it can be distributed down the distribution tree.

To watch this video, browse to the link or scan the QR code with your smart phone:

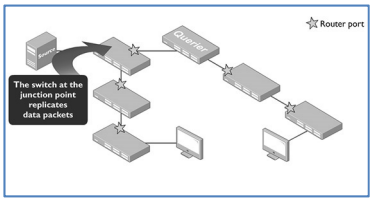
http://youtu.be/CrQGFUGFfgc



Of course, if switches that are on the path from the source to the Querier have received requests for the stream, they will forward the stream directly to the ports that they received the reports on; they do not ***have*** to receive the stream via the Router Port in order to forward it.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/zUvBLC5DeVQ



So, there we have it. Layer 2 multicasting sets up a multicast distribution tree by using the IGMP Querier as a kind of *Rendezvous Point*, where the requests sent by receivers are guaranteed to meet the streams sent by the sources. Not all the distribution of the multicast has to go via the Querier – if reports on their way up to the Querier intersect multicast streams before they reach the Querier, then those multicast streams are sent to the receiver there and then (as well as continuing to be forwarded to the Querier).

The details of the operation of the IGMP/MLD protocol are described in the chapter Multicast Listener Discovery on page 321.

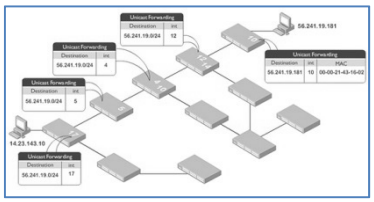
Layer 3 unicast and multicast forwarding

Now we understand Layer 2 unicast and multicast forwarding, let's now compare Layer 3 unicast and multicast forwarding.

The core activity in **Layer 3 unicast** routing is the process of finding the destination IP addresses in packets, then looking up a table of routes to find the one which best matches this destination IP address; then forwarding the packet out an interface that is referenced in that route table entry.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/LmR6DDV5YdU



The routers' route tables can be populated in various way—statically configured entries; entries pointing to the subnets directly connected to interfaces; entries learnt by RIP; entries learnt by OSPF, etc. But, regardless of how the entries get into the route table, the process of using them is the same—find the best route to a packet's destination IP, and then send the packet down that route.

There is one and only one egress interface for any given packet. Even if there are multiple equally good routes to the destination (Equal Cost MultiPath – ECMP), only one of those routes is chosen for any given packet.

Layer 3 multicasting is quite a different process to Layer 3 unicast forwarding. In contrast, Layer 3 multicasting operates quite analogously to Layer 2 multicasting. A receiver still sends IGMP reports to indicate that it wishes to receive a group. The receiver does not care whether the source of the multicast is in the same subnet as itself, or a different subnet, so the host has only one way to request a multicast group – i.e. by sending IGMP reports.

The IGMP reports arrive at a router that is the gateway from the subnet to the wider Layer 3 network. The router now has to tell other routers in the Layer 3 network to send it the group(s) that the receiver(s) has requested. The routers in the Layer 3 network then work together to create a forwarding tree that will deliver the group(s) to the receiver(s).

This *set up the forwarding tree when the group is requested* approach is rather different to Layer 3 unicast forwarding, where routers fill their tables with routes to subnets and then simply wait for packets to come along that need forwarding to those subnets.

How exactly do the routers in the network *"work together to create a forwarding tree"?*

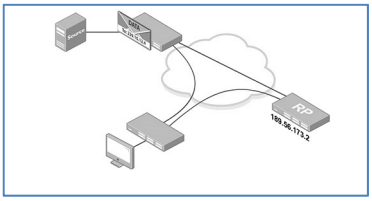
Well, that is where the Layer 3 multicast routing protocol comes in.

PIM Sparse Mode (PIM SM), for instance, designates particular routers to be the Rendezvous Points for given sets of groups. When a router needs to get hold of a given group, it sends a request to the designated Rendezvous Point for that group (and all the routers in between take note of this request, thereby setting up a forwarding path down to the requesting router). If there is a source in the network that is transmitting a stream to this group, then the Rendezvous Point will be receiving this stream (that is achieved by another aspect of the PIM SM protocol), and will be able to forward it to the requesting router. So this is really a Layer 3 version of what goes on in a Layer 2 multicast network, except that the Layer 2 network has only one Querier, whereas the Layer 3 network can have multiple Rendezvous Points, each looking after a different set of groups.

To be honest, getting to the point where the Rendezvous Point is forwarding the stream to the requesting router is not actually the end of the story. Once the receiving router receives the stream, it learns the source IP of the stream, and then sets up a new request directly back up the path to that source IP (which may or may not be via the Rendezvous Point), so that it is eventually receiving the stream via the most efficient path available.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/8e3M3kG6L18



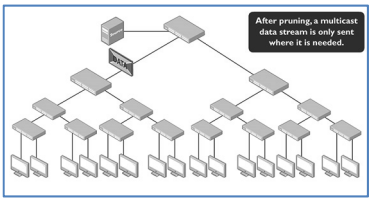
In PIM Dense Mode, the process works quite differently. Instead of having Rendezvous Points that know about which streams are available, the network lets every router directly find out about every stream that is available. Whenever a new stream starts being transmitted into the network, all the routers that receive the stream forward it to all their neighbors (except that from which they received it) so that every router in the network becomes aware of the existence of this stream. Then, starting from the edge of the network (i.e. from those routers that have no downstream neighbors) those routers who have no downstream receivers for this group start telling their upstream neighbors to stop sending it to them – a process known as **pruning**.

That way, the forwarding tree for the new stream settles down to be just what is required to get the stream to all the receivers who want it.

But, the router nearest the source of the stream continues to send out periodic reminders that the stream is still being produced, so that all the routers know it is still available if they need to request it.

To watch this video, browse to the link or scan the QR code with your smart phone:

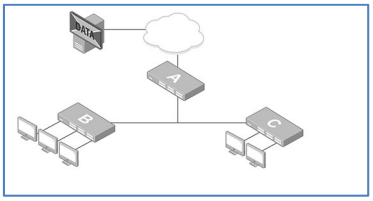
http://youtu.be/WogtCSBNZOI



Therefore, when a router which is not currently receiving a given stream needs to get that stream, it knows where to send its request for it.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/-S1wHs9RvCU



You may be thinking *"Isn't this business of flooding a new stream* **everywhere** *rather against the bandwidth conserving spirit of multicasting?"* In which case, you would be right. But, the benefit of this approach is that it makes PIM Dense Mode a much simpler protocol than PIM Sparse Mode, as it does away with the RPs and the rather complicated processing that they entail.

The details of the PIM SM and PIM DM protocols are explained in their respective chapters.

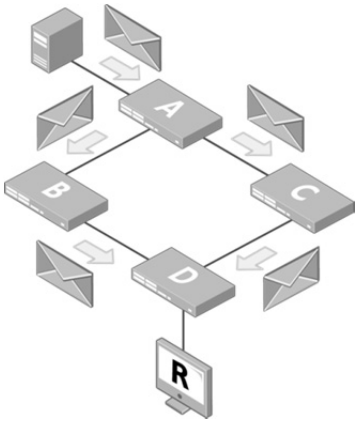
Terminology

As with any field of activity, IP multicasting has built up its own language of specific terms. Such terminology often becomes a bit of a barrier to understanding the subject. Let's get the terminology out into the open, and explained, so it becomes a help rather than a hindrance.

Reverse Path Forwarding (RPF)

Multicasting faces a slight dilemma that is particularly notable in PIM Dense Mode. On the one hand, there is the desire for all routers to send streams to all possible downstream ports; on the other hand, there is the need to prevent data being duplicated in looped networks.

Consider the simple example illustrated below – the multicast stream from the source arrives at Router A. Router A sends the stream to both its downstream neighbors – B and C. They, in turn, both send the stream to their downstream neighbor – D.



So, poor old D receives two copies of the stream. When host R requests this stream, what does D do? How can it know if the stream it is receiving from B is the same as that received from C? D is just a router; it cannot analyze the content of video to decide if the streams are duplicates of each other.

Should it just send both copies to R, and let R sort it out? But that is a waste of bandwidth, and potentially overloads R. And, what if R can't "*sort it out*"?

Rather than have these difficult choices, it is much better to have a simple clear method to decide which streams to accept, and which to simply ignore.

The simple clear method is the **Reverse Path Forwarding** criterion. Stated simply, the decision that this criterion dictates is:

- If the interface that a stream arrives on is the interface out which the router would forward unicast packets to the source IP of the stream, then accept the stream. If a stream is arriving on an interface that does not satisfy this criterion, then ignore it (**drop the packets**).

The term **Reverse Path Forwarding** refers to the fact that you are deciding whether or not the interface that the stream is arriving on is the interface on which you would forward packets destined onto the Reverse Path back up to the source of the stream.

The term Reverse Path Forwarding is typically abbreviated to **RPF**. The correct interface on which to receive a stream is frequently referred to as the **RPF interface**.

How does the RPF concept help out D in our example above? After a little thought, you may conclude that our example illustrates rather a flaw in this RPF idea. If the routes D-B-A-S and D-C-A-S are equal cost, then D can't say which is the unique RPF interface – they are both RPF interfaces. Unicast packets destined to S would just as likely be sent via B as via C.

So, yes, the RPF rule needs a bit of refinement to deal with the ECMP situation. Some extra criterion needs to be introduced as the tie breaker when there are ECMP routes back to the source. The RFCs do not specify what the criterion should be, so different implementations can choose different criteria like

- Look through all the egress interfaces used by the routes in the ECMP group, and choose the interface with the highest IP address.

OR

- Look through all the egress interfaces used by the routes in the ECMP group, and choose the interface which is connected to the PIM neighbor with the highest IP address.

The method used by AlliedWare Plus is to perform a hash calculation on the IP address of the source, and the IP address of the next hop neighbor. This calculation is performed for the next hop neighbor on each of the routes in the ECMP group. The next hop neighbor that results on the highest hash value is chosen as the RPF neighbor for the route back to that source.

(S,G)—pronounced *S comma G*

This little construction looks somewhat mathematical – a definition of a vector or some little piece from Group Theory or the like. Seeing a page dotted with this, or similar constructs like (*, G) or (RPT,G) can be a little worrying – "*arrggh, what's this? Algebra? Vectors? Some other abstract mathematical concept?*"

But, in fact, the (S,G) construct is not anything particularly mathematical; it is just a convenient shorthand for the phrase "*a stream from source IP address S to group address G*". Similarly, the construct (*, G) means "*a stream from ANY source IP to the group IP address G*".

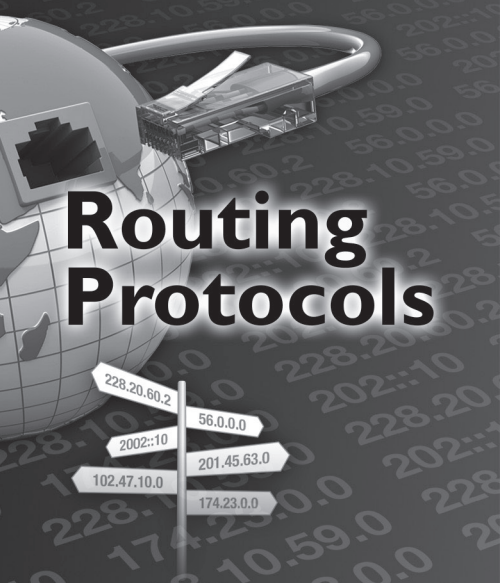
Because any discussion of multicasting so often refers to streams to a given group G – either from a specific IP or from ANY IP – having a simple shorthand way to represent that idea saves a lot of typing, and adds some precision to the discussion.

Graft, Prune

When discussing multicasting, it is common to talk of the multicast data being sent down a distribution ***tree***. A tree makes a very good analogy for the way that multicast data is transmitted from a source to multiple receivers. From the root of a tree you can trace unique paths to all the leaves. There is one and only one path from the root to any given leaf; and the paths to different leaves will follow the same route for as long as they can, and only diverge when they have to.

Because of this *tree* analogy, other tree-related terms have entered the language of multicasting. When a router realizes that nothing downstream of it needs to receive a given group, it can inform its upstream neighbor that it no longer wishes to receive that group. The router is said to **prune** itself off the distribution tree for that group.

Similarly, when it finds that once again something downstream of it needs to receive that group, it has to request that its upstream neighbor sends it that group. The router is then said to **graft** itself back onto the distribution tree for that group.



Routing Protocols

This chapter covers the following topics:

- Introduction
- Basic Features of MLD
- Details of MLDv1
- Summary of Timers and Counters used in MLDv1
- Details of MLDv2
- Packet Types
- Content of MLDv2 packets
- Overview of the Operation of an MLDv2 Querier
- Summary of Differences between MLDv1 and MLDv2

CHAPTER 10

Multicast Listener Discovery

Introduction

Multicast Listener Discovery (MLD) is the IPv6 equivalent of Internet Group Management Protocol (IGMP). Fortunately, as with most aspects of IPv6 routing, the features and operation of MLD closely resembles those of its IPv4 equivalent. In fact, the RFCs defining MLD quite explicitly state that they are adaptations of IGMP to IPv6.

The packet types, timers, actions etc in MLD are very much the same as those in IGMP.

As with IGMP, there are multiple versions of MLD. Because MLD has been defined more recently than IGMP, the version numbers of MLD are actually one behind the version numbers of IGMP.

This means that:

- MLDv1 is equivalent to IGMPv2.
- MLDv2 is equivalent to IGMPv3.

This chapter will:

- Start with some of the basic facts of MLD – the general packet format, addressing, etc.
- Describe MLDv1 in detail.
- Describe MLDv2 in detail.
- Summarize the differences between MLDv1 and MLDv2.

Basic Features of MLD

The basic actions that occur in the MLD protocol are:

1. Hosts that wish to receive certain multicast groups send out MLD reports requesting the groups.
2. Routers and switches that receive these reports create forwarding entries to forward the groups in question out via the interface on which the report was received.
3. Routers send out MLD queries at intervals, to find out who still wants to receive each group.
4. If a router or switch has not recently seen any reports requesting a given group on a given interface, it will remove the forwarding entry that is delivering that group to that interface.

As we will see in the detailed descriptions of MLDv1 and MLDv2, there is more to it than this, but the general gist of the operation of MLD is described in those 4 actions.

In the actions above, a router is an MLD querier. The other switches in the network (i.e. those between the listening hosts and the querier) will perform MLD snooping. In this chapter we will only discuss the operation of MLD routers, and will not consider MLD snooping in any detail.

As with a lot of the signaling packets in IPv6, MLD packets are ICMPv6 packets. The different types of MLD packets have different ICMPv6 type values, as will be discussed in the details sections below.

The source address of MLD packets is the link-local address of the device sending the packet (in MLDv2, it is actually permissible to send MLD packets with an all-zeros source address if the link-local address has not yet been calculated).

MLD packets always have a hop-count of 1.

MLD packets contain a Hop-by-Hop Options extension header that includes the Router Alert Option. The effect of this option is that all routers need to process MLD packets, irrespective of whether the router believes it is actively listening for the multicast address to which the MLD packet is destined.

Details of MLDv1

As mentioned above, MLDv1 is modeled on IGMPv2.

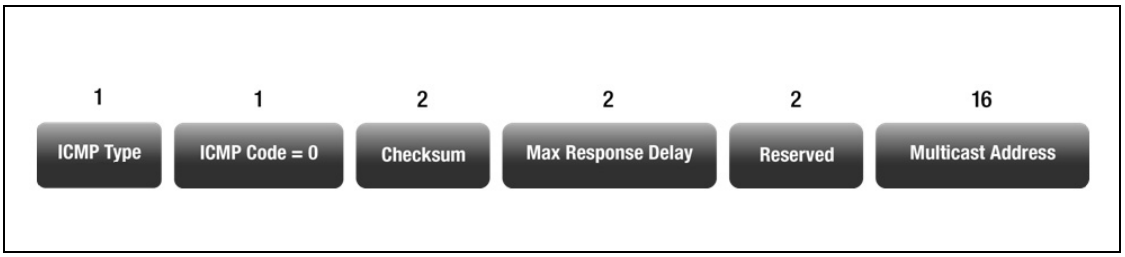
MLDv1 uses three types of packets:

1. MLD reports – sent by hosts wishing to receive a group.
2. MLD queries – sent by the querier to check who is still listening for a group (or, for ANY group).
3. MLD 'listener done' packets – sent by a host when it no longer wishes to receive a certain group (equivalent to the IGMPv2 Leave).

MLD queries fall into two categories:

- General query – asks hosts to send reports for ANY groups they are listening to
- Multicast-Address-Specific Query – asks hosts to send reports if they are listening to a specific group.

MLDv1 packets have a simple format:



Packet Type	Source Address	Destination Address	ICMP Type	Multicast Address in Packet
General query	Sender's link-local address	FF02 : : 1	130	: :
Multicast address-specific query	Sender's link-local address	The specific group address being queried about	130	The specific group address being queried about
Report	Sender's link-local address	The address of the group the host wishes to receive	131	The address of the group the host wishes to receive
Listener Done	Sender's link-local address	FF02 : : 2	132	The address of the group the host no longer wishes to receive

All MLD packets have an ICMP code value of 0.

Operation of an MLDv1 querier

The main duties of an MLD querier are to:

- Send general queries, to find out who is still listening.
- Handle incoming reports, creating forwarding entries for the groups requested in those reports. If the MLD querier is also acting as a Layer 3 router of IPv6 multicast (using PIMv6 or similar), the MLD query process also needs to tell the Layer 3 multicast routing process to send upstream requests for the groups being requested in incoming MLD reports.
- Handle incoming Done messages. Removing the forwarding entry for the groups specified in the Done message, after first checking that no other host on the same interface is still listening to that group.

In addition, a querier has to:

- Listen for incoming queries from other queriers that might be operating in the same Layer 2 network. If the querier receives a query from a device with a lower IP address, then it has to stop operating as a querier, and let the lower-address device operate as the querier for the local network.
- Time out forwarding entries for which no reports have been received recently.

Let's look into each of these duties in more detail. Most of these operations are controlled by counters and timers. At the end of the discussion of the various duties, there is a table summarizing all the counters and timers. The table summarizes the meanings of the counters/timers, their default values, their relationships to other counters and/or timers, and the AlliedWare Plus command to configure the value of the parameter.

Sending general queries

An MLD querier needs to keep track of the locations of hosts listening to given groups. The key to achieving this *keeping-track* duty is to send out queries to say, "*If you are listening to* ***any*** *IPv6 multicast group, then please send in reports for all the groups you are listening to*." These queries are the MLD General Queries.

When a querier starts up, it needs to send General Queries soon after startup, to get a picture of where all the listeners are. Then, to keep this picture up-to-date, it needs to continue sending General Queries at regular intervals.

Queries at startup

When an MLD querier first starts up, i.e. when the device boots up, or when MLD querying is configured on an interface (or globally for the whole device), it should send out a set of MLD General Queries. If the device is configured as an MLD querier on multiple interfaces, then it should send this set of MLD General Queries on all the interfaces for which it is configured as

an MLD querier. In the case that MLD querier is newly configured on one interface, and had already been active on other interfaces, the set of general queries need only be sent out the newly active interface.

The number of General Queries sent at this startup phase is controlled by a counter referred to as the **Startup Query Count**. The time interval between these General Queries being sent is called the **Startup Query Interval**.

So, a newly active MLD querier interface will send out Startup Query Count General Queries, each Startup Query Interval apart.

Regular periodic queries

After the querier has sent out its initial burst of General Queries at startup, it settles into a pattern of sending General Queriers out all querier interfaces at regular intervals.

The period at which it regularly sends these General Queries is referred to as the **Query Interval**.

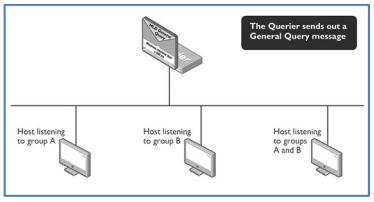
Reducing the number of reports received

An interesting aspect of the MLD query is the **Maximum Response Delay**. This is the amount of time within which the querier expects listening hosts to respond to the query. Initially, it might seem a little pointless to specify a Maximum Response Delay, because the querier does not go into a mode whereby it stops listening for reports; a querier is **always** listening for reports. It is not as though the querier is saying, "*If you do not respond within the Maximum Response Delay, it is too late, your report will be ignored*."

What is the purpose of the Maximum Response Delay? The answer is that it is part of the process of reducing the number of unnecessary responses the querier receives. Instead of every host sending in reports as soon as they receive a query, the hosts actually choose to wait a short time before they send in their reports. The amount of time they wait before sending in their reports is a randomly chosen value between 0 and the Maximum Response Delay specified in the General Query. If a given host sees that another host in the same Layer 2 network sends in a report for one of the same groups that it is listening to, then it realizes that it does not need to also send the same report, and therefore does not send it.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/_DdHaXfD6VQ



Rather than every host immediately sending reports for all the groups it is listening to, each host chooses a delay time for each group it is listening to, and only sends a report if it has not seen someone else send a report for that group within that delay time.

The value that the querier puts into the Maximum Response Delay of its query packets is referred to as the Query Response Interval.

Handling incoming reports

Hosts that are listening to certain groups send out MLD reports to indicate that that they wish to receive data streams being sent to those group addresses. The reports may be sent in response to queries, as described above, or they may be unsolicited queries, which are sent when the host first starts listening to a given group (e.g. when the viewer changes channels).

Irrespective of whether a report is sent in response to a query, or is unsolicited, the actions taken by the MLD querier upon receiving the report are the same.

Simply, if the querier is not already forwarding the requested group out of the interface on which the report is received, then the querier will create a forwarding entry to start forwarding that group out of that interface. When the forwarding entry is created, the querier also associates a timer with the entry. If no more reports are received on that interface, requesting that group, before the timer expires, then the forwarding entry is removed. The time period of this timer is referred to as the Multicast Listener Interval.

If the querier already has a forwarding entry for the requested group on the receiving interface, then the timer for that entry is simply reset back to the start.

The process of sending out periodic General Queries, and eliciting reports in response, has the effect of resetting the timers on the forwarding entries corresponding to the reports that arrive in response to the General Queries.

Handling Listener Done messages

When a host no longer wishes to listen to a given group, it should send out a Listener Done message to report the fact that it is no longer wishing to receive data for that group.

When a querier receives a Listener Done message, it should not immediately just remove the forwarding entry for the group in question, on the interface in question. This is because other hosts downstream of that interface could still be listening to this group.

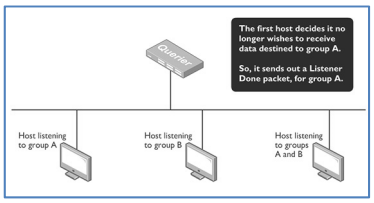
The querier needs to have a way to check if other hosts are still listening. The way it performs the check is to send out Multicast-Address-Specific queries.

The purpose of the Multicast-Address-Specific query is to ask, "*Is anyone still listening to this specific Group?*" If there are hosts who are still listening to that group, then the querier should receive a report requesting that group. As with the case of the General Query, the hosts listening to the specified group do not all respond at once, but set random delay timers, based

on the Maximum Response Time in the Query packet, and decide not to reply if they see another host sends the required report before they do.

To watch this video,browse to the link or scan the QR code with your smart phone:

http://youtu.be/Z6VXyko0vRE



Because of the seriousness of the act of cutting off the sending of data that hosts still want to receive, the querier will typically send out multiple Multicast-Address-Specific Queries, to be sure that at least one of them is seen by the hosts that need to see it. The number of queries that the Querier will send is referred to the **Last Listener Query Count**. The time between these queries is the **Last Listener Query Interval**.

The Maximum Response Time to put into Multicast-Address-Specific Queries is also the Last Listener Query Interval.

Once all the Multicast-Address-Specific Queries have been sent out, and no report for that specific Group received, the forwarding entry for that Group, on the interface in question, is deleted.

Handling queries received from other queriers

Only one MLD querier is required in any given Layer 2 domain. In fact, it is not desirable to have more than one querier in a Layer 2 domain, as it generates extra traffic.

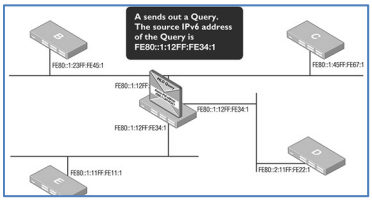
If there are multiple devices connected to a Layer 2 domain that have been configured as MLD queriers, a method is needed whereby they can decide amongst themselves which one of them will operate as the active querier.

The method is very simple:

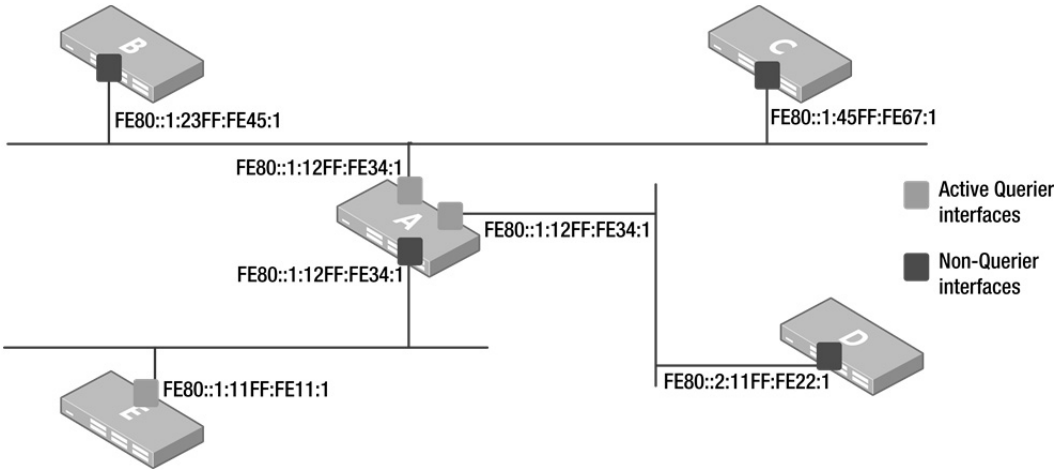
- If a querier receives a query from another device, it looks at the source IPv6 address of the query. If the source IPv6 address is numerically lower than the link-local address it is using as the source address of MLD packets it is sending out of that interface, then it stops operating as a querier (goes into a mode called 'non-querier'). This way the device using the lowest IPv6 address on that Layer 2 domain ends up being the only active querier in the Layer 2 domain.
- If a non-querier has not received any queries, with lower source IPv6 addresses than its own, for a certain period on a given interface, then it resumes the role of active querier on that interface. The time period it waits is called the **Other Querier Present Interval**.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/IuKJHd1chts



Note, a device that is connected to multiple different Layer 2 domains, and configured as an MLD querier on its interfaces into those multiple domains, can be an active querier on some of those interfaces, and a non-querier on others of the interfaces.



Timing out forwarding entries

Although a host should send a Listener Done packet when it no longer wishes to receive data for a given group, there are plenty of circumstances in which the querier will no longer need to send a group to a given host, even though the querier had not received a Listener Done packet from that host:

- The host was simply disconnected from the network.
- The host was powered down, or crashed, suddenly.
- The host did send a Listener Done packet, but it was lost before reaching the querier.
- The multicast client application in the host is faulty, and does not send Listener Done messages.

As you can see, the querier cannot rely on the receipt of Listener Done messages as the only trigger for deleting group forwarding entries.

That is why there is a delete timer associated with every forwarding entry created in a querier. If the delete timer ever counts down to zero, that indicates that no MLD reports have been received for that group on that interface for a reasonable length of time, so it is safe to assume that no hosts downstream of that interface still wish to receive the group. Hence, the forwarding entry can be removed.

It is important that unnecessary forwarding entries are cleaned out. If a querier never deleted forwarding entries, then it would simply continue to forward data for every group for which it had ever received a report, regardless of whether every one of those groups still had any hosts interested in receiving it. This would be a waste if network bandwidth, and could easily lead to congestion in the network.

Timers and Counters Used in MLDv1-Summary

Parameter	Meaning	Default Value	Relationship to other parameters	AlliedWare Plus configuration command
Query Interval	Period at which querier sends General Queries	125 sec	Must be greater than the Query Response Interval	`IPv6 MLD query-interval` (per interface)
Query Response Interval	The value put into the Maximum Response Delay of General Query packets. Sets the maximum time hosts wait before sending in reports.	10 sec	Must be less than the Query Interval	Not configurable
Multicast Listener Interval	The period, during which no reports are received for a given Group on a given interface, before the querier deletes the forwarding entry for that Group on that interface.	Defined by relationship to other parameters	Defined as Robustness Variable X Query Interval + Query Response Interval	Not configurable
Other Querier Present Interval	The period, during which no Queries, with a lower source IPv6 address, are received on a given interface, before a device transitions from non-querier to Active Querier on that interface.	The RFC states that this is defined by its relationship to other parameters. AlliedWare Plus sets the default to 255 seconds.	Should be Robustness Variable X Query Interval + (Query Response Interval / 2)	`IPv6 MLD querier-timeout` (per interface)
Startup Query Interval	The time between the multiple General Queries sent on a newly active Querier interface.	Query Interval / 4	Should be Query Interval / 4	Not configurable
Startup Query Count	Number of General Queries sent when a querier interface starts up.	Equal to Robustness Variable	Should be the same as Robustness Variable	Not configurable

Parameter	Meaning	Default Value	Relationship to other parameters	AlliedWare+ configuration command
Last Listener Query Count	The number of Multicast-Address-Specific Queries sent after a *Listener Done* message is received.	Equal to Robustness Variable	Should be the same as Robustness Variable	`IPv6 MLD last member-query-count` (per interface)
Last Listener Query Interval	The time between each of the Multicast-Address-Specific Queries sent after a *Listener Done* message is received.	1000	N/A	`IPv6 MLD last-member-query-interval` (per interface)
Robustness Variable	The number of times that certain packets are repeated, to allow for the possibility that packets will be lost.	2	Used to calculate the expected values of several other parameters, as seen above.	`ipv6 mld robustness-variable` (per interface)
Unsolicited Report Interval	When a host first wishes to receive a given group, it sends out a number (equal to the Robustness Variable) of unsolicited reports to request this group. This parameter is the time between these reports.	10 seconds	N/A	N/A

Details of MLDv2

The fundamental change that is introduced by MLDv2 is awareness of the **source** of multicast streams.

In MLDv1, neither the listening host nor the querier care about the source from which a multicast stream is sent. The listening host just says, "*Send me a multicast stream destined to Group G, I don't care where it comes from.*" Likewise the querier simply creates forwarding entries that forward data whose destination is the Group address G; the forwarding entry does not check on the source address of the packets it is forwarding.

But, MLDv2 introduces the ability to support **Source-specific** multicast, in which the forwarding or not-forwarding of a multicast stream to group G can depend on the source address from which the stream is sent.

Whilst this might seem quite a small addition, the changes required to reliably support source-specific multicast are quite significant.

A major reason why the support of source-specific multicast requires significant changes to MLD is the level of flexibility that has been provided for the way that hosts can specify their requirements for the source address of the streams they are listening for.

If it was simply a matter of each host specifying one-and-only-one source for each Group they are listening for, then the changes required in MLD would be much simpler. However, MLD has actually been extended to allow hosts to specify

- a list of different sources from which they would accept a given Group

OR

- a list of the sources from which they would NOT accept a given Group (implicitly, they will accept it from any other source).

An MLD querier can't simply keep a list of the sources from which its downstream hosts have requested to receive each Group. It needs to:

- keep track of which sources the downstream hosts will accept a Group from
- keep track of which sources the downstream hosts will not accept a Group from
- update the lists when a host changes its source-list for a given Group from a *will accept* list to a *won't accept* list or vice versa
- decide when a given source address is no longer in any downstream host's *will accept* list for a given Group G.

Therefore, in MLDv2, a querier has more to keep track of, and a greater range of events to handle than it did in MLDv1. To make this all work in a sensible fashion, MLDv2 brings in

some new concepts, changes the types of packets that are exchanged, and adds a number of new fields to the packets.

Let us look at these new concepts, packet types, packet data, etc and gain an understanding of how MLDv2 works.

Include and Exclude

Key concepts introduced into MLDv2 are those of the **source filter** and **source filter mode**.

The **source filter** refers to the fact that a host will only accept a Group G from some sources, and not from others. So the host is said to the 'filtering' the incoming packets, based on their source address. The host will filter out those packets that are not sent from a source from which the host will accept group G.

The **source filter mode** refers to whether the host is filtering on the basis of a list of sources from which it will accept group G (and, so, dropping packets for group G that originate from any other sources), or is filtering on the basis of the sources from which it will not accept packets that are destined to G (and, so, dropping packets for group G that originate from those banned sources, but accepting from all other sources).

- If the host is filtering based on a **will accept** list, its source filter mode is said to be **Include**.
- If the host is filtering based on a **won't accept** list, its source filter mode is said to be **Exclude**.

To prevent too much complication, the operation of source-specific multicast restricts a host to filter in either Include Mode or Exclude Mode for any given group on any given interface. Any mixture of Include Mode AND Exclude mode being used together for a Group G on a given interface is specifically precluded.

An MLDv2 querier, then, will receive reports from downstream hosts – with some of the hosts saying "*Include these sources*," and other hosts saying "*Exclude these sources*." Somehow, the querier has to satisfy all these different requests. One simple way to do so would be to simply forward all the streams it receives for a given Group, and let the hosts sort out which of those streams they will and will not accept. However, that approach is contrary to the spirit of IP multicasting, which aims always to conserve bandwidth, and forward only those packets that need to be forwarded.

Hence, an MLDv2 querier needs to put some effort into working out which streams it does and does not need to forward.

The basic rules that an MLDv2 querier applies are:

1. If all the downstream hosts on a given interface I are operating in Include mode for a given Group G, then simply keep a list of sources that is the aggregate of the sources from which the various hosts have indicated they will accept G. Then, only forward a stream to group G if its source is in that list.

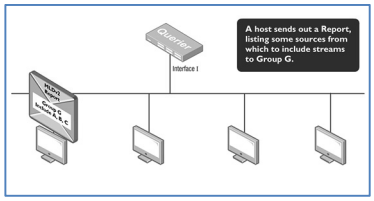
   Therefore, the querier is, by default, not forwarding streams from a given source unless it is in the list of requested sources.

   In this case, the querier is said to be operating in **Router Include** mode for Group G on Interface I. Of course, this mode does mean that some hosts will be receiving group G from sources other than those which they requested. Unless all the downstream hosts happened to request exactly the same set of sources, there will be some hosts that requested sources that others did not. However, the Querier needs to err on the side of no host missing out on a source that it did request, so it must forward streams from any source that any one of the downstream hosts requested.

   There will still be a requirement for each host to "*sort things out for themselves*," and actively take note of the sources from which they are receiving G, and filter out the packets that are from sources they did not request.

   To watch this video, browse to the link or scan the QR code with your smart phone:

   http://youtu.be/QO-U-UVBXgg

2. If at least one host downstream on a given interface I is operating in Exclude mode for a given group G, then the querier must cut over to a different way of operating. Because, when a host is operating in Exclude mode, it is saying "*Send me Group G from ANY source except the following….*"

   The querier needs to go over to a mode whereby, by default, it is forwarding G from ANY source unless it is sure that NO downstream hosts wish to receive G from this source.

   In this case, the querier is said to be operating in **Router Exclude** mode for group G on interface I.

In this mode, the querier will forward G from a given source **unless**:

- every downstream host that is operating in exclude mode has this source in its exclude list

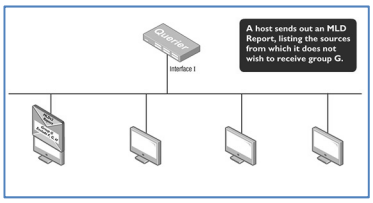
  AND

- no downstream host that is operating in include mode has this source in its include list.

  This means the querier has to see that all the downstream excluders exclude a given source, and none of the downstream includers include this source before deciding that it definitely does not need to forward group G from this source. Once again, the querier is erring on the side of ensuring that no downstream host is missing out on receiving G from a source from which it desires to receive G (or, from a source from which it is not explicitly stating it will not accept G). So, again, each host will have to do a certain amount of "*sorting things out for themselves*" to filter out streams from sources that they are not interested in.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/cu3uxvfX-fU



Packet Types

MLDv2 has less packet types than MLDv1, but the content of MLDv2 packets is more complex than that of MLDv1 packets.

Signaling from hosts

In MLDv1, the signaling from hosts is quite simple. A host either says:

- I wish to receive group G (MLDv1 report)

OR

- I am not longer interested in receiving group G (MLDv1 Listener Done).

The host-side signaling in MLDv1 is naturally served by two packet types – the Report and the Listener Done.

In MLDv2, however, the picture is not quite so simple. Certainly, an MLDv2 host will report that it wishes to receive a certain group, but it also needs to report which sources it wishes (or, does not wish) to receive this group from. Also, it can change the set of sources from which it wishes to receive the group. If it decides to reduce the set of sources from which it wishes to receive the group, that is sort of an act of leaving (in that it is leaving the streams that bring G from certain sources), but it is not fully a Leave (as it still wishes to receive G from at least some sources). Also, if the host indicates that it wants to stop receiving from some sources, and, at the same time, wishes to start receiving from other sources, then that is an act of leaving and an act of joining all at the same time.

For these reasons, it does not really make sense to split the host-side signaling of MLDv2 into Reports and Listener Dones. Instead, MLDv2 hosts just send reports.

These reports can indicate a variety of messages, though, like:

- I want to start receiving G from the following sources.
- I am still wishing to receive G from the following sources (in response to a Query).
- I no longer wish to receive G from the following sources.
- I wish to receive G from anything but the following sources.
- I am still wishing to receive G from anything but the following sources (in response to a query).
- I used to be in Include mode, but I have changed to Exclude mode, and here is the list of sources from which I do not wish to receive G.
- Etc.

The full list of possible messages, and how they are structured in the Report packet, is provided below in the section Reports on page 340.

Another significant difference between the MLDv2 Report and the MLDv1Report is that the MLDv2 Report can send information about multiple groups all in the same packet. By contrast, in MLDv1, each Report packet was able to request only one Group.

For this reason, the MLDv2 reports are not sent to the Group address to which the Report pertains, as the Report does not necessarily pertain to just one Group. Rather, MLDv2 reports are sent to a reserved Multicast address FF02::16.

Also, in recognition of the fact that the MLDv2 Report is really a quite different beast to the MLDv1 Report, the MLDv2 Report is allocated a new ICMP type – namely 143 (MLDv1 Reports have ICMP type 131).

Summary of MLDv2 packet types

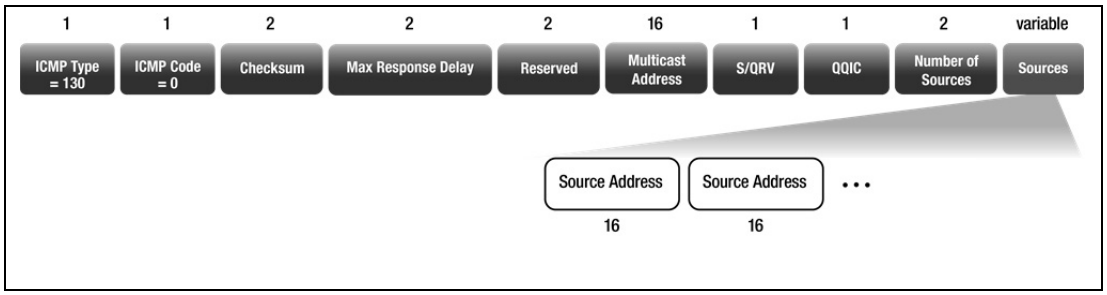
Packet Type	Source Address	Dest Address	ICMP Type	Multicast Address in packet
General Query	Sender's link-local address	FF02::1	130	::
Multicast-address-specific Query	Sender's link-local address	The specific group address being queried about	130	The specific group address being queried about
Multicast-address-and-source-specific Query	Sender's link-local address	The specific group address being queried about	130	The specific group address being queried about
Report	Sender's link-local address	FF02::16	143	Multiple multicast addresses

Content of MLDv2 Packets

Queries

The MLDv2 Query is not vastly different from the MLDv1 Query. The main difference is that the Query can contain a set of IPv6 source addresses (for the case of the Multicast-Address-And-Source-Specific Query).

The Query packet structure is as shown below:



Let's work through the fields in the packet, and consider the meaning of each field.

Type

This is the ICMP Type, which is 130 for MLD Queries (both MLDv1 and MLDv2 Queries).

Code

This is the ICMP Code, which is 0 for all MLD packets (MLDv1 and MLDv2 packets)

Checksum

This is a standard ICMPv6 checksum.

Maximum Response Code

This field has the same meaning as in MLDv1, i.e. the time within which hosts should respond to the Query. However, the format of the field is different in MLDv2.

If the value in the field is less than 32768, then the value simply represents the Maximum Response Delay in milliseconds (just as in the MLDv1 query).

However, to allow for longer delays (if the querier wants to reduce the burstiness of the responses to General Queries), the field can contain floating-point value. If the raw value in the field is greater than 32768, then the value is actually an encoded floating-point number that can be unencoded by manipulating the bits according to an algorithm defined in RFC3810.

Multicast Address

In General Queries, this field is set to the IPv6 address : : (i.e. all zeros). In Multicast-Address-Specific Queries and Multicast-Address-And-Source-Specific Queries, the field contains the Multicast address to which the Query pertains.

S Flag

This little Flag represents an interesting advance of MLDv2 with respect to MLDv1.

As with MLDv1, an MLDv2 non-querier needs to listen to, and act on, MLD reports. The non-queriers need to keep a *shadow* of the forwarding states and timers for all the Groups that are being forwarded into the network, so that if they have to take over as querier, they can do so with minimum disruption.

When the querier sends out Multicast-Address-Specific Queries or Multicast-Address-And-Source-Specific Queries, in response to a Report that a host no longer wishes to receive a certain Group (or, no longer wishes to receive the Group from a certain source), the non-queriers need to reduce their keep-alive timers for that Group (or that Group-and-Source combination).

When a Report is received that indicates that a host still is interested in that Group (or that Group-and-Source combination), the querier, and all the non-queriers will once again increase the keep-alive timer for the Group (or the Group-and-Source combination).

But, the protocol needs to allow for the possibility that the non-queriers did not see the initial Query from the querier, or missed the Report that arrived from the still-interested host. So, the querier will always repeat the Multicast-Address-Specific Query or Multicast-Address-And-Source-Specific Query a few times (in fact, **Last Listener Query Count** times), to try to make sure that everybody sees all the right packets.

But, this repeating of the Query causes a potential problem. If a non-querier missed the first Query, and the responding Report, but does receive the second (or subsequent) Query, then it will go and reduce its keep-alive timer for the Group (or Group-and-Source combination). But, actually, the querier knows, at this point, that the non-queriers should not be reducing their keep-alive timers, because the querier (having received the Report) already knows that there is a host that still wants to receive this Group (or Group-and-Source combination).

The querier needs a way to say to the non-queriers "*Look, I am sending out this Query, even though I already know the answer, because I always repeat Queries, for robustness sake. So, don't go lowering your keep alive timer for the Group (or Group-and-Source combination) to which this query pertains*." This is what the S Flag is for. The "S" stands for "Suppress Router-Side processing", i.e., it tells routers (non-queriers) to suppress their inclination to lower keepalive timers upon receiving the Query.

MLDv1 does not have an equivalent flag. This is a bit of a hole in the protocol that MLDv2 has plugged.

QRV

This acronym stands for Querier's Robustness Variable. It is sent in the packet for the benefit of non-queriers, so that the non-queriers will change their robustness variable to match that of the current querier. If a non-querier takes over as querier, it will continue to use the same robustness variable as the previous querier was using. Again, this is aimed at maintaining as much continuity as possible when there is a change of querier.

QQIC

This is another value that is sent out so that non-queriers can use the same parameter values as the current querier. The acronym stands for Querier's Query Interval Code. The word "*Code*" is taken on because, like the Maximum Response Code, the value in this field can be either a simple integer or an encoded floating-point number.

If the value is less than 128, then it is a simple integer. If it is **greater** than 128, it is an encoded value, which can be decoded according to an algorithm described in RFC3810.

Number of Sources

For General Queries and Multicast-Address-Specific Queries, no sources appear in the packet, so this value is zero, and this is the end of the packet.

For Multicast-Address-And-Source-Specific Queries, this number indicates how many IPv6 source addresses will now follow.

Source Addresses

This is the list of sources to which a Multicast-Address-And-Source-Specific Query pertains.

When hosts which do wish to receive Group G from one or more of the Sources in this list process the Multicast-Address-And-Source-Specific Query, they send back a Report which lists the sources from which they do wish to receive G.

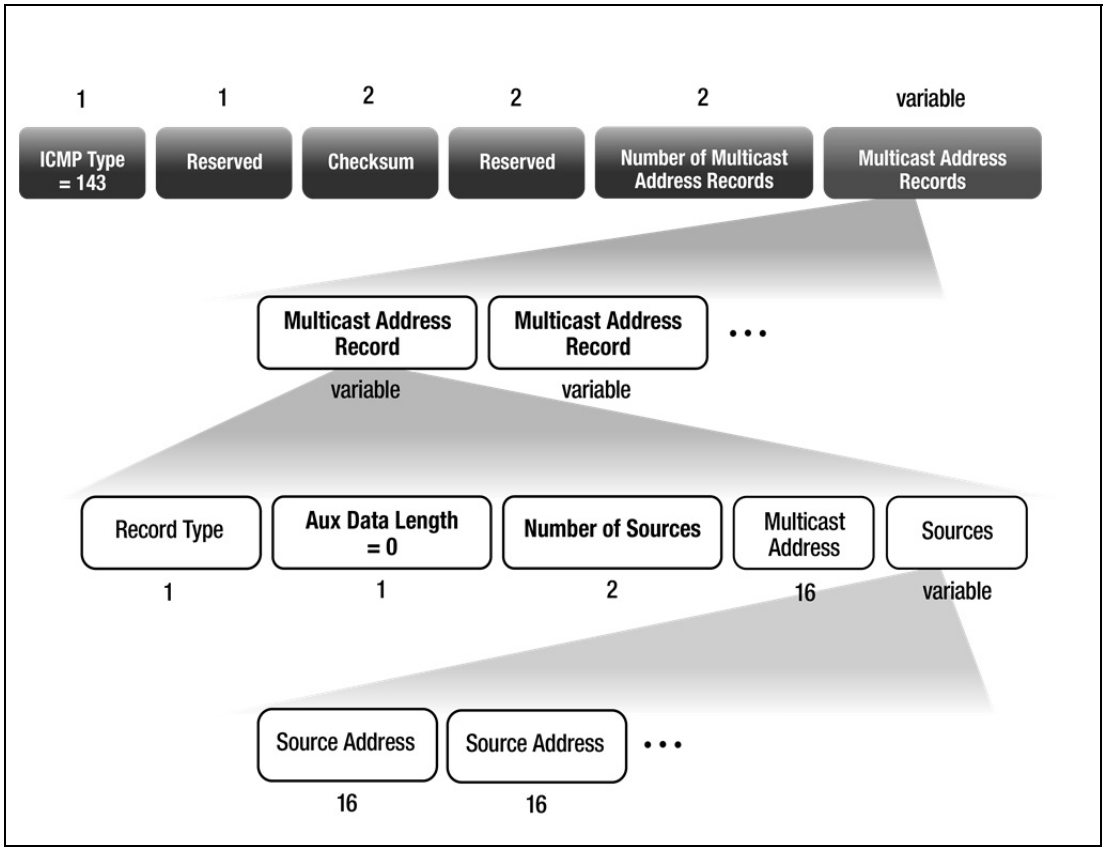
If the host is in Include mode for group G, it will look through the sources listed in the Query, and pick out all those which are in its include list. The responding Report will list all those sources it picked out. If there was no overlap between the list of sources in the Query, and the sources in the host's include list, the host does not send a Report at all.

If the host is in Exclude mode for group G, it will look through the sources listed in the Query, and pick out all those which are not in its exclude list. The responding Report will list all those sources it picked out. If all of the sources in the Query were also in the host's exclude list, the host does not send a Report at all.

Reports

The MLDv2 Report has a relatively complex structure.

Effectively the Report is a header followed by a set of *Multicast Address Records*, each of which contains information about one of the Multicast Groups the Report sender is listening to.



Each Multicast Address Record has the following internal format, let's look at the detail of each field in the packet:

MLDv2 Report field details

Type

The ICMP type of the MLDv2 Report is 143. Note that the superfluous ICMP Code field has been removed, and is now just a **Reserved** field.

Checksum

This is a standard ICMPv6 checksum.

Number of Multicast Address Records

This field indicates how many Multicast Address Records are to follow. Now let's look into the content of a Multicast Address Record.

Record Type

The Report Record Type is something that really highlights the difference between MLDv1 and MLDv2. In MLDv1, there is only one type of Report; as the only thing an MLDv1 Report can say is "*I am interested in receiving Group G.*" But, as mentioned above in the section Signaling from hosts on page 335, an MLDv2 Report can say all sorts of things about a host's relationship with a Group.

Understanding the meaning of all the possible values of the Record Type field is quite central to understanding the MLDv2 protocol.

The first thing to note is the Record Types fall into three categories:

1. **Current State Records**. These are responses to Queries, and are just reports of the current state of the host's relationship with Group G on the Interface from which it is sending the Report. I.e., it just says "*I am in Include Mode for Group G on this interface, and my list of Include sources is as follows…*" or "*I am in Exclude Mode for Group G on this interface, and my list of Exclude sources is as follows…*"

2. **Filter Mode Change Record**. When there is a change of filter mode that a host is using for selecting acceptable sources from which to receive Group G on a given interface, the host sends out a Filter Mode Change Record for G from that interface. So, these Records are saying "*I was using Include-Mode source filtering for G on this interface, but I am now changing to Exclude Mode, and here is the list of from which I now do not want to receive G…*" or vice versa.
   A Filter Mode Change Record also indicates that a host has just started listening to Group G on interface I. If a host starts listening to G, and has a list of sources from which it would like to receive G, then it send a Report containing a Filter Mode Change Record to say "*I am entering Include Mode for Group G on this Interface, and my list of Include sources is …*"

3. Source List Change Record. This sort of record is sent if the host is using Include-Mode filtering for G wants to add more sources to its Include List, or remove sources from the Include list. Similarly, it is sent if the host is using Exclude-Mode filtering for G, and wishes to change the content of its Exclude list.

Within each of these three categories, there are two record types.

Current State Record Types

MODE_IS_INCLUDE

This record states that the host is using Include-Mode Source filtering for the Group G on the Interface I. The sources listed within the record are the sources from which the host will accept G on interface I. The source list cannot be empty, as that would mean that the host will not accept G from ANY source, which effectively means that the host is not listening to Group G at all.

You might think that this is OK, as it is a way of Leaving a Group. However, the right Record Type to use for Leaving a Group is described below in the discussion of the CHANGE_TO_INCLUDE_MODE and BLOCK record types. A response to a Query (which is what the MODE_IS_INCLUDE record type is used for) is not the appropriate place to say that you want to leave a group.

MODE_IS_EXCLUDE

This record states that the host is using Exclude-Mode Source filtering for the Group G on Interface I. The sources listed within the record are the sources from which the host will not accept G on interface I. The source list could be empty, in which case the host is saying it will accept G from ANY source.

Filter Mode Change Record Types

CHANGE_TO_INCLUDE_MODE

This record states that the host has just changed from using Exclude-Mode Source filtering for G on the Interface I to Include-Mode Source filtering for G on this interface. The sources listed within the record are the sources from which the host will accept G on interface I.

If the list of sources is empty, then the host is saying that, on this interface, it no longer wishes to receive G from ANY source, i.e. this interface is leaving group G.

CHANGE_TO_EXCLUDE_MODE

This record states that the host has just changed from using Include-Mode Source filtering for G on the Interface I to Exclude-Mode Source filtering for G on this interface. Or, the host has just started listening to G on this interface, and is using Exclude-Mode source filtering for G. The sources listed within the record are the sources from which the host will not accept G on interface I.

The source list could be empty, in which case the host will accept G from ANY source.

Source List Change Record Types

ALLOW_NEW_SOURCES

If the host is using Include-Mode source filtering for Group G on Interface I, and wishes to add more sources to the Include list, then it will send an ALLOW_NEW_SOURCES record for G. The sources in the record are the list of new sources to be added. The record does not contain the full content of the Host's Include list for Group G on interface I, but just the sources that it has added to the Include List.

A special case, though, is when a host first starts listening to a Group G on an interface I, and is using Include-Mode source filtering. In this case, the host will send an ALLOW_NEW_SOURCES record, and the list of sources in the record will be the full contents of the host's Include list for Group G on Interface I.

Similarly, if the host is using Exclude-Mode source filtering for Group G on Interface I, and wishes to remove some sources from the Exclude list, then it will send an ALLOW_NEW_SOURCES record for G. The sources in the record are the list of sources it has just removed from the Exclude list.

BLOCK_OLD_SOURCES

If the host is using Include-Mode source filtering for Group G on Interface I, and wishes to remove some sources from the Include list, then it will send a BLOCK_OLD_SOURCES record for G. The sources in the record are the list of sources to be removed.

It is possible that the list of sources in the record is the full content of the host's Include list for group G on interface I. In that case, the host is effectively reporting that this interface is leaving Group G.

If the host is using Exclude-Mode source filtering for Group G on Interface I, and wishes to add sources to the Exclude list, then it will send a BLOCK_OLD_SOURCES record for G. The sources in the record are the list of sources it has just added to the Exclude list.

A host may undergo a change which means that, for a Group G on interface I, it is adding some new sources, and blocking some old ones, all at the same time. In that case, it has to send an ADD_NEW_SOURCES record and a BLOCK_OLD_SOURCES record for G.

A summary of the Record Types is provided in the table below:

Value	Category of record	Name	Abbreviated name
1	Current State	Mode_Is_Include	IS_IN
2	Current State	Mode_Is_Exclude	IS_EX
3	Filter mode change	Change_to_Include_Mode	TO_IN
4	Filter mode change	Change_to_Exclude_Mode	TO_EX
5	Source list change	Allow_New_Sources	ALLOW
6	Source list change	Block_Old_Sources	BLOCK

Auxiliary Data Length

This is the number of 32-bit words in the Auxiliary Data field that appears at the end of the record. In fact, the whole concept of Auxiliary Data is simply added for possible future extensions. As far the MLDv2 protocol itself is concerned, there is no use for the Auxiliary Data field. The value of the Auxiliary Data Length is always 0.

Number of sources

This is simply the number of IPv6 source addresses that will follow the multicast address. The number may be zero.

Multicast address

The IPv6 multicast address of the group to which this record pertains.

Sources

The list of sources relevant to this record. As described above in the section Record Type on page 341, the sources have different meanings for different record types.

Overview of the Operation of an MLDv2 Querier

Having described the way that MLDv2 hosts and queriers store information about the sources from which they wish to receive groups, and the way this information is represented in packets, let's now take a brief run through the packet exchanges that occur in MLDv2, the actions that queriers take upon receiving certain packets, and the timers that trigger other actions in the querier.

Many aspects of the operation of the MLDv2 querier are the same as those for an MLDv1 querier, so this section will concentrate mostly on the areas where the MLDv2 querier operates differently to an MLDv1 querier.

Sending General Queries

Just as with MLDv1, an MLDv2 querier sends:

- A set of General Queries at the startup of a querier interface. (The number of General Queries sent at this time is **Startup Query Count**, and they are sent **Startup Query Interval** seconds apart).
- Periodic General Queries every **Query Interval** Seconds.

Also, the mechanism for electing the active querier in MLDv2 is the same as that in MLDv1 – namely that if a querier receives, on a particular interface, a Query from a lower IPv6 source address than it is using for MLD on that interface, then it goes into non-querier mode on the interface in question.

As described above, in Record Type on page 341, the hosts respond by sending in Reports that contain Current State records (i.e. Records of Type IS_IN or IS-EX). The Reports sent in by the hosts will contain records for multiple different multicast Group addresses if the hosts are listening to multiple groups.

A difference between MLDv1and MLDv2 is that in MLDv2, the hosts do not implement the "*wait and see if someone else sends in a Report*" process described above in Reducing the number of reports received, on page 325. Instead, hosts respond to all Queries for which they have a meaningful response.

They:

- Respond to ALL General Queries if they are listening to any multicast groups.
- ALWAYS respond to Multicast-Address-Specific Queries for Groups they are listening to.
- ALWAYS respond to Multicast-Address-And-Source-Specific Queries if they are listening for that Group from any of the sources listed in the Query.

RFC3810 lists some reasons why this change was made. The gist of it is that:

- MLDv2 Reports are more complex than MLDv1 Reports, so it is not desirable for hosts to have to analyze everyone else's Report to work out if those other Reports have rendered their pending Report superfluous.
- It may be useful for the querier to get responses from ALL hosts, as the querier may wish to keep track of exactly which hosts are listening to which Groups from which sources, even though MLDv2 itself does not require queriers to do so.
- Because MLDv2 Reports can report on several groups all in one packet, there need only be one Report per host, rather than the one-Report-per-group-per-host that would have occurred with MLDv1 if MLDv1 did not implement the Report suppression mechanism.

Reacting to reports

The actions that MLDv2 queriers must take upon receiving reports are more varied than in MLDv1.

In MLDv1, it was simply:

"Someone is asking to receive group G, OK I will start forwarding that group."

Or

"Someone is saying they don't want to receive G anymore. OK, I will check if anyone else is still listening for that group. If no one responds, I will stop forwarding it."

However, as we have seen above, MLDv2 implements a number of different types of messages that a Report can convey. The querier needs to take different actions in response to these different messages.

Also, the action that the querier takes will depend on whether the querier is currently operating in Router Include mode or Router Exclude for Group G on receiving interface.

For example:

- If the querier is operating in Router Include mode for Group G on interface I, and receives a Report containing an ALLOW record for G, then it simply merges its existing Include list for G with the list of sources in the ALLOW record.
- If the querier is operating in Router Include mode for Group G on interface I, and receives a Report containing a BLOCK record for G, then it has to:
  - Check to see if there is any overlap between its current Include list for G on the receiving interface, and the list of sources in the record.

- If there is an overlap, then the set of sources in the overlap may need to be removed from its include list, so it sends out a Multicast-Address-And-Source-Specific Query for G, containing this list of may-have-to-be-removed source addresses.
- If any hosts respond to the query, the sources that appear in these responses must be removed from the list of may-have-to-be-removed sources.
- Once all the responses are in, then any sources remaining in the list of may-have-to-be-removed sources are removed from the querier's Include List for G on the interface in question.

- If the querier is operating in Router Include mode for Group G on interface I, and receives a Report containing a TO-EX record for G, then it has to:
  - Change to operating in Router Exclude Mode for G on this interface (as stated in Include and Exclude on page 333, if a querier has even one downstream host operating in Exclude mode, then the querier itself has to operate in Router Exclude mode).
  - Compare its previous Include List for G on the receiving interface to the list of sources in the To_EX record.
    - If there is any overlap between the previous include list and the list in the TO_EX record, then potentially the querier needs to still forward streams from those sources in the overlap. Even though the host sending the To_EX record has said "*I don't want to receive G from these sources*," it is quite possible that other downstream hosts do want to receive G from these sources. So, it needs to send out a Multicast-Address-And-Source-Specific Query for G, and the list of sources in the overlap. As responses are received that indicate hosts do still want to receive G from some of these sources, then the sources listed in the response are added to a 'sources from which G still needs to be forwarded' list. The querier knows that all the other sources in the overlap are sources from which it no longer needs to forward G out interface I. Any forwarding entries for sending G, from these sources, out Interface I can then be removed.
    - If the TO_EX record contains any sources that were not in the querier's previous Include List for G, then the querier can be sure it no longer needs to forward from those sources. So any forwarding entries for sending G, from these sources, out Interface I are immediately removed.
    - If the querier's previous Include List for G contained any sources that are not in the TO_EX record, then the querier knows that it no longer has to specifically keep track of these sources, as they now fall into the "*forward G from all sources that are not specifically excluded*" category.

- If the querier is in Router Exclude mode for Group G on interface I, and receives a Report containing a TO_IN record for G, then it really cannot be sure which source it should be using for forwarding G to the receiving interface. This is because:
  - Possibly only one host downstream of this interface was in Exclude mode for G, and the TO_IN record came from that host, in which case the querier needs to go back to Router Include mode for G on this Interface.

OR

  - Possibly there are still downstream hosts on this Interface that are in Exclude mode for G, in which case the querier needs to stay in Exclude mode.
  - The querier needs to just send out a Multicast-Address-Specific Query for G on this interface, and get a full picture of the relationships that the downstream hosts have to G.

In summary, there is quite a matrix of combinations of initial states, and record types that can be received in Reports; and the actions required for each of these combinations are varied and sometimes quite complex. It is beyond the scope of this document to go into all the details of all the combinations. There is a complete description in RFC3810.

Timing out forwarding entries

The process of keeping track of forwarding entries depends, as with many things in MLDv2, on whether the querier is operating in Router Include Mode or Router Exclude Mode for Group G on interface 1.

If the router is operating in Router Include mode for Group G on Interface I, then the process is relatively simple:

- There is a forwarding entry for G for each source in the Include list.
- Each of these entries has its own timer.
- The timer is refreshed every time the querier receives a report that a host wishes to receive G from the source in question.
- When the timer times out, the forwarding entry is removed.
- When the last forwarding entry for forwarding G out interface I from any source is deleted, then the querier stops tracking any state for group G.

If the router is operating in Router Exclude Mode for Group G on Interface I, then things are a bit more complicated.

The querier effectively has three categories of sources for Group G on interface I:

1. Those sources that are definitely excluded – i.e. those sources that all downstream hosts in Exclude mode have in their exclude lists, and no downstream host has in an Include list. There are no forwarding entries for G from these sources, as they are explicitly excluded. But, the querier keeps track of these sources in an Exclude list. Entries can time out of the Exclude list if Reports listing them in Exclude lists are not received for a while. When a source times out of the Exclude list, it goes into the "*all others*" category described below.

2. Those sources that are definitely requested by downstream hosts. This is referred to as the "*Requested list*." There are forwarding entries for these sources, as they have been explicitly requested. Sources will time out of this list if some time elapses without the querier receiving any Reports that have the source in an Include list. The fate of the source upon timeout depends on whether it appears in all Exclude lists received from downstream hosts.
   If is appears in ALL Exclude lists received from downstream hosts, then it moves to the "*definitely excluded*" category. Otherwise it moves to the *all others* category.

3. All others. If a source is not excluded by ALL downstream hosts that are in Exclude mode, and is not being explicitly requested by any downstream hosts in Include mode, then it becomes one of the sources from which G will be forward out interface I by default. This *by default* forwarding is due to the fact that in Exclude mode, a group is accepted from any source that is not explicitly excluded. If a stream to G appears from a source that is in the *all others* category (i.e. a source that is not in either of the other 2 categories), then a forwarding entry will be created for this source, to forward G out of interface I. This forwarding entry will not time out unless the whole forwarding of G out interface I is timed out, due to a period of the querier receiving no Reports at all that contain records for Group G.

Summary of MLDv2 Timers and Counters

Parameter	Meaning	Default Value	Relationship to other parameters	AlliedWare Plus Configuration command
Parameters that are also present in MLDv1				
Query Interval	Period at which querier sends General Queries	125 sec	Must be greater than the Query Response Interval	IPv6 MLD query-interval (per interface)
Query Response Interval	The value put into the Maximum Response Delay of General Query packets. Sets the maximum time hosts wait before sending in reports.	10 sec	Must be less than the Query Interval	Not configurable
Multicast Listener Interval	The period during which no reports are received that have records for a given group (or group-and-source combination) on that interface.	Defined by relationship to other parameters	Defined as: Robustness Variable x Query Interval + Query Response Interval	Not configurable
Other Querier Present Interval	The period, during which no Queries, with a lower source IPv6 address, are received on a given interface, before a device transitions from non-querier to Active Querier on that interface.	The RFC states that this is defined by its relationship to other parameters. AlliedWare Plus sets the default to 255 seconds.	Should be: Robustness Variable x Query Interval + (Query Response Interval / 2)	IPv6 MLD querier-timeout (per interface)
Startup Query Interval	The time between the multiple General Queries sent on a newly-active Querier interface.	Query Interval / 4	Should be: Query Interval /4	Not configurable.
Startup Query Count	Number of General Queries sent when a querier interface starts up.	Equal to Robustness Variable	Should be: The same as Robustness Variable.	Not configurable.

Parameter	Meaning	Default Value	Relationship to other parameters	AlliedWare Plus Configuration command
Lat Listener Query Count	The number of Multicast-Address-Specific Queries or Multicast-Address-And-Source-Specific Queries sent when the querier needs to work out if any downstream hosts are still listening to a given Group, or listening to that Group from one or more specified sources.	Equal to Robustness Variable	Should be: The same as Robustness Variable.	IPv6 MLD last-member-query count (per interface)
Last Listener Query Interval	The time between the sending of Multicast-Address-Specific Queries or Multicast-Address-And-Source-Specific Queries. It is also the Maximum Response Time put into those Queries.	1000 (1 second)	N/A	`IPv6 MLD last-member-query-interval` (per interface)
Robustness Variable	The number of times that certain packets are repeated, to allow for the possibility that packets will be lost.	2	Used to calculate the expected values of several other parameters, as seen above.	`ipv6 mld robustness-variable` (per interface)
Unsolicited Report Variable	When a host first wishes to receive a given group, it sends out a number (equal to the Robustness Variable) of unsolicited reports to request this group. This parameter is the time between these reports.	1 second	N/A	N/A
Parameters that are not present in MLDv1				
Older-Version Querier Present Interval	If an MLDv2 host receives MLDv1 queries, then it goes into an MLDv1 compatibility mode. This parameter is the length of time it then waits for MLDv1 queries before exiting MLDv1-compatibility mode.	Defined by relationship to other parameters	Defined as: Robustness Variable x Query Interval + Query Response Interval	N/A

Parameter	Meaning	Default Value	Relationship to other parameters	AlliedWare Plus Configuration command
Older-Version Host Present Interval	**If an MLDv2 querier receives an MLDv1 Report for a given Group G, then it goes into MLDv1 compatibility mode, just for G. This parameter is the length of time it then waits for MLDv1 Reports for G before exiting MLDv1-compatibility mode for G.**	**Defined by relationship to other parameters**		**Defined as:** **Robustness Variable x Query Interval + Query Response Interval**

Summary of Differences between MLDv1 and MLDv2

- The fundamental change between MLDv1 and MLDv2 is the fact that MLDv2 implements source filtering.

  This has flow-on effects like:

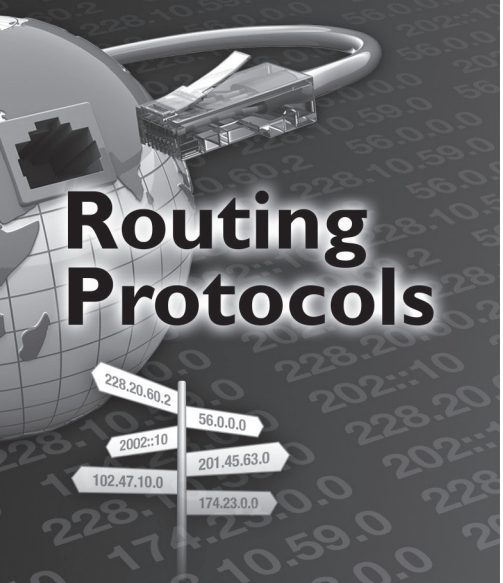
  - MLDv2 queriers keep separate forwarding entries for each source from which it is forwarding a group G, and can delete individual (S,G) forwarding entries, while retaining other entries for forwarding G from other sources.
  - MLDv2 introduced Multicast-Address-And-Source-Specific Queries.
  - MLDv2 Reports have been fundamentally altered. Rather than containing just a multicast group that is being requested, they contain Multicast Address Records that contain a set of information relating to their relationship to the Group G.
  - MLDv2 has done away with the *Listener Done* message, as the redesigned Report packet has subsumed any purpose that a *Listener Done* message would fulfill.
- Multiple Groups can be reported in the same MLDv2 Report packet.
- MLDv2 reports have a new ICMP type value, and are sent to a constant destination multicast address.
- The S flag has been introduced to the Query message, to deal with difficulties that can arise when non-queriers do not receive reports that the querier does receive.
- The querier puts the values of its Robustness Variable and Query Interval into its Queries, so that non-queriers can adopt these values, thereby enabling a smoother transition if the current querier is replaced by one of the non-queriers.

- The range of possible values of Maximum Response Times has been increased by using an algorithm that can encode a floating-point value into the Maximum Response Time field of Queries.
- MLDv2 hosts never suppress their Reports. All hosts respond to all Queries to which they have a meaningful response.

Backward compatibility to MLDv1 has been built into MLDv2.

Note - MLDv2 Multicast address records that are equivalent to an MLDv1 Report and an MLDv1 *Listener Done* are:

- An MLDv1 Report is equivalent to a CHANGE_TO_EXCLUDE_MODE Multicast Address Record with an empty source list. Such a Multicast Address Record says "*I have started listening to G on this interface, and I will accept G from ANY source*" .This is exactly what an MLDv1 Report says.
- An MLDv1 *Listener Done* is equivalent to a CHANGE_TO_INCLUDE_MODE Multicast Address Record with an empty source list. Such a Multicast Address Record says "*I have been using Exclude mode source filtering for G on this interface, but now, I am changing to Include Mode, and actually do not want to accept G from ANY sources at all.*" This has the same effect and an MLDv1 *Listener Done*.



Routing Protocols

This chapter covers the following topics:

- Summary of how the protocol operates
- Details of the PIM-DM protocol
- State Refreshes
- Interface state machines
- The Assert Process
- Generation ID
- Interpreting show command output
- PIM Dense Mode packet formats
- Packet types

CHAPTER 11

PIM Dense Mode

Summary of How the Protocol Operates

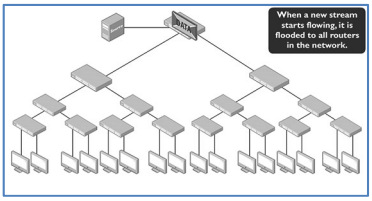
Before looking in the finer details of the PIM (Protocol Independent Multicast) Dense Mode protocol, let us get a general understanding of how the protocol operates.

New streams are flooded to all routers in the network

When a new stream starts being generated by a multicast source, the first router that the stream arrives at will forward the stream to all its neighbors, and they will fo rward it to all their neighbors, and so on.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/6iLlzpj6fJI

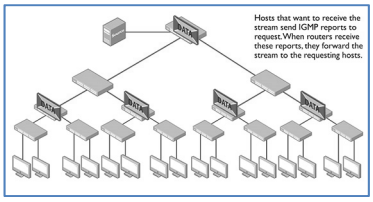


Streams are delivered to hosts that ask for them

Hosts that want to receive the stream send IGMP Reports to request it. When routers receive these Reports, they forward the stream to the requesting hosts.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/6wZGUqZIR-8



Routers that do not need to receive the stream send a Prune message upstream

If a router has no downstream neighbors, but it has received IGMP Reports requesting the stream in question, it will deliver the stream to the ports on which those IGMP Reports arrived.

If a router has no downstream neighbors, and has received no IGMP Reports requesting the stream in question, then it does not need to receive the stream. As a result, it sends a message back up to its upstream neighbor saying "*don't send me this stream any more*". This is called a **Prune** message.

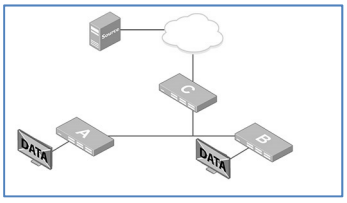
When the upstream neighbor receives the Prune message, it will wait a few seconds. If it hears no message from any other downstream neighbors, it will stop forwarding the stream out the interface on which it received the Prune message.

But, why did the upstream neighbor wait a few seconds before stopping the stream? That is to allow for the case where there is another router downstream of the interface that does want to continue receiving the stream. This other router needs to say "*don't stop that stream, I still need it*".

Because PIM packets are multicast packets sent to a reserved multicast address, the Prune packet will be sent to all other routers in the same VLAN as the pruning router. If one of these routers still wishes to receive the stream, it will see the Prune packet, and think "*hold on, if the upstream neighbor acts on this Prune, and stops sending that stream, my downstream clients are not going to be happy*". So, it sends a **Join** packet to override the other router's Prune.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/ZX-8zeZ_bP8

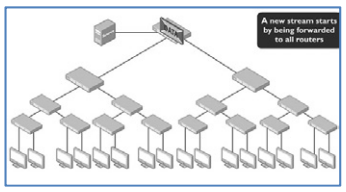


This Pruning (and the occasional overriding Joins) will continue back upstream until the stream is being sent to all the receivers that want it, and nowhere else.

At this point, the network is at a steady state, with regard ti this stream.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/XMJUc0UQwWk



State Refresh messages remind all routers of which streams, from which sources, are available

Now, what happens when a receiver requests a stream that its local router is not currently receiving?

There are two ways we could imagine the protocol handling this situation.

1. One way would be for the switch, upon receiving the request, to send out an upstream request for this stream. But, if the router has multiple upstream neighbors, which neighbor does it send the request to? If the router does not know the address of the source device for this stream, then it will not know which neighbor is the one on the path back up to the source. What the router would have to do is send the request to all its neighbors, and they would have to send it to all THEIR neighbors, and so on until the request arrived at a router that was receiving the stream being requested. This approach would work, but it would be rather untidy, and would require routers to keep track of the requests they had forwarded, so that they did not end up sending requests around in loops. Also, when the request did arrive at a router that was able to forward the stream, other routers in the network would not necessarily know this had occurred, and would waste their time continuing to unnecessarily forwarding the request on upstream on paths not directed towards the source of the stream.

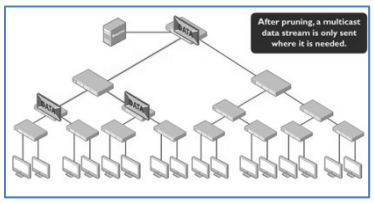
2. The second way that the protocol could deal with a router receiving a request for a stream it is not currently receiving is to make sure that all routers know about all streams that are being transmitted on the network.

   This would require that each router that was closest to the source of a given stream would send out a regular message to all the other routers in the network saying "*FYI, I am receiving a stream from source S to group G. If you ever get a request for a stream to group G, then send a request towards me*".

The method that PIM Dense Mode uses is the second method described above. These periodic messages that are sent out are called **State Refresh** messages. The name comes from the fact that initially, a new stream is always flooded to all routers in the network, and then Pruned back by those routers that do not currently need it. The periodic messages being sent out by the router nearest the source are effectively refreshing routers' memory of the fact that they had received that stream.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/WogtCSBNZOI



To summarize the general operation of the protocol:

1. A new stream is flooded to all routers in the network.
2. Those routers that do not need to receive the stream send a Prune message to their upstream neighbor, requesting that they no longer receive the stream.
3. The upstream neighbor will stop sending the stream to the pruning router, provided no other router connected to the same interface of the upstream router requests that it keep receiving the stream.
4. So that all routers in the network know which streams are being transmitted in the network, and where they are coming from, the router nearest each multicast source sends out State Refresh messages to inform all routers of the presence of a given stream, and where its source is.
5. When a router receives a request for a stream it is not currently receiving, it will send a request in the direction towards the source of the stream.

Now let's work through all this in more detail.

Details of the PIM-DM Protocol

Let us go through each aspect of the protocol, to understand its operation in more detail

Neighbor discovery

The first thing that a router running PIM DM needs to do when its PIM interfaces come up is establish relationships with its neighbors.

As is not uncommon, PIM's neighbor discovery process involves the exchanging of Hello packets. PIM routers constantly send out Hello packets on all their active PIM interfaces at a regular interval. The default value for this Hello Interval is 30 seconds.

The Hello Interval can be altered by using the command:

```
ip pim hello-interval
```

However, it is advisable not to change the Hello Interval, as there is little benefit in doing so.

The process of neighbor discovery is very simple. Once a PIM router starts receiving Hello packets from another router, then the receiving router adds that other router to its list of neighbors. It is as simple as that, no negotiation of a neighbor relationship is necessary; simply being aware of the other router's existence is sufficient for it to become a neighbor.

Losing a neighbor

As long as Hello packets continue to be received from another router, that router continues to be deemed a neighbor.

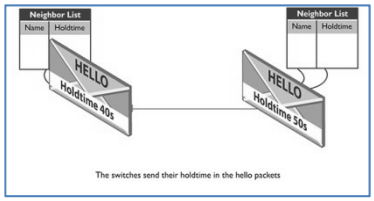
If a router stops receiving Hellos from a neighbor, at what point does it drop that neighbor out of its neighbor list?

In fact, the method that PIM uses to determine how long to wait before deciding a neighbor is no longer a neighbor is quite interesting. When a router sends out Hello packets, there is a field in the packets called the Hold Time. This hold time is the period that the receiving router should wait before deciding that the sending router is no longer a neighbor if the sending router stops sending Hellos.

It is the sending router that defines how long the receiving router should wait until un-neighboring the sending router. Rather than the receiving router deciding how long it should wait before removing the sending router from its neighbor list, it is actually the sending router that tells the receiving router how long to wait.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/c1Tg2KX8hkI



Pruning and Grafting

The two key activities in the PIM DM protocol, are asking to not receive unwanted streams (**pruning**), and asking to receive wanted streams (**grafting**).

Pruning

A router sends Prunes on a per-stream basis. A Prune packet specifically says *"I do not wish to receive the stream (S,G)"*. The Prune packet is sent out the Reverse Path Forwarding (RPF) interface on which the stream (S,G) is accepted.

There are two likely actions that will occur in response to a Prune packet.

1. The upstream neighbor that is the next hop on the path back to the source S will receive the Prune, and act on it.

2. Any other routers connected to the same segment as this RPF interface, and which actually want to continue receiving the stream (S,G), will see the Prune, and override it with an *"actually, I want to continue receiving that stream"* Join packet.

A Prune is sent in the following circumstances:

- A packet in stream (S,G) is received in the RPF interface, and the router has no downstream neighbors, or directly connected receivers, who have recently requested (S,G). Having sent this Prune, the router should not send another Prune for this same (S,G) for a while. The default value for this waiting period (known as the **Prune Limit Timer**) is 210 seconds.
  The reason for this wait is that if another router on the same segment does override the Prune, then the router will continue to receive the stream (S,G) on its RPF interface. If it sent a Prune every time it received a packet in the stream, it would end up sending an awful lot of Prunes. So, the idea is that it should just shut up about this stream for a while, and have another go at pruning it later on. Of course, if other routers on the same segment as the RPF interface do decide, in the meantime, that they do not want to receive (S,G), they will Prune it instead. It is always possible that the stream will stop before the 210 seconds are up anyway.

- A State Refresh packet for (S,G) is received on the RPF interface, and the **Prune Indicator** bit in the State Refresh is set to 0, and the router has no downstream neighbors, or directly connected receivers, who have recently requested (S,G).

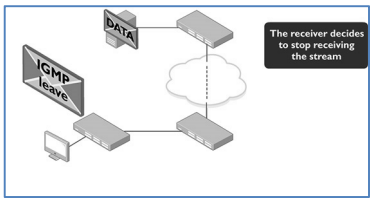
  The *Prune Indicator* bit in the State Refresh is an indicator of the upstream router's belief as to whether it should be sending the stream (S,G) downstream on the interface that faces the router in question. If the *Prune Indicator* is set to 0, then that indicates that the upstream router believes it should be sending the stream (S,G) out through that interface (i.e. (S,G) is **not** pruned on that interface). If the *Prune Indicator* is set to 1, then that indicates that the upstream router believes it should not be sending the stream (S,G) out through that interface (i.e. (S,G) **is** pruned on that interface).

- The router has been forwarding the stream (S,G) to downstream neighbors or to directly connected receiving hosts, and the last downstream receiver indicates it no longer wishes to receive the stream. Given that the router now has no good reason to receive (S,G), it sends a Prune upstream to request that it no longer receive the stream. Again, if it continues to receive the stream, it must wait for the **Prune Limit Timer** period before sending another Prune for this (S,G).

- If a topology change of some sort occurs such that there is a change in the unicast route back to the source S of the stream (S,G), it is possible that the RPF interface for this (S,G) can change – i.e. that the unicast route to S is now out another interface. In this case, if the router has no downstream neighbors, or directly connected receivers, who have recently requested (S,G), then it should send a Prune to the new RPF neighbor, to pre-emptively inform this neighbor that it has no desire to receive the stream (S,G).

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/snMKk4dwyY4



Grafting

Once a router has gone and pruned itself off the distribution tree for a given (S,G), it needs a way of connecting itself back onto that tree when it finds that someone downstream of it wants to receive that stream. This rejoining to the tree is achieved by sending a **Graft** message.

When a router sends a Graft message, it expects a Graft-Acknowledgement (**Graft-Ack**) in response, to be sure that the upstream neighbor actually received the Graft. Most of the time, of course, the arrival of the requested stream is acknowledgement enough of the Graft

message. However, there will be times when the stream is paused, or just very intermittent, and will not start arriving quickly. To cover these cases, the Graft-Ack is the mechanism that ensures the sender of the Graft can see that the Graft has been received by the upstream neighbor.

The router waits 3 seconds to receive the Graft-Ack, before retrying the Graft. It will continue retrying until it gets the Graft-Ack.

In fact, a State Refresh message, with *Prune Indicator* set to 0, can suffice as an acknowledgement of a Graft. If the State Refresh, coming from the upstream router, for stream (S,G) has its Prune Indicator set to 0, that demonstrates that the upstream router knows it needs to forward (S,G) down towards us. Having seen this State Refresh message, there is no longer any need to hang round waiting for a Graft-Ack, it is clear that the upstream router has got the message.

A Graft is sent in the following circumstances:

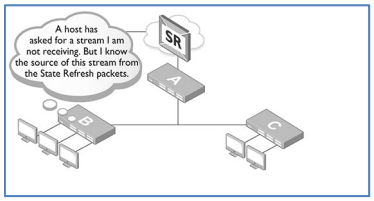
1. The router receives a request, from the downstream side, for a stream (S,G), and the router knows that the stream is being transmitted in the network (as it is receiving State Refresh packets for this (S,G)). And the router is not currently receiving that stream – i.e. it has actively pruned itself from the distribution tree for this (S,G).

   Note – if the router receives a request for a group that it is not receiving State Refreshes for, then it will not send a Graft anywhere to request this stream. The main reason being, that it will not know the source address for the stream, and so does not know where to send the Graft. In this case, it simply has to drop the request that came from the downstream device.

2. If a topology change of some sort occurs such that there is a change in the unicast route back to the source S of the stream (S,G), it is possible that the RPF interface for this (S,G) can change – i.e. that the unicast route to S is now out another interface. In this case, if the router has downstream neighbors, or directly connected receivers, who it knows want to receive (S,G), then it should send a Graft to the new RPF neighbor, so it can continue receiving (S,G) and sending it on downstream.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/-S1wHs9RvCU



State Refreshes

Let us take a look at the State Refresh mechanism.

Originating State Refresh messages

State Refresh messages are generated on a per (S,G) basis. For every stream (S,G) that is being transmitted into the network, there should also be State Refresh messages being generated.

The messages are initiated by the router(s) that are closest to the source of the stream, and are sent right through the network. Actually, it is possible that the State Refresh messages will not reach every PIM router in the network. The Time To Live (TTL) of the State Refresh messages is set to the same value as the TTL of the stream they represent, and the TTL is decremented at each router that forwards the State Refresh messages. If the TTL expires before the State Refresh messages arrive at the most distant routers, then those distant routers remain unaware of the existence of the stream (S,G). This is as it should be, because that stream will not be able to reach those distant routers before its TTL expires.

If a router receives packets in a stream (S,G) on one of its interfaces, and the source address, S, is in the same subnet as the router's IP address on the receiving interface, then the router knows it is a "*First Hop*" router for that stream, and therefore responsible for initiating the State Refresh messages for that stream.

It sends the State Refresh to all downstream interfaces on which it is not an Assert Loser (the concept of Assert Loser will be discussed later on in section The Assert Process on page 369).

The originator sends a State Refresh for a given group (S,G) once per minute, by default. If the originator has not received any packets in the stream (S,G) for 210 seconds, then it stops generating State Refresh packets for (S,G).

Receiving State Refresh Messages

When a State Refresh message arrives at a router, the router makes some checks before it decides whether it can accept this packet. It checks:

- Has the packet arrived on my RPF interface for the source S of the (S,G) that this State Refresh relates to?
- Has this State Refresh not arrived too soon after the previous State Refresh I received for this (S,G)?
- Is the TTL in the State Refresh packet above the acceptance threshold (which may not necessarily be 1)?

If the packet passes all of these checks, then it is accepted, and the router will perform various actions.

Most of all, the router receiving the State Refresh reconfirms to itself that the stream (S,G) that the State Refresh refers to is still available in the network.

Each router keeps a list of all the (S,G) streams that it is currently receiving, or for which it is currently receiving State Refreshes. If it goes for 3 minutes without receiving either stream content packets or State Refreshes for a given (S,G), it decides that stream is no longer being sent, and times it out of its list of known active streams.

The router can also take the opportunity to correct any mistakes on the part of the upstream neighbor.

If the router has downstream clients to whom is wants to be forwarding the stream (S,G), and it sees that the Prune Indicator in the State Refresh packet is set to 1, then it needs to inform the upstream neighbor who sent the State Refresh that it is sorely mistaken. As discussed below in Forwarding State Refresh messages on page 365, a Prune Indicator value of 1 indicates that the sender of the State Refresh believes there are NO receivers of the (S,G) downstream of the interface out which is sent the State Refresh. But, if the receiving router knows that it has downstream receivers for (S,G), then these receivers must be downstream of the interface that the State Refresh message came from. So, that upstream router is mistaken.

Similarly, if the router has no downstream clients to whom it wants to be forwarding the stream (S,G), and it sees that the Prune Indicator in the State Refresh packet is set to 0, then that is also cause to have to inform the upstream neighbor that it is in error. A Prune Indicator value of 0 indicates that the sender of the State Refresh believes there are receivers of the (S,G) downstream of the interface out which is sent the State Refresh. But, if the receiving router knows that it has no downstream receivers for (S,G), then it needs to inform the upstream neighbor that, as far as it knows, the Prune Indicator value should be 1. It does this by sending a Prune message, for (S,G), to the upstream neighbor. Of course, it may well turn out that the reason that the Upstream Neighbor is setting the Prune indicator to 0 is because other routers on that same subnet are forwarding (S,G); in which case, one of those routers will need to send a Join that overrides the Prune that has just been sent.

If the router has downstream PIM DM neighbors, and the TTL of the State Refresh has not been exhausted, then it will forward the State Refresh to those downstream neighbors.

Forwarding State Refresh messages

When a router accepts a State Refresh on its upstream interface (the RPF interface for the stream (S,G) that the State Refresh refers to), then it needs to forward this State Refresh to its downstream interfaces.

It does not simply forward the State Refresh on exactly as it was when it received it. It needs to update a few things in the packet before forwarding it.

- The source address of the State Refresh is set to the router's IP address on the interface that the State Refresh is being transmitted through. If the router is forwarding the State Refresh to multiple downstream interfaces, the packets sent out through different interfaces will have different source IP addresses.
- The Prune Indicator bit in the packet will be set to reflect whether or not the router believes it should currently be forwarding (S,G) out the interface in question. If the router believes it has no downstream listeners for (S,G) on a given interface (i.e., everything downstream if this interface is Pruned off the distribution tree for (S,G)), then the Prune Indicator is set to 1. Conversely, if the router believes that it still has listeners for (S,G) downstream of this interface (i.e. there are links from this interface that are not pruned off the distribution tree for (S,G)), then the Prune Indicator is set to 0. So, again, the Prune Indicator value set in the packet will vary depending of the Prune state of the interface it is being transmitted through.
- The TTL of the State Refresh needs to be decremented by 1.
- The State Refresh also contains the Metric, Preference, and Mask of the unicast route to the source, S, of (S,G). The router updates all these fields to the relevant values for its unicast route back to S.

Interface State Machines

At the heart of the PIM implementation in a router are ***state machines*** that keep track of what Prune/Graft/Join/Ack messages an interface has received, and when it should be sending Prune/Graft/Join/Ack messages.

You may think that state machines are not a suitable topic of discussion in a document on using and debugging PIM. This sort of thing sounds more like the territory of programmers than users.

To a certain extent, you would be right. However, an understanding of how a router models its handling of multicast streams, and how RFCs describe the handling of streams, makes it much easier to understand the output of the PIM show commands in AlliedWare Plus, and to understand how and why that output changes in reaction to given events in the network. Also, the state machine is not actually the difficult concept that it may sounds, and is a convenient framework within which to discuss how the protocol actually operates.

Let's get an understanding of the state machines that are involved in PIM Dense Mode.

Upstream interface state machine

For any given stream (S,G), a router will have an RPF interface—the one interface via which it will accept this stream. That interface is also referred to as the upstream interface for this (S,G). To make it easy to understand how the router should react to events that occur in relation to this upstream interface, there is a per-(S,G) state machine for the RPF interface, that consists of:

- Three possible PIM-DM states that the interface can be in
- A defined set of events that can cause it to transition between states
- Defined actions it should perform when in any given state

The three states are: Forwarding, Pruned, and ACK Pending:

Forwarding

Having a state called "*Forwarding*" defined on the upstream interface may seem a little odd. This state does not mean that (S,G) is being forwarded **out** the RPF interface; instead it means that **if** the stream (S,G) were to arrive on the RFP interface, it will be forwarded to at least one other interface.

Also, Forwarding is the default initial state that the router applies to the (S,G) when it first receives the stream, while it is working out whether it should actually forward it or not.

Pruned

If the router knows that it is does not have any downstream receivers to which it needs to forward (S,G), then it sends a Prune message to its upstream neighbor, and puts the RPF interface for (S,G) into the Pruned state.

Ack Pending

This is the "*in-between*" state, just after the router has realized it now needs to be forwarding (S,G). It has sent off the Graft to request (S,G) from the upstream neighbor, but has not yet received that Ack to that Graft. Typically, the interface should not stay in this state for long.

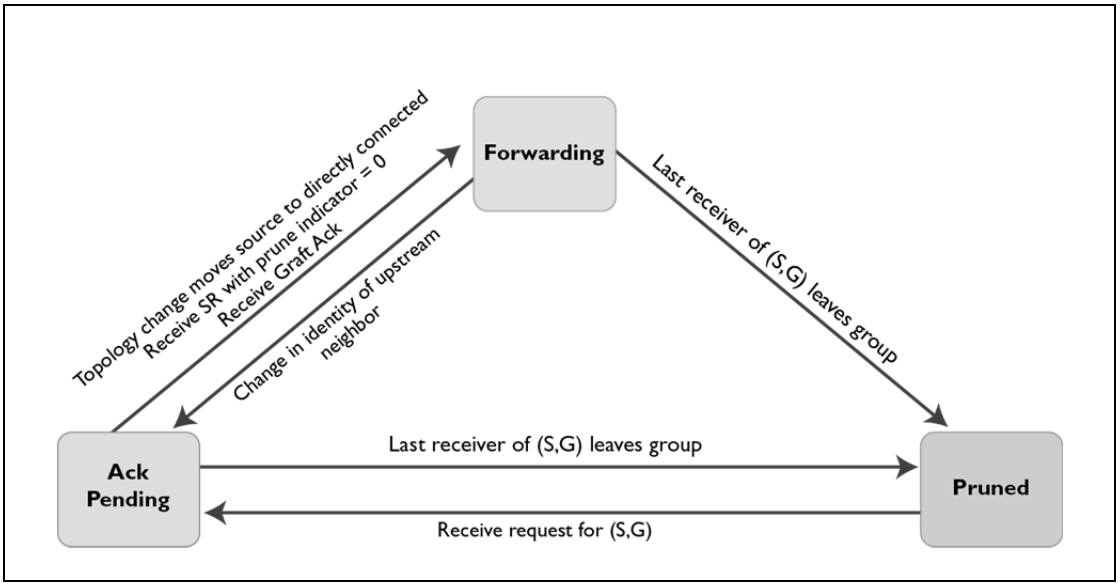
Transition Events

The events that will cause the interface to transition from one state to another are:

Event	State Transition	Action
Last downstream receiver of (S,G) leaves the group	F -> P	Send Prune
A downstream receiver requests a stream we know of, but are not currently forwarding	P -> AP	Send Graft
A topology change affects which router is the upstream neighbor on the path back to the source of an (S,G) we are forwarding	F->AP	Send Graft to the new upstream neighbor
Receive a Graft Ack	AP->F	
Receive State Refresh from upstream neighbor, with Prune Indicator is 0	AP->F	
A topology change causes a change such that the source of an (S,G) we are forwarding is now on a directly connected VLAN	AP->F	

In addition, RFC 3973 details a number of timer-related actions that the router should perform when certain events occur when it is in certain states, but we will not into that level of detail here.

Graphically, the transition events above can be represented as:



Downstream interface state machine

A downstream interface (any interface that is not directed towards the source, S, of stream (S,G)), again has three states defined, but they are not exactly the same as the states defined for the upstream interface:

Pruned (P)

A Prune has been received for (S,G) on this interface, and no overriding Join has been received. So, (S,G) is not being forwarded out through this interface.

Prune-Pending (PP)

A Prune has been received for (S,G) on this interface, but the router is still waiting to see if an overriding Join will be received, so (S,G) is still being forwarded out through this interface.

NoInfo (NI)

There have been no Prunes for (S,G) received on this interface for the period of the Prune Timeout (sent in the Prune packet) OR a Graft or Join has been received since the most recent Prune.

Now, this does not necessarily mean that the router is forwarding (S,G) out through this interface. It could well be that there are no downstream neighbors, and no directly connected receivers, on this interface, in which case the router would have sent a Prune for (S,G) to the upstream neighbor, as it has no need to receive (S,G).

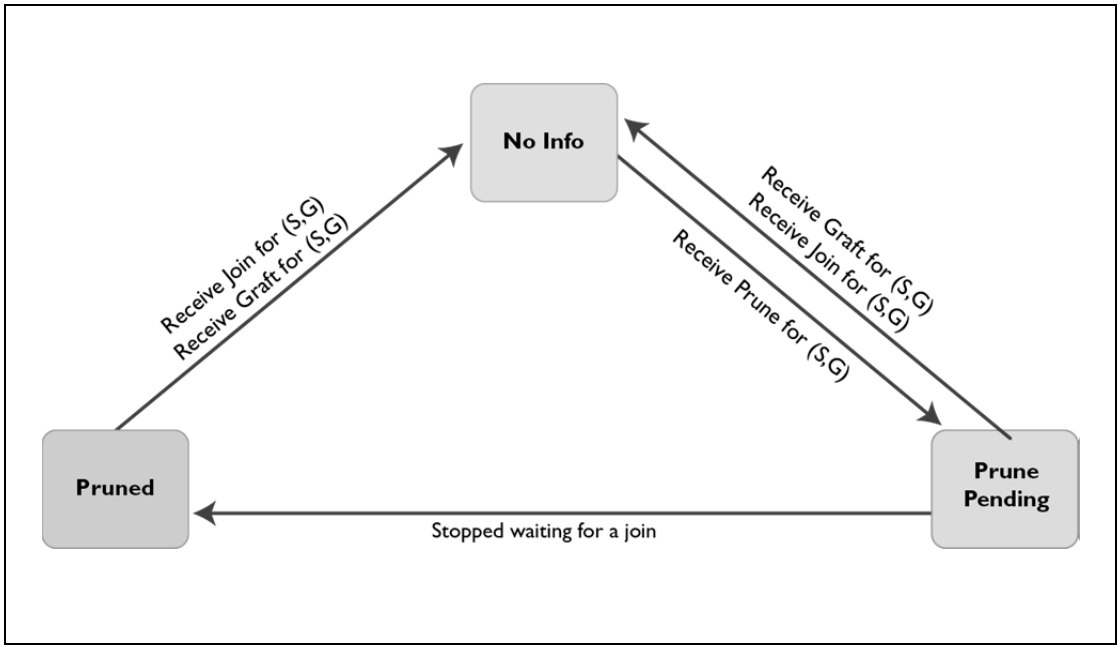
The States do not entirely reflect what the router is doing with the stream (S,G); rather they reflect the history of the receiving of Prunes/Grafts/Acks etc for (S,G) on that interface.

Transition Events

The events that will cause the interface to transition from one state to another are:

Event	State transition	Action
Receive a Prune for (S,G)	NI ->PP	
Receive a Join for (S,G)	P -> NI PP -> NI	
Receive a Graft for (S,G)	P -> NI PP -> NI	Send Graft Ack
Timeout the wait for a Join	PP->P	
A topology change causes this interface to become the RPF interface for (S,G)	P->NI PP->NI	

Graphically, this can be represented as:

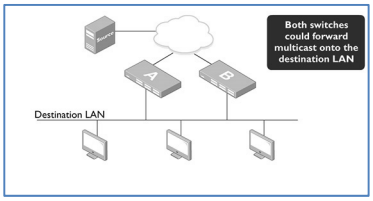


The Assert Process

If two or more routers are connected to the same LAN, and they are each receiving a given stream (S,G), then potentially they could all deliver that stream onto their common LAN.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/TkrxGZvTJiw



However, that would cause problems, as multiple copies of the stream would arrive onto the LAN, and hosts on the LAN, and further downstream, would need to pick out just one copy of each packet.

Also, it is a waste of bandwidth – which is rather contrary to the spirit of multicasting.

Rather than cause this multi-copy mess, the routers need to decide amongst themselves who will forward the stream onto the LAN, and who will butt out.

And, indeed, PIM routers **do** decide among themselves who will get the honor of forwarding the stream (S,G) onto a shared LAN. This decision process is referred to as the "*Assert*"

process. In short, the routers each send Assert messages onto the shared LAN, putting forward their credentials for the role of "*Forwarder of (S,G)*". The router with the best credentials wins.

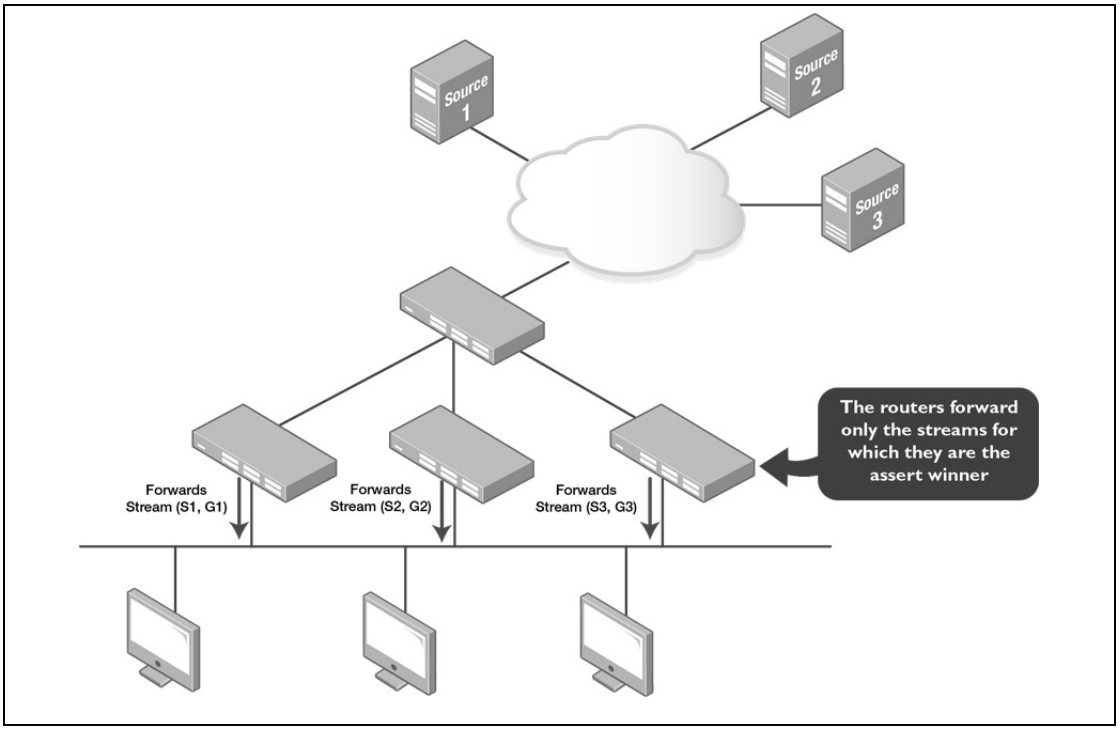
Any given router enters the Assert bake-off when it receives either a data packet for the (S,G) stream, or another router's Assert, on a downstream interface.

In response to receiving that packet on the downstream interface, the router sends out its own Assert that contains the Administrative Distance and Metric of its route to the source, S, of (S,G).

The router whose route to S has the best (lowest) Administrative Distance wins.

If two or more of the routers have routes to S with equal lowest Administrative Distance, the router whose route has the lowest metric wins. If there is still no clear winner, the final tie breaker is that the router whose interface onto the shared LAN has the highest IP address wins.

The Assert Winner then continues to forward (S,G) onto this LAN (if there are downstream receivers requesting this stream), and the other routers stop forwarding that stream onto the LAN.



Assert State Machine

As with the sending and receiving of Grafts, Prunes, and Joins, the Assert process is quite well described by a state machine.

The Assert State Machine for a given stream (S,G) on a given interface I has three states:

1. Winner (W)
2. Loser (L)
3. No Info (NI)

By default, an interface is in the NoInfo state. But, as soon as it receives a packet from the stream (S,G) and it is evident that this interface is not the RPF interface to S, then the interface transitions to the Assert Winner state, and kicks off the Assert negotiation.

The Assert negotiation involves sending out Assert messages, to establish who has the best credentials to be the one that forwards (S,G) onto the shared LAN.

If a router receives an Assert that has better credentials than its own, it will transition to the Assert Loser state.

Also, State Refresh messages include info on a router's route to the source of (S,G). If a router receives a State Refresh for (S,G) on the downstream interface, and it is evident that the sender of the State Refresh has a better route back to S, then that is also a reason for the router to transition to the Assert Loser state.

Once a router is in the Loser state for a given (S,G) on interface I, it cannot transition straight to the Winner state.

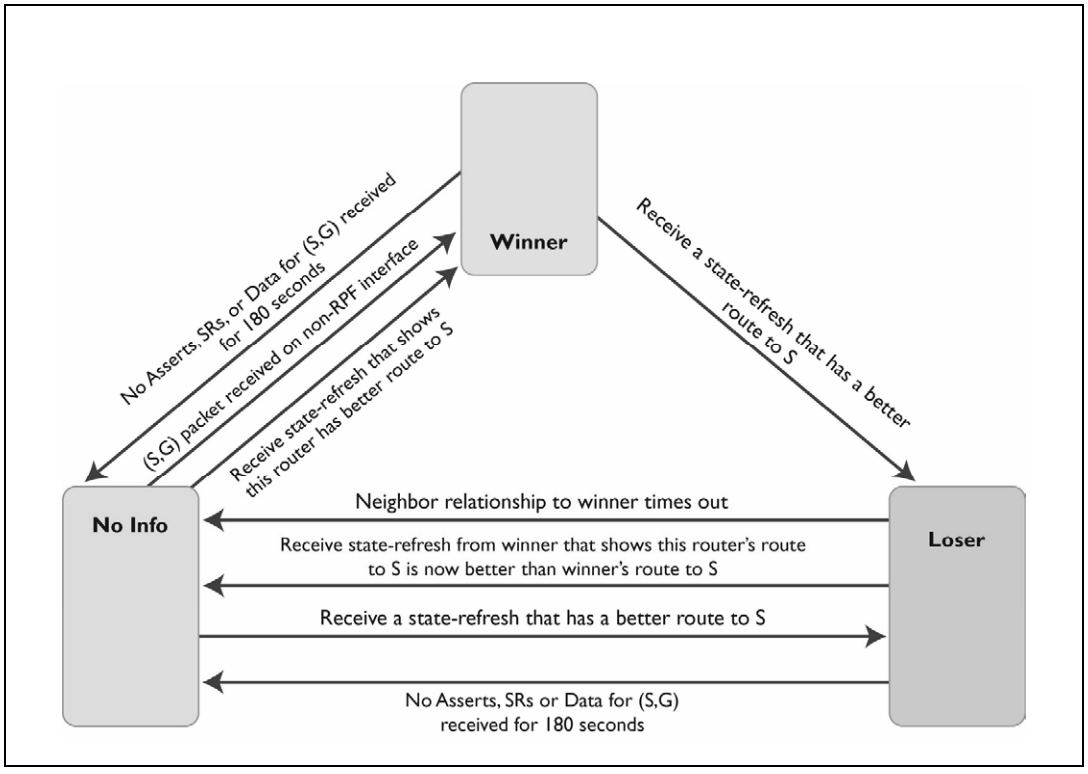
It can transition to the NoInfo state if:

- Its neighbor relationship with the winner times out.
- It receives an Assert or State Refresh from the winner which shows the winner to now have a worse route back to S.
- It does not see any Asserts or State Refreshes for (S,G) on interface I for 180 seconds.

The transition events are:

Event	State transition	Action
A data packet in stream (S,G) received on interface that is not RPF to S	NI->W	Send Assert packet
Receive an Assert or State Refresh from Winner that makes it evident that the Winner's route to S is now worse than router's own route to S	L->NI	
Receive an Assert or State Refresh from another router that makes it evident that the router's own route to S is better than the sender's route to S	NI->W	Send Assert packet
Receive an Assert or State Refresh from another router that has a better route to S than the current router does	W->L NI->L	Send Prune packet
No Assert, State Refresh or Data packet for (S,G) received for 180 seconds	W->NI L->NI	
Neighbor relationship with Winner times out	L→NI	

Graphically, the state machine can be represented as below.



Generation ID

The prime purpose of PIM signaling is to enable multicast forwarding trees to be created. For a forwarding tree for a given stream to be effective, all the routers in the path need to agree that they are forwarding the stream in question. And, conversely, for streams to not be forwarded unnecessarily, routers need to remember which of their downstream interfaces have been pruned off given trees.

This all works fine as long as all the routers in the network are consistently retaining the information they have learnt from the PIM packets they have received from their neighbors. But, what happens when a router reboots? Invariably, PIM forwarding state information is not stored in non-volatile memory. So, when a router comes up again after reboot, it will have no idea what streams it should or should not be forwarding, or even which streams are even present in the network.

Eventually, of course, the router will relearn all this information, as neighbors' regular PIM signaling packets (State Refreshes, regular Prune Updates, etc) arrive. However, relying on regular Updates to rebuild a router's picture of what it should be doing can take minutes. It is preferable if this process can be sped up.

And, certainly, it can be sped up. If neighbors know that a router has just rebooted, they can decide to send signaling packets immediately, instead of waiting for the next scheduled update. But, how does a router say to its neighbors "*I have just rebooted*"? Well, that is where the concept of Generation ID comes in.

The Generation ID is a random value that is created on a per-interface basis, at the startup of the PIM processing on that interface. As it is a random value, it should be different each time it is created. Then, all Hello packets sent out through the interface contain this value in their GenerationID field.

If a router sees a sudden change in the Generation ID in the Hellos arriving from one its neighbors, it can conclude that the PIM process on this neighbor has just restarted, and so can guess that this neighbor has completely lost any memory which streams exist in the network, and of which streams it should or should not be forwarding. At this point the router can take the opportunity to send unscheduled PIM signaling packets to this neighbor.

Specifically if the restarting neighbor is:

- Downstream in any (S,G) tree, then the current router should send State Refresh packets to the just-restarted neighbor for that (S,G) tree. That way the just-restarted neighbor learns which streams are present in the network.
- Upstream in any (S,G) tree, and the current router is pruned from that tree, then it should quickly send a Prune up to the just-restarted neighbor, to quickly inform the just-restarted neighbor to not send it that stream.

Interpreting Show Command Output

The command that shows you the content of the PIM Dense mode tree is **show IP PIM dense mroute**. This command will output a set of information for each (S,G) entry currently in the tree. Let us look at a couple of representative entries, and explain the meaning of the various pieces of information that are output for the entry.

The first example is from a router that is the first-hop router for a given stream:

```
(45.231.4.40, 239.100.10.3)
  Source directly connected on vlan1
  State-Refresh Originator State: Originator
  Upstream IF: vlan1
    Upstream State: Forwarding
    Assert State: NoInfo
  Downstream IF List:
    vlan2, in 'olist':
      Downstream State: NoInfo
      Assert State: NoInfo
    vlan4:
      Downstream State: Pruned
      Assert State: NoInfo
```

Let's work through this output line-by-line.

1 `(45.231.4.40, 239.100.10.3)`
This is simply the (S,G) that identifies the stream. This entry relates to the stream from 45.231.4.40, destined to the group address 239.100.10.3.

2 `Source directly connected on vlan1`
This identifies the fact that the current router is the first-hop router for this stream. There is no other router between the source (45.231.4.40) and the VLAN1 interface of the current router.

3 `State-Refresh Originator State: Originator`
As described in the section Originating State Refresh messages on page 363, the first-hop router has the job of originating the State Refresh messages that are sent through the network to inform all routers that this stream is being produced.

4 `Upstream IF: vlan1`
VLAN1 is the interface that is connected to the source of the stream, so it is the upstream interface for this stream.

5 `Upstream State: Forwarding`
As described in the section Upstream interface state machine on page 366 the

"*Forwarding*" state on an upstream interface means that this stream is being forwarded to at least one downstream interface – i.e. not all downstream interfaces are in the Pruned state with regard to this stream.

6 `Assert State: NoInfo`
On an upstream interface, the Assert state will invariably be NoInfo, as there should never be any exchange of Assert messages on an upstream interface.

7 `Downstream IF List:`
The downstream interface list will contain all the PIM interfaces on the router except the upstream interface for this stream. Irrespective of whether or not the stream is being forwarded to one of these interfaces, they are in the downstream interfaces list as they are all interfaces to which the stream COULD be forwarded.

8 `vlan2, in 'olist':`
The fact that VLAN2 is in the olist means that VLAN2 is an interface that the stream is being forwarded to.

9 `Downstream State: NoInfo`
As described in the section Downstream interface state machine on page 368, NoInfo state on a downstream interface effectively means "*Not pruned*" – i.e. either it has not received any Prunes for a good while, or it has received an explicit Graft or Prune override. So, the rather non-descript looking term "*NoInfo*" really means "*The stream is being forwarded out through this interface, unless it is an Assert Loser*".

10 `Assert State: NoInfo`
On a downstream interface, as described in the section Assert State Machine on page 371, an Assert state of NoInfo typically means that it has received no Assert messages or other packets (State Refreshes or packets in the stream) that might indicate that another router is forwarding this stream into the same LAN that this downstream interface is connected to. The Assert state of NoInfo, could mean that there is currently a change of Assert winner going on in this LAN. If true, the interface will not remain in the NoInfo state for long.

11 `vlan4:`
vlan4 is also a downstream interface for this stream.

12 `Downstream State: Pruned`
This interface is receiving regular Prune messages for this stream from one or more downstream routers, and the Prunes are not being overridden. That means that there are no listeners downstream of this interface who wish to receive this stream. So, the stream is not being forwarded out of this interface.

13 `Assert State: NoInfo`
Just the same as line 10 above. The Assert state of an interface is quite independent of its Prune/Graft state.

The next example illustrates an entry in the case that a router is at least one hop away from the source of the stream:

```
(10.10.10.10, 239.1.1.1)
  RPF Neighbor: 192.168.1.2, Nexthop: 192.168.1.2, vlan1
  Upstream IF: vlan1
    Upstream State: Pruned
    Assert State: NoInfo
  Downstream IF List: empty
```

Again, let's work through this output line-by-line.

1 `(10.10.10.10, 239.1.1.1)`
This time the entry relates to the stream from 10.10.10.10, destined to the group address 239.1.1.1.

2 `RPF Neighbor: 192.168.1.2, Nexthop: 192.168.1.2, vlan1`
The RPF neighbor is the first router on the path back towards the source of the stream.

The next hop is almost always the same as the RPF neighbor address. It is possible for the next hop address to differ from the RPF neighbor address if the interface on the RPF neighbor router is multihomed.
In this case, if the RPF neighbor router is using one address on that interface as the source of its PIM packets, but the current router considers a different address on that interface to be the next hop on the route back to source S, then the RPF neighbor address will differ from the next hop address.

vlan1 is the egress interface towards the source.

3 `Upstream IF: vlan1`
Further confirmation that vlan1 is the interface facing towards the source.

4 `Upstream State: Pruned`
As described in the section Upstream interface state machine on page 366, the "*Pruned*" state on an upstream interface means that there are no devices downstream of the router that are listening for this stream, so is sending Prunes upstream to tell the upstream neighbor that it no longer wishes to receive the stream.

5 `Assert State: NoInfo`
Just as for the other example, the NoInfo state in the Assert State machine on an upstream interface is the expected state.

6 `Downstream IF List: empty`

PIM Dense Mode Packet Formats

PIM has its own IP protocol type – protocol number 103. All PIM packets have IP protocol number 103.

Most PIM packets are sent to the multicast address 224.0.0.13.

The exceptions to this are Graft and Graft Ack messages. Grafts are sent to the unicast address of the upstream neighbor on the RPF path toward to source of the stream (S,G). Graft Ack messages are sent to the unicast address of the sender of the Graft.

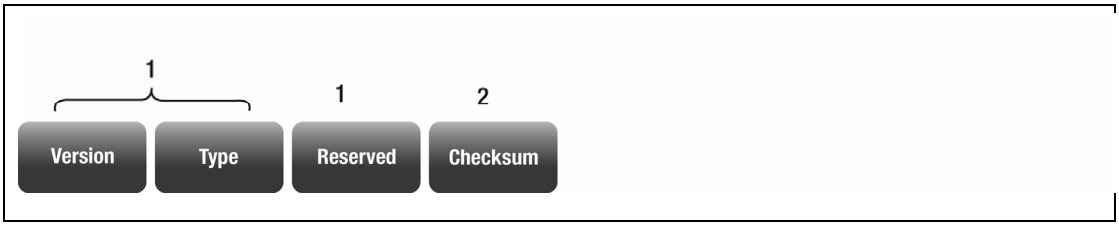
There are six packet types used in PIM Dense Mode:

1. Hello packets
2. Join/Prune Packets
3. Graft Packets
4. Graft Ack Packets
5. Assert Packets
6. State Refresh packets

Let us look at the format of each of these packet types.

PIM header

First, we need to look at the PIM header, which is at the start of all PIM packets.

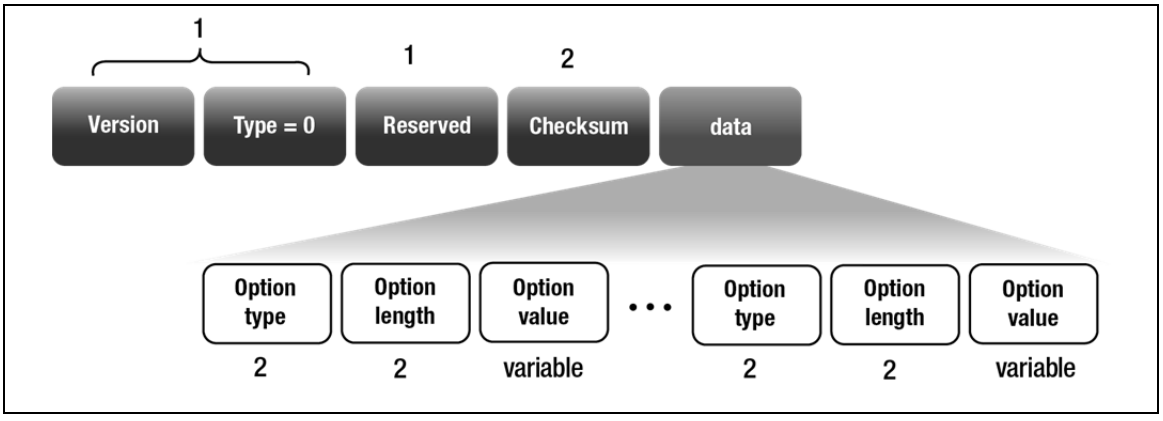


The fields in the header are:

Fields in Header	Description
PIM version	The current version number of PIM is 2
Type	The Type values for the different packet types are: 0 = Hello 3 = Join/Prune 5 = Assert 6 = Graft 7 = Graft Ack 9 = State Refresh
Checksum	The checksum is calculated over the entire PIM message

Hello packet format

Really, a Hello packet is just a PIM header followed by one or more option fields.



The important options are:

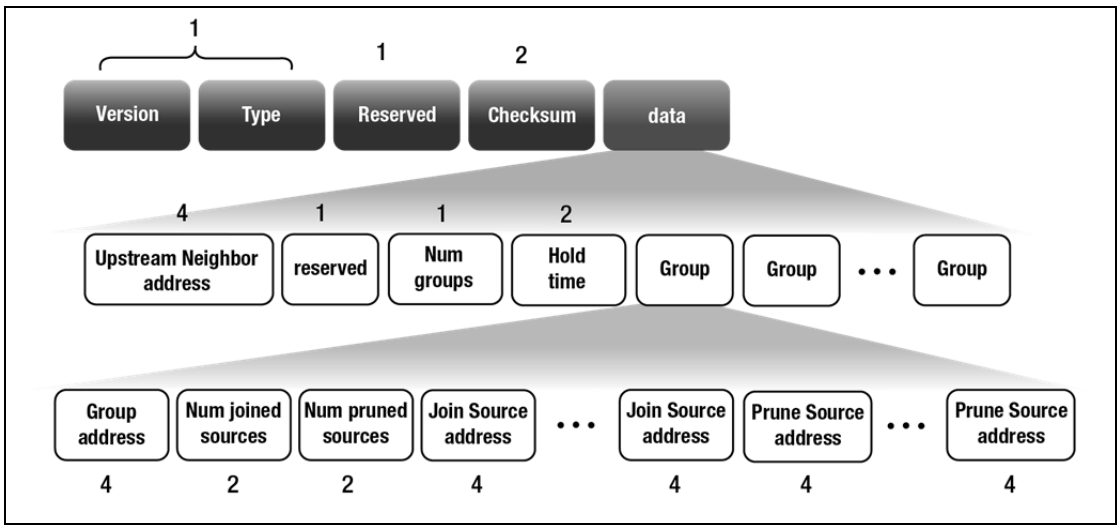
Option	Option Name	Description
Option 1	Hello Hold Timer	This option must be present in a Hello packet. It tells the recipient how long to wait before timing out the neighbor relationship with the sender if the recipient stops receiving Hellos from the sender. Unlike some other protocols, in which the neighbor timeout is configured on the recipient, in PIM the timeout is defined by the sender.
Option 20	Generation ID	This option field carries the value of the Generation ID, described above in the section Generation ID on page 373.
Option 21	State Refresh Capable	This option indicates that the sender supports State Refresh sending and receiving. There are some PIM Dense mode implementations that are compliant to earlier drafts that did not define State Refreshes. Routers cannot immediately assume that their neighbors support State Refresh. If a router does support State Refresh, then it must put this option in its Hello packets.

Join/Prune packet format

The PIM Join and Prune messages are effectively the same message type.

A single Join/Prune can Join and/or Prune multiple (S,G) streams at once.

The packet contains a list of multicast groups, and associated with each multicast group, there is a list of the source addresses of the 0 or more streams to that multicast group that the router wishes to join, and a list the source addresses of the 0 or more streams to that multicast address that the router wishes to have pruned.



Field	Description
Upstream Neighbor Address	Address of the neighbor on the RPF path to the sources of all the streams mentioned in the packet.
Hold Time	This tells the upstream neighbor how long to keep the streams pruned, if the packet is pruning any streams.
Number of Groups	The number of multicast groups for which this packet contains Joins or Prunes.
Multicast Group Address	The group address of one of the groups being joined or pruned.
Number of Joined Sources	The number of sources in the list of source addresses from which the router would like to receive streams to the group in question.
Number of Pruned Sources	The number of sources in the list of source addresses from which the router would like to prune streams to the group in question.
Join Source Address 1..n	The list of source addresses from which the router would like to receive streams to the group in question.
Prune Source Address 1..n	The list of source addresses from which the router would like to prune streams to the group in question.

Graft and Graft-Ack packets

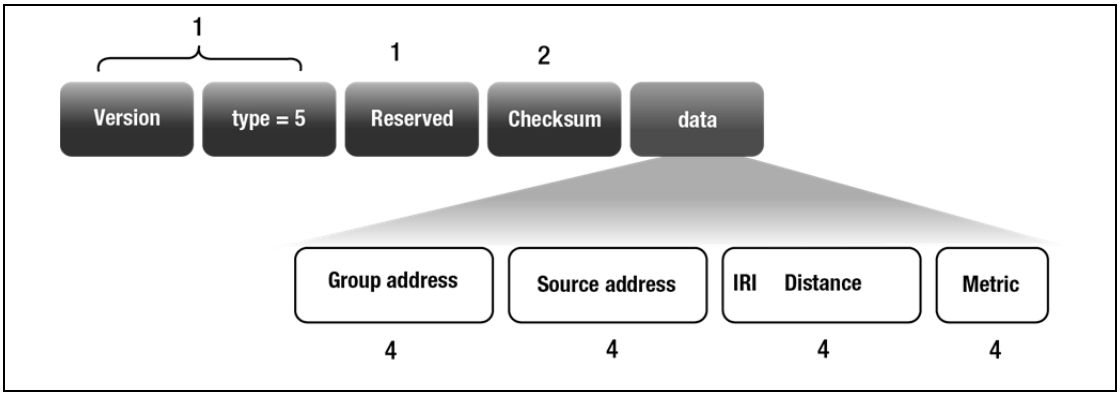
Graft and Graft-Ack packets also use the same format as the Join/Prune packet. The only difference being the packet type in the PIM header. The Graft packet has Type=6 and the Graft-Ack packet has Type=7.

A Graft puts the source address(es) of the stream(s) it wishes to receive into the Join location in the packet. It sets the holdtime to 0.

A Graft-Ack is effectively a reflection back of the Graft that it is Acking. The only difference being the value in the Type field, and the fact that the "*upstream neighbor address*" is set to the address of the sender of the graft (who is actually a downstream neighbor of the sender of the Graft Ack).

Assert packet format

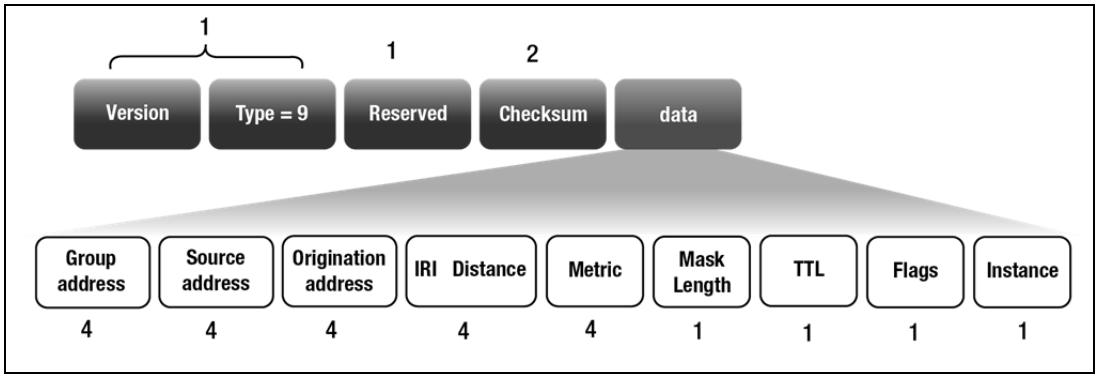
An Assert Packet relates to just one (S,G)



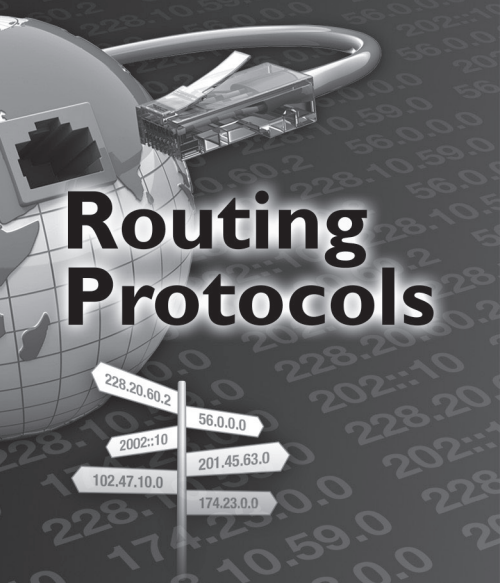
Field	Description
Multicast Group Address	Group address, G, of the (S,G) that this packet relates to.
Source Address	The Source address, S, of the (S,G) that this packet relates to.
R	The Rendezvous Point Tree bit. Relevant only to PIM Sparse Mode, not used in PIM Dense Mode.
Metric Preference	The Administrative Distance of the sender's route back to the source, S.
Metric	The Metric of the sender's route back to the source, S.

State Refresh Message Format

A State Refresh packet relates to just one stream (S,G).



Multicast Group Address	The Group address, G, of the (S,G) that this packet relates to.
Source Address	The Source address, S, of the (S,G) that this packet relates to.
Originator Address	The address of the router directly connected to the source that originated the State Refresh.
R	The Rendezvous Point Tree bit. Relevant only to PIM Sparse Mode.
Metric Preference	The Metric of the sender's route back to the source, S.
Masklen	The mask length of the sender's route back to the source.
TTL	Time To Live of the State Refresh message.
P	Prune Indicator. If the sender of the State Refresh (SR) believes that (S,G) is pruned on the interface out of which it transmitted this SR, then the flag is set to 1. Otherwise, the flag is set to 0.
N	Prune Now flag. This is a hangover from an earlier version of PIM Dense Mode.
O	Assert Override flag. This is also a hangover from an earlier version of PIM Dense Mode.
Interval	Set by the originating router to the Interval (in seconds) between consecutive State Refresh messages for this (S,G) stream.



Routing Protocols

This chapter covers the following topics:

- Introduction
- An overview of the PIM-SM protocol
- BootStrap Router election and RP selection
- Distributing RP information
- Tree Information Base
- Joining and Pruning
- Understanding the presentation of the Tree Information Base
- PIM Sparse Mode packet formats
- Assert Packet Format

CHAPTER 12

PIM Sparse Mode

Introduction

This description on PIM Sparse Mode builds on the preceding description of PIM Dense Mode. It will be assumed that you have already read the PIM Dense Mode description.

Whilst PIM Sparse Mode has a slightly different philosophy to PIM Dense Mode, the two protocols have a lot in common. To quite an extent, PIM Dense Mode is a subset of PIM Sparse mode. So, those elements of PIM Sparse Mode that exist in PIM Dense Mode will not be described in detail here, but you will be referred to the relevant description in the PIM Dense Mode chapter.

As is often stated, the prime difference between PIM Sparse Mode and PIM Dense Mode is that:

- Dense Mode forwards streams until told not to
- Sparse Mode does not forward a stream until asked

While this sounds like a relatively minor difference, it results in Sparse Mode needing quite a lot more complexity than Dense Mode. The big difference that results from these two contrasting approaches is that: in Dense Mode all routers know of all the streams in the network and, most significantly, know the source IPs of all the streams, whereas in Sparse Mode, a router will not know the source IP of a stream is it not currently forwarding.

As a result, Sparse Mode routers need a way to find out the source IPs of the streams for which they receive downstream requests. To solve this, Sparse Mode introduces the concept of Rendezvous Points – specific designated routers that receive notification of all streams destined to specific ranges of multicast addresses (or, possibly, all multicast addresses). In turn, the introduction of Rendezvous Points (RPs) into the protocol requires that the protocol also provide a way that routers find out the identities of the Rendezvous Points. Hence Sparse Mode has the concept of a Bootstrap Router (BSR) that knows the identities of the RPs and provides this information to all other routers. In addition, the protocol needs a process by which the RPs are notified of new streams, and a process whereby a router, having learnt the source IP of a stream from the RP, then accesses the stream via a direct path from the source.

All in all, this all ends up making PIM Sparse Mode noticeably more complicated than PIM Dense Mode.

An Overview of the PIM-SM Protocol

As with most routing protocols, PIM Sparse Mode is multifaceted, and contains operational sequences that deal with a number of different situations. There is no linear path to follow when trying to get an overview of the protocol. However, if we split the overview into two portions, then we can treat each portion in a quite linear fashion, and still cover most of the important features of the protocol.

The two angles from which we will overview the protocol will explain:

1. How routers discover their network environment, and how they react to downstream joins and leaves.
2. What happens to a stream.

The router's eye view

When a PIM Sparse-Mode router comes up, there are two key activities it must perform initially

- finding neighbors
- discovering who the BSR and the RPs are

The process of neighbor discovery is exactly the same as that in PIM Dense Mode. Simply, the routers send out Hello packets, and keep track of the other routers from which they have received Hello packets. The process of electing a BSR and distributing RP information is a bit more complex, but is kept relatively simple.

Electing a BSR and distributing Rendezvous Point information

First, a BSR needs to be elected; then the routers that are RP candidates must inform the BSR of their candidacy; then the BSR must distribute the RP information to all the routers in the network.

As a router comes up, it will either be participating in the BSR election, and/or passing its RP candidacy to the elected BSR, or simply receive RP information from an elected BSR.

Whatever the case, the net result needs to be that the router ends up with a set of Group-to-RP mappings. So, when the router receives requests for a given group, it knows the address of the RP from which to request that group.

Once the router has found its neighbors, and worked out the Group-to-RP mapping, it is ready to receive requests for streams.

Requests for streams

Requests can come in the form of IGMP Reports from directly connected hosts or in the form of PIM Joins from downstream neighbors. The router acts the same way in both cases.

- If the router receives a new request for a group that it is already forwarding to one or more interfaces, then it simply adds a new downstream interface to the list of egress interfaces. There is no need to signal anything upstream or to send an Ack to the requestor; it simply starts forwarding the stream out the port the request arrived on.

- If the router was not already forwarding the requested stream, then there is rather more involved:

  - The router does not know what source device (if any) might be transmitting the requested stream, so it cannot simply send a request towards that device to ask for a copy of the stream. Instead, it has to rely on an RP to provide the stream. By looking up its Group-to-RP mapping (that it has learnt from the BSR) the router knows the address of the RP that should be able to provide the requested stream.

  - The router sends a Join request off towards the RP. The request is not unicast to the RP, it is sent as a multicast towards the neighbor that is the next hop on the path towards the RP. This neighbor, in turn, will forward the request to its next-hop neighbor on the path towards the RP. The request will continue to be forwarded from neighbor to neighbor in this way until either it arrives at a router that is already receiving the requested stream, or it reaches the RP.

  - If the request reaches a router that is already receiving the stream in question, before getting to the RP, then this router will start sending the stream down in the direction from which the Join arrived, and will not forward the Join request any further upstream.

  - If the request ends up going all the way to the RP, then the RP will start forwarding the stream down in the direction from which the Join request arrived. If the RP has never actually received the requested stream, then it will simply do nothing – there is no mechanism in PIM for a router to say "*sorry, that stream does not exist*".

  - Once the requesting router receives the stream, it will see the source IP of the stream. At that point, it can take the opportunity to send a request directly towards the source, rather than having to receive the stream via the RP. The RFC does not specify how long after starting to receive the stream that a router should start sending Join requests directly towards the source. However, AlliedWare Plus, like most implementations, starts immediately. As soon as it receives a packet in the stream, it learns the source IP of the stream, and then starts sending requests directly towards the source.

  - Soon enough, the stream will start arriving along a path directly from the source. At that point, the router prunes itself from the stream that is arriving from the RP.

At that point, the router's forwarding of the stream has reached its final steady state.

What happens to a Stream

Now, let's cut over to an overview of how a stream gets treated by PIM-SM.

The treatment of a stream

A source simply sends a stream. It has no idea which devices (if any) will receive the stream. It is the routers in the network that pick up the stream, and forward it to where it needs to go.

The router nearest the source, known as the **first-hop router**, has the responsibility of making sure that the RP for a given stream gets a copy of that stream.

The first-hop router, like all routers in the network, needs to learn a group-to-RP mapping from the BSR. Having learnt that mapping, it knows the address of the RP to which it must forward any given stream.

Of course, the first-hop router can't forward the stream to the RP by multicast, as the routers in between will not know to forward the stream. Instead, the first-hop router has to encapsulate the packets of the stream with a unicast header, and unicast them to the RP.

This is a process known as **Registering** the stream with the RP.

Once the RP starts receiving the stream by this unicast tunneling method, it has two choices of how to proceed:

1. If it has had requests for this stream, then it will unencapsulate the packets, and forward them out the interface(s) on which it has received the Join requests. At the same time, it will send PIM Join requests upstream towards the source of the stream, to establish a path that can deliver the stream by multicast. Once the Joins have arrived at the first-hop router, and that router sends the stream by multicast, the RP requests that the first-hop router stop tunneling the stream in unicast register packets.
2. If the RP has no currently active requests for the stream, it simply signals to the first-hop router to stop sending the register packets for the stream.

So, there we have the overview of the operation of PIM Sparse mode. Now let's look at the operation in more detail.

BootStrap Router Election and RP Selection

The election of the bootstrap router, as with most election processes in network protocols, is a matter of all the candidates sending out packets that announce their credentials for the role, and everybody agreeing who has the best credentials.

Not every router in the network will necessarily be a candidate for the role of BSR. Most PIM implementations default to a router not being a candidate for BSR election, but the user being able to configure it to be a candidate.

Under AlliedWare Plus, the command to configure a switch as a bootstrap router candidate is:

```
ip pim bsr-candidate <interface> [<hash>] [<priority>]
```

Where:

interface is the name of one of the PIM interfaces on the Router. The router will use the address of this interface on its BSR negotiation packets.

hash is a parameter that is used in choosing the RP. Under AlliedWare Plus, the hash value defaults to 10 if not specified in the command.

priority is the router's priority for becoming BSR. The higher the priority, the better the chance that the router has of becoming BSR. Under AlliedWare Plus, the priority **defaults** to **64** if not specified in the command.

The credential that a router uses to establish its suitability for the BSR role is a combination of the user-configurable priority value, and the IP address of the specified interface.

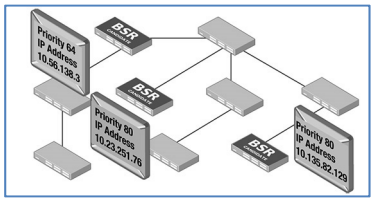
All the routers that have been configured to be BSR candidates send out their BootStrap Messages (BSMs). All routers in the network must forward the BSMs they receive. They forward the BSMs out all PIM interfaces on which they have PIM neighbors, including the interface on which the BSM was received. The idea is to get the BSMs spread as widely through the network as quickly as possible.

To avoid BSMs bouncing back and forth as routers forward the BSMs back out the interface they were received on, there is a check performed – if a BSM is received from a neighbor that is not the RPF neighbor back to the originator of the BSM, then it is dropped.

As the various BSR candidates receive each others' BSM packets, they work out who is the best candidate. As soon as a router receives a BSM from a better-qualified candidate, it will stop sending its own BSMs. Soon enough, there is only one router still sending BSMs, and it has been elected BSR.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/Aok4qFiWSrg



Note – the BSR can be pre-empted. If a BSR has been elected, but then another router in the network is configured to be a BSR with higher priority than the currently elected BSR, then the current BSR will back down, and let the new one be elected BSR.

Distributing RP Information

Just as routers in the network can be configured as BSR candidates, they can also be configured as candidates to be the RP for a range of Groups.

Typically, you should configure different routers to be RP candidates for different sets of groups, so that the load can be shared around the network, rather than one router being the RP for the whole multicast address range.

Under AlliedWare Plus, the command to configure a router as an RP candidate is:

```
ip pim rp-candidate <interface>
[priority <priority>|interval <interval>| grouplist <grouplist>]
```

Once the BSR has been elected, then any routers that have been configured as RP candidates are required to send Candidate RP advertisement (C-RP-Adv) packets to the BSR. These packets inform the BSR of the range(s) of group addresses for which the router wishes to be RP, and its configured priority.

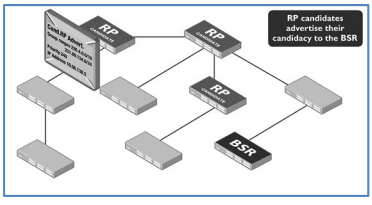
The BSR receives the RP candidacies that are advertised to it, and accumulates a list of group ranges that have RP candidates, and the list of candidates for each of those group ranges. There is no hard and fast rule about whether a BSR must accept all the candidate RPs that it has received for any given group range, or whether it may apply some policy that limits the number of candidate RPs that it will accept for a group range. This is a choice that is allowed to be implementation-specific.

The list of group ranges, and their associated lists of candidate RPs, is referred to as the **RP-set**.

The content of the RP-set is advertised in the BSMs that the BSR continues to originate at regular intervals.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/U2K-8kA3sG4



Choosing the right RP from the set

All the routers in the network will receive the lists of group ranges, and candidate RPs for those group ranges, from the boot-strap router.

What do the receiving routers do with these lists of RPs? Basically, they store the information away for when they need it.

The RP information is needed in two circumstances:

1. When the router needs to send a Join request to the RP for a group that a downstream device is requesting.

2. When the router is the first-hop router for a stream, and needs to unicast this stream to the RP, so as to inform the RP that the stream exists (a process called **Registering** the stream).

In both of these situations, the router needs to choose just one RP to use for the group in question. Moreover, the router needs to choose exactly the same address as any other router in the network would choose as the RP for that same group.

To understand why all routers in the network need to choose the same RP address for any given stream, just think what would happen if the first-hop router chose to register a given stream with RP1, but another router sent its Join request for that stream to RP2. In short, the requesting router would not receive the stream, as RP2 would not know anything about it.

All the routers in the network need to have exactly the same algorithm for choosing the RP address corresponding to a given group.

The algorithm that is used in PIM Sparse Mode has two stages:

1. Each candidate RP has a priority value associated with it. (This is the **priority** parameter in the **ip pim rp-candidate** command). Choose the candidate with the lowest priority value.

2. If there is a tie – i.e. multiple candidates all with the same lowest priority, then the tie-breaker is a hash function. The router takes a portion of the group address and puts it into a hash function that will come up with a number "N". Based on that result, the router will use the Nth RP in the list of tied lowest-priority-value RPs

Exactly which portion of the group address the router chooses to use is determined by the "*Hash*" parameter that the BSR was configured with. This hash parameter is actually a mask length. A hash value of X says "*insert the first X bits of the group address into the hash calculation*".

The hash value that is configured on the BSR is advertised in all the BSMs that it sends out, so all routers in the network will use the same mask length in the calculations for choosing RPs.

Registering a stream with the RP

As described in the overview, the first-hop router has a duty to register a stream with the RP.

Let's go step-by-step through how that happens.

The first step in the process is that of deciding who will do the registering. There could be multiple PIM routers connected to the LAN that contains the multicast source. If they all took it upon themselves to register the source's streams with the RP, then the RP would be bombarded by multiple copies of each stream. That would be wasteful of bandwidth, and put extra strain on the RP.

If there are multiple routers on the LAN, they need to elect a single router to perform the registration. This is referred to as the **Designated Router**, or DR.

Electing the DR

The election of DR is quite simple, and is integrated into the process of finding neighbors.

Essentially, all PIM routers send Hello messages periodically from all their active PIM interfaces. One of the fields in the Hello packet is the DR priority. As routers receive each others' Hello packets, they check the DR priority. The router with the largest value of DR priority is the winner. The tie breaker in the case of more than one router having the same highest value of DR priority is IP address. Of the tied routers, the one with the **highest** IP address on the subnet in question is the **winner**.

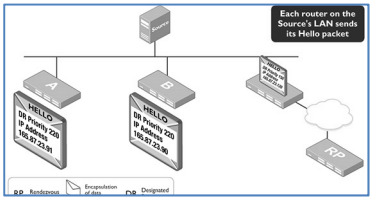
Under AlliedWare Plus, the commands to configure the DR priority on an interface are

```
awplus# configure terminal
awplus(config)# interface <vlanx>
awplus(config-if)# ip pim dr-priority <priority value>
```

Where the DR priority can take any value from 0 to 4294967249.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/5HNMLj2Hopc



A DR can be pre-empted at any time. If a new router joins the LAN, that has a better claim on the DR role than the current DR does, then the current DR will bow out, and the new router will take on the role. Similarly, if the DR priority of a router already connected to the subnet is increased, so that it now has the best claim to the DR role, it will take on the role.

The registration process

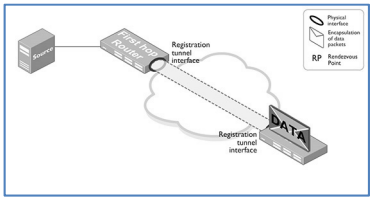
When the first-hop DR starts receiving a stream (S,G) that is not currently being handled by the registration process, it must proceed to register this stream.

It needs to work out the RP for this stream (by the RP selection process described above in the section BootStrap Router election and RP selection on page 388.

Then, it effectively sets up us a tunnel from itself to the RP, through which it will send the stream. In fact, it is very useful to think of there being a virtual tunnel connection from this router to the RP, and for there to be a virtual PIM interface on this tunnel. Then, within the protocol, this virtual PIM interface can be treated quite similarly to a normal PIM interface.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/4_-mgIDPAL0



When the first-hop DR first starts registering the stream, it acts as though the RP had implicitly sent a join request to this virtual interface. So, the first-hop DR sends the stream into the tunnel.

"*Sending the stream into the tunnel*" means, in fact, that the first-hop DR wraps a unicast header around the packets of the stream (S, G) and sends them to the unicast address of the RP. The packets are not sent as UDP, but are sent as PIM register packets.

If the data packets of the stream, when arriving from S, are large enough that resulting register packet, with its extra unicast header, will exceed the MTU of the egress interface, then the data packets are fragmented, and sent in two register messages.

The TTL of the packets is decremented by 1 before they are encapsulated, so the journey to the RP is treated as just one hop, from the TTL point of view, even though the encapsulated packet might pass through several routers. This does mean that the TTL of the stream when it is delivered from the RP might be somewhat higher than it will be when delivered hop-by-hop, by multicast, from the first-hop router. However, it is not possible to predict in advance how many hops the eventual receivers of the stream will be from the first-hop router. What we do know for sure is that for all receivers downstream of the first-hop router, the TTL will be decremented by at least 1 when the stream is delivered hop-by-hop, by multicast, from the first-hop router. So, decrementing the TTL by 1 before sending to the RP is the most sensible thing to do.

How the RP reacts

The RP receives the register packets and decapsulates them. Having done this, the RP may drop the packets, or may forward them on to down-stream clients and neighbors that have sent Joins for the group, G, that is the destination of the stream.

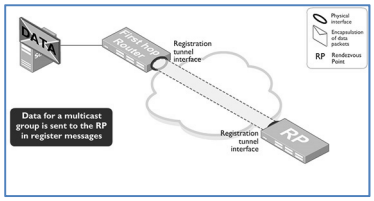
Also, the RP will work to get the first hop router to stop sending the stream through and encapsulated tunnel as quickly as possible. The overhead of sending the stream through an encapsulated tunnel is not desirable; so the RP needs to get this stopped as quickly as possible.

So, the RP sends PIM Joins upstream towards the first hop router, to establish a path via which the stream can be directly multicasted from the first hop router to the RP, rather than being tunneled in unicast. (How the sending of Joins upstream establishes a multicast forwarding path is described below in the section Obtaining a stream from the RP on page 401.

Once the multicast forwarding path to get (S,G) to the RP is established, and the stream starts arriving at the RP by multicast, it is time to inform the first hop router that it no longer needs to send the stream in a unicast tunnel. The packet type that carries out this mission is called a **Register-Stop packet**. The RP sends Register-Stops for (S,G) to the first-hop router. When these packets are received at the first-hop router, it treats them as though they were Prunes arriving on the register-tunnel interface. As a result, the first-hop router stops sending the stream into the tunnel.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/5oJGJ4AdibA



Null-registers

When the first hop router stops sending the stream into the tunnel, it reserves the right to start doing so again if it seems that the RP is no longer insisting that the registering be stopped. Effectively, the first-hop router sets a timer, called the Register-Stop timer. If it does not receive another Register-Stop for (S,G) by the time the timer has expired, then it will resume sending the stream into the tunnel.

To make sure that the RP realizes that the (S,G) stream still exists, and to remind it to send a Register-Stop; the first-hop router will send a special "*Null-Register*" packet to the RP just before the Register-Stop timer is about to expire. This "*Null-Register*" tells the RP that the stream (S,G) is still flowing. In response the RP re-iterates its Register-Stop messages to the first hop router.

This ping-pong of Null-Register in one direction and Register-Stop back again, every minute or so, becomes the equilibrium state for the interaction between the first-hop router and the RP with regard to stream (S,G). This equilibrium can last indefinitely.

A summary of the steps in registering a stream with an RP

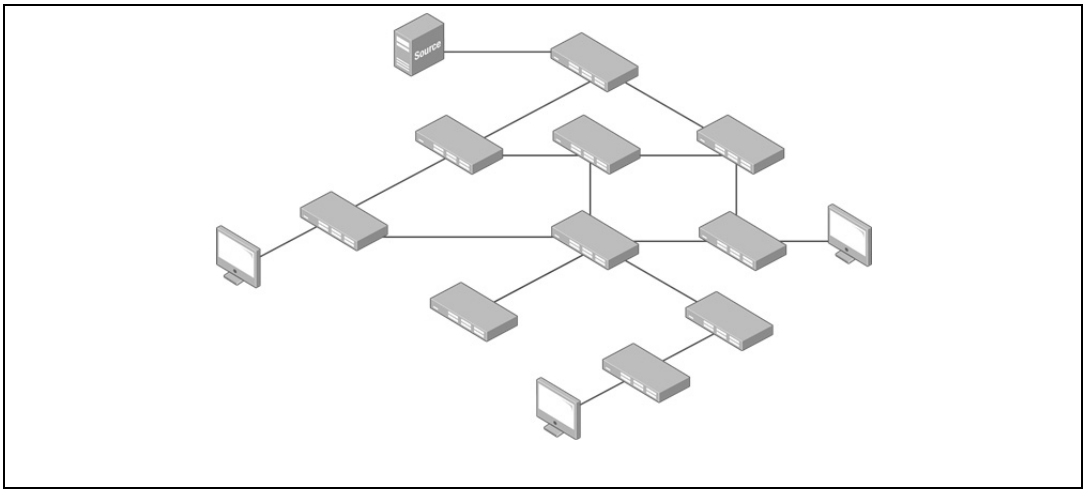
- Routers on the source's LAN elect a DR.
- The DR looks up the RP for each group being streamed by the source.
- The DR encapsulates data packets in register packets, and unicasts to RP.
- Register packets arrive at RP, so the stream is Registered with RP.
- The RP sends Joins upstream towards source, so the stream can be multicasted to it.
- When Joins reach router on source's LAN, the stream starts multicasting to RP.
- When the multicasted stream arrives at RP, it sends Register-Stops to DR on source's LAN.
- When the DR on source's LAN receives Register-Stops, it stops encapsulating stream in Registers.
- The DR on source's LAN continues to send periodic Null Registers.
- The RP responds to Null Registers with Register Stops.

We have now seen how routers discover where the RP is; and how the RP finds out which streams are available in the network. The next step in the process of delivering streams to clients is for routers to request streams from the RP.

However, before we go on to look at that step of the process, we need to understand a concept that is quite central to PIM. The concept in question is the Tree Information Base.

Tree Information Base

In a multicast network, the forwarding path for any given stream is a tree, from its source, with branches reaching to all the receivers of the stream. This is typically referred to as the **Forwarding Tree** or **Distribution Tree** for that stream.



In PIM, the idea of the Distribution Tree is extended somewhat, so that it takes in not just the tree that represents where a stream is being forwarded to from its source, but also:

- The trees that represent where a stream is being forwarded to from an RP.
- The set of forwarding entries that have been created in a chain of routers, due to a Join request from a receiver, for a stream that does not actually exist in the network.
- The 'memory' of a forwarding tree – i.e. "*I have pruned that stream*" entries in a chain of routers who had been forwarding a stream, but are no longer doing so.

PIM needs to keep track not just of the streams that are currently being forwarded, but also the paths that have been set up waiting for a stream to appear, and the paths that have been recently been closed down.

This is as it should be – the purpose of the PIM protocol is to manage the L3 forwarding of multicast in a network. So, PIM is all about setting up and maintaining multicast forwarding trees. So, PIM needs to give routers a method for keeping track of what is currently being forwarded and for setting up paths ready for streams to be forwarded along, and paths along which certain streams should not be forwarded. But, what does it actually mean to say that PIM keeps track of all these different 'trees'? In essence, it means that each router separately keeps a record of all the trees that is part of.

This is what the Tree Information Base is – it is a database on each router in which that router stores all the multicast trees that it belongs to, and its current state in each of those trees.

This way, the router knows, for each tree:

- Whether it should forward packets that arrive via this tree.
- If so, where it should forward them to.
- If its waiting for the upstream router to ACK a request to Prune itself from the tree.

Tree Information Base examples

The description above is a little abstract. To understand better what the Tree Information Base actually contains, let's think of some examples of the trees that a router would be part of, and what information it would store in relation to those trees.

1. If the Router is forwarding a stream directly from source S (i.e. not via an RP) to group G, then it is in the Source-Based Distribution Tree for (S,G). There will be an entry in the Tree Information Base for this stream that contains:
    - The source and group: (S,G)
    - The ingress interface
    - The upstream neighbor
    - Which downstream interface(s) the stream is being forwarded to
    - Various timers – like how long it will wait for the next refresh of the downstream Join request, how long it would wait before timing out the entry if no more packets for the stream arriving on the upstream interface, how long before it needs to send the next refreshing Join request up to the upstream neighbor etc.
2. If the router has received an IGMPv2 request, on some interface, for a group G, then it will need a branch connecting it to the RP for that group.

As an IGMPv2 request does not specify the required source for the stream, then the tree that this particular branch belongs to is characterized as a (*,G) tree. Effectively it is a chain of routers saying "*if I receive a stream from **any** source for group G, I will forward it along this path.*"

There may not be any streams available in the network for group G, but that will not cause this tree to time out. As long as the sender of the IGMP request continues to refresh that request, this chain of requests all the way up to the RP will continue to be refreshed.

This tree causes the routers along the path to have a Tree Information Base entry stream that contains:

- The group address G, and the fact that it will forward packets from ANY source: (*,G)
- The upstream interface towards the RP

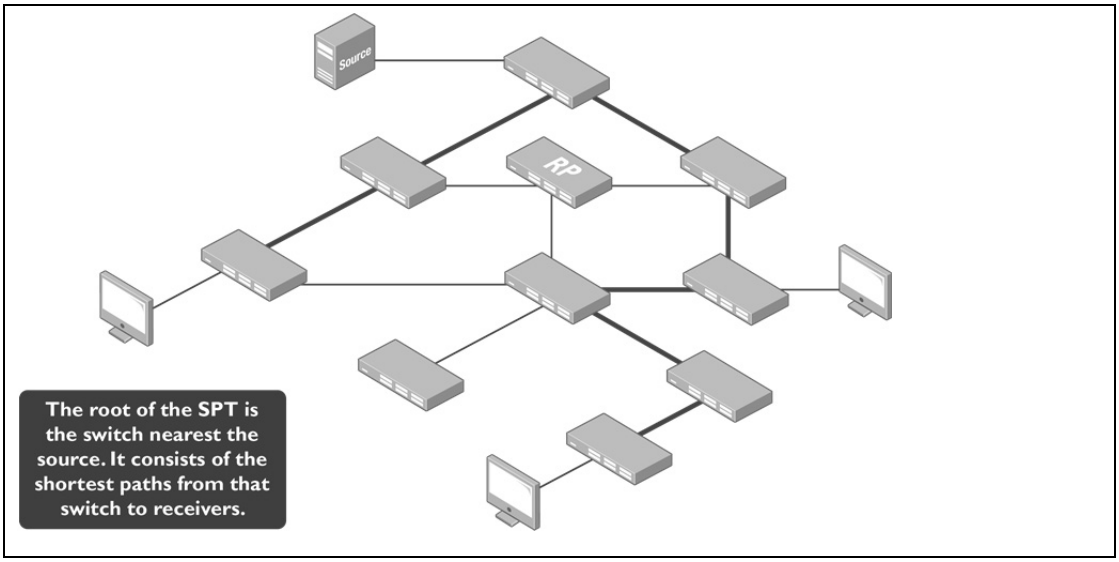
- The upstream neighbor
- Which downstream interface(s) are receiving Join requests for this group
- Various timers – like how long it will wait for the next refresh of the downstream Join request, how long before it needs to send the next refreshing Join request up to the upstream neighbor etc.

It is important to note that the particular paths any given stream will be forwarded on will differ from stream to stream (depending on exactly which end-clients have requested which groups), so the protocol needs to manage an individual tree for every stream. You can think of the network as being overlaid with dozens, or even hundreds or thousands of distribution trees, one for each group that is being streamed. Many of the trees may end up covering exactly the same sets of links, but the trees do all need to be considered separately, as each one can gain or lose branches at any moment. Hence, the Tree Information Base can be quite large, as it will contain separate entries for every stream that is passing though the router, and every group that a downstream client has requested.

Categories of Tree Information Base Entries

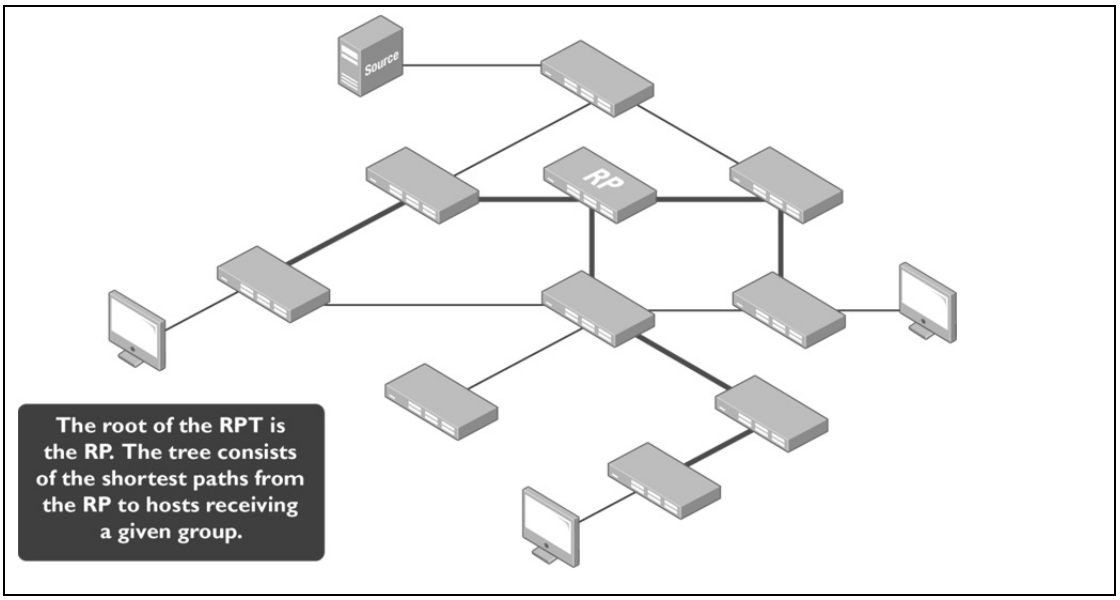
There are four types of Distribution trees that PIM keeps track of, and generates Tree Information Base entries for.

1. (S,G) entries



As discussed above, these entries relate to the straightforward case of a distribution tree for the stream from source S, destined to group G. Any Tree Information Base entry in the (S,G) category, tracks the router's current involvement in such a distribution tree – whether the router is just happily forwarding the stream to one or more downstream interfaces, whether the router is just in the process of getting itself attached to this tree (sending upstream Joins to ask to receive the stream), whether the router is in the process of detaching itself from this tree (sending a Prune upstream), or has just recently detached itself from the tree.

2. (*,G) entries

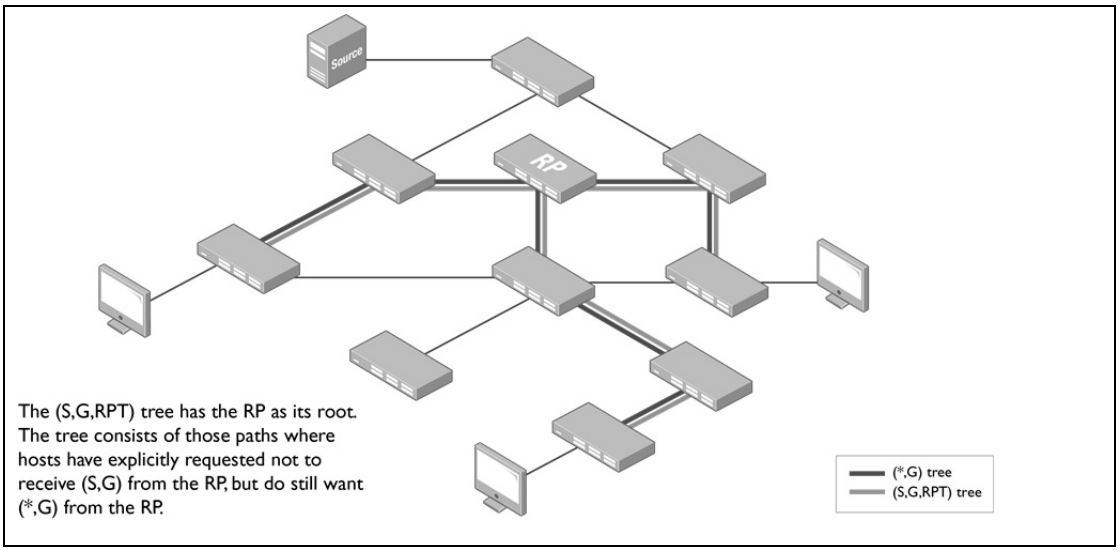


These entries are for a tree that is set up to receive a stream (any stream) for group G from the RP for that group. As discussed above, when a router receives an IGMPv2 Join request for group G, it sets up (or joins onto) a (*,G) tree, which will be the path down which a stream for group G will be forwarded from the RP.

As the Join request is forwarded from hop to hop on the way up to the RP, each router along the way becomes a part of this tree, and creates a (*,G) entry in its Tree Information Base, so that when it receives any stream destined to G, it knows where to forward it.

In the section Swapping over to the Shortest Path Tree on page 402, we will discuss the fact that once a stream starts actually flowing down the (*,G) tree, it will quite soon be cut over to an (S,G) tree instead.

3. (S,G,rpt) entries



With these types of entries, we move away from real distribution trees to more a type of "*negative*" distribution tree that is telling routers what NOT to forward.

Let's understand what a (S,G,rpt) represents, and why it is needed.

If the router has a (*,G) entry for some group G, that will be part of a tree that extends to the RP.

When the router starts actually receiving a stream to G, from some source S, it will cut over to receiving that stream directly from S (this will be discussed more in the section Swapping over to the Shortest Path Tree on page 402). At that point, the router will need to say to all the routers between itself and the RP "*don't send me the stream (S,G)",* because it is already receiving that stream via a different path.

But, that presents a little bit of a dilemma. The router does still want the RP to send it any stream to G that is coming from any source **apart** from S. So, it needs to keep the (*,G) tree in place. Hence it needs some way to say "*still send me streams to G that don't come from S, but don't send me the stream (S,G)".*

The solution to this dilemma is a **negative** distribution tree. The (S,G,rpt) tree is just such a "*negative*" tree. The router sets up a (S,G,rpt) entry, and sends a Prune for (S,G,rpt) upstream towards the RP. Then, the routers along the path of the (*,G) tree forward this Prune up towards the RP. As each of the routers receives these Prunes, and forwards them on, they create their own (S,G,rpt) entry, which says "*don't forward the stream (S,G) if it is arriving from the RP*."

This process establishes a path along which routers will NOT forward the stream (S,G), i.e. a negative distribution tree for (S,G).

4. (*,*,RP) entries

The (*,*,RP) tree is actually a forwarding tree (as against a 'negative' tree). It has the special purpose of enabling a router to receive all the streams that are being registered at an RP.

The reason for this facility is to enable a router at the border of a PIM network to receive all streams, and forward them into some other multicast network.

Such a router creates a set of (*,*,RP) trees by sending (*,*,RP) Joins towards each RP. Then, all the streams registered at each RP will be forwarded to that router.

Of course, if you do have this happening in your network, you need to make sure there is sufficient bandwidth along the (*,*,RP) trees to handle all the streams that will be sent down them.

Obtaining a Stream from the RP

Now that we have looked at the concept of a forwarding tree, and defined some of the associated terminology, we are ready to return to the main story of how a router gets the RP to send it a stream.

When a router receives a Join request from a client, saying "*send me a stream for multicast group G*", the router needs to join to the (*,G) tree for that group G.

The process of joining a tree is really just a matter of sending a PIM Join request to the upstream neighbor on the path you wish to establish.

Therefore, the steps the router performs are:

1. Determine the address of the RP for group G as described above in Choosing the right RP from the set on page 391.
2. Look up its unicast route to this address, to find the right interface from which to send the Join request.
3. Create the Join request, and send it.

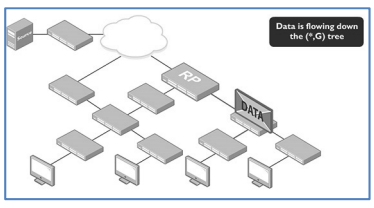
The first upstream router receives the request and creates an (*,G) entry in its own Tree Information Base. This (*,G) entry will tell the router that if it receives packets from the direction of the RP, destined for group G, then they should be forwarded out the interface that the Join request arrived on.

This router then sends a (*,G) Join request out its RPF interface towards the RP. The next router up the chain receives that Join request, and processes it, and so on.

Eventually, the chain of routers that have received the (*,G) Join request will extend up to the RP. At that point, if the RP is receiving a stream destined to G it will forward it down this newly formed (*,G) tree.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/wLtjQ69l4RI

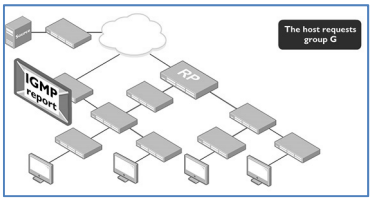


Reaching the tree before the RP

The process of forwarding the (*,G) Join requests does not always proceed all the way to the RP. If, at some point along the path towards the RP, a (*,G) Join request arrives at a router that is already connected to a (*,G) tree, then there is no need to continue forwarding Joins on up to the RP. The portion of the tree from this router to the RP is already established, there is no need to go establishing it again.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/WMks_ggXqGw



Swapping over to the Shortest Path Tree

At this point in the story, the router has received the Join request from the downstream client, it has sent its PIM Join upstream towards the RP, the Join has been relayed on upstream, and router is attached to the (*,G) tree that emanates from the RP for group G.

At this point, if the stream is present in the network, and has been successfully registered with the RP, then it will start flowing down the (*,G) tree, and will arrive at the router.

As the router receives the stream, it will forward it on to the client that sent the original Join request.

Right now, you might be thinking "*OK, that's it, the stream is successfully getting to the router and the router is sending it on to the requesting client; everyone is happy; job done*".

But, PIM Sparse Mode is not content to stop there. The main value of multicasting is to optimize the use of bandwidth. It is more than likely that receiving a stream via the RP is not actually the optimum use of bandwidth. The odds are that the path from source to client via the RP is not the most direct path possible.

Rather than just settle for this potentially sub-optimal forwarding path, and possible waste of bandwidth, PIM Sparse Mode makes the effort to find the most direct path via which to deliver the stream.

When the stream arrives at the router, the router then has the opportunity to find out one very important piece of information about the stream – namely its source. Once the router has that information, it can set about joining a tree that provides a direct path from that source.

The router does the following:

- looks up its unicast route back to S, and therefore the RPF interface towards S
- sends PIM Join packets for (S,G) out that RPF interface

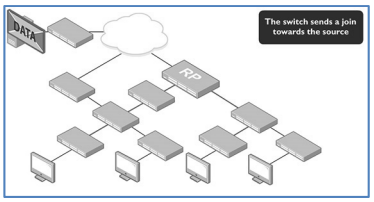
Then, each router on the path back to S will:

- Receive the (S,G) Join request.
- Create their own (S,G) entry in their Tree Information Base, so when they receive packets in the (S,G) stream, they will forward them out the interface on which the Join arrived.
- Create their own copy of the (S,G) Join request, and send it upstream toward S.

This will create an (S,G) distribution tree extending back up to S. Each router on the path has its (S,G) entry that tells it to forward (S,G) packets downstream towards the initiating router.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/CRbo4c7WGqU

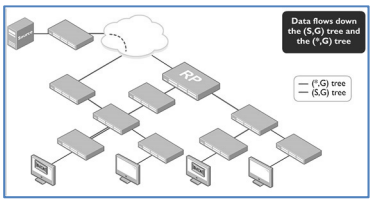


When the (S,G) stream starts arriving at the router via path directly from S, it then no longer needs to receive the stream from the RP. So, it needs to tell the RP and the routers in between, to stop sending it that stream.

As discussed above in Categories of Tree Information Base Entries on page 398, (in the section (S,G,rpt) entries), the router cannot completely remove itself from the (*,G) tree; but rather creates the 'negative' (S,G,rpt) tree, to stop the (S,G) stream being delivered to it on the path from the RP.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/incl_AjvI7Y



At this point, the process of establishing the delivery of the multicast stream to this router is finally complete.

Joining and Pruning

In the description above, there has been a fair of talk about sending Joins and Prunes. This joining and pruning has been referred to in the context of working through the process of getting a stream established. Now, let us look briefly at the joining and pruning processes in their own right.

The first thing to make clear is that PIM Sparse Mode does not have the concept of a Graft. In PIM Sparse Mode, the two tree-related actions are joining and pruning:

- As with PIM Dense mode, Joins and Prunes both use the same Join/Prune packet.
- As with PIM Dense Mode, the Join/Prune packet can include a number of entries, some of which are Joins and some of which are Prunes.

Moreover, a single Join/Prune packet can contain Joins and/or Prunes for multiple types of trees – (S,G), (S,G,rpt), (*,*,RP) ,(*,G). The nature of the tree that a particular Join or Prune entry in the packet relates to is indicated by the identity of the source address requested for the stream, and by the values of specific flags in the entry. This is discussed in more detail below in Differentiating between (*,G), (S,G), (S,G,rpt) and (*,*,RP) entries in the Join/Prune packet. on page 419.

In particular; (*,*,RP) and (*,G) entries specify the RP address as the source address for the stream being requested, whereas (S,G), (S,G,rpt) entries specify the **real** source of the stream.

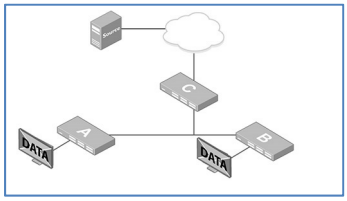
In brief:

- When a router wants to join a given tree, it sends upstream a Join for that tree. This Join will take the form of a Join entry in a Join/Prune packet. The nature of the tree – (*,G) , (S,G), (S,G,RPT) etc. – is indicated by certain values in the Join entry.
- When a router wants to leave a given tree, it sends upstream a Prune for that tree. This Prune takes the form of a Prune entry in a Join/Prune packet. The nature of the tree – (*,G) , (S,G), (S,G,RPT) etc. is indicated by certain values in the Prune entry.

Also, there is a Prune override mechanism in PIM Sparse mode. If a router that wishes to stay connected to a given tree sees a Prune for that tree being sent by another router connected to the upstream interface, it will have to override that Prune. To do so, it will quite quickly send upstream a Join for the group that was just being pruned. Then, the upstream router will continue sending the stream in question.

To watch this video, browse to the link or scan the QR code with your smart phone:

http://youtu.be/ZX-8zeZ_bP8



Interface State Machines

The interface state machines are a convenient way to describe how the router updates the state of its Tree Information Base entries when certain events occur, and what packets the router sends at those times.

The managing of (S,G), (*,G) and (*,*,RP) entries all follow a very similar pattern, so we will discuss them all together. The (S,G,rpt) is rather different, so we will look at it separately.

Interface state machine for (*,G), (S,G) and (*,*,RP)

Upstream interface

The upstream interface state machine for these entries is quite nice and simple. In essence, the upstream state machine simply has two events:

- When a downstream interface of the router receives a Join request for an (S,G), (*,G) or (*,*,RP) that it is not currently joined to, then it will aim to join the tree in question, and sends an appropriate Join request upstream.
- When a last downstream listener for an (S,G), (*,G) or (*,*,RP) that the router is currently joined to leaves or prunes or simply times out, then the router will aim to remove itself from the tree in question, and sends an appropriate Prune upstream.

When the router is not currently joined to a given tree, then the upstream interface is said to be in the NotJoined (NJ) state with regard to that tree. When the router is currently joined to a given tree, then the upstream interface is said to be in the Joined (J) state with regard to that tree.

The state transitions are laid out in the table below:

Event	State Transition	Actions
Receives downstream request to join a tree it is not currently joined to	**NJ -> J**	**Sends an appropriate Join upstream.**
Sees the last downstream member leave a tree that it is currently joined to.	**J → NJ**	**Sends an appropriate Prune upstream.**

Once the router has joined a given tree, it will continue to send Joins for this (S,G), (*,G) or (*,*,RP) from the upstream interface at regular 60-second intervals.

There is no periodic sending of Prunes. Once the router has sent its Prune, and left the tree, then that is enough. It does not need to continue to remind upstream routers that it has left the tree.

Downstream interface

A downstream interface can have three states with regard to a given (S,G), (*,G) or (*,*,RP)

1. Join – the router is currently joined to the tree in question, there is at least once device downstream of this interface that is sending regular Join requests to tell the router to stay joined to the tree.

2. Prune Pending – the router is currently joined to the tree in question, but a downstream router has sent a Prune, to ask to be removed from this tree. The router puts the interface into the Prune Pending state briefly, while it listens to see if some other device attached to this interface will send a Join to override the Prune.

   If such an overriding Join is received, the interface transitions back to the Join state. If no overriding Join is received within the time that the router waits, then that interface is pruned from the tree in question and transitions to the NoInfo state.

3. NoInfo – there are no devices downstream of this interface joined to the tree in question, so the interface is not really involved in the tree.

The state transitions are laid out in the table below:

Event	State Transition	Actions
Receive a Join for the relevant tree	→J	Irrespective of previous state
Receive a Prune for the relevant tree	J → PP	
Timeout the wait for a Join that overrides a Prune	PP → NI	Send a Prune-Echo downstream
Timeout the wait for the next regular Join message	J → NI	

Some notes

When a router receives a (*,G) Join, it checks the RP address in that Join, which will be the originator's idea of who is the RP for group G. If the receiving router does not agree that this is the correct RP for that G, then it will drop the (*,G) Join packet, and not process it.

The length of time that the router waits for regular Join packets is not set by the receiving router. Instead, the sending router decides how often it will send the Joins, and puts the timeout value into its Join packets. The time that the receiving router waits to see the next Join is the timeout value specified in the Join that has just been received.

In AlliedWare Plus the timeout value that the router will put into Join packets is set globally (not per interface) by the commands:

```
awplus# configure terminal
awplus(config)# ip pim jp-timer <1-65535>
```

A Prune-Echo is an extra Prune that the router itself sends just in case a neighbor on the downstream LAN had sent an overriding Join, but the router had not seen that overriding Join. So, the Prune-Echo is effectively the router saying "*listen neighbors, I received a Prune, and nobody over-rode it, so I am going to stop forwarding the stream in question. If anybody really does still want to continue receiving this stream, please say so now*".

Understanding the Presentation of the Tree Information Base

In AlliedWare Plus the command show ip pim sparse-mode mroute detail outputs the content of the Tree Information Base. Here is a typical example, followed by an explanation of the pieces of information that appear in this output.

```
IP Multicast Routing Table

(*,*,RP) Entries: 0
(*,G) Entries: 38
(S,G) Entries: 64
(S,G,rpt) Entries: 41
FCR Entries: 6

(10.21.103.9, 229.100.100.2) Uptime: 00:10:14
  RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: NOT JOINED, SPT Bit: on, JT: off, KAT Expiry: 132 secs
  Downstream:

(*, 229.100.100.3) Uptime: 00:08:00
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: JOINED, SPT Switch: Enabled, JT Expiry: 0 secs
   Macro state: Join Desired,
  Downstream:
   vlan272:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
  Local Olist:
   vlan272

(10.21.103.9, 229.100.100.3) Uptime: 00:10:13
  RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: JOINED, SPT Bit: on, JT Expiry: 0 secs, KAT Expiry: 148 secs
   Macro state: Join Desired,
  Downstream:
  Inherited Olist:
   vlan272

(10.21.103.9, 229.100.100.3, rpt) Uptime: 00:08:00
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: NOT PRUNED, OT: off
   Macro state: RPT Join,
  Downstream:
  Inherited Olist:
   vlan272

(*, 229.100.100.222) Uptime: 00:09:43
```

```
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: JOINED, SPT Switch: Enabled, JT Expiry: 17 secs
   Macro state: Join Desired,
  Downstream:
   vlan272:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
   vlan2521:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
  Local Olist:
   vlan272, vlan2521

(*, 239.100.10.101) Uptime: 00:09:11
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: JOINED, SPT Switch: Enabled, JT Expiry: 49 secs
   Macro state: Join Desired,
  Downstream:
   vlan2521:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
  Local Olist:
   vlan2521

(10.20.22.51, 239.100.10.101) Uptime: 00:10:13
  RPF nbr: None, RPF idx: None
  Upstream:
   State: JOINED, SPT Bit: on, JT: off, KAT Expiry: 17 secs
   Macro state: Join Desired,
  Downstream:
  Inherited Olist:
   vlan2521

(10.20.22.51, 239.100.10.101, rpt) Uptime: 00:10:13
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: PRUNED, OT: off
   Macro state: RPT Join, Prune Desired,
  Downstream:
  Inherited Olist:
   vlan2521

(*, 239.100.10.102) Uptime: 00:09:43
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: JOINED, SPT Switch: Enabled, JT Expiry: 17 secs
   Macro state: Join Desired,
```

```
 Downstream:
   vlan272:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
   vlan2521:
     State: NO INFO, ET: off, PPT: off
     Assert State: NO INFO, AT: off
      Winner: 0.0.0.0, Metric: INF, Pref: INF, RPT bit: on
     Macro state: Could Assert, Assert Track
  Local Olist:
   vlan272, vlan2521

(10.20.22.51, 239.100.10.102) Uptime: 00:10:14
  RPF nbr: None, RPF idx: None
  Upstream:
   State: JOINED, SPT Bit: on, JT: off, KAT Expiry: 16 secs
   Macro state: Join Desired,
  Downstream:
  Inherited Olist:
   vlan272, vlan2521

(10.20.22.51, 239.100.10.102, rpt) Uptime: 00:10:14
  RP: 10.21.1.53, RPF nbr: 10.24.10.1, RPF idx: vlan2273
  Upstream:
   State: PRUNED, OT: off
   Macro state: RPT Join, Prune Desired,
  Downstream:
  Inherited Olist:
   vlan272, vlan2521
```

Explanation of above output

This output shows the content of the router's Tree Information Base. As described above in the section Tree Information Base on page 396, the data that is presented here is the router's view of its state within the various multicast trees that it is attached to. As we have seen in the section Tree Information Base on page 396, there are different types of trees – (S,G), (*,G), (S,G,rpt) and (*,*,RP). The table presented here has examples of all of these except (*,*,RP).

A lot of the information that is displayed for each Tree Information Base entry describes the current state of the entities within the PIM software that implement specific aspects of the PIM Sparse Mode RFC.

The RFC is written in a way that has the software implementation very much in mind. The RFC prescribes timers that should be set when particular events occur, and when these timers should be reset, and what should be done when the timers expire. Additionally, the RFC makes much use of pseudo-code macros that describe how the software should determine whether it should Join or Prune given streams, whether it should enter an Assert negotiation, etc.

Given that these software-implementation oriented entities appear in the RFC, it makes sense for the software implementation of PIM to refer to these entities, and for the show commands to output the state of these entities. If we can see the values of timers, and the states of macros, then with an understanding of the RFC, we know what decisions the software should be making, and we can also surmise what PIM signaling packets the router has been receiving and sending.

However, at first glance, this information looks rather terse and obscure, and difficult to interpret.

Of course, the only way to fully understand the meaning and significance of each of these items of information is to completely digest the RFC. However, let's go through each item in turn here and get a reasonable level of understanding of what they mean.

RP and RPF information

RP: This is the address of the Rendezvous Point for this group. Note that (S,G) entries do not mention any RP, as those entries are for paths that come direct from the source, and don't need to use the RP.

RFP nbr: This is the IP address of the next PIM neighbor on the path back up the tree for this entry. For an (S,G) entry it is the RPF neighbor on the path back to the source; for other entries it is the RPF neighbor on the path back to the RP.

RPF idx: This is the name of the interface that connects to the RPF neighbor.

You will notice that a few of the entries above have "*None*" for both RPF nbr and RPF idx. These are entries for streams whose source is directly connected to this router, so there is no RPF neighbor between this router and the source.

Upstream Interface

First, let's look at the information that is displayed for an upstream interface:

State: This is the state as described above in the section Interface state machines on page 405.

SPT switch: This item of information is only relevant to (*,G) entries. It indicates whether the router would attempt to switch over to the SPT when it starts receiving a stream for G from some source S.

SPT Bit: This item is specific to (S,G) entries. It is set to true when the router knows that it is receiving the stream (S,G) via the SPT.

During the transition from receiving (S,G) via to the RPT to receiving (S,G) via the SPT, there will typically be a period where the router has sent its Join to the SPT, and waiting for the stream to start arriving via the SPT. During that period, the SPTBit is False, which tells the router that it should keep on forwarding the (S,G) stream that it is receiving via the RPT. But,

once the SPTBit is set, the router will no longer forward any packets of the (S,G) stream that are received via the RPT.

Typically, the SPTBit gets set as soon as the router first receives a packet in the (S,G) stream via its RPF neighbor towards S.

Once the SPTBit has been set on the (S,G) entry, the router will send a Prune towards the RP to say "*don't send the stream to me any more*". (Of course, if there are still routers downstream of the current router that want to receive the (S,G) stream from the RP, it will not send the Prune until they have ceased wanting to receive the (S,G) stream from the RP).

This Prune sets up a branch of the (S,G,rpt) "*negative tree*" between the router and the RP.

JT Expiry: Length of time until the Join timer expires, and the router will need to send another Join upstream to refresh its attachment to this tree. Note that on entries where the router is directly connected to the source's subnet (i.e. when RPF nbr is 'None' and RPF idx is 'None') the Join timer is set to Off. This is because there is no one upstream to send Joins to.

KAT Expiry: Length of time until the Keep Alive Timer expires. The Keep Alive Timer is the mechanism for deciding whether the stream in question is still arriving at the router. The KAT is reset to 0 when a packet of the stream arrives at the router. If the KAT ever reaches its full expiry time, then that means that no packets in the stream have arrived at the router for a time period equal to the expiry time of the KAT. This is grounds for deciding that the stream has stopped, and there is no need to continue to maintain a tree for forwarding this stream. At that point, the Tree Information Base entry in question is deleted.

Note that the KAT is only relevant to (S,G) entries. This is because

(*,G) trees are created in the hope of receiving a stream, their existence is not dependent on there being a stream to actually forward

(S,G,rpt) trees are not intended to forward streams at all.

Macro State: The macros states refer directly to pieces of pseudocode that are defined in the RFC, and are convenient shorthand for describing aspects of the state of a Tree Information Base entry.

The macros fall into three categories:

Join/Prune Macros

JoinDesired is a convenient way of saying that there are downstream hosts or routers that want to join onto the tree in question, and that this router is the Assert winner on at least one of the interfaces facing to those downstream hosts or routers.

I.e., the router desires to send a Join upstream on the tree in question.

Rather than say "*the router has just received a Join or an IGMP report on a downstream interface for which this router is the Assert winner*", it is possible to just say "*the JoinDesired macro has changed state to True*".

RPTJoinDesired is specific to (*,G) and (*,*,RP) entries. Its specific meaning is that the router has received Joins or IGMP reports on Assert winner downstream interfaces such that it needs to join a tree that is delivering streams from the RP.

PruneDesired is specific to (S,G,rpt) entries. It has the meaning that the router is joined to the (*,G) or (*,*,RP) tree, but either:

Is now receiving (S,G) directly from the source S, so no longer needs to receive that stream from the RP

OR

Has received (S,G,rpt) Prunes on its downstream interface, so no longer needs to receive (S,G) from the RP.

I.e. the router needs to send up (S,G,rpt) Prunes towards the RP.

Registration Macros

CouldRegister means that the router is directly connected to the Source S of a stream (S,G), and the stream is currently being transmitted from S, and the router is the DR on the LAN that contains S. If the router knows of an RP for the group G of this stream, then it would register the stream to that RP.

Assert Macros

Discussed in the following section: Downstream Interfaces on page 413.

Downstream Interfaces

ET: Is the Expiry Timer. This timer going off indicates that a PIM Join for the tree in question has not been received on this interface for quite a while, so the tree should be pruned off this interface.

Note, on interfaces where the (*,G) entry has been created due to receiving IGMP reports rather than (*,G) Pim Joins, this timer is not active, as there are no Joins to wait for.

PPT: The Prune pending timer. If a Prune has been received on this interface, the router waits briefly to see if any other downstream neighbor sends a Join to override the Prune. The Prune Pending Timer keeps track of whether the router has waited long enough to decide that an overriding Join is not coming.

Assert State: Described in the PIM-DM chapter on page 369, in the discussion of the Assert process.

AT: Assert Timer. If this router is the Assert Winner, then the Assert Timer governs when the router needs to send the next Assert. If the router is the Assert Loser, then the Assert Timer governs how long the router waits without seeing asserts before it decides the Assert Winner has gone away, and it is necessary to elect a new Assert Winner.

Winner: The IP address of the Assert Winner for this tree on this LAN. An address of 0.0.0.0 means that the current router is the only PIM router transmitting onto this LAN.

Metric: The Assert Metric, calculated as described above in in the PIM-DM chapter, in the discussion of the Assert process.

Pref: The preference of the router's unicast route to the source or RP for this entry.

RPT bit: indicates whether the Asserts are for a (*,G) entry or for an (S,G)entry.

Assert Macros

Assert Tracking Desired applies to downstream interfaces. It means that this downstream interface is connected to the tree in question, and that Assert events on the LAN that interface connects to could result in another router taking over as the point where the tree connects. So, it is necessary for Assert events to tracked, in case they affect the connection of this tree to this interface.

Could Assert also applies to downstream interfaces. It means that Joins or IGMP reports for (S,G) or (*,G) or (*,*,RP) have been received on this interface, So the router would forward (S,G) out of this interface if this router was the Assert winner. So, if there is a need to get into an Assert battle on this interface's LAN for the tree entry in question, then the router would enter into that battle.

Local oList and Inherited oList

An oList is effectively the list of downstream interfaces for a tree entry.

The Local oList on an entry is the set of downstream interfaces that have been added to the entry due to having received Joins specifically for that tree. E.g. receiving (*,G) PIM Joins, or IGMPv2 reports for group G, on a given interface, will cause that interface to be added to the local oList for the relevant (*,G) entry.

Similarly, receiving (S,G) PIM Joins on a given interface will cause that interface to be added to the local oList for the relevant (S,G) entry.

An Inherited oList for an entry is the combination of the Local oList for that entry and the Local oLists for any other more general entries that whose trees could also deliver the current entry's stream. For example, the inherited oList for (S,G) would be the combination of the Local oList for (S,G) and the Local oList for (*,G).

The Inherited oList for (S,G) is the combination of those interfaces that have received (S,G) Joins, and those that have received (*,G) Joins and/or IGMPv2 reports for G.

In effect:

(*,G) entries cannot have an inherited oList, as they have nothing to inherit from. The inherited oList for an (S,G) entry is the actual set of interfaces that the (S,G) stream is being forwarded to.

PIM Sparse Mode Packet Formats

PIM has its own IP protocol type – protocol number 103. All PIM packets have IP protocol number 103.

Most PIM packets are sent to the multicast address 224.0.0.13.

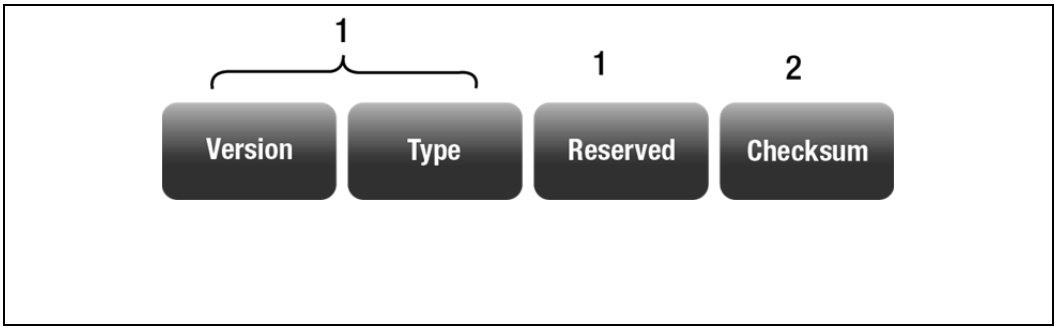
There are seven packet types used in PIM Sparse Mode:

1. Hello packets
2. Register Packets
3. Register-Stop Packets
4. Join/Prune Packets
5. Bootstrap Packets
6. Candidate-RP-Advertisement Packets
7. Assert Packets

Let us look at the format of each of these packet types.

First, we need to look at the PIM header, which is at the start of all PIM packets:

The PIM Header



The fields in the header are:

PIM version: The current version number of PIM is 2.

Type: The Type values for the different packet types are:

0 = Hello

1 = Register

2 = Register-stop

3 = Join/Prune

4 = Bootstrap

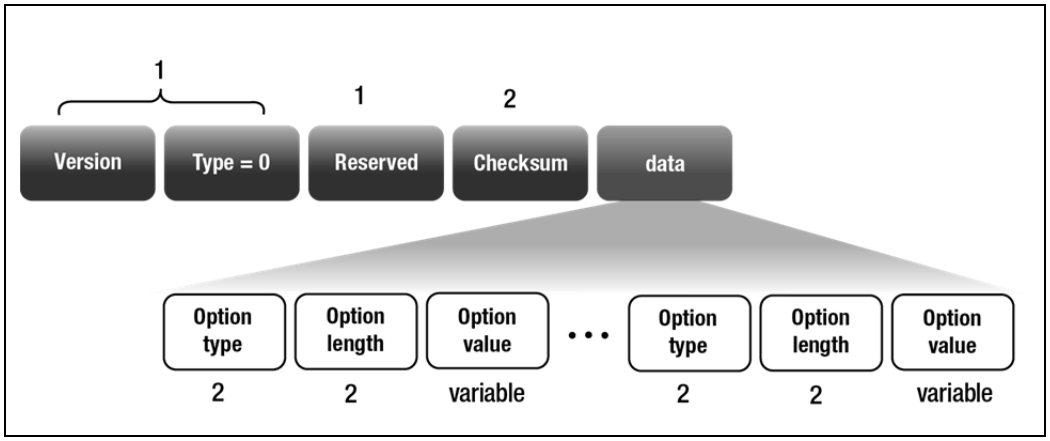
5 = Assert

8 = Candidate-RP-Advertisement

Checksum: The checksum is calculated over the entire PIM message.

Hello Packet Format

Really, a Hello packet is just a PIM header followed by one or more option fields.



The important options are:

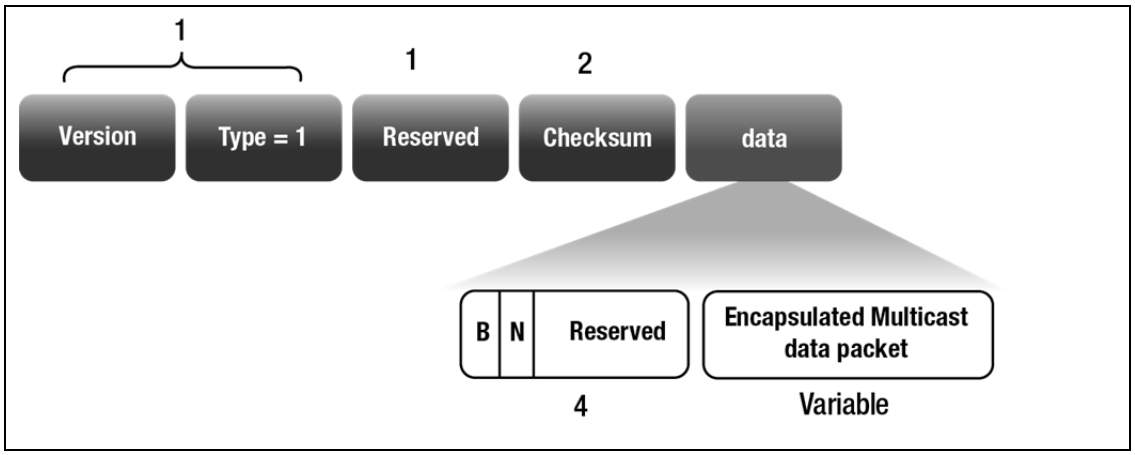
Option 1: Hello Hold Time

This option must be present in a Hello packet. It tells the recipient how long to wait before timing out the neighbor relationship with the sender if the recipient stops receiving hellos from the sender.

So, unlike some other protocols, in which the neighbor timeout is configured on the recipient, in PIM the timeout is defined by the sender.

Register Packet Format

A Register message encapsulates multicast data packets that are being tunneled to an RP. So, a unicast header is wrapped around the multicast data packet. The source address in the unicast header is the address of the first-hop router that is sending the register. The destination IP address in the unicast header is the RP's address.



B is the Border Bit. It indicates whether the stream being registered is generated by a source that is in the current PIM domain, or whether the stream is arriving from another PIM domain, and is being tunnelled to the RP by a border router.

- 0 – source is in current PIM domain
- 1 – stream is arriving from another domain

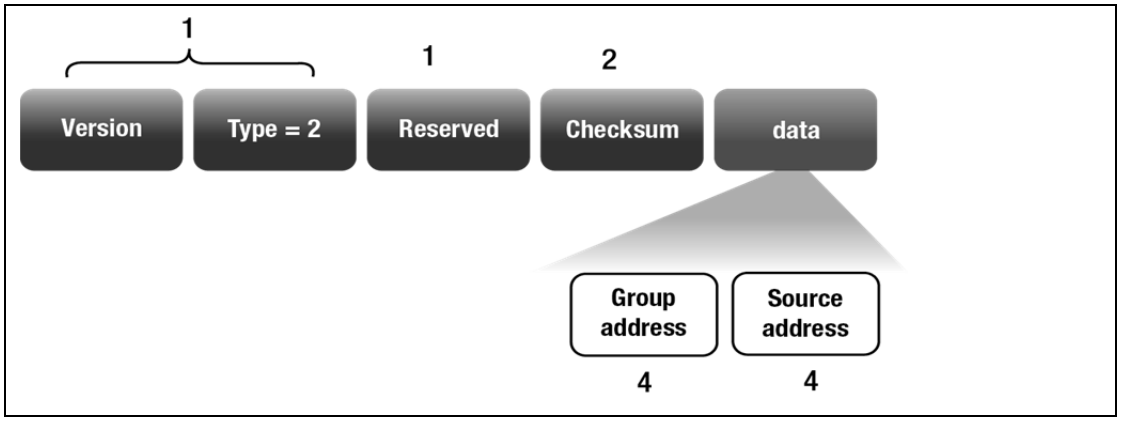
N is the Null-Register bit.

- 1 – this is a null-register packet
- 0 – this is not a null-register packet.

Multicast data packet – this is the encapsulated data packet.

Register-Stop Message Format

Like the Register packet, the Register-Stop is a unicast packet. The source IP is the IP address of the RP sending the packet. The destination is the the IP address of the router that is sending the register messages.

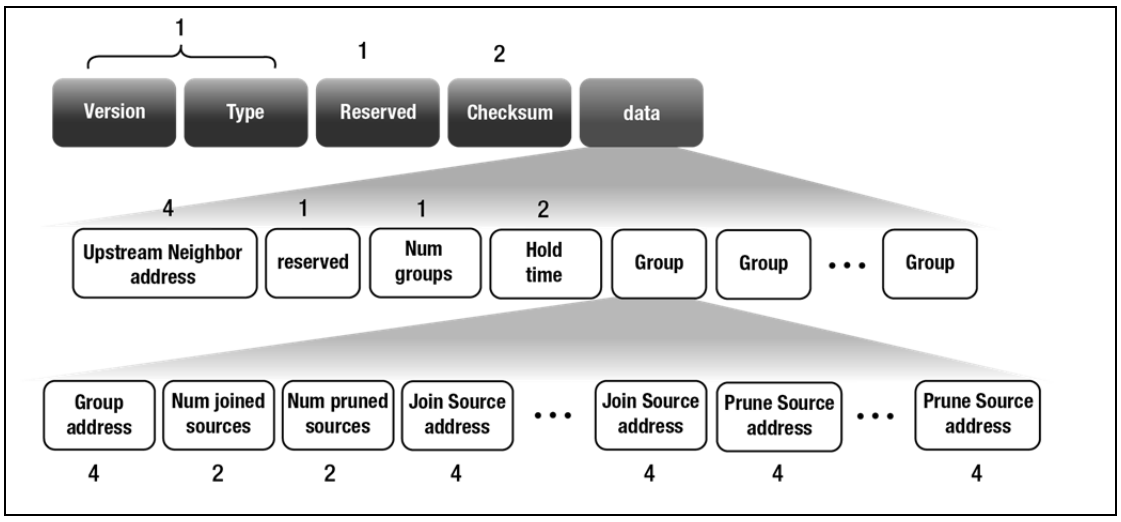


Group Address: This is the multicast destination address of the (S,G) stream in question.

Source Address: This is the unicast source address of the (S,G) stream in question. If, for any reason, the RP just wants to stop the registering of any stream for Group address G, it can use the value 0.0.0.0 for the source address.

Join/Prune Packet Format

The PIM Join and Prune messages are effective the same message type. A single Join/Prune can Join and/or Prune multiple (S,G) streams at once. The packet contains a list of multicast groups, and associated with each multicast group, there is a list of the source addresses of the 0 or more streams to that multicast group that the router wishes to join, and a list the source addresses of the 0 or more streams to that multicast address that the router wishes to have pruned.



Upstream Neighbor Address: The upstream neighbor address is the address of the neighbor on the RFP path to the sources of all the streams mentioned in the packet.

Hold Time: This tells the upstream neighbor how long to keep the streams pruned, if the packet is pruning any streams. So, unless the router sends a Graft in the meantime, the upstream neighbor should not send these streams down to the current router for at least the length of the holdtime.

Number of Groups: Number of multicast groups that this packet contains Joins or Prunes for.

Multicast Group Address: The group address of one of the groups being joined or pruned.

Number of Joined Sources: The number of sources in the list of source addresses from which the router would like to receive streams to the group in question.

Number of Pruned Sources: The number of sources in the list of source addresses from which the router would like to prune streams to the group in question.

Join Source Address 1..n: The list of source addresses from which the router would like to receive streams to the group in question.

Prune Source Address 1..n: The list of source addresses from which the router would like to prune streams to the group in question.

Differentiating between (*,G), (S,G), (S,G,rpt) and (*,*,RP) entries in the Join/Prune packet.

The same Join/Prune packet can contain Join and/or Prune entries for each of the 4 types of forwarding tree.

How do you tell which entries are relevant to which type of tree?

(*,G) entries are characterized by:

- The requested source address in the entry is the RP address for the requested group.
- Both the W (wild card) and R (rpt) flags associated with the entry are set to 1.

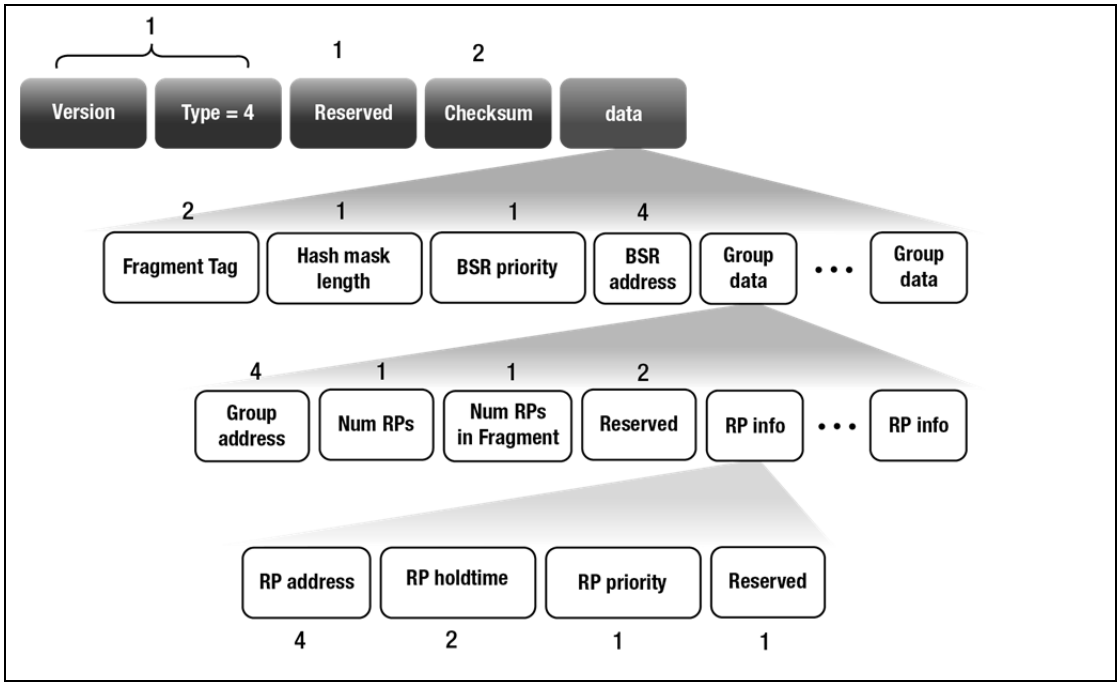
(S,G,rpt) entries are characterized by:

- The requested source address in the entry is the address S.
- The W (wild card) flag associated with the entry is set to 0. The R (rpt) flag associated with the entry are set to 1.

(*,*,RP) entries are characterized by:

- The requested source address in the entry is the address of the RP in question.
- The multicast IP address in the entry is 224.0.0.0/4.

Bootstrap Message Format



N is the No-forward bit

0 = It is OK to forward this message

1 = This message may not be forwarded

Fragment Tag: If a bootstrap message is very large, it is fragmaented over more than one IP packet. This is not using the standard IP fragmentation process, but is a fragmentation performed before the packets are handed to the IP stack.

So, the protocol needs to keep track of which fragments belong to the same BootStrap message. The Fragment tag is a randomly generated number. All the fragments of the same message contain the same tag value (which will be different to the tag values in packets that are fragments of other bootstrap messages).

Hash Mask Len: The number of bits in the mask that used in the hash process (see the section BootStrap Router election and RP selection on page 388.

BSR Priority: The priority value for this BSR candidate.

SR Address: The address of the BSR.

`Group Address 1..n:` The packet contains the RP candidate information for multiple ranges of multicast group addresses. Each Group Address field contains a value that is an encoded combination of the group address and mask that defines a range of group addresses.

RP Count 1..n: The number of RP candidates that are being put forward for a given range of group addresses.

Frag RP Cnt 1..n The set of RP candidates for a Group range could be split across two or more fragment packets, then this value indicates how many of the relevant RP candidates appear in the current fragment. Of course, if the set of RP candidates is not split across fragments, then this value will be equal to the RP Count for this group range.

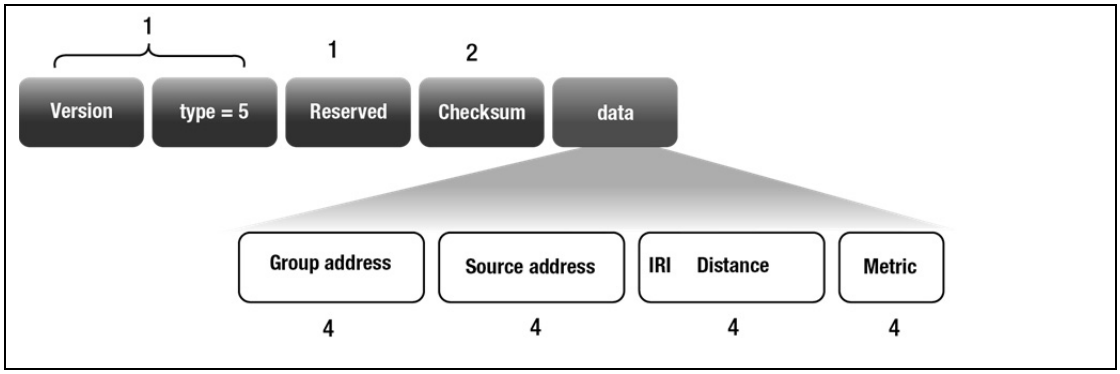
RP address 1..m: The IP addresses of all the RP candidates for the current Group address range.

RP1..m Holdtime: The Holdtime (a value in seconds) for each RP.

RP1..m Priority: The priority value for each RP.\

Assert packet format

An Assert packet relates to just one (S,G)



Multicast Group Address: The Group address, G, of the (S,G) that this packet relates to.

Source Address: The Source address, S, of the (S,G) that this packet relates to

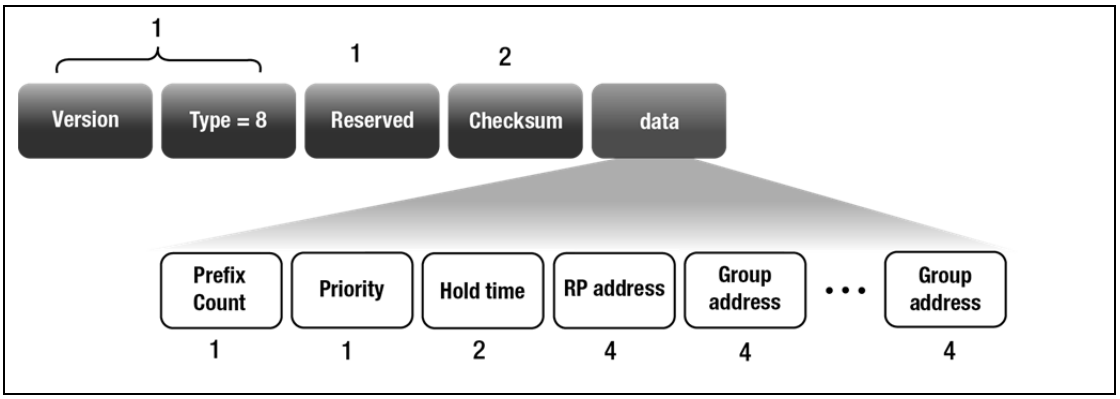
R: The Rendezvous Point Tree bit. Not used in PIM Dense Mode.

Metric Preference: The Administrative Distance of the sender's route back to the source, S.

Metric: The Metric of the sender's route back to the source,S.

Candidate-RP-Advertisement Message Format

These are unicast packets in which candidate RPs inform the BSR of their candidacy.



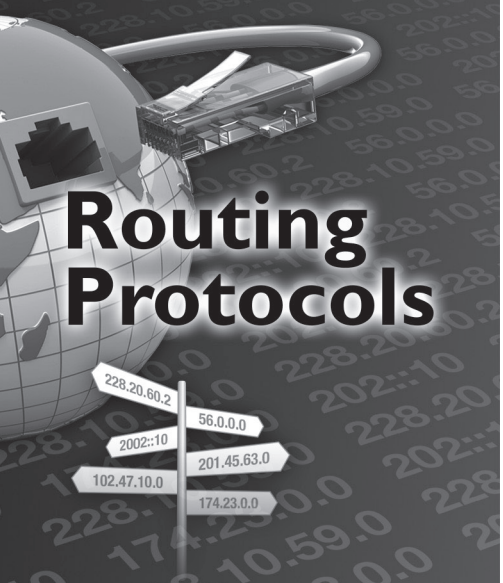
Prefix Count: The number of different ranges of group addresses for which this router is announcing its candidacy.

Priority: The priority value that is being associated with this RP candidacy. This priority value applies to all the sets of group addresses that this router is announcing candidacy for in this packet.

Holdtime: The period of time for which this candidacy remains valid. If a new advertisement is not sent within the holdtime, then the BSR assumes that this router is no longer an RP candidate for these sets of group addresses.

RP Address: The IP address of the interface on which this router wants its RP to appear.

Group Address-1..n: The sets of ranges of group addresses that this RP candidacy applies to. Each group address is provided in a form that encodes the address and mask together.



Routing Protocols

This chapter covers the following topics:

- Introduction
- IGMPv3
- New terminology
- Group address range
- PIM SSM
- IGMPv2 backward compatibility

CHAPTER 13

PIM SSM

Introduction

One of the significant characteristics of PIM Sparse Mode and PIM Dense Mode is the fact that hosts, and most of the routers, in the network do not know the source address of the multicast groups they wish to join.

Keeping the network in the dark about the source addresses of the groups makes the network management a bit simpler.

It means that you:

- Don't need a process by which hosts are told the source addresses of the streams in advance (although, that is only a small saving of hassle, as you still need a process to tell the hosts the group addresses of the streams in advance).
- Can change the multicast servers around as much as you like, without having to go telling all the hosts that the server address has changed.

But, it has quite big disadvantages:

- It makes the multicast routing protocol more complicated. A lot of the functionality within PIM Sparse Mode and PIM Dense Mode is necessitated by the fact that the hosts and routers do not know the source of a group being requested. The whole business of Rendezvous Points in Sparse Mode and State Refreshes in Dense Mode are ways that those protocols deal with the fact that streams' source addresses are not known to the requesting hosts.
- If you are receiving multicast feeds from multiple external content providers, then you need to be careful that the group addresses to which these providers are sending do not overlap with each other.
- You are somewhat vulnerable to multicast DOS attacks. If an attacker knows the group address of a stream in use in your network, then they can simply send in multicast packets destined to that group address. These packets will interfere with the genuine stream, as they will be forwarded to hosts listening to that group, irrespective of what source IP they come from.

In light of these disadvantages, a variant of Multicast routing, called Source Specific Multicast (SSM), has been defined.

In SSM routing:

- The hosts requesting streams need to know the source address of the stream they are requesting, and must specify the source in their request
- Routers differentiate between streams to the same group address, but from different source addresses. If they have been requested to send a stream (S1,G), but not a stream to the same group, from a different source (S2,G), they will forward (S1,G), but not (S2,G).

A version of PIM Sparse Mode has been created that supports SSM. Unsurprisingly, it has been named **PIM SSM**.

IGMPv3

Before we look into the details of PIM SSM, let's look at the modifications that had to be made to IGMP in order to support Source Specific Multicasting.

The IGMPv2 protocol has no concept of source addresses. No IGMPv2 packet types (Report, Query or Leave) have any field for carrying source addresses.

So, to enable Hosts to specify the source address from which they wish to receive a stream, a new version of IGMP, which is source aware, must be defined.

This is IGMPv3.

There are significant differences between IGMPv2 and IGMPv3. For example:

- IGMPv3 does not have a **Leave** packet-type. This is replaced by a request to receive a group from no sources.
- An IGMPv3 report can request multiple groups all in the same packet.
- IGMPv3 reports specify the source address from which the host wishes to receive a stream. This is relatively flexible – it can be specified as a single source IP, as a list of acceptable source IPs, a list of unacceptable source IPs (i.e. "*I am looking for a stream to group G coming from any source except the following...*") or "any source".
- A new type of Query is introduced in IGMPv3 – namely the source-and-group-specific-query, which asks "*is anybody still listening to group G being sent from the following source(s)?*"

There are other details of IGMPv3 that differ from IGMPv2., These are described in detail in the Multicast Listener Discovery chapter (as explained in that chapter, the differences between MLDv2 and MLDv1 are effectively the same as the differences between IGMPv3 and IGMPv2) . The key point, as far as this chapter is concerned, is that IGMPv3 was introduced to support Source Specific Multicast, and provides the ability to specify the source(s) from which a host wishes to receive a stream.

New Terminology

Let's look at some new terminology that was introduced in the PIM SSM RFC (RFC 4607). Using this terminology will make our discussion of PIM SSM simpler and clearer.

Any-Source multicast

Because the concept of Source Specific Multicast was being introduced, it was convenient to coin a term for the existing 'other sort' of multicast that was not source specific. The term "*Any-Source Multicast*" (ASM) was created to refer to an existing situation in which the hosts did not specify the source from which they wished to receive their multicast, and the routers therefore forwarded the multicast from any source that was available.

Channel

The data stream from a source S, destined to a group G – i.e. the stream sent down the (S, G) distribution tree – is referred to as a channel. In contrast, the multicast forwarding that occurs in Any-Source Multicast, whereby hosts just request a group, and are uninterested in which source it comes from, is referred to as forwarding to a Host Group.

Subscribe

The term subscribe is used to refer to the act of a host requesting to receive a certain group from a certain source (a channel). In SSM, hosts *subscribe* to a *channel*. Similarly, when a host indicates that it no longer wishes to receive that channel, it is said to *unsubscribe* from the channel. For clarity, the terms Join and Leave are applied specifically to the ASM case where a host indicates it wishes to receive a group, but does not specify the source. So, in the ASM case, a host *joins* a *Host Group*. And similarly, when it no longer wishes to receive that data, it *leaves* the Host Group.

Group Address Range

The subnet 232.0.0.0/8 is the set of group addresses that are used for SSM. In fact, the first class C subnet within the range (232.0.0.0/24) is reserved, and should not be used as the destination addresses for Multicast channels. However, all other address in the 232.0.0.0/8 subnet are available for multicast channels.

Addresses in this range cannot be used for ASM. RFC4607 states that a router must drop (S,G) packets, with G in the 232.0.0.0/8 subnet, unless a downstream device has specifically requested this exact channel. So downstream requests for (*,G) are not a good enough reason to forward the packets. Hence 232.0.0.0/8 addresses cannot be used for ASM purposes.

Also, the RFC states that if a router receives (*,G) requests, with G in the 232.0.0.0/8 subnet, then it should not process these requests and should not propagate these requests to any neighbors.

In fact, we will see below, in the section IGMPv2 backward compatibility on page 430, that in reality, it is not practical to complete disallow hosts from making (*,G) requests, with G in the 232.0.0.0/8 subnet. However, a router that receives such requests needs to have a mechanism for unambiguously converting such requests into (S,G) requests. As we will see in the section IGMPv2 backward compatibility, AlliedWare Plus does have such a mechanism.

PIM SSM

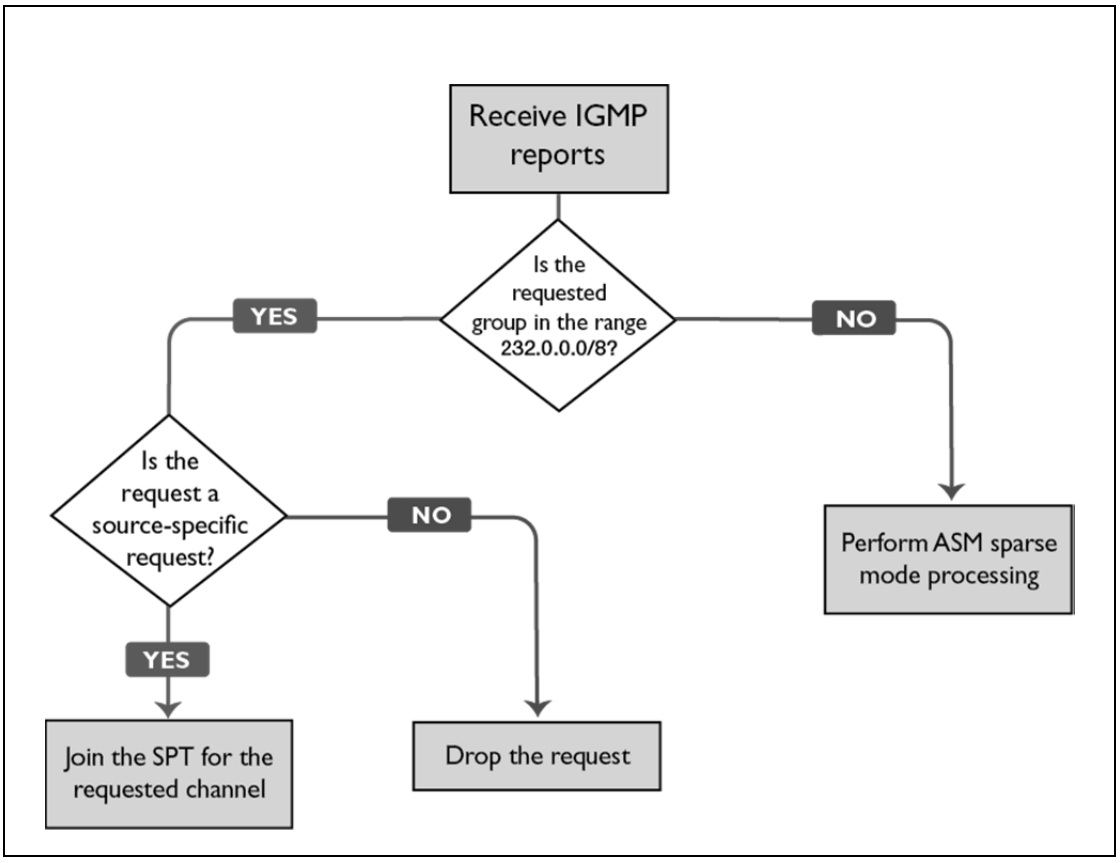
In essence, PIM SSM is PIM Sparse Mode without Rendezvous Points. Because hosts know the source from which they wish to receive streams, the multicast routers know immediately what source address they are sending their Joins towards. There is no need for the whole process of streams being registered with RPs, and routers initially having to ask RPs to send them streams until they learn the source address from the data packets.

The process of a router joining a distribution tree in PIM SSM effectively skips the whole shared-path-tree phase, and starts straight from the step of joining the Source Path Tree.

Fortunately, the process of joining the Source Path Tree (SPT) already involves sending PIM joins that specify the source address from which the router wishes to receive the stream – i.e. (S,G) Joins. So, PIM is already set up for supporting SSM.

In practice, most routers that implement PIM SSM will also implement PIM Sparse Mode, and will quite likely be performing both SSM and ASM at the same time. The processing logic is along the lines of:

1. Receive IGMP reports.
2. If a report is requesting a group whose address is not in the SSM range (232.0.0.0/8) then perform normal (ASM) PIM Sparse Mode processing - i.e. start by joining the Shared Tree for the requested group, then cut over to SPT when the stream starts arriving.
3. If a report is requesting a group whose address is in the SSM range then:
    - if the request is not a Source-Specific request, then drop it.
    - if the request is a Source-Specific request, then immediately join the SPT for the channel being requested.



IGMPv2 Backward Compatibility

As has been described earlier, Source Specific Multicasting expects that hosts subscribing to a channel will specify the IP address of the source from which they wish to receive the channel.

However, the fact is that there is a large installed base of equipment that supports on IGMPv2, and not IGMPv3. It would be extremely annoying to not be able to use SSM in a network simply because some of the equipment connected to it does not support IGMPv3.

Consider the case of a service providing delivering TV as multicast over Ethernet. If this provider is receiving content from upstream content providers who only support PIM SSM and will not accept any (*,G) Joins, then the service provider must implement PIM SSM in their network. However, it is highly likely that at least some of the subscribers connected to the network will be using Set Top Boxes that are not capable of IGMPv3. This service provider is then stuck between a rock and a hard place, they either need to go around and replace ALL subscribers' Set Top Boxes with IGMPv3 capable devices, or they need their content providers to relax their (S,G) Join requirements. Neither of these options is going to be easy.

Fortunately, there is a third option, the multicast routers in the network could help them out, and provide a work-around that converts IGMPv2 reports into Source-Specific reports.

This third option is exactly what AlliedWare Plus provides.

To configure this feature, proceed as follows:

1. Create an access-list to define a range of multicast group addresses.

```
access-list  10  permit   232.1.67.0   0.0.0.255
```

2. Enable SSM mapping of IGMPv1/v2 reports.

```
ip igmp ssm-map enable
```

3. Create a mapping which informs the router that all group addresses matching the access list are deemed to come from a certain source address.

```
ip igmp ssm-map static  10  89.156.213.78
```

The effect of this command is that if the router receives any IGMPv1 or IGMPv2 reports for Groups in the 232.1.67.0/24 range, then it will treat those reports as though they were a Source-Specific request, and the requested source was 89.156.213.78. Hence, the router will probably then proceed to join the distribution tree for (89.156.213.78,G), where G is the group requested by the host sending the IGMP packets.

Note the SSM-mapping will not look at IGMPv3 packets. So if the router receives IGMPv3 requests for groups in the 232.1.67.0/24 range, and those requests do not specify just a single source, then the requests will simply be dropped. The SSM-mapping feature is purely to provide backward compatibility to the earlier IGMP versions that did not support the concept of specifying the source from which to receive a stream.

CHAPTER 14

PIMv6

PIMv6 Overview

This is a very short chapter.

The simple fact is that PIM for IPv6 is effectively identical to PIM for IPv4.

In fact, the PIM RFCs (RFC3973 for Dense Mode, RFC4601 for Sparse Mode), explicitly apply to both IPv4 and IPv6. The text of the RFCs is mostly IP-version agnostic; and just occasionally has mention of slight differences (mostly relating to addressing) between implementing PIM on IPv4 and on IPv6.

So, the logic of the operation of the PIM-SM and PIM-DM protocols is the same for both IPv4 and IPv6.

There are a few IPV6-specific aspects of PIM that should be discussed, for reference:

- The multicast destination address used for PIMv6 is ff02::d
- The source address used for PIMv6 packets is the link-local address on the interface via which the packet is transmitted.
- It is possible to embed the Rendezvous Point IPv6 address into IPv6 Multicast Group addresses. The mechanism for achieving this is described in RFC3956. This is a useful mechanism that removes the need for the Bootstrap process, as it enables a PIMv6 router to know immediately the address of the RP from which to obtain the stream to a given group address.
- This is one of the capabilities that is made possible by the sheer size of IPv6 addresses. With all that space to play with, it is conceivable to come up with a scheme for embedding information within an address. In IPv4, where the address is so much shorter, this sort of process is not really possible.
- The IPv6 address range reserved for Source Specific Multicasting is ff3x::/32.

Glossary

Access Control List (ACL)

Address Family

Address Family Identifier

Adjacency

Administrative Distance

Anycast

Area

ARP

Autonomous System Border Router (ASBR)

Autonomous System (AS)

Autonomous System Number (ASN)

Backbone

Bandwidth

Border Gateway Protocol (BGP)

Broadcast

Convergence

Core

Classful Network Address

Designated Router (DR)

Dynamic Host Configuration Protocol (DHCP)

Dijkstra Algorithm

Domain Name Server (DNS)

External Border Gateway Protocol (eBGP)

Equal Cost MultiPath – ECMP

EUI-64

Flooding of LSAs

Flooding Scope for LSAs

Graceful Restart

Graft

Internal Border Gateway Protocol (iBGP)

Internet Protocol (IP)

Key Chain

Latency

Link-local address

Link State Advertisement (LSA)

Metric

Multicast Listener Discovery (MLD)

Multihomed

Multicast

Non-Broadcast Multi-Access (NBMA)

Neighbor Discovery

Neighbor Solicitation

Neighbor Advertisement

Next Hop

Not So Stubby Area (NSSA)

Path Attribute

Point-to-point

Port

Protocol

Prune/Graft

Quality of Service (QoS)

Querier

Redirect

Rendezvous Point

Route Flapping

Route Flap Dampening

Route Redistribution

Route Summarization

Router ID

Router Solicitation

Router Advertisement

Routemap

Service Provider

Solicited Node Address

Split Horizon

Stateless Address Auto Configuration

Subnet

Subnet Mask

Throughput

Unicast

Voice over IP (VoIP)

Note:

See the next page for Glossary definitions.

Glossary - Definitions

Access Control List (ACL)

An Access Control List (ACL) is one filter, or a sequence of filters, that are applied to an interface to either block, pass, or when using QoS, apply priority to, packets that match the filter definitions. ACLs are used to restrict network access by hosts and devices and to limit network traffic.

Address Family

Address Family is really just a succinct way of saying Network Protocol Type. The term Address Family is used in the context of routing protocols that are designed to exchange routes for multiple network protocols. For example, if a routing protocol can carry routes for IP4, IPv6, IPX, Appletalk and others, then it is said to support multiple 'address families'. Also, an implementation of this routing protocol may allow the user to specify configuration options that are different for different address families. In that case, then the configuration options will be split into address-family-specific groupings.

Address Family Identifier

The numbers that identify which protocol a route belongs to are referred to as Address Family identifiers. These identifiers are listed at:

http://www.iana.org/assignments/address-family-numbers/address-family-numbers.xml

Adjacency

This is when two OSPF routers have compatible configurations and have reached at least the two-way exchange of Hello packets. If the routers have gone as far as synchronizing their link-state databases, then they are said to be "*Fully Adjacent*".

Administrative Distance

The administrative distance is a rank given to a route based on the protocol that the route was received from. Administrative distance indicates a level of trustworthiness of a route where the lower the administrative distance the higher the integrity of a route.

Anycast

Anycast is a network addressing and routing scheme where data is routed to the nearest or best destination as viewed by the routing topology. Compared to unicast with a one-to-one association between network address and network endpoint, and multicast with a one-to-many association between network address and network endpoint; anycast has a one-to-many association between network address and network endpoint. For anycast, each destination address identifies a set of receiver endpoints, from which only one receiver endpoint is chosen.

Anycast is often implemented using BGP to simultaneously advertise the same destination IP address range from many sources, resulting in packets address to destination addresses in this range being routed to the nearest source announcing the given destination IP address.

Area

OSPF allows the grouping of networks into a set, called an **Area**. The internal topology of an area is hidden from the rest of the AS. This technique minimizes the routing traffic required for the protocol. When multiple areas are used, each area has its own separate topological database. Routing can be between areas (inter-area routing) or within areas (intra-area routing). A backbone area forms the central path between all other areas. A router that forms a connection between an area and the backbone is known as an Area Border router.

ARP

When unicast packets are delivered onto an Ethernet network, the destination MAC address in the packets must the MAC address of the target device. If a router or host has packets to deliver into an Ethernet, but knows only the IP address of the target, then it needs a way to find out the target's MAC address as well.

ARP (Address Resolution Protocol) is the means by which it finds out the MAC address of a device that possesses a given IP address. The protocol is simple - requests saying "*who has IP address a.b.c.d?*" are broadcast onto the Ethernet segment, and the appropriate host replies with "*I am a.b.c.d, and here is my MAC address*".

The recipient of the reply thereby learns IP-to-MAC mappings on its directly connected Ethernet segment(s), and stores this information in an ARP cache.

Autonomous System Border Router (ASBR)

An OSPF router that has links to non-OSPF routers, and has learnt routes via those routers, is said to be at the border of the OSPF autonomous system. Therefore it is referred to as an Autonomous System Border Router. The key attribute of an ASBR is that it can advertise ASExternal routes into the OSPF network.

Autonomous System (AS)

An autonomous system is defined as a number of networks, all of which share the same routing and administration characteristics.

An AS is either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of an entity (such as a university, a business enterprise, or ISP). An autonomous system is also sometimes referred to as a routing domain. Each autonomous system is assigned a globally unique number called an Autonomous System Number (ASN).

Note that in the context of the OSPF protocol, an autonomous system is the set of routers, under a specific administration, that are all using OSPF. If there are other routers, under the same administration, that are using another routing protocol (e.g. RIP or BGP), then those routers are considered to be outside the OSPF autonomous system.

Autonomous System Number (ASN)

A unique Identification number that is assigned by IANA to each registered autonomous system.

Originally, ASNs had a range from 0 to 65535, but in 2007, the range was extended out to 4,294,967,295.

Backbone

A backbone is the set of (typically) high-bandwidth links at the core of a network.

In OSPF, the term Backbone has a special meaning, as the name given to the Area 0 that must be present in any OSPF network.

Bandwidth

In computer networks, this is a measurement amount of bit-rate or data (in bits) that can be transmitted per second over a particular network connection, or through a system.

Border Gateway Protocol (BGP)

BGP is an exterior gateway protocol that determines the best path in networks, performs optimal routing between multiple autonomous systems or domains, and exchanges routing information with other BGP systems. The RFCs 1771 (BGP4), 1654 (first BGP4 specification), and 1105, 1163, 1267 (older version of BGP) describe BGP and BGP4.

Broadcast

Some network types, notably Ethernet and Token Ring, support the concept of a packet that will be sent to all hosts in a Layer 2 domain. This is referred to as Broadcasting; and a packet that will be sent to all hosts is referred to as a Broadcast packet.

This provides an efficient way for a host to send a message to a collection of other hosts.

Convergence

Over time, a range of types of communication, that had previously used different transport mechanisms, have migrated to using IP as their communication medium.

For example, telephony is migrating to Voice over IP (VoIP), video transmission is moving from RF signaling over coax to IP data transmission, industrial control systems are migrating to IP. This process of multiple different communication systems standardizing on a single medium is termed 'convergence', as the systems are converging on a common communication infrastructure.

Core

See **Backbone**

Classful Network Address

A classful network address is one in which the combination of zero and non-zero bytes conforms to the mask of the network class that the address falls within. (For example, 23.0.0.0 is an example of a classful network address in the Class A range, and 143.245.0.0 is an example of a classful network address in the Class B range.)

Designated Router (DR)

The term "*Designated Router*" is used in both OSPF and PIM.

In an OSPF network, the routers connected to a given subnet hold an election among themselves to be the originator of the Network LSA for the subnet in question. This router is called the Designated Router. It creates the necessary Network LSA, and forms full adjacencies with all its neighbors. Similarly, a Backup Designated Router also forms full adjacencies with all the routers in the subnet. Having only two routers do all this greatly reduces OSPF network traffic.

In PIM, if multiple PIM routers are connected to the same Ethernet segment, then a single one of these routers will be elected to perform the transmission of PIM signaling packets. This is the Designated Router for that segment.

Dynamic Host Configuration Protocol (DHCP)

A method of automatically allocating IP addresses. A DHCP server holds a pool of IP addresses from which it draws individual ones as it allocates them to users when they log on.

Dijkstra Algorithm

The Dijkstra Algorithm is a solution to the "*shortest path*" problem - i.e. it is a method for finding the shortest path between any two nodes in a set of nodes connected by links

The algorithm is used in OSPF to convert the network into a tree that contains the 'shortest' paths to each node. In this application of the algorithm, the definition of shortest is not simply 'the least number of branches from the root'. It associates a length value with each branch, and aims to find the path from the root, to any given node, that involves the least aggregate branch lengths.

Domain Name Server (DNS)

DNS allows you to access remote systems by entering human readable device host names rather than IP addresses. DNS works by creating a mapping between a device name, such as **http://www.alliedtelesis.com**, and its IP address. These mappings are held on DNS servers.

External Border Gateway Protocol (eBGP)

Refers to how a router views a BGP peer relationship, where the peer is in another AS. See also, Internal BGP (iBGP), where the peer is in the same AS.

Equal Cost MultiPath – ECMP

In general, routers forward packets along the least-cost route to their destination. Cost is calculated in different ways for different routing protocols, but is generally a measure of the reliability and bandwidth of a path.

If a router has multiple routes to the same destination, and these routes all have the same cost, then it can choose to use any of these routes. Typically, in this situation, a router will share traffic across these multiple routes, using a sharing algorithm of the router's choosing.

This act of sharing traffic across multiple equal-cost routes is referred to as Equal-Cost Multipath routing.

EUI-64

An algorithm for calculating an IPv6 interface ID from a MAC address.

Flooding of LSAs

OSPF LSAs are made to be shared. Their purpose is to transmit information between OSPF routers. Different types of LSAs have different sets of routers they must be shared with (see Flooding Scope).

Flooding is the process of getting an LSA to every router that is should reach, whether within a single link, one area, or in multiple areas.

Flooding Scope for LSAs

Different types of OSPF LSA are destined to be shared with different sets of routers.

The range of routers that an LSA needs to be shared with is the LSA's flooding scope.

There are 3 levels of Flooding Scope:

- Link-local – the LSA is forwarded only to other routers in the same subnet as the originator
- Area – the LSA is forwarded to all routers in the same area as the originator
- AS-wide – the LSA is forwarded to all routers in the AS (often with some specific exceptions)

Graceful Restart

On occasions, a router needs to restart its process that controls one or other of the routing protocols that it is operating. It is desirable if this restarting of the routing protocol process on the router does not disturb the operation of the network. Graceful Restart is the name given to the mechanisms that are defined in various routing protocols that enable a router and its neighbors to interoperate in such a way that the router can restart a routing protocol process without disturbing the forwarding of user data.

In essence, there are two key components to graceful restart:

1. The router in question informs its neighbors that it will be incommunicado for a brief time while it restarts the process in question, so that the neighbors will not react adversely to the router's temporary lack of responsiveness

2. The router in question does not remove routes from its routing table while it is restarting the routing protocol process, so that it will continue to forward user data.

Thereby, the router can restart its routing protocol process gracefully, and the network achieves non-stop forwarding of user data.

Graft

The paths down which a multicast stream is forwarded in a network will typically form a tree, with the multicast source at the root, and the recipients constituting the leaves. If a router detects that hosts downstream of it desire to receive a given stream, then the router needs to signal up towards the source of the stream, to request receipt of the stream. In this manner, the router is said to Graft a new branch onto the forwarding tree for that stream.

In the PIM-DM protocol, the packets that a router sends to request receipt of a stream are termed Graft packets.

Internal Border Gateway Protocol (iBGP)

A characteristic of a BGP neighbor relationship, specifically when the two routers are internal to the same BGP AS.

Internet protocol (IP)

IP is the communication protocol used throughout the Internet for directing data. All traffic needs to have IP address information for the routers to steer it to the correct destination.

Key Chain

When you want your routers to automatically change the authentication key they are using, you need to set up a set of keys that they will work through over time, and tell the routers when to change from using one key in the set to using the next key.

This combination of a set of keys and the set of times for changing keys is referred to as a **key chain**.

Latency

The time that information takes to move across a network. For voice networks, latency is the delay from when a word is spoken to the time it is heard by the listener. Low latency is usually highly desirable for real-time applications such as voice and video.

Link-local address

A link-local address is an IP (Internet Protocol) address that is only used for communications in the local network, or for a point-to-point connection. Routing does not forward packets with link-local addresses. IPv6 requires a link-local address is assigned to each interface, which has the IPv6 protocol enabled.

Link State Advertisement (LSA)

At the core of the OSPF protocol is the exchange, between OSPF routers, of information about the immediate network environment of each router. The routers inform each other of what links they are connected to, the characteristics of those links, and addresses in use on those links. From this information, routers can build up a map of the whole network, and then compute the paths to given destinations.

Because the key information being exchanged is the state of the links that are attached to each router, the data structures that are used to carry this information are called Link State Advertisements, or LSAs.

There are a number of categories of LSA used in OSPF, with each category carrying a different type of information about the network. In fact, not all types of LSA strictly carry Link State information; some actually just carry route entries. But, the name LSA has been applied to all the categories, for the sake of consistent terminology.

Metric

A metric is a number that a routing protocol uses to represent the desirability of a route. Routing protocols have to decide on the best (most desirable) route to a given destination. So, an algorithm is applied to the routes in a routing protocol's route table, and thereby calculate a metric for each route. If there are multiple routes to the same subnet, then best route is the one with the lowest metric.

The algorithms used by routing protocols to calculate metric vary widely. RIP just uses a count of the number of hops to the destination. OSPF sums up the costs of the links on the path to the destination, where a link's cost is proportional to the inverse its bandwidth. Some protocols use a range of variables like utilization, latency, reliability, MTU, loss rate on the links in the path.

Multicast Listener Discovery (MLD)

Multicast Listener Discovery (MLD) enables each IPv6 router to discover the presence of multicast listeners (that is, nodes wishing to receive multicast packets) on its directly attached links, and to discover specifically which multicast addresses are of interest to those neighboring nodes.

Multihomed

Connecting a network, or AS, to multiple other networks. When the other networks are ISPs, then BGP can provide some measure of redundancy in the event that any one link fails.

Multicast

One device sends out data that is intended to be received and processed by a selected group of the devices it reaches.

Non-Broadcast Multi-Access (NBMA)

Network providers often provide a WAN network as a 'cloud' of switching infrastructure, to which multiple sites are connected by a single physical link from each site into the cloud. Popular protocols for this type of service are Frame Relay, ATM, and X.25.

Each of these protocols supports the concept of 'logical channels' via which each router communicates to another router. From a router's point-of-view, the single physical link into the provider's cloud appears as a bundle of logical channels, with a single router at the other end of each channel.

The name Non-Broadcast Multi-Access network refers to the fact that multiple routers have access to each other via the cloud, but there is no concept of a broadcast packet that will be forwarded to all the connected routers. Communication between routers must be performed in a unicast manner, via each separate logical channel.

Neighbor Discovery

Neighbor discovery is an ICMPv6 function that allows a router or host to identify other devices on its links. It is the IPv6 equivalent of ARP.

Neighbor Solicitation

Neighbor solicitation is the process whereby IPv6 hosts request information about their neighbors. The Neighbor Solicitation message allows a device to check that a neighbor exists and is reachable, and to initiate address resolution.

Neighbor Advertisement

Neighbor advertisements are used by nodes to respond to a Neighbor Solicitation message. The Neighbor Advertisement message confirms the existence of a host or router, and also provides Layer 2 address information when needed.

Next Hop

IP routing involves forwarding packets from one router to the next, until they reach their destination. Routers do not need to know the full path to a packet's destination; they just need to know the next router to forward the packet on to. This 'next router' is referred to as the next hop of an IP route.

Not So Stubby Area (NSSA)

OSPF areas that have some of the properties of a Stub Area, but not all. These areas:

- Do receive Summary LSAs from other areas.
- Do contain ASBRs. So, External LSAs are generated within the area. The External LSAs generated in an NSSA are Type-7 LSAs.
- Can advertise their External LSAs to other areas (translated to Type-5 LSAs)
- Do not receive External LSAs from other areas.

Path Attribute

A variety of parameters that are associated with routes and exchanged in BGP routing updates. BGP has an elaborate best path algorithm that is controlled by these path attributes, and allows network engineers flexibility in how routers choose the best BGP routes.

Point-to-point

A connection between routers when there is no subnet between them. In an OSPF point-to-point network, a direct Layer 3 connection can exist between a single pair of OSPF routers.

Port

The connection point for a communications line, such as Cat-6 or fiber optic cables, into a network router or switch.

Protocol

Set of rules or procedures that enable two or more systems to exchange information.

Prune/Graft

When discussing multicasting, it is common to talk of the multicast data being sent down a distribution **tree**. When a router realizes that nothing downstream of it needs to receive a given group, it can inform its upstream neighbor that it no longer wishes to receive that group. The router is said to **Prune** itself off the distribution tree for that group.

Similarly, when it finds that once again something downstream of it needs to receive that group, it has to request that its upstream neighbor sends it that group. The router is then said to **Graft** itself back onto the distribution tree for that group.

Quality of Service (QoS)

QoS enables you to both prioritize traffic and limit its available bandwidth. The concept of QoS is a departure from the original networking protocols, in which all traffic on the Internet or within a LAN had the same available bandwidth. Without QoS, all traffic types are equally likely to be dropped if a link becomes oversubscribed.

This approach is now inadequate in many networks, because traffic levels have increased and networks often carry time-critical applications such as streams of realtime video data. QoS also enables service providers to easily supply different customers with different amounts of bandwidth. Configuring Quality of Service involves two separate stages: Classifying traffic into flows, according to a wide range of criteria, and then acting on these traffic flows.

Querier

In Layer 2 multicasting, this is the switch that is specifically designated the task of sending out IGMP queries (packets which periodically ask receivers "*do you still wish to receive the stream(s) you have requested?*").

Redirect

Directing a host to a better gateway for a given destination. In other words, routers may inform hosts of a better first hop router for a destination.

Rendezvous Point

PIM Sparse Mode requires specific designated routers to receive notification of all streams destined to specific ranges of multicast addresses (or, possibly, all multicast addresses).

When a router needs to get hold of a given group, it sends a request to the designated Rendezvous Point for that group (and all the routers in between take note of this request, thereby setting up a forwarding path down to the requesting router). If there is a source in the network that is transmitting a stream to this group, then the Rendezvous Point will be receiving this stream, and will be able to forward it to the requesting router.

Route Flapping

If information being advertized about a route is frequently changing, the route is said to be flapping. The flapping may take the form of the route appearing and disappearing frequently, or may take the form of the path to the route changing frequently.

Route Flap Dampening

A feature has been developed within BGP that helps reduce the effect of flapping routes.

The way that Route Flap Dampening works is as follows:

If a router is configured to perform Route Flap Dampening, it keeps track of every route it has learnt from eBGP neighbors. Each route has a **Flapping Penalty value** associated with it. This penalty value is initialized to **zero**.

Each time a route is withdrawn, due to receiving a withdraw request from a neighbor, the penalty value is increased by 1000. Although the route is removed from the BGP route table when the withdraw event occurs, BGP still keeps a record of this route, so that its flap penalty value is remembered. There is no addition to the penalty when the route is learnt again; the penalty is only incremented each time the route is withdrawn.

If the penalty value on a route gets above a certain value, called the **Suppress Limit**, then the router stops advertising the route.

Route Redistribution

The act of importing routes from one routing protocol to another. E.g. taking routes learnt by BGP and importing them into OSPF and advertising them to OSPF neighbors.

Route Summarization

This is when routers receive information about a number of individual subnets, but will advertise a single route that encompasses the full address range that covers all those individual subnets.

Router ID

Within some routing protocols, every router in the network requires a unique indentifier. This is referred to as the routerID.

Router IDs typically take the form of an IPv4 address (even for IPv6 routing protocols), and can be explicitly configured, or automatically chosen from among the IP4 addresses on the routers interfaces.

Router Solicitation

IPv6 hosts can automatically locate routers on the LAN. The host achieves this by using two different ICMPv6 messages. They are:

- Router Solicitation (ICMPv6 Type 133)
- Router Advertisement (ICMPv6 Type 134)

When a host is first connected to a LAN, it will send an IPv6 Router Solicitation packet to request information about routers on the network. Each router which is active on the LAN will respond to this packet by sending a Router Advertisement (RA) with its address to all nodes in the group. It informs the host what network address(es) is(are) in use on the subnet. It also informs the host if it is a default gateway.

Router Advertisement

When an IPv6 router receives Router Solicitation packets, it replies with Router Advertisements.

As well as responding to Router Solicitation events, a router will also send out Router Advertisements packets at regular intervals.

Routemap

These are a structured combination of match criteria and actions. They can be used to filter out routes and also to alter the attributes in Update messages.

There are various levels of structure within a route map:

- A route map is an entity with a name
- Each route map consists of multiple entries, identified by sequence numbers
- Each entry can consist of multiple clauses

In effect, an entry defines an individual filter. It can have a **match** clause that defines what it will match on, and it can have multiple **set** clauses that can specify actions to be taken.

Service Provider

A company that provides organizations with communications, storage, and many other services. Service Providers generally refer to third-party or outsourced suppliers, including telecommunications service providers (TSPs), application service providers (ASPs), storage service suppliers (SSPs), and Internet service providers (ISPs).

Solicited Node Address

A special multicast address, used in neighbor discovery, that narrows down the number of hosts that need to process the request.

Split Horizon

Split Horizon eliminates the slow process of eliminating dead routes and also preventing routing loops, by not sending routing information back in the direction from which it came. Split Horizon can improve convergence time in complex, highly-redundant environments.

Stateless Address Auto Configuration

A process whereby an enabled IPv6 host can work out its own IPv6 address. When IPv6 has been enabled on a network interface, the device can automatically configure itself with an IPv6 address by using ICMPv6 router discovery messages. This is known as Stateless Address Auto-Configuration (SLAAC). SLAAC occurs when a host configures its own address—the address is **generated**, not allocated.

Subnet

The logically visible subdivision of an IP network. The practice of dividing a network into two or more networks is called subnetting.

Subnet Mask

An IP address has two components, the **network** address and the **host** address. A subnet mask separates the IP address into the network and host addresses.

Throughput

Amount of useful data bandwidth transported through a system. Usually measured in Kbps, Mbps, or Gbps.

Unicast

In unicast, data is sent to a single network destination identified by a unique address. This is in contrast to broadcast, where the same data is sent to all possible destinations or broadcast where data is sent to only interested destinations.

Voice over IP (VoIP)

Phone service over the Internet